ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ:Ηλεκτρονικής και Υπολογιστών
ΕΡΓΑΣΤΗΡΙΟ ΣΥΣΤΗΜΑΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία

της φοιτήτριας του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών της Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών

Χρυσοχού Αγγελική του Νικολάου

Αριθμός Μητρώου: 5167

Θέμα

«Αξιοποιώντας το Real-Time Linux σε Ενσωματωμένα Συστήματα»

Επιβλέπων
Αναπληρωτής Καθηγητής κ. Θραμπουλίδης Κλεάνθης

Αριθμός Διπλωματικής Εργασίας:

Πάτρα, Σεπτέμβριος 2010

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διπλωματική Εργασία με θέμα

«Αξιοποιώντας το Real-Time Linux σε Ενσωματωμένα Συστήματα»

Της φοιτήτριας του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Χρυσοχού Αγγελική του Νικολάου

Αριθμός Μητρώου: 5167

Παρουσιάστηκε δημόσια και εξετάστηκε στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις 10/09/2010

Ο Επιβλέπων

Αναπληρωτής Καθηγητής
κ. Κλεάνθης Θραμπουλίδης

Ο Διευθυντής του Τομέα

Καθηγητής κ. Ευθύμιος Χούσος

Αριθμός Διπλωματικής Εργασίας:

Θέμα: «Αξιοποιώντας το Real-Time Linux σε Ενσωματωμένα Συστήματα»

Φοιτήτρια: Επιβλέπων:

Χρυσοχού Αγγελική **κ. Κλεάνθης Θραμπουλίδης**

Περίληψη

Τα τελευταία χρόνια τα ενσωματωμένα συστήματα πραγματικού χρόνου χρησιμοποιούνται σε ολοένα μεγαλύτερη γκάμα εφαρμογών. Κινητά τηλέφωνα, συσκευές αναπαραγωγής ψηφιακών δίσκων, εκτυπωτές, ψηφιακές φωτογραφικές μηχανές, αποτελούν λίγα παραδείγματα συσκευών που χρησιμοποιούν την τεχνολογία των ενσωματωμένων συστημάτων. Στην παρούσα διπλωματική εργασία ερευνήθηκαν οι δυνατότητες αξιοποίησης του Real-Time Linux σε ενσωματωμένα συστήματα.

Μελετήθηκε διεξοδικά ο πυρήνας του λειτουργικού συστήματος Linux και ο αλγόριθμος χρονοδρομολόγησης που χρησιμοποιεί. Αναζητήθηκαν τρόποι μετατροπής του Linux σε λειτουργικό σύστημα για συστήματα πραγματικού χρόνου. Για το σκοπό αυτό, μελετήθηκε η έννοια του πραγματικού χρόνου όσον αφορά την κατασκευαστική διάσταση του συστήματος και το πρόγραμμα της εφαρμογής. Μιά κατηγορία συστημάτων παραγματικού χρόνου, είναι τα λειτουργικά συστήματα με δυνατότητες πραγματικού χρόνου. Αυτά οργανώνουν και καθορίζουν την χρήση των πόρων των συστημάτων ώστε να είναι ιδανική για εφαρμογές πραγματικού χρόνου πέρα από το περιβάλλον για ανάπτυξη και εκτέλεση των προγραμμάτων που παρέχουν.

Γίνεται αναφορά στην έννοια της πολυεπεξεργασίας (multiprocessing), της χρήσης δηλαδή δύο ή και παραπάνω κεντρικών μονάδων επεξεργασίας (CPU) σε ένα υπολογιστικό σύστημα. Λόγω της αύξησης της θερμοκρασίας στα ηλεκτρονικά κυκλώματα με την αύξηση των ταχυτήτων ρολογιού στις κεντρικές μονάδες επεξεργασίας και την μείωση του μεγέθους των ηλεκτρονικών, η ταχύτητα του ρολογιού αλλά και η μείωση του μεγέθους περιορίζονται ώστε να υπάρχει αντοχή στις θερμοκρασίες που αναπτύσσονται. Το γεγονός αυτό σε συνδυασμό με την ιδέα του multiprocessing οδήγησε στην ιδέα των πολλαπλών ανεξάρτητων πυρήνων ανά κεντρική μονάδα επεξεργασίας (multicore systems) για βελτίωση της αποδοτικότητας, ακόμα και των συστημάτων ευρείας κατανάλωσης.

Η διαφορά των πολυπύρηνων (multicore) συστημάτων με τα multiprocessing συστήματα, έγκειται στην συνύπαρξη των πυρήνων σε ένα ολοκληρωμένο κύκλωμα (chip) αντί για πολλές κεντρικές μονάδες επεξεργασίας και η ομοιότητα τους, στο ότι ουσιαστικά τα multicore συστήματα εξομοιώνουν λειτουργίες πολυεπεξεργασμού. Πλεόν, πολλά ενσωματωμένα συστήματα διαθέτουν πολυπύρηνους επεξεργαστές καθιστώντας απαραίτητη την ανάπτυξη λειτουργικών συστημάτων πραγματικού χρόνου που να αξιοποιούν στο έπακρο τις δυνατότητες τους.

Δοκιμάστηκε και αξιολογήθηκε η επέκταση ASMP-Linux που αποτελεί έναν τρόπο μετατροπής του Linux σε λειτουργικό σύστημα πραγματικού χρόνου. Το ASMP-Linux αξιοποιεί τις ικανότητες πολυεπεξεργασίας ενός συστήματος με τη δυνατότητα δημιουργίας διαμερισμάτων πραγματικού χρόνου σε κάθε στοιχείο επεξεργασίας. Αναπτύχθηκε εφαρμογή αξιολόγησης σε γλώσσα προγραμματισμού C. Τα αποτελέσματα μελετήθηκαν διεξοδικά μέσω γραφημάτων και εξαγωγής στατιστικών μέτρων όπως η μέση τιμή και η τυπική απόκλιση.

Μελετήθηκε η περίπτωση χρήσης της εφαρμογής ελέγχου του Festo MecLab, που αναπτύχθηκε από τον διδακτορικό φοιτητή Γεώργιο Δούκα. Το Festo MecLab αποτελεί μια προσομοίωση γραμμής παραγωγής με σταθμούς στοίβαξης, μεταφοράς και χειρισμού. Η εφαρμογή ελέγχου εκτελέστηκε επιτυχώς στο λειτουργικό σύστημα πραγματικού χρόνου ASMP-Linux και περιγράφεται στο παράρτημα Α.

Στα πλαίσια της διπλωματικής εργασίας εκτελέστηκε στο Τεχνολογικό Πανεπιστήμιο της Βιέννης μέσω του προγράμματος Erasmus project που αφορά την ανίχνευση και χρονοσφράγιση προγραμμάτων ενσωματωμένων συστημάτων. Η τεχνική αναφορά του project παρατίθεται στο παράρτημα Β.

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών, υπό την επίβλεψη του αναπληρωτή καθηγητή κ. Κλεάνθη Θραμπουλίδη, με την βοήθεια των μελών της ομάδας Software Engineering Group (SEG) και των μεταπτυχιακών φοιτητών του εργαστηρίου συστημάτων υπολογιστών.

Ευχαριστίες αποδίδονται στον κ. Κλεάνθη Θραμπουλίδη για την καθοδήγηση και υποστήριξη που προσέφερε. Ευχαριστίες αποδίδονται και στον διδακτορικό φοιτητή του τμήματος Γεώργιο Δούκα για την συμβολή του και παροχή βοήθειας στην εκπόνηση της συγκεκριμένης εργασίας, αλλά και στους μεταπτυχιακούς και διδακτορικούς φοιτητές του εργαστηρίου στο οποίο εκπονήθηκε η εργασία.

Τέλος, ευχαριστούμε πολύ τον βοηθό καθηγητή Raimund Kirner του Real-Time Systems Group του τεχνολογικού πανεπιστημίου της Βιέννης (Technische Universitaet Wien) για την καθοδήγηση που παρείχε κατά την εκπόνηση του project που παρουσιάζεται στο παράρτημα Β και αφορά την ανίχνευση και χρονική ανάλυση εφαρμογών σε ενσωματωμένα συστήματα. Το project πραγματοποιήθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας μέσω του προγράμματος Erasmus και των επαφών που διατηρεί με το συγκεκριμένο πανεπιστήμιο ο επιβλέπων καθηγητής.

Εισαγωγή

Ο όρος ενσωματωμένα συστήματα χρησιμοποιείται για να περιγράψει το συνδυασμό hardware και software καθώς και μηχανικών ή ηλεκτρονικών περιφερειακών για την επίτευξη μιας συγκεκριμένης λειτουργίας ή μιας ορισμένης ομάδας λειτουργιών. Συνήθως αποτελούν μέρος μεγαλύτερων συστημάτων προκαλώντας τους μηχανικούς να βελτιστοποιήσουν το σύστημα τους ώστε να επιτευχθεί μείωση του κόστους και του μεγέθους με παράλληλη αύξηση της αξιοπιστίας και της απόδοσης. Τα μεγέθη των ενσωματωμένων συστημάτων κυμαίνονται από πολύ μικρές συσκευές, για παράδειγμα ένα ψηφιακό ρολόι χειρός, μέχρι συστήματα ελέγχου των μονάδων παραγωγής πυρηνικής ενέργειας. Μπορούμε να έχουμε συστήματα χαμηλής πολυπλοκότητας, με ένα μόνο ολοκληρωμένο κύκλωμα μικροελεγκτή, ή υψηλής με πολλές μονάδες επεξεργασίας, περιφερειακά και πρόσβαση σε δίκτυο. Πρακτικά, μπορεί κανείς να συναντήσει την τεχνολογία των ενσωματωμένων συστημάτων οπουδήποτε. Ο φούρνος μικροκυμάτων, το πλυντήριο ρούχων, το αυτοκίνητο, περιέχουν όλα ενσωματωμένα υπολογιστικά συστήματα για τον έλεγχο τουλάχιστον μιας λειτουργίας, όπως η ρύθμιση των στροφών του κάδου πλύσης ανάλογα με το επιλεγόμενο πρόγραμμα, ή ηλεκτρονικά συστήματα υποβοήθησης της οδήγησης.

Ένα σύστημα πραγματικού χρόνου (real time system) είναι ένα υπολογιστικό σύστημα στο οποίο η σωστή λειτουργία και συμπεριφορά εξαρτάται όχι μόνο από την ορθότητα των υπολογισμών, αλλά και από το χρονικό διάστημα που απαιτείται για να παραχθούν τα αποτελέσματα τους.Στα συστήματα αυτά, ο χρόνος είναι ουσιώδης. Το λογικό αποτέλεσμα μιας εντολής που δίνεται εκτός των επιθυμητών χρονικών ορίων λαμβάνεται συνήθως ως λάθος ή αποτυχία του συστήματος. Η αποτυχία ενός συστήματος ορίζεται όταν ένα σύστημα δεν ικανοποιεί μια ή και παραπάνω από τις απαιτήσεις που δίνονται για τον ορισμό του. Τα συστήματα πραγματικού χρόνου μπορεί να είναι ενσωματωμένα ή αντιδραστικά (reactive) συστήματα. Μια κατηγοριοποίηση των συστημάτων αυτών είναι σε αυστηρό (hard), ελαστικό (soft) και σταθερό (firm) πραγματικό χρόνο ανάλογα με το πόσο αυστηρά είναι τα χρονικά όρια αναμονής της απόκρισης του συστήματος.

Η έννοια της πολυεπεξεργασίας αναφέρεται είτε στην ύπαρξη δύο ή και παραπάνω κεντρικών μονάδων επεξεργασίας σε ένα σύστημα με την ικανότητα της κατανομής διεργασιών σε αυτές, είτε στην ταυτόχρονη εκτέλεση πολλών διαδικασιών λογισμικού, οπότε ονομάζεται πολυπρογραμματισμός (multiprogramming, multitasking). Ένα σύστημα μπορεί να έχει είτε μόνο την ιδιότητα της πολυεπεξεργασίας είτε μόνο του πολυπρογραμματισμού ή και τις δύο ή καμία από τις δύο. Η πολυεπεξεργασία υλοποιείται είτε με αυτόνομες μονάδες εργασίας, ή όπως εξελίχθηκε, με πολλούς μικροεπεξεργαστές (πυρήνες) σε ένα chip, δηλαδή με πολυπύρηνους (multicore) επεξεργαστές. Η τεχνική αυτή ονομάζεται και Chip Multiprocessing ή CMP και η εξέλιξη σε αυτήν ήταν κατά κάποιο τρόπο απαραίτητη, μιας και η αύξηση της συχνότητας ρολογιού ενός επεξεργαστή προκαλεί την αύξηση της θερμοκρασίας, ενώ απαιτεί την μείωση του μεγέθους τρανζίστορ με αποτέλεσμα την μείωση της ικανότητας λειτουργίας σε αυξημένες θερμοκρασίες.

Το Linux είναι ένα ελεύθερο τύπου-Unix λειτουργικό σύστημα που αρχικά δημιουργήθηκε από τον Linus Torvalds στο Πανεπιστήμιο του Ελσίνκι το 1991. Είναι μια ανεξάρτητη POSIX υλοποίηση και στα χαρακτηριστικά του συμπεριλαμβάνει πραγματικό multitasking, πραγματικό πολυχρηστικό περιβάλλον, virtual memory, shared libraries, demand loading, TCP/IP networking και πολλά άλλα χαρακτηριστικά. Το Linux μπορεί να χρησιμοποιηθεί για πολλούς σκοπούς, όπως το networking, η ανάπτυξη προγραμμάτων, ακόμα και για πλατφόρμα για απλούς χρήστες.

Το ASMP-Linux (Asymmetric Multiprocessor Linux) αποτελεί μια τροποποιημένη εκδοχή του πυρήνα του Linux που μπορεί να να χρησιμοποιηθεί σε συστήματα πολυεπεξεργασίας, καθώς και σε ενσωματωμένα συστήματα με απαιτήσεις πραγματικού χρόνου. Αναπτύχθηκε το 2007 από την ομάδα προγραμματισμού συστημάτων του πανεπιστήμιου της Ρώμης “Tor Vergata”. Το ASMP-Linux είναι ένα λειτουργικό σύστημα ανοιχτού κώδικα που χρησιμοποιεί μια ασύμμετρη προσέγγιση του πυρήνα. Σε ένα συμβατικό συμμετρικό πυρήνα, οι συσκευές εισόδου/εξόδου και οι επεξεργαστές θεωρούνται ομοειδή, εφόσον το φορτίο του συστήματος δεν λαμβάνεται υπόψιν. Οι ασύμμετροι πυρήνες αντίθετα, θεωρούν προνομιακές τις διαδικασίες πραγματικού χρόνου και τις συσκευές που χρησιμοποιούν ή χρειάζονται και τις προφυλάσσουν από τις υπόλοιπες δραστηριότητες του συστήματος.

Η διπλωματική εργασία έχει ως στόχο τη μελέτη του Linux κυρίως όσον αφορά τη χρήση του σαν λειτουργικό σύστημα πραγματικού χρόνου και τις δυνατότητες αξιοποίησης του στην ανάπτυξη ενσωματωμένων συστημάτων. Στο πλαίσιο αυτό, παρουσιάζεται η εγκατάσταση του ASMP-Linux στον εξυπηρετητή IBM x3455. Η εγκατάσταση πραγματοποιήθηκε κάνοντας compile εναν πυρήνα Linux έκδοσης 2.6.19-1. Ακόμα, αναπτύχθηκε εφαρμογή αξιολόγησης του σαν λειτουργικό σύστημα αυστηρού πραγματικού χρόνου. Πιο συγκεκριμένα:

Στο **κεφάλαιο 1** παρουσιάζονται οι βασικές έννοιες των ενσωματωμένων συστημάτων. Μελετώνται οι προκλήσεις που αφορούν στον προγραμματισμό αυτών υπό την έλλειψη συνηθισμένων διεπαφών χρήστη όπως οθόνη, στο μέγεθος και την ποιότητα των ηλεκτρονικών που χρησιμοποιούνται στην κατασκευή τους με την απαίτηση για μείωση του μεγέθους και αύξηση της αντοχής για λειτουργία σε αντίξοες συνθήκες των ενσωματωμένων συστημάτων. Επιπλέον μελετώνται ο σχεδιασμός και ο έλεγχος των ενσωματωμένων συστημάτων.

Το **κεφάλαιο 2** ασχολείται με τον ορισμό και περιγραφή της έννοιας του πραγματικού χρόνου και την εφαρμογή αυτών σε συστήματα τα οποία ως επί το πλείστον είναι και ενσωματωμένα. Ο “πραγματικός χρόνος” κατηγοριοποιείται σε soft real time, ή ελαστικό πραγματικό χρόνο και hard real time, ή αυστηρό πραγματικό χρόνο. Η διαφορά των δύο βρίσκεται στην τήρηση των χρονικών ορίων, και στα χρονικά όρια, πόσο μεγαλύτερα όσον αφορά την διάσταση του χρόνου είναι ή πόσο μεγάλο περιθώριο λάθους ορίζεται ως επιτρεπτό. Παρουσιάζονται ακόμα βασικές έννοιες των λειτουργικών συστημάτων πραγματικού χρόνου.

Το **κεφάλαιο 3** αναφέρεται στα πολυεπεξεργαστικά συστήματα (multiprocessing systems), στα υπολογιστικά συστήματα δηλαδή που περιέχουν δύο ή και παραπάνω κεντρικών μονάδων επεξεργασίας (CPU). Στις μονάδες επεξεργασίας κατανέμονται οι διάφορες διεργασίες προς εκτέλεση με βάση κατάλληλους αλγορίθμους προγραμματισμού και χρονοδρομολόγησης (scheduling). Περιγράφονται οι κατηγορίες των συστημάτων πολυεπεξεργασίας, όπως η συμμετρική και ασύμμετρη πολυεπεξεργασία. Παρουσιάζονται τα πολυπύρηνα συστήματα και ο ρόλος που αυτά παίζουν στα ενσωματωμένα συστήματα πραγματικού χρόνου. Τέλος παρουσιάζονται οι προκλήσεις που εμφανίζονται κατά τον προγραμματισμό και χρονοδρομολόγηση των ενσωματωμένων συστημάτων.

Στο **κεφάλαιο 4** αναφερόμαστε στο λειτουργικό σύστημα Linux που είναι ένα ελεύθερο λογισμικό ανοικτού κώδικα λειτουργικό σύστημα που βασισμένο στον Linux kernel. Το κεφάλαιο αυτό περιγράφει πως αυτό το λειτουργικό σύστημα μπορεί να χρησιμοποιηθεί στην ανάπτυξη ενσωματωμένων συστημάτων και δίνει έμφαση στα πλεονεκτήματα που μπορεί να προσφέρει. Τέλος αναλύεται διεξοδικά ο αλγόριθμος χρονοδρομολόγησης που χρησιμοποιεί το Linux και αναφέρονται οι δυνατότητες του Linux σαν λειτουργικό σύστημα πραγματικού χρόνου για ενσωματωμένα συστήματα.

Στο **κεφάλαιο 5** περιγράφεται το πρακτικό μέρος της παρούσας διπλωματικής εργασίας. Μετά από έρευνα πάνω στις λύσεις που προσφέρονται για λειτουργικά συστήματα πραγματικού χρόνου βασισμένα στο Linux, αποφασίστηκε να δοκιμαστεί η επέκταση ASMP-Linux που προσδίδει σύμφωνα με τους δημιουργούς της χαρακτηριστικά αυστηρού πραγματικού χρόνου στον πυρήνα του Linux. Εγκαταστάθηκε η προαναφερθείσα επέκταση στον εξυπηρετητή IBM x3455 που υπάρχει στο εργαστήριο συστημάτων υπολογιστών και αναπτύχθηκε εφαρμογή αξιολόγησης των δυνατοτήτων του. Αναλύονται τα αποτελέσματα της αξιολόγησης αυτής.

Στο **κεφάλαιο 6** παρατίθενται τα συμπεράσματα της διπλωματικής εργασίας καθώς και ιδέες για μελλοντική έρευνα και ενασχόληση.

Το **παράρτημα Α** αφορά την μελέτη περίπτωσης χρήσης του λειτουργικού συστήματος πραγματικού χρόνου ASMP-Linux. Εκτελέστηκε στο ASMP-Linux η εφαρμογή ελέγχου του συστήματος Festo MecLab που αποτελεί μια προσομοίωση μιας γραμμής παραγωγής. Η εφαρμογή ελέγχου αναπτύχθηκε στα πλαίσια του διδακτορικού του Γεωργίου Δούκα.

Στο **παράρτημα Β** παρουσιάζεται η τεχνική αναφορά που αποτελεί εργασία της συντάκτριας πραγματοποιημένη στο Τεχνολογικό Πανεπιστήμιο της Βιέννης (Technische Universitaet Wien) ως μέρος της παρούσας διπλωματικής εργασίας.

Πίνακας Περιεχομένων

1 Ενσωματωμένα Συστήματα

1.1 Εισαγωγή

Ο όρος ενσωματωμένα συστήματα χρησιμοποιείται για να περιγράψει το συνδυασμό hardware και software καθώς και μηχανικών ή ηλεκτρονικών περιφερειακών για την επίτευξη μιας συγκεκριμένης λειτουργίας ή μιας ορισμένης ομάδας λειτουργιών. Συνήθως αποτελούν μέρος μεγαλύτερων συστημάτων προκαλώντας τους μηχανικούς να βελτιστοποιήσουν το σύστημα τους ώστε να επιτευχθεί μείωση του κόστους και του μεγέθους με παράλληλη αύξηση της αξιοπιστίας και της απόδοσης. [1]

Αντίθετα με ένα ενσωματωμένο σύστημα, ένας προσωπικός υπολογιστής ή υπολογιστής γενικού σκοπού, ο οποίος συνδυάζει με παρόμοιο τρόπο hardware, software και περιφερειακά, μπορεί να επιτελέσει πολλές διαφορετικές λειτουργίες με την χρήση κατάλληλου λογισμικού. Έτσι, μετατρέπεται σε επεξεργαστή κειμένου ή κέντρο πολυμέσων ανάλογα με τις προτιμήσεις του χρήστη. Η διαφορά των δύο έγκειται ακριβώς στην δέσμευση επίτευξης μιας συγκεκριμένης λειτουργίας, παρά στο μέγεθος τους. Επιπλέον ένας προσωπικός υπολογιστής μπορεί να απαρτίζεται από υποσυστήματα τα οποία είναι ενσωματωμένα όπως ένα ασύρματο ποντίκι και ένας δρομολογητής δικτύου.

Η διάκριση μεταξύ των δύο δεν είναι πάντα ξεκάθαρη εφόσον πολλά ενσωματωμένα συστήματα παρέχουν κάποιες δυνατότητες προγραμματισμού ενώ συστήματα όπως οι υπολογιστές χειρός μοιράζονται στοιχεία με αυτά, λειτουργικά συστήματα, μικροεπεξεργαστές, δεν ανήκουν όμως στην οικογένεια των ενσωματωμένων συστημάτων.

Τα μεγέθη των ενσωματωμένων συστημάτων κυμαίνονται από πολύ μικρές συσκευές, για παράδειγμα ένα ψηφιακό ρολόι χειρός, μέχρι συστήματα ελέγχου των μονάδων παραγωγής πυρηνικής ενέργειας. Μπορούμε να έχουμε συστήματα χαμηλής πολυπλοκότητας, με ένα μόνο ολοκληρωμένο κύκλωμα μικροελεγκτή, ή υψηλής με πολλές μονάδες επεξεργασίας, περιφερειακά και πρόσβαση σε δίκτυο.

Πρακτικά, μπορεί κανείς να συναντήσει την τεχνολογία των ενσωματωμένων συστημάτων οπουδήποτε. Ο φούρνος μικροκυμάτων, το πλυντήριο ρούχων, το αυτοκίνητο, περιέχουν όλα ενσωματωμένα υπολογιστικά συστήματα για τον έλεγχο τουλάχιστον μιας λειτουργίας, όπως η ρύθμιση των στροφών του κάδου πλύσης ανάλογα με το επιλεγόμενο πρόγραμμα, ή ηλεκτρονικά συστήματα υποβοήθησης της οδήγησης. Κινητά τηλέφωνα, συσκευές αναπαραγωγής ψηφιακών δίσκων, εκτυπωτές, ψηφιακές φωτογραφικές μηχανές, αποτελούν λίγα ακόμα παραδείγματα συσκευών που χρησιμοποιούν την τεχνολογία των ενσωματωμένων συστημάτων.

Πιο εξελιγμένα συστήματα βρίσκονται σε τομείς όπως η ιατρική, αεροπορία, ναυτιλία και βιομηχανία. Πολύπλοκες μονάδες παραγωγής με διάφορους βαθμούς αυτοματοποίησης και ρομποτικούς μηχανισμούς αποτελούνται συνήθως από ένα πλήθος ενσωματωμένων συστημάτων, ενώ στην ιατρική απαιτείται απρόσκοπτη και αποτελεσματική λειτουργία των συσκευών που χρησιμοποιούνται, όπως ένας αξονικός ή μαγνητικός τομογράφος, η παραγωγή ενισχυμένου φωτός με εξαναγκασμένη εκπομπή ακτινοβολίας (laser). Λόγω του μικρού μεγέθους και της αυξημένης αξιοπιστίας, πολλά ενσωματωμένα συστήματα απαρτίζουν συνήθως αυτές τις συσκευές. Ένας ακόμα τομέας που απαιτεί εκτεταμένη χρήση τέτοιων συστημάτων, είναι η εξερεύνηση του διαστήματος.

Ενσωματωμένα συστήματα κατασκευάζονται που να είναι ανθεκτικά στην έλλειψη βαρύτητας, σε διαφόρου είδους ακτινοβολίες, ακραίες θερμοκρασίες, να έχουν χαμηλή κατανάλωση ενέργειας και μικρό μέγεθος. Έτσι μη επανδρωμένες αποστολές καθίστανται δυνατές.

Συνοψίζοντας τα γενικά χαρακτηριστικά ενός ενσωματωμένου συστήματος:

- Είναι ένα σύστημα κατασκευασμένο να εκτελεί τη λειτουργία του χωρίς η με μερική ανθρώπινη παρέμβαση.
- Είναι βελτιστοποιημένο ώστε να επιτελεί τις εργασίες με τον πιο αποδοτικό τρόπο.
- Αλληλεπιδρά με στοιχεία του περιβάλλοντος, όπως αισθητήρες θερμοκρασίας και πίεσης ή το σύστημα ελέγχου και οδήγησης ενός κινητήρα.
- Η ύπαρξη επεξεργαστή και προγράμματος συνήθως αποκρύπτεται από των χρήστη των συσκευών.

Σύμφωνα με το [2] τυπικές προδιαγραφές των ενσωματωμένων συστημάτων είναι:

- Τεχνικές:
  - Συγκεκριμένη λειτουργία.
  - Εκτέλεση σε πραγματικό χρόνο.
  - Χαμηλή κατανάλωση ισχύος.
- Οικονομικές:
  - Χαμηλό κόστος.
  - Αξιόπιστη λειτουργία.
  - Μικρός χρόνος σχεδιασμού και εισόδου στην αγορά.
  - Επαναχρησιμοποίηση (re-usability).

[1], [3], [4], [5]

1.2 Ιστορία και Εξέλιξη

1.2.1 Ιστορία

Τα πρώτα υπολογιστικά συστήματα που εμφανίστηκαν στο τέλος της δεκαετίας του τριάντα, μοιάζανε κατασκευαστικά με αυτό που σήμερα ονομάζουμε ενσωματωμένα συστήματα, με την έννοια ότι είχαν μια συγκεκριμένη εργασία προς εκτέλεση, δεν μπορούν όμως να χαρακτηριστούν έτσι λόγω του κόστους, του μεγέθους και της μη αποδοτικότητας που τα χαρακτήριζε.

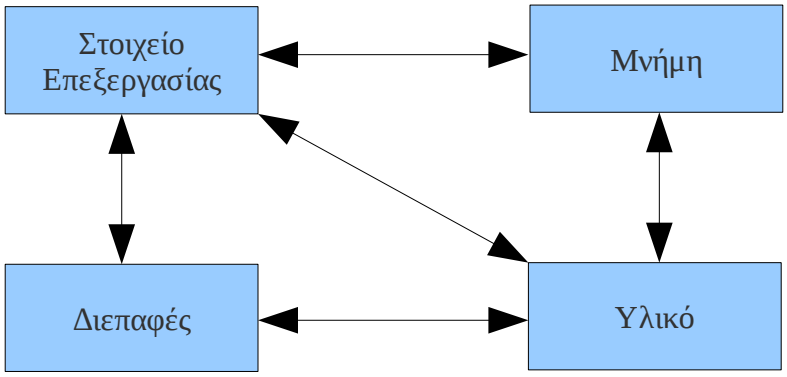
Το πρώτο ενσωματωμένο σύστημα θεωρείται κατά γενική αποδοχή το υπολογιστικό σύστημα καθοδήγησης, πλοήγησης και ελέγχου που χρησιμοποιήθηκε στα οχήματα των διαστημικών αποστολών Apollo. Το Apollo Guidance Computer (AGC) αναπτύχθηκε στο εργαστήριο οργάνων του Massachusetts Institute of Technology (MIT instrumentation laboratory) υπό την διεύθυνση του επιστήμονα και μηχανικού Charles Stark Draper και του υπεύθυνου σχεδιασμού Eldon C. Hall το 1961. Η λειτουργία του ήταν η συλλογή και παροχή πληροφοριών πτήσης και ο αυτόματος υπολογισμός και έλεγχος της πλοήγησης. [6]

Το AGC ήταν ένας από τους πρώτους υπολογιστές που περιείχε ολοκληρωμένα κυκλώματα. Η block I έκδοση του είχε 4100 ολοκληρωμένα κυκλώματα που το καθένα αποτελούταν από μία λογική πύλη NOR τριών εισόδων, ενώ η μεταγενέστερη (1966) block II έκδοση διπλές λογικές πύλες NOR τριών εισόδων, συνολικά περίπου 5600 πύλες. Η συχνότητα ρολογιού του επεξεργαστή ήταν 2048 Mhz, είχε 4 kilobyte μνήμη τυχαίας προσπέλασης μαγνητικού πυρήνα, 32 kilobyte μνήμη μόνο ανάγνωσης και διεπαφή χρήστη με οθόνη και πληκτρολόγιο που ονομαζόταν DSKY, από το display and keyboard τα οποία φαίνονται στην Εικόνα 1. Το AGC ήταν ένα σύστημα πραγματικού χρόνου, έννοια που θα συζητηθεί στο επόμενο κεφάλαιο. Εξαιτίας της χρήσης των πρωτοεμφανιζόμενων τότε ολοκληρωμένων κυκλωμάτων, θεωρούταν το πιο αναξιόπιστο μέρος της αποστολής. [6], [7], [8]

Εικόνα 1: Apollo Guidance Computer

Όσον αφορά τη μαζική παραγωγή ενσωματωμένων συστημάτων, την πρωτιά κατέχει το υπολογιστικό σύστημα καθοδήγησης Autonetics D17 των πυραύλων Minuteman I και II η χρήση του οποίου έγινε μαζική το 1966. Η κατασκευή του για τον Minuteman I ακολουθούσε λογική τρανζίστορ και για μνήμη χρησιμοποιούσε σκληρό δίσκο ενώ στον Minuteman II αντικαταστάθηκε από υπολογιστή που βασιζόταν σε ολοκληρωμένα κυκλώματα. Το γεγονός ότι κατασκευάστηκαν δεκάδες χιλιάδες κομμάτια οδήγησε στη μείωση του κόστους των πυλών NAND από 1000$ σε 3$ ανά πύλη. [2], [10]

Από το 1961 τα βασικά στοιχεία των ενσωματωμένων συστημάτων ακολουθούν το δομικό Διάγραμμα 1.



Διάγραμμα 1: Δομικό Διάγραμμα Ενσωματωμένων Συστημάτων

1.2.2 Εξέλιξη

Έκτοτε, το χάσμα στις επιδόσεις των δομικών στοιχείων όπως επίσης μεταξύ υλικού και λογισμικού, αλλαγές και εξελίξεις στον προγραμματισμό, στην ηλεκτρονική αλλά και στις ανάγκες που παρουσιάζονται, οδήγησαν στην εξέλιξη των ενσωματωμένων συστημάτων όπως τα γνωρίζουμε σήμερα.

Ορισμένα γεγονότα που αποτέλεσαν δομικούς λίθους στην εξέλιξη τους κατά το [11] ακολουθούν παρακάτω:

- 1968: Η Fairchild εισάγει τον πολύ δημοφιλή τελεστικό ενισχυτή 741 που προοριζόταν για χρήση σε ενσωματωμένα συστήματα. [12]
- 1970:
  - Η Texas Instruments κατασκευάζει το πρώτο ολοκληρωμένο κύκλωμα Programmable Logic Array. [13]
  - Η Intel κατασκευάζει το 1103 chip που αποτέλεσε το πρώτο Dynamic Random Access Memory chip (DRAM). [14]
- 1971: Η Intel παρουσιάζει τον πρώτο μικροεπεξεργαστή (microprocessor) με ονομασία 4004. [15]
- 1972: Ο Dennis Ritchie αναπτύσσει την γλώσσα προγραμματισμού C. [16]
- 1973: Κατασκευάζεται η διεπαφή δικτύου Ethernet. [17]
- 1979: Η Bell Labs κατασκευάζει τον πρώτο Digital Signal Processor (DSP). [18]
- 1980: Οι Colin Hunter και James Ready εισάγουν στην αγορά το πρώτο λειτουργικό σύστημα πραγματικού χρόνου για ενσωματωμένα συστήματα Versatile Real-Time Executive (VTRX). [19]

- 1983: Ο Richard Stallman ανακοινώνει τη δημιουργία του λειτουργικού συστήματος GNU. [20]
- 1984:
  - Η εταιρία Xilinx συμμετέχει στην κατασκευή του πρώτου Field Programmable Gate Array (FPGA). [21]
  - Εφεύρεση της NOR και NAND flash μνήμης από τον Dr. Fujio Masuoka. [22]
- 1985:
  - Η Mips Computer Systems κατασκευάζει τον πρώτο της Reduced Instruction Set Computer (RISC) μικροεπεξεργαστή R2000. [23]
  - Η Arcon Computers παρουσιάζει τον ARM1 μικροεπεξεργαστή. [24]
  - Διαμορφώνεται το Joint Test Action Group (JTAG) με σκοπό να βρεθεί πρότυπος τρόπος ελέγχου και αποσφαλμάτωσης των συστημάτων μετά την παραγωγή τους. Η τεχνολογία που αναπτύχθηκε έγινε IEEE πρότυπο το 1990. [25]
- 1991: Ο Φιλανδός φοιτητής Linus Torvalds ανακοινώνει την πρώτη έκδοση του λειτουργικού συστήματος Linux. [26]
- 1993: Οι εταιρίες Apple, IBM και Motorola παρουσιάζουν την αρχιτεκτονική PowerPC. [27]
- 1996: Η Sun αναπτύσσει τη γλώσσα προγραμματισμού Java. [28]
- 1997: Προτείνεται η γλώσσα μοντελοποίησης Unified Modeling Language (UML) v1.0. [29]
- 2001: Οι Sony, Toshiba και IBM αναπτύσσουν τον μικροεπεξεργαστή αρχιτεκτονικής Cell που χρησιμοποιήθηκαν μαζικά στην κονσόλα Playstation 3. [30]
- 2004: Οι IBM, Intel και AMD παρουσιάζουν τις πρώτες τους πολυπύρηνες κεντρικές μονάδες επεξεργασίας. [31], [32]

1.2.3 Περιγραφή

Σήμερα, υπάρχουν διάφοροι τρόποι υλοποίησης ενσωματωμένων συστημάτων, ανάλογα με το στοιχείο επεξεργασίας που χρησιμοποιούν, την υλοποίηση της πλακέτας (board) του συστήματος και το εύρος των δυνατοτήτων τους.

Οι ενσωματωμένοι επεξεργαστές μπορούν να είναι είτε μικροεπεξεργαστές (microprocessors) είτε μικροελεγκτές (microcontrollers). Ο μικροεπεξεργαστής περιέχει όλα τα βασικά δομικά στοιχεία μιας κεντρικής μονάδας επεξεργασίας (CPU) όπως αριθμητική και λογική μονάδα, μνήμη, αποθηκευμένο σύνολο εντολών για εκτέλεση, μεταφορά δεδομένων και έλεγχο ροής προγράμματος, σε ένα ολοκληρωμένο κύκλωμα (chip). Μια κατηγορία των μικροεπεξεργαστών, οι επεξεργαστές ψηφιακού σήματος (Digital Signal Processor-DSP) με βελτιστοποιημένη αρχιτεκτονική για τις ανάγκες των υπολογισμών της επεξεργασίας σήματος, χρησιμοποιείται συχνά στα ενσωματωμένα συστήματα. Ο μικροελεγκτής παρέχει μειωμένο κόστος και μέγεθος αφού αποτελείται από έναν απλό

μικροεπεξεργαστή με κάποια περιφερειακά υλοποιημένα σε ένα μόνο chip. Μπορεί να περιέχει για παράδειγμα τον ταλαντωτή κρυστάλλου, watchdog timer, σειριακή και αναλογική είσοδο/έξοδο και μνήμη τυχαίας προσπέλασης (RAM). Σε αντίθεση με τους μικροεπεξεργαστές, οι μικροελεγκτές προορίζονται για πολύ πιο συγκεκριμένες εργασίες. Για τον λόγο αυτό η απλότητα στον σχεδιασμό τους είναι απαραίτητη ώστε να επιτευχθεί μείωση του κόστους και αύξηση της αξιοπιστίας.

Ποικίλουν επίσης και οι αρχιτεκτονικές των ενσωματωμένων επεξεργαστών. Μπορεί να συναντήσει κανείς μικροεπεξεργαστές που χρησιμοποιούν αρχιτεκτονική Harvard ή Von Neumann, Reduced Instruction Set Computer (RISC), Very Long Instruction Word (VLIW), ακόμα και μη-RISC αρχιτεκτονικές. Οι παραπάνω αποτελούν γενικές κατηγορίες αρχιτεκτονικής συνόλου εντολών. Στην πραγματικότητα υπάρχουν σχεδόν τόσες αρχιτεκτονικές όσα είναι και τα μοντέλα των μικροεπεξεργαστών, αφού η κάθε εταιρία αναπτύσσει το δικό της σύνολο εντολών το οποίο επαναχρησιμοποιεί σε ορισμένα, λίγα συνήθως, μοντέλα. [33], [34], [35]

Σε εφαρμογές στις οποίες το μέγεθος δεν είναι πρόβλημα συχνά επιλέγεται ένα Single Board Computer (SBC), ένας πλήρης δηλαδή υπολογιστής υλοποιημένος σε μια πλακέτα κυκλώματος, με θύρες επέκτασης ή χωρίς. Αυτός μπορεί να περιέχει μέρη συμβατά με γενικού σκοπού προσωπικούς υπολογιστές και συνήθως έχει εγκατεστημένο ένα λειτουργικό σύστημα. Αντίστοιχη υλοποίηση μπορεί να έχει και ένας μικροελεγκτής σε ένα Single Board Microcontroller, με περιορισμούς στο εύρος των εργασιών που μπορεί να αναλάβει όπως προαναφέρθηκε. Υπάρχουν διάφοροι τύποι SBC, αυτά που χρησιμοποιούνται σε ένα backplane, έναν δίαυλο επικοινωνίας με άλλα SBC δηλαδή, αυτά που είναι σε stackable μορφή όπου χρησιμοποιούνται σαν δομικά υλικά ενός μεγαλύτερου συστήματος, αυτά που μπορούν να λειτουργήσουν αυτόνομα σε standalone μορφή και υβριδικά, που μπορούν να υποστηρίξουν πάνω από ένα τρόπο λειτουργίας. Μερικά παραδείγματα single board computers είναι οι πλακέτες PC/104(+), PCI-104 και EBX. [36]

Ένας άλλος τύπος ενσωματωμένων συστημάτων είναι τα System on Chip (SoC), συστήματα με στοιχείο επεξεργασίας, μνήμη και περιφερειακά υλοποιημένα σε ένα μόνο chip. Τα SoC μπορούν να αναπτυχθούν είτε πάνω σε ένα Application Specific Integrated Circuit (ASIC) είτε σε FPGA.

ASIC χαρακτηρίζεται ένα ολοκληρωμένο κύκλωμα που είναι προσαρμοσμένο για συγκεκριμένη εφαρμογή, για παράδειγμα το κύκλωμα ενός κινητού τηλεφώνου. Ενώ τα πρώτα ASIC ήταν σχεδιασμένα με λογική πίνακα πυλών (gate array), στη συνέχεια εφαρμόστηκαν και άλλες λογικές σχεδιασμού όπως είναι η λογική τυποποιημένου κυττάρου (standard cell design), η πλήρως προσαρμοσμένη (full custom design) και η δομημένη (structured/platform design). Σύμφωνα με τη λογική πίνακα πυλών, τα ενεργά στοιχεία του κυκλώματος (transistor και άλλα) είναι τυπωμένα στο κύκλωμα αλλά ασύνδετα κατά την παραγωγή. Μετά την διαδικασία του φυσικού σχεδιασμού της λειτουργίας του κυκλώματος εφαρμόζονται οι κατάλληλες διασυνδέσεις. Οι πίνακες πυλών μπορούν να είναι channeled, με ένα κενά κανάλια σιλικόνης να διαχωρίζουν τις σειρές από ζεύγη transistor (μέγιστη χωρητικότητα ανά κύκλωμα: 20000 πύλες) , ή channel-less χωρίς κανάλια διαχωρισμού (μέγιστη χωρητικότητα: 100000 πύλες). Τα gate array ASIC κατέχουν σχεδόν το μισό μερίδιο της αγοράς ακόμα και σήμερα.

Η λογική τυποποιημένου κυττάρου χρησιμοποιεί βιβλιοθήκες δομικών στοιχείων του κυκλώματος, αντιστάσεων, πολλαπλασιαστών και άλλων (κύτταρα), και βιβλιοθήκες πυλών και flip-flop (τυποποιημένα κύτταρα). Αυτό σημαίνει ότι τα ηλεκτρικά κυκλώματα των στοιχείων του chip είναι προσχεδιασμένα και αμετάβλητα, ενώ παρέχεται ευελιξία στην τοποθέτηση και τη διασύνδεση τους. Αυτός ο τρόπος σχεδιασμού προσφέρεται για κυκλώματα υψηλής πυκνότητας και απόδοσης.

Σε ένα πλήρως προσαρμοσμένο ASIC όλα τα κυκλωματικά στοιχεία και οι διασυνδέσεις τους είναι ξεχωριστά σχεδιασμένα και τοποθετημένα πάνω στο chip. Παρά τον μεγάλο χρόνο σχεδιασμού και το εξειδικευμένο προσωπικό που απαιτείται για την κατασκευή τέτοιων ASIC, είναι ο τρόπος που οδηγεί στο chip με το μικρότερο μέγεθος για μια εφαρμογή και προσφέρει απεριόριστη ευελιξία. Λόγω του κόστους και χρόνου παραγωγής όμως, κατέχουν ένα μερίδιο της τάξεως του 10% στην αγορά.

Προσχεδιασμένα στοιχεία από κατασκευαστικές εταιρίες ASIC ή από τρίτους χρησιμοποιούνται στην δομημένη λογική σχεδιασμού, ώστε να επιτευχθεί μείωση του κύκλου παραγωγής. Ο σχεδιασμός εδώ μοιράζεται στοιχεία με την λογική standard cell, αλλά με επιπλέον κυκλώματα προκαθορισμένα, όπως η τροφοδοσία, ρολόγια και δομές δοκιμής. [37], [38], [39]

Τα FPGA διαφέρουν από τα ASIC στο ότι είναι ολοκληρωμένα κυκλώματα έτοιμα να διαμορφωθούν από τον σχεδιαστή ή τον πελάτη μετά την παραγωγή. Αποτελούνται από μια πλακέτα στην οποία υπάρχουν στοιχεία προγραμματιζόμενης λογικής που ονομάζονται μπλοκ λογικής (logic blocks) και μια ιεραρχία διασυνδέσεων η οποία μπορεί να προσαρμοστεί στην υλοποίηση του τελικού κυκλώματος στην πλακέτα. Ακολουθούν την λογική των breadboard που χρησιμοποιούνται στην ηλεκτρονική δηλαδή. Τα μπλοκ λογικής μπορούν να διαμορφωθούν όσον αφορά τη λειτουργία που επιτελούν, είτε σε απλές πύλες NAND, NOR κ.ο.κ., είτε σε πολύπλοκες συνδυαστικές συναρτήσεις. Επιπλέον περιέχουν δομές μνήμης, απλές όπως flip-flops ή πιο πολύπλοκες. Για την διαμόρφωση των FPGA όπως και των ASIC χρησιμοποιούνται συνήθως γλώσσες περιγραφής υλικού (Hardware Description Language-HDL), για παράδειγμα VHDL ή Verilog. Σαν στοιχείο επεξεργασίας μπορούν να έχουν ελαστικούς πυρήνες (soft cores) διαθέσιμους σε συνθέσιμη γλώσσα περιγραφής υλικού, ελαστικούς πυρήνες βελτιστοποιημένους για FPGA ή άκαμπτους πυρήνες (hard cores), δηλαδή μη προσαρμόσιμες μακροεντολές περιεχόμενες στο σχέδιο του FPGA. [37], [40]

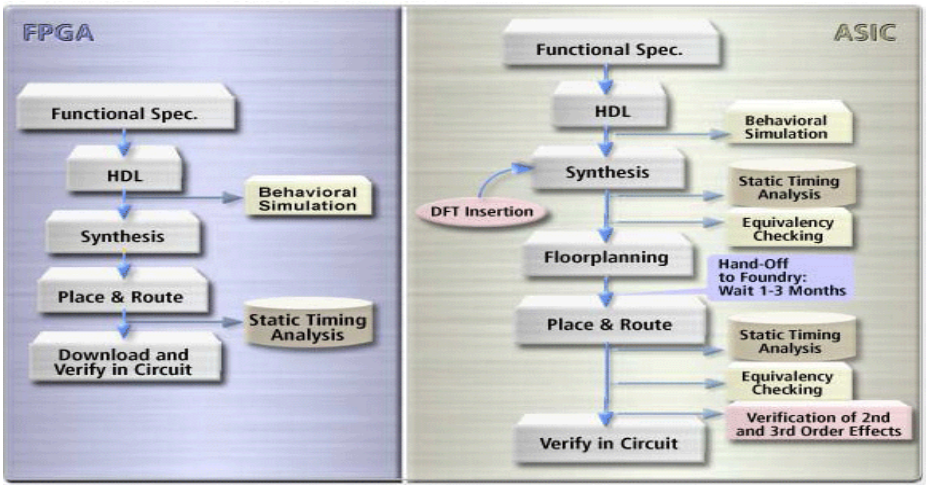
Τα FPGA προσφέρουν τα παρακάτω πλεονεκτήματα [41]:

- Μικρότερο χρόνο εισόδου στην αγορά λόγω της μείωσης των απαιτούμενων βημάτων σχεδιασμού και κατασκευής.
- Δεν απαιτούν προκαταβολικά μη επαναλαμβανόμενα έξοδα (Non Recurring Expenses-NRE), κόστη που συνήθως εμπλέκονται με τον σχεδιασμό ASIC.
- Έχουν απλούστερο κύκλο σχεδιασμού.
- Δυνατότητα για αλλαγή του κυκλώματος on site, αφού είναι field programmable.

Τα ASIC όμως, διατηρούν τα δικά τους πλεονεκτήματα [41]:

- Ικανότητα πλήρως προσαρμοσμένης λογικής σχεδίου με αποτέλεσμα την κατασκευή “εξατομικευμένων” συστημάτων.
- Χαμηλότερο κόστος μονάδας για υψηλής πυκνότητας chip.
- Μικρότερος συντελεστής μορφής.
- Μπορούν να υποστηρίξουν υψηλότερες συχνότητες ρολογιού.

Η Εικόνα 2 [41]δείχνει τις διαφορές στη ροή σχεδιασμού μεταξύ FPGA και ASIC.



Εικόνα 2: FPGA vs ASIC

1.3 Προκλήσεις στην Ανάπτυξη Ενσωματωμένων Συστημάτων

Εξαιτίας της ύπαρξης αυστηρών περιορισμών αλλά και νέων τάσεων της τεχνολογίας ένα πλήθος προκλήσεων παρουσιάζεται στον σχεδιασμό και κατασκευή ενσωματωμένων συστημάτων.

1.3.1 Προκλήσεις κατά τον Σχεδιασμό

Τα ενσωματωμένα συστήματα αποτελούνται από υλικό, λογισμικό και εξαρτώνται άμεσα από το περιβάλλον. Οι συνθήκες λειτουργίας παίζουν καθοριστικό ρόλο στο πόσο ανθεκτικό, ασφαλές, προσβάσιμο πρέπει να είναι ένα τέτοιο σύστημα, ακόμα και στο κατά πόσο αυτό θα πρέπει να λειτουργεί απρόσκοπτα δίχως συντήρηση αν αυτή δεν είναι εφικτή. Ένα σύστημα διόρθωσης πορείας στο τηλεσκόπιο Hubble πρέπει να παραμένει λειτουργικό σε συνθήκες ακραίας θερμοκρασίας, ακτινοβολίας χωρίς συντήρηση και επιδιορθώσεις για πολύ μεγάλα διαστήματα, ίσως για χρόνια. Στα ενσωματωμένα συστήματα δεν υπάρχουν περιθώρια για προγραμματιστικά λάθη γιατί μπορούν να οδηγήσουν σε μοιραία αποτελέσματα, όταν αυτά για παράδειγμα συμβούν στον πλοηγό ενός αεροσκάφους.

Ο σχεδιασμός του συστήματος αποτελεί έτσι μια πρόκληση όσον αφορά το υλικό και το λογισμικό. Επιπλέον ο διαχωρισμός της υπολογιστικής (λογισμικό) και της φυσικής υπόστασης (πλατφόρμα και περιβάλλον) του συστήματος, διαχωρισμός που αποτελεί μια από της κεντρικές ιδέες για την ανάπτυξη της υπολογιστικής επιστήμης, δεν είναι πλέον εφικτός. Αντίθετα, πρέπει να υπάρχει μια ολιστική προσέγγιση, με σχεδιασμό υλικού, σχεδιασμό λογισμικού και θεωρία ελέγχου να υλοποιούνται και να εξετάζονται με ένα συνεχή και συνδυαστικό τρόπο. Οδηγούμαστε λοιπόν στην τάση του συν-σχεδιασμού (co-design) που θα συζητηθεί σε επόμενο υποκεφάλαιο.

Οι σχεδιαστές ενσωματωμένων συστημάτων έχουν να αντιμετωπίσουν

- την πολυπλοκότητα:

Στις ηλεκτρονικές συσκευές ευρείας κατανάλωσης παρουσιάζεται σύγκλιση απαιτήσεων για αποθήκευση, συνδεσιμότητα, ολοκληρωμένη λειτουργικότητα, λόγου χάρη σε μια συσκευή αναπαραγωγής mp3 αρχείων ή μια ψηφιακή βιντεοκάμερα. Στην αυτοκινητοβιομηχανία αυξάνεται η απαίτηση για μείωση των καλωδιώσεων και του αριθμού των επεξεργαστών. Σαν αποτέλεσμα, το σύστημα χαρακτηρίζεται από έλλειψη αφαιρετικότητας και μεγαλύτερα επίπεδα λειτουργικότητας.

- την ευελιξία:

Επαναχρησιμοποίηση, αλλαγές και αναβάθμιση στο λογισμικό είναι πλέον απαραίτητες προϋποθέσεις στον σχεδιασμό ενσωματωμένων συστημάτων.

- το μέγεθος και το βάρος:

Στις περισσότερες εφαρμογές απαιτείται ολοένα μεγαλύτερη μείωση του μεγέθους και του βάρους των συστημάτων που χρησιμοποιούνται (πχ. στην κινητή τηλεφωνία). Τα συστήματα υψηλής απόδοσης όμως, είναι συνήθως μεγαλύτερα και βαρύτερα αφού ένας επεξεργαστής με πολλούς ακροδέκτες ή συστήματα με μεγάλες απαιτήσεις σε μνήμη καταλαμβάνουν μεγάλες περιοχές στις πλακέτες. Γίνεται χρήση υψηλών επιπέδων ολοκλήρωσης για να αντιμετωπιστεί το πρόβλημα προσθέτοντας στην πολυπλοκότητα.

- την τροφοδοσία και ψύξη:

Ισχυροί επεξεργαστές απαιτούν μεγαλύτερη τροφοδοσία που συνεπάγεται την αύξηση του μεγέθους και του βάρους ενώ αύξηση στην κατανάλωση απαιτεί αύξηση της ψύξης του συστήματος.

- την υπολογιστική ικανότητα:

Στα ενσωματωμένα συστήματα η υπολογιστική ικανότητα δεν ορίζεται μόνο από τον αριθμό των εντολών ανά δευτερόλεπτο. Σημαντικό ρόλο στην απόδοση του παίζουν και παράγοντες όπως η εξυπηρέτηση των διακοπών, το overhead μεταγωγής πλαισίου (context switching) και η απόδοση εισόδου/εξόδου. Λόγω των απαιτήσεων πραγματικού χρόνου, πρέπει να υπάρχει προβλεψιμότητα (predictability) και αιτιότητα (determinacy) στην συμπεριφορά του συστήματος. Τα προβλέψιμα συστήματα παρουσιάζουν run-time συμπεριφορά που εξάγεται εύκολα με παρατήρηση του πηγαίου κώδικα, και τα αιτιατά συστήματα, παρέχουν εντολές το αποτέλεσμα της εκτέλεσης των οποίων δεν μεταβάλλεται και δεν εξαρτάται από προηγούμενες εντολές ή χρονικές στιγμές.

- την αξιοπιστία:

Λόγω των αντίξοων συνθηκών όπως προαναφέρθηκε στις οποίες ορισμένα συστήματα καλούνται να λειτουργήσουν και επειδή συνήθως είναι ζωτικής σημασίας η λειτουργία τους, μεγάλο ρόλο στον σχεδιασμό των ενσωματωμένων συστημάτων κατέχει η αύξηση της αξιοπιστίας. Για την επίτευξη του στόχου αυτού, σημαντική είναι η διατήρησή του αριθμού των ακροδεκτών χαμηλό και της ψύξης σε ικανοποιητικά επίπεδα.

- το κόστος:

Το κόστος των ενσωματωμένων συστημάτων που χρησιμοποιούνται σε συσκευές καθημερινής χρήσης πρέπει να διατηρείται όσο το δυνατόν χαμηλότερο, γεγονός που έρχεται σε αντίφαση με την αύξηση των δυνατοτήτων που απαιτείται.

- την αρχιτεκτονική:

Τα αρχιτεκτονικά χαρακτηριστικά των ενσωματωμένων επεξεργαστών διαφέρουν από αυτά των γενικού σκοπού υπολογιστών. Σε εφαρμογές αυστηρού πραγματικού χρόνου, οι προθεσμίες πρέπει να τηρούνται κάθε φορά, χωρίς καμία εξαίρεση. Το να τύχει δύο υπορουτίνες να ανταγωνίζονται για να πάρουν δεδομένα από την κρυφή μνήμη είναι μη αποδεκτό. Μια αποτυχία κρυφής μνήμης (cache miss), μπορεί να οδηγήσει σε αποτυχία λειτουργίας. Για τον λόγο αυτό, η αρχιτεκτονική των επεξεργαστών στα ενσωματωμένα συστήματα πρέπει να σχεδιάζεται με γνώμονα τη λειτουργία του συστήματος.

[42], [43], [44], [45], [46]

1.3.2 Προκλήσεις κατά τον Προγραμματισμό

Ο προγραμματισμός των ενσωματωμένων συστημάτων θέτει νέες προκλήσεις που δεν υπάρχουν στον προγραμματισμό προσωπικών υπολογιστών. Σε αυτό προσθέτουν παράγοντες που συζητήθηκαν προηγουμένως, όπως η αυξανόμενη πολυπλοκότητα, οι περιορισμένες δυνατότητες μνήμης και φυσικά οι εφαρμογές πραγματικού χρόνου.

Στα ενσωματωμένα συστήματα που δεν έχουν την δυνατότητα εγκατάστασης λειτουργικού συστήματος, οι εργασίες αυτού πρέπει να εκτελούνται από το πρόγραμμα τις εφαρμογής. Τυπικές εργασίες ενός λειτουργικού συστήματος, είναι η χρονοδρομολόγηση (scheduling), η διαχείριση της μνήμης και των λειτουργιών εισόδου/εξόδου, η δημιουργία στρωμάτων αφαιρετικότητας για το υλικό και η επικοινωνία του στοιχείου επεξεργασίας με τα περιφερειακά. Το πρόγραμμα τις εφαρμογής δηλαδή, πρέπει με τους περιορισμένους πόρους που παρέχονται να εξομοιώνει τις δυνατότητες ενός λειτουργικού συστήματος προσωπικού υπολογιστή. Ακόμα όμως και όταν υπάρχει η δυνατότητα εγκατάστασης λειτουργικού συστήματος σε ένα ενσωματωμένο σύστημα, αυτό συνεχίζει να έχει περιορισμένους πόρους αλλά και απαιτήσεις, όπως η προβλεψιμότητα και αιτιότητα στις εργασίες που αυτό επιτελεί.

Οι περιορισμοί στο μέγεθος αλλά και στον τύπο της μνήμης αποτελούν ένα ακόμη εμπόδιο στον προγραμματισμό ενσωματωμένων συστημάτων. Η χρήση volatile μνήμης δεν είναι εφικτή τις περισσότερες φορές. Ένα κινητό τηλέφωνο ή ένα ψηφιακό ρολόι δεν μπορεί να εκκινεί από ένα σκληρό δίσκο, έτσι καταφεύγουν στη χρήση μνήμης τύπου flash ή μόνο ανάγνωσης. Αυτοί οι τύποι μνήμης όμως, περιορίζουν τον όγκο αποθήκευσης που μπορούν να χειριστούν περιορίζοντας αναλόγως το μέγεθος του κώδικα και των δεδομένων του προγράμματος εφαρμογής. Ακόμη και η μνήμη τυχαίας προσπέλασης που χρησιμοποιείται πρέπει να αντιμετωπίζεται με μεγάλη προσοχή από τους προγραμματιστές κυρίως ώστε να μην υπάρξουν “διαρροές” καθώς τα προγράμματα είναι πολύ πιθανό να τρέχουν σχεδόν για πάντα. Για παράδειγμα, κλείνοντας μια τηλεόραση από το πλήκτρο του τηλεχειριστηρίου, βάζοντας την σε standby, κάποιες συνιστώσες εισέρχονται στην αναμονή αλλά το πρόγραμμα συνεχίζει να τρέχει περιμένοντας κάποια εντολή από το τηλεχειριστήριο ή τα πλήκτρα στην τηλεόραση. Αν το πρόγραμμα σταματούσε να τρέχει λόγω έλλειψης μνήμης, η τηλεόραση δεν θα ανταποκρινόταν σε εντολές από το τηλεχειριστήριο αφού δεν θα υπήρχε η οντότητα που ελέγχει περιοδικά αν έχει πατηθεί κάποιο πλήκτρο. Ένα byte μόνο να διαρρεύσει στο δρόμο εκτέλεσης μιας εντολής, μπορεί να φέρει το σύστημα σε μη λειτουργική κατάσταση στο μέλλον.

Ο κώδικας ενός προγράμματος για ένα ενσωματωμένο σύστημα, πρέπει να είναι πολύ αποδοτικός διότι δεν μπορούν να χρησιμοποιηθούν ισχυροί επεξεργαστές, λόγω κόστους, κατανάλωσης και πολυπλοκότητας. Για παράδειγμα, θα ήταν ανούσιο ένας φούρνος μικροκυμάτων να χρησιμοποιεί έναν Intel Core 2 Duo. Δεν πρέπει να να επιλέγονται αλγόριθμοι με υψηλές απαιτήσεις σε υπολογιστική ικανότητα χωρίς λόγο.

Ένας παράγοντας που ριζικά διαφοροποιεί τον προγραμματισμό για ενσωματωμένα συστήματα από αυτόν για γενικού σκοπού συστήματα, είναι η επικοινωνία και οι διεπαφές με το υλικό. Ένας προγραμματιστής κοινών εφαρμογών, μπορεί να αναπτύξει το πρόγραμμα του στα πλαίσια ενός μοντέρνου λειτουργικού συστήματος όντας σε πλήρη άγνοια για το υλικό στο οποίο το πρόγραμμα προορίζεται να τρέξει λόγω της ύπαρξης στρωμάτων αφαιρετικότητας. Αντιθέτως, ένας προγραμματιστής εφαρμογών για ενσωματωμένα συστήματα συνήθως καλείται να αναπτύξει το πρόγραμμα του στα πλαίσια κάποιου λειτουργικού συστήματος πραγματικού χρόνου (Real Time Operating System-RTOS) το οποίο δεν μπορεί να παρέχει τέτοια επίπεδα αφαιρετικότητας, γεγονός που οφείλεται στους περιορισμένους πόρους του συστήματος και της μεγάλης ποικιλίας υλικού. Ο κώδικας του θα πρέπει να αλληλεπιδρά απευθείας με το συγκεκριμένο υλικό για το οποίο προορίζεται.

Η έλλειψη τυπικών διεπαφών εισόδου/εξόδου δυσχεραίνει επιπλέον τον προγραμματισμό στα ενσωματωμένα συστήματα. Οι προσωπικοί υπολογιστές, διαθέτουν οθόνη, πληκτρολόγιο και ποντίκι που χρησιμεύουν στο να μπορεί να κανείς να “δει” τι συμβαίνει μέσα στο πρόγραμμα για να μπορεί να αποσφαλματωθεί ο κώδικας. Τα περισσότερα ενσωματωμένα συστήματα δεν διαθέτουν τέτοιες διεπαφές, με αποτέλεσμα ο προγραμματιστής να μην γνωρίζει τι συμβαίνει μέσα στο σύστημα. Εναλλακτικά, χρησιμοποιούνται οι παρακάτω τρόποι αποσφαλμάτωσης:

- Εξωτερική αποσφαλμάτωση με χρήση καταγραφής ή της εξόδου της σειριακής θύρας, ή κάποιον εξυπηρετητή αποσφαλμάτωσης (debug server).
- Εσω-κυκλωματική αποσφαλμάτωση (In Circuit Debugging-ICD) με χρήση κατάλληλων συστημάτων που συνδέονται στο σύστημα προς αποσφαλμάτωση μέσω της διεπαφής JTAG ή NEXUS.
- Χρήση εσω-κυκλωματικού εξομοιωτή (in circuit emulator) στη θέση του στοιχείου επεξεργασίας.
- Αποσφαλμάτωση σε πλήρη εξομοιωτή (emulator), σε πρόγραμμα που τρέχει σε προσωπικό υπολογιστή και προσομοιώνει τη λειτουργία και το υλικό του συστήματος.

[47], [48], [49]

1.3.3 Προκλήσεις για τα Λειτουργικά Συστήματα Ενσωματωμένων Συστημάτων

Υπάρχουν ορισμένες επιθυμητές ιδιότητες σε ένα λειτουργικό σύστημα που προορίζεται για ενσωματωμένα συστήματα, όπως να μπορεί αποτελείται από βασικά δομικά στοιχεία τα οποία να συνδυάζονται ανάλογα με την περίπτωση ώστε να δημιουργούν το κατάλληλο λειτουργικό σύστημα. Επιπλέον, πρέπει να είναι επεκτάσιμο και εύκολο να διαμορφωθεί, με μικρό αποτύπωμα μνήμης και να υποστηρίζει αρκετούς επεξεργαστές και οδηγούς συσκευών.

Ένα λειτουργικό σύστημα μπορεί να προσφέρει τα εξής σε ένα ενσωματωμένο σύστημα:

- Πρωτόκολλα όπως TCP/IP και στοίβα δικτύου.
- Υπόβαθρο για εκκίνηση (boot) των επεξεργαστών.
- Διαχείριση μνήμης και multithreading.
- Συστήματα αρχείων.
- Βιβλιοθήκες και πηγαίο κώδικα τρίτων μερών.
- Διευκόλυνση στην αποσφαλμάτωση εφαρμογών.
- Υποστήριξη από μια κοινότητα ή εταιρία.
- Οδηγούς συσκευών.

Σε αρκετές περιπτώσεις, δεν συμφέρει η εγκατάσταση λειτουργικού συστήματος και η ανάπτυξη της εφαρμογής στα πλαίσια αυτού, είτε λόγω έλλειψης πόρων, είτε για οικονομικούς λόγους. Τα μειονεκτήματα εγκατάστασης λειτουργικού συστήματος συνοψίζονται παρακάτω:

- Αύξηση της πολυπλοκότητας.
- Αύξηση του χρόνου εκκίνησης.
- Overhead, επιβάρυνση δηλαδή των πόρων του συστήματος.
- Αφαιρεί εν μέρει τον έλεγχο του υλικού μέσω του κώδικα, μείωση της δυνατότητας χρήσης προγραμματισμού χαμηλού επιπέδου.
- Κόστος άδειας χρήσης στην περίπτωση που είναι εμπορικό λειτουργικό σύστημα.

Η ανάπτυξη λειτουργικών συστημάτων για ενσωματωμένα συστήματα αποσκοπεί στην βελτιστοποίηση του ισοζυγίου πλεονεκτημάτων-μειονεκτημάτων και στην αύξηση της ευελιξίας και φορητότητας των εφαρμογών.

[50], [51], [52], [53], [54]

1.3.4 Νέες Τάσεις στην Ανάπτυξη Ενσωματωμένων Συστημάτων

Λόγω της ιδιαιτερότητας των ενσωματωμένων συστημάτων οι σχεδιαστές αυτών ακολουθούν συγκεκριμένες τάσεις τις τεχνολογίας. [44], [55], [56], [57], [58], [59], [60]

- Εξομοίωση (simulation) της λειτουργίας του υλικού με πρόγραμμα που τρέχει σε κάποιον γενικού σκοπού υπολογιστή. Ένα σχέδιο δοκιμής εφαρμόζεται στο μοντέλο ενός κυκλώματος και στη συνέχεια εξετάζεται η απόκριση του κυκλώματος. Βοηθάει στον εντοπισμό των λαθών πριν την κατασκευή του υλικού, μειώνοντας το κόστος και καθιστώντας τις τυχόν αλλαγές πολύ πιο εύκολες. Επιπλέον, μειώνει τον χρόνο δοκιμών και τελικά όλου του κύκλου κατασκευής. Η εξομοίωση μπορεί να είναι είτε:

- Υψηλού επιπέδου (εξομοίωση του συστήματος και του περιβάλλοντος του).
- Επιπέδου συστήματος (εξομοίωση του συστήματος μέσω μοντέλων των στοιχείων από τα οποία αποτελείται).
- Βασισμένη σε κύκλους ρολογιού.
- Οδηγούμενη από γεγονότα.

➢ Δημιουργία πρωτοτύπων (prototyping) για έλεγχο της λειτουργίας, είτε με πραγματικό υλικό είτε εικονικό μέσω εξομοίωσης.

➢ Σχεδιασμός βασισμένος σε μοντέλα (model-based design), όπου τα μέρη του συστήματος μετατρέπονται σε μοντέλα που εξομοιώνουν τη λειτουργία τους μέσω γλωσσών περιγραφής υλικού και χρησιμοποιούνται σε εικονικά περιβάλλοντα.

➢ Συν-σχεδιασμός (co-design). Ταυτόχρονη ανάπτυξη, σχεδιασμός και μοντελοποίηση του υλικού (ηλεκτρονικά και μηχανικά μέρη) και του λογισμικού, αντίθετα με την κοινή πρακτική κατά την οποία πρώτα αναπτύσσεται το υλικό και μετά το λογισμικό. Απαιτεί εργαλεία που παρέχουν το κατάλληλο πλαίσιο μέσω γλωσσών περιγραφής υλικού και εξομοίωση του περιβάλλοντος εργασίας και ελέγχου του συστήματος προς ανάπτυξη.

➢ Συν-επαλήθευση (co-verification). Ταυτόχρονη επαλήθευση υλικού και λογισμικού μέσω εργαλείων εξομοίωσης. Ουσιαστικά αποτελεί βήμα του συν-σχεδιασμού. Τα εργαλεία συν-επαλήθευσης, πρέπει να παρέχουν ακριβή μοντέλα υλικού και λογισμικού καθώς και την δυνατότητα εκτέλεσης ακολουθιών προγραμμάτων για έλεγχο της αλληλεπίδρασής των ελεγκτών με τους οδηγούς λογισμικού.

➢ Πανταχού παρούσα Υπολογιστική (Ubiquitous Computing). Είναι μια έννοια που αφορά στην ύπαρξη μικρών οικονομικών και αξιόπιστων υπολογιστικών μονάδων σε αντικείμενα κοινής χρήσης, διασυνδεδεμένων μεταξύ τους με κατανεμημένο τρόπο λειτουργίας. Η έννοια εισήχθηκε στον κόσμο της πληροφορικής από έναν ερευνητή στα εργαστήρια της Xerox σχεδόν πριν 20 χρόνια. Η εξέλιξη των ενσωματωμένων συστημάτων μας φέρνει πιο κοντά στην πανταχού παρών υπολογιστική όπως την οραματίστηκε ο Mark Weiser:

- Ο σκοπός ενός υπολογιστή είναι να λειτουργεί σαν βοήθημα στις εργασίες των ανθρώπων.
- Ο καλύτερος υπολογιστής λειτουργεί σαν αόρατος υπηρέτης.
- Ο υπολογιστής πρέπει να εκτείνει το ασυνείδητο και όχι την διαίσθηση.
- Η τεχνολογία πρέπει να δημιουργεί ηρεμία, να μας ενημερώνει αλλά να μην απαιτεί την προσοχή μας.

Θα μπορούσε κανείς να πει ότι είναι η εποχή που ακολουθεί, μετά από αυτή των μεγάλων mainframes και αυτή της διάδοσης των προσωπικών υπολογιστών, κατά οποία ο υπολογιστής περνάει στο φόντο της ζωής μας. Ο σχεδιασμός ενσωματωμένων συστημάτων ακολουθεί και αυτή την τάση της τεχνολογίας δημιουργώντας έξυπνες συσκευές προς την διευκόλυνση πολλών εργασιών, όπως πλυντήρια ρούχων με ανίχνευση του βάρους και υλικού του φορτίου, ρολόγια χειρός με επιπλέον λειτουργίες για μέτρηση των παλμών της καρδιάς, φωτογραφικές μηχανές με λειτουργία ανίχνευσης προσώπων και πολλές άλλες συσκευές στις οποίες ο ρόλος του υπολογιστή θυμίζει όλο και περισσότερο τον αόρατο υπηρέτη του Mark Weiser.

Αναφορές

[1] M. Barr, A.J. Massa, and Michael Barr, “Programming embedded systems: with C and GNU development tools,” *Introduction*, O'Reilly, 2006, pp. pp. 1-2; http://books.google.com/books?id=nPZaPJrw_L0C&pg=PA1.

[2] Δρ. Μηνάς Δασυγένης, “Σχεδιασμός Ενσωματωμένων Συστημάτων: Μνήμη Εντολών και Δεδομένων”
; http://www.embedded.gr/duth2008.

[3] &. Heath, *Embedded systems design*, Newnes, 2003; http://books.google.com/books?id=BjNZXwH7HlkC&pg=PA2.

[4] M.B.(.E. Barr and [[Michael Barr (software engineer)|Michael Barr]], “Netrino Technical Library,” *Embedded Systems Glossary*; http://www.netrino.com/Embedded-Systems/Glossary.

[5] “Design of Embedded Systems: New Challenges”; http://www.ercim.org/publication/Ercim_News/enw52/cabillic.html.

[6] &.C. Hall, *Journey to the Moon: The History of the Apollo Guidance Computer*, [[American Institute of Aeronautics and Astronautics|AIAA]], 1996.

[7] “M.I.T. Apollo Guidance Computer”; http://ed-thelen.org/comp-hist/vs-mit-apollo-guidance.html.

[8] &.E. Tomayko, “NASA SP-2000-4224 — Computers Take Flight: A History of NASA's Pioneering Digital Fly-By-Wire Project,” *The NASA History Series*, 2000; http://www.klabs.org/history/history_docs/reports/dfbw_tomayko.pdf.

[10] “Minuteman Missile - Strategic Air Command - Nuclear Warhead”; http://www.strategic-air-command.com/missiles/Minuteman/Minuteman_Missile_History.htm.

[11] “Embedded.com - Milestones in embedded systems design”; http://www.embedded.com/timeline/.

[12] “Fairchild Semiconductor in 1965”; http://www.fairchildsemi.com/mediaKit/history_1965.html.

[13] “Monolithic Memories announces: a revolution in logic design,” *Electronic Design*, vol. 26, Mar. 1978, pp. 148B, 148C.

[14] “Intel Museum - Historical Documents”; http://www.intel.com/museum/archives/history_docs/index.htm.

[15] “The Intel 4004 Home”; http://www.intel4004.com/.

[16] “Dennis Ritchie Home Page”; http://cm.bell-labs.com/who/dmr/.

[17] “IEEE 802.3 ETHERNET”; http://www.ieee802.org/3/.

[18] “Bell Labs Top 10 Innovations”; http://www.alcatel-lucent.com/wps/portal/!ut/p/kcxml/04_Sj9SPykssy0xPLMnMz0vM0Y_QjzKLd4w3MXMBSYGYRq6m-pEoYgbxjggRX4_83FT9IH1v_QD9gtzQiHJHR0UAaOmbyQ!!/delta/base64xml/L3dJdyEvd0ZNQUFzQUMvNElVRS82X0FfNDZL.

[19] “Real-Time Operating Systems”; http://www.csee.wvu.edu/~jdm/classes/cs258/OScat/rtime.html.

[20] “The GNU Operating System”; http://www.gnu.org/.

[21] “Xilinx : Our History”; http://www.xilinx.com/company/history.htm.

[22] “Unsung hero - Forbes.com”; http://www.forbes.com/global/2002/0624/030.html.

[23] “Architectures - MIPS Technologies - MIPS Everywhere - MIPS Technologies”; http://www.mips.com/products/architectures/.

[24] “ARM Milestones”; http://www.arm.com/aboutarm/milestones.html.

[25] “IEEE Standards Description: 1149.1-1990”; http://standards.ieee.org/reading/ieee/std_public/description/testtech/1149.1-1990_desc.html.

[26] “Linux Online - Linus Torvalds Bio”; http://www.linux.org/info/linus.html.

[27] “IBM Power Architecture: A High-Performance Architecture with a History”; http://www-03.ibm.com/systems/p/hardware/whitepapers/power/ppc_arch.html.

[28] “JAVASOFT SHIPS JAVA 1.0”; http://web.archive.org/web/20080205101616/http://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960123.10561.xml.

[29] “History of UML”; http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/history_of_uml.htm.

[30] “IBM Research | IBM Research | The Cell architecture”; http://domino.research.ibm.com/comm/research.nsf/pages/r.arch.innovation.html.

[31] “33211A_Multi-Core_WP_en.pdf”; http://multicore.amd.com/Resources/33211A_Multi-Core_WP_en.pdf.

[32] “EETimes.com - Intel will demo its first multi-core CPU at IDF”; http://www.eetimes.com/news/semi/showArticle.jhtml;jsessionid=T0JB3F15IL0UTQE1GHPCKHWATMY32JVN?articleID=46200165&_requestid=852850.

[33] B.B. B and B.B. Brey, *The Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Pro Processor Architecture, Programming, and Inter- facing*, Prentice Hall, 2000.

[34] &.G. Liptak, *Instrument Engineers' Handbook: Process control and optimization*, CRC Press, 2006; http://books.google.com/books?id=TxKynbyaIAMC&pg=PA11.

[35] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach, 3rd Edition*, Morgan Kaufmann, 2002.

[36] “A Linux-oriented Intro to Embeddable Single Board Computers - Linux For Devices Articles - Linux for Devices”; http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/A-Linuxoriented-Intro-to-Embeddable-Single-Board-Computers/.

[37] “ASIC Design Home Page”; http://www.ict.tuwien.ac.at/asicdesign/lehre/384089.html.

[38] R. Ball and Richard Ball, “The promise of structured Asic,” *Electronics Weekly*, Oct. 2004; http://www.electronicsweekly.com/Articles/2004/10/26/33416/The promise of structured Asic.htm.

[39] “upatras eclass | Σχεδιασμός Ολοκληρωμένων Συστημάτων με Τεχνικές VLSI | Έγγραφα”; http://eclass.upatras.gr/modules/document/document.php?openDir=/032375561fo0.

[40] C. Maxfield, *The design warrior's guide to FPGAs*, Elsevier, 2004.

[41] “Getting Started with FPGAs - FPGA vs. ASIC”; http://www.xilinx.com/company/gettingstarted/fpgavsasic.htm#pcs.

[42] “Andrews_BNPP_05.pdf”; http://ertos.nicta.com.au/publications/papers/Andrews_BNPP_05.pdf.

[43] “challenges.pdf.”

[44] “main.pdf”; http://www-verimag.imag.fr/~sifakis/RecentPublications/main.pdf.

[45] “sdnc90a.pdf.”

[46] “Embedded.com - An uncertain future”; http://www.embedded.com/columns/murphyslaw/20300590.

[47] J. Catsoulis, *Designing embedded hardware*, O'Reilly Media, Inc., 2005.

[48] “Embedded System Design Issues (the Rest of the Story)”; http://www.ece.cmu.edu/~koopman/iccd96/iccd96.html.

[49] I.&. Gupta, *Embedded Realtime Systems Programming*, Tata McGraw-Hill, 2003.

[50] “0173-4EmbeddedOSflyer.pdf”; http://www.arm.com/pdfs/0173-4EmbeddedOSflyer.pdf.

[51] “12.pdf”; http://www.cs.virginia.edu/~zaher/rtss-wip/12.pdf.

[52] “Embedded.com - Embedded systems survey: Operating systems up for grabs”; http://www.embedded.com/columns/surveys/163700590?_requestid=725995.

[53] “Embedded Operating System Porting”; http://www.embeddedarm.com/services/embedded-os-porting.php.

[54] “Embedded Operating Systems and Linux - Με την υποστήριξη των Εγγράφων Google”; http://docs.google.com/viewer?a=v&q=cache:TFobAlY0lSsJ:www.nepalinux.org/docs/embedded_linux.pdf+embedded+operating+systems&hl=el&gl=gr&pid=bl&srcid=ADGEESiGkaSU4jxnQeNo2wzX1X3m6abYhe16iv7ixvjqeWOo1WTY5RCwkeFuSY0qxydAp1C_4X37yogSA6fXH_rBisCrgRRGL9ktkfk46yt8ABjAodEROejRHTFfCz_je4Nyd8_KTrCB&sig=AHIEtbRtTtnWmj9lB_MN5LEqYFldtb6YYg.

[55] “2007-01-0776.pdf.”

[56] “systemsengineering010916.pdf.”

[57] “ISORC07paper.pdf.”

[58] A. Greenfield, *Everyware*, New Riders, 2006.

[59] “Ubiquitous Computing”; http://www.cc.gatech.edu/classes/cs6751_97_fall/projects/say-cheese/marcia/mfinal.html.

[60] “Ubiquitous Computing and Embedded Operating Systems Design”; http://www.ercim.org/publication/Ercim_News/enw47/banatre.html.

2 Συστήματα Πραγματικού Χρόνου

2.1 Εισαγωγή

Ένα σύστημα πραγματικού χρόνου (real time system) είναι ένα υπολογιστικό σύστημα στο οποίο η σωστή λειτουργία και συμπεριφορά εξαρτάται όχι μόνο από την ορθότητα των υπολογισμών, αλλά και από το χρονικό διάστημα που απαιτείται για να παραχθούν τα αποτελέσματα τους. [1] Στα συστήματα αυτά, ο χρόνος είναι ουσιώδης. Το λογικό αποτέλεσμα μιας εντολής που δίνεται εκτός των επιθυμητών χρονικών ορίων λαμβάνεται συνήθως ως λάθος ή αποτυχία του συστήματος. Η αποτυχία ενός συστήματος ορίζεται όταν ένα σύστημα δεν ικανοποιεί μια ή και παραπάνω από τις απαιτήσεις που δίνονται για τον ορισμό του. [2]

Για παράδειγμα, τα σύγχρονα συστήματα αποφυγής κλειδώματος των τροχών κατά το φρενάρισμα ενός αυτοκινήτου (Anti Blocking System-ABS) είναι ενσωματωμένα συστήματα πραγματικού χρόνου. Όταν ο οδηγός του αυτοκινήτου πατήσει το φρένο ο ελεγκτής παίρνει μετρήσεις από το περιβάλλον εφαρμογής, δηλαδή από τους διάφορους αισθητήρες, για την ταχύτητα περιστροφής των τροχών, τον τύπο του οδοστρώματος, την κατεύθυνση της κίνησης. Ανάλογα με τις μετρήσεις αυτές αυξομειώνει την υδραυλική πίεση των φρένων στους τροχούς ώστε να μην μπλοκάρουν προκαλώντας την ολίσθηση του αυτοκινήτου. Το αποτέλεσμα (φρενάρισμα) αλλά και ο χρόνος μέσα στον οποίο αυτό θα παραχθεί είναι σημαντικά για την ασφάλεια των επιβατών. Είναι προφανές ότι οποιαδήποτε καθυστέρηση στην ανάλυση των αποτελεσμάτων μπορεί να αποβεί μοιραία. [3]

Στην πραγματικότητα η έννοια του πραγματικού χρόνου είναι κάπως αφηρημένη. Κατά το [4] έχουμε μερικούς ορισμούς για τα συστήματα πραγματικού χρόνου:

- *Ένα οποιοδήποτε σύστημα για το οποίο ο χρόνος παραγωγής της εξόδου είναι σημαντικός. Αυτό συμβαίνει συνήθως διότι η είσοδος αντιστοιχεί με κάποια είδους ενέργεια στον φυσικό κόσμο και η έξοδος πρέπει να συσχετιστεί με την συγκεκριμένη ενέργεια. Η καθυστέρηση που συμβαίνει από τον χρόνο εισόδου μέχρι τον χρόνο εξόδου, πρέπει να είναι αρκετά μικρή για να υπάρχει αποδεκτή "επικαιρότητα" στα πλαίσια ολόκληρου του συστήματος.*
- *Οποιαδήποτε ενέργεια ή σύστημα επεξεργασίας πληροφοριών που πρέπει να ανταποκριθεί σε εξωτερική είσοδο μέσα σε καθορισμένα και πραγματικά χρονικά όρια.*
- *Ένα σύστημα πραγματικού χρόνου είναι ένα σύστημα που απαιτείται να αντιδρά σε διέγερση από το περιβάλλον του (συμπεριλαμβανομένου και του περάσματος του φυσικού χρόνου) μέσα σε χρονικά διαστήματα που καταδεικνύονται από το περιβάλλον.*

Οι παραπάνω ορισμοί καλύπτουν ένα ευρύ φάσμα υπολογιστικών εφαρμογών και δραστηριοτήτων. Ακόμα και ένα λειτουργικό σύστημα όπως είναι το UNIX θα μπορούσε να θεωρηθεί πραγματικού χρόνου, με την έννοια ότι ο χρήστης του συστήματος περιμένει απόκριση στις ενέργειες του εντός κάποιων δευτερολέπτων. Συνήθως όμως, δεν είναι καταστροφικό να ξεπεραστεί αυτό το χρονικό διάστημα. Είναι ξεκάθαρος έτσι ο διαχωρισμός με συστήματα στα οποία η αποτυχία απόκρισης μέσα στα αναμενόμενα χρονικά όρια είναι το ίδιο με μια λανθασμένη απόκριση ή με απουσία απόκρισης.

Αυτή η ιδιότητα είναι που χαρακτηρίζει τα συστήματα πραγματικού χρόνου με βάση τον ορισμό που δόθηκε στην πρώτη παράγραφο.

Τα συστήματα πραγματικού χρόνου μπορεί να είναι ενσωματωμένα ή αντιδραστικά (reactive) συστήματα. Αντιδραστικά είναι τα συστήματα των οποίων η χρονοδρομολόγηση (scheduling) οδηγείται από αντίδραση σε εισόδους που δίνονται από το περιβάλλον του συστήματος, όπως το σύστημα καθοδήγησης ενός γερανού που αντιδρά στο πάτημα των αντίστοιχων πλήκτρων και στην κίνηση μοχλών από τον χειριστή του. [2]

Κατά το [4] τυπικές χρήσεις συστημάτων πραγματικού χρόνου βρίσκουμε:

- Στον έλεγχο διεργασιών.
- Στην παρασκευή προϊόντων.
- Σε γενικευμένα ενσωματωμένα συστήματα.

Μια κατηγοριοποίηση των συστημάτων αυτών είναι σε αυστηρό (hard), ελαστικό (soft) και σταθερό (firm) πραγματικό χρόνο ανάλογα με το πόσο αυστηρά είναι τα χρονικά όρια αναμονής της απόκρισης του συστήματος.

Τα συστήματα αυστηρού πραγματικού χρόνου (hard real time systems) είναι αυτά στα οποία είναι απολύτως απαραίτητο να ληφθεί απόκριση μέσα σε καθορισμένα χρονικά όρια ενώ στα συστήματα ελαστικού πραγματικού χρόνου (soft real time), είναι μεν σημαντικό να ληφθεί απόκριση μέσα στα καθορισμένα όρια αλλά μια καθυστερημένη, συνήθως με ορισμένα χρονικά περιθώρια, απόκριση είναι αποδεκτή και το σύστημα μπορεί να συνεχίσει την ορθή λειτουργία του. Το σύστημα ελέγχου πτήσης για παράδειγμα ενός στρατιωτικού αεροσκάφους, είναι σύστημα αυστηρού πραγματικού χρόνου αφού μη τήρηση μιας προθεσμίας είναι καταστροφική. Αντίθετα θα μπορούσε το σύστημα ελέγχου μιας διαδικασίας να ανεχθεί κάποιες ενδιάμεσες καθυστερήσεις, παρόλο που η δειγματοληψία της εισόδου ενός αισθητήρα πρέπει να γίνεται σε ορισμένα και τακτά χρονικά διαστήματα. Το τελευταίο θεωρείται ένα σύστημα ελαστικού πραγματικού χρόνου. Φυσικά, μπορεί ένα σύστημα πραγματικού χρόνου να απαρτίζεται από υποσυστήματα και των δύο προαναφερθέντων κατηγοριών. [4]

Ο ορισμός της αποδεκτής απόκρισης του ελαστικού πραγματικού χρόνου έχει δύο μορφές σύμφωνα με το [4]:

- Το ανώτατο χρονικό όριο αποδεκτής απόκρισης μπορεί περιστασιακά να ξεπεραστεί στα πλαίσια ενός μέγιστου αριθμού τέτοιων συμβάντων.
- Η απόκριση μπορεί να είναι αποδεκτή αν και καθυστερημένη μέσα σε ένα επιπλέον χρονικό όριο καθυστέρησης.

Στην τρίτη κατηγορία συστημάτων σταθερού πραγματικού χρόνου (firm real time), ένα ανώτατο χρονικό όριο μπορεί να ξεπεραστεί, αλλά μια καθυστερημένη απόκριση θεωρείται μη αποδεκτή, δεν προσφέρει κάποιο πλεονέκτημα και θεωρείται σαν απουσία απόκρισης. Συνήθως λειτουργούν με χρήση πιθανοτήτων για τους ορισμούς των ορίων τους και μπορούν να θεωρηθούν υποκατηγορία των συστημάτων ελαστικού χρόνου. [4] Με απλά λόγια, μερικές αποτυχίες απόκρισης δεν είναι καταστροφικές, αλλά περισσότερες από έναν ορισμένο αριθμό μπορεί να είναι. [2]

Ένα παράδειγμα συστήματος σταθερού πραγματικού χρόνου είναι το σύστημα ελέγχου πορείας (cruise control) ενός αυτοκινήτου το οποίο αναλαμβάνει τον έλεγχο του γκαζιού ώστε το αυτοκίνητο να διατηρεί μια σταθερή ταχύτητα που ορίζεται από τον οδηγό.

Η ταχύτητα του οχήματος δειγματοληπτείται σε τακτά χρονικά διαστήματα και σύμφωνα με τις τιμές αυτής αυξομειώνεται η ώθηση που δίνει στον κινητήρα το γκάζι του οχήματος. Αν οι αισθητήρες ταχύτητας ή το υπεύθυνο λογισμικό δεν δώσουν εγκαίρως στον αλγόριθμο ελέγχου την τρέχουσα ταχύτητα, μπορεί να χρησιμοποιηθεί η προηγούμενη μετρηθείσα τιμή της ταχύτητας ώστε να υπολογιστούν τα σήματα ελέγχου αντί της αναμονής για την καθυστερημένη τιμή. Λόγω των μικρών διαστημάτων δειγματοληψίας της ταχύτητας, το πιθανότερο είναι να υπάρχει μικρή διαφορά στις τιμές και το σύστημα θα έχει αποδεκτή λειτουργία. Το καθυστερημένο δείγμα δεν χρειάζεται και μπορεί να παραβλεφθεί από το σύστημα. [5]

2.2 Ανάπτυξη Συστημάτων Πραγματικού Χρόνου

2.2.1 Εισαγωγή

Ένα τυπικό μπλοκ διάγραμμα ενός συστήματος πραγματικού χρόνου δίνεται από το [1] στην Εικόνα 1:

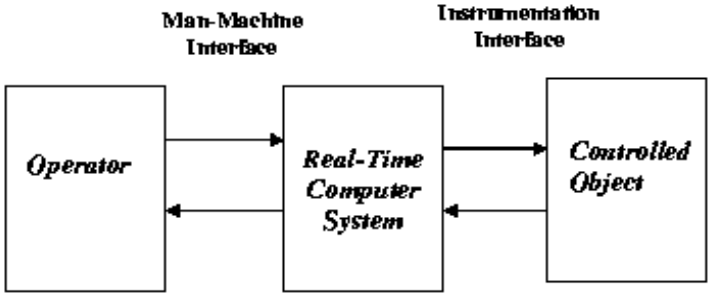


Figure 1: Real-Time System

Εικόνα 1: Δομικό Διάγραμμα Συστήματος Πραγματικού Χρόνου

Μεταξύ του συστήματος πραγματικού χρόνου και του χειριστή αυτού, υπάρχει η διεπαφή ανθρώπου-μηχανής (Man-Machine Interface) που αποτελείται από συσκευές εισόδου/εξόδου όπως πληκτρολόγιο και οθόνη ενώ μεταξύ του συστήματος πραγματικού χρόνου και του αντικειμένου προς έλεγχο υπάρχει η διεπαφή οργάνων (Instrumentation Interface). Αυτή συμπεριλαμβάνει αισθητήρες (sensors) και ενεργοποιητές (actuators) που χρησιμεύουν στην μετατροπή των αναλογικών σημάτων σε ψηφιακών

και αντίστροφα. Μια δεξαμενή σαν αντικείμενο ελέγχου που τροφοδοτείται από μια βαλβίδα, μπορεί να έχει για διεπαφή οργάνων έναν αισθητήρα στην ανώτατη στάθμη της και έναν ενεργοποιητή, που ανοίγει και κλείνει την βαλβίδα. Όταν ο αισθητήρας ανιχνεύσει υγρό στην ανώτατη στάθμη, στέλνει ένα ψηφιακό σήμα στο σύστημα ελέγχου το οποίο με τη σειρά του αποστέλλει σήμα στην βαλβίδα να πάει σε κατάσταση off.

Στο Διάγραμμα 1 φαίνονται οι κλάδοι της πληροφορικής και της επιστήμης του ηλεκτρολόγου μηχανικού που είναι απαραίτητο να μελετηθούν κατά την σχεδίαση ενός συστήματος πραγματικού χρόνου. [2]

Διάγραμμα 1: Στοιχεία Σχεδιασμού Συστημάτων Πραγματικού Χρόνου

Ο σχεδιαστής συστημάτων πραγματικού χρόνου πρέπει να κάνει προσεκτική μελέτη του υλικού και λογισμικού που θα χρησιμοποιηθεί , με βάση την εξισορρόπηση της σχέσης κόστους-ποιότητας, τον ταυτόχρονο ή και παράλληλο προγραμματισμό του συστήματος, και το αν η υλοποίηση του θα είναι κατανεμημένη. Επιπλέον, απαραίτητη είναι η αντίληψη των συνεπειών των εντολών ενός προγράμματος όσον αφορά το χρόνο. Αυτό μπορεί να γίνει εφικτό κατανοώντας την μετατροπή των εντολών αυτών σε γλώσσα μηχανής. Από τις πιο σημαντικές προϋποθέσεις στον σωστό σχεδιασμό ενός τέτοιου συστήματος είναι η κατανόηση της έννοιας του χρόνου και των γεγονότων και της σωστής αναπαράστασης τους.

- Η φύση του χρόνου:

Σύμφωνα με το [2], η φύση του χρόνου είναι μεγάλης σημασίας αφού οι προθεσμίες (deadlines), το ανώτατο δηλαδή χρονικό όριο αναμονής απόκρισης ενός συστήματος πραγματικού χρόνου, είναι στιγμιαίες. Οι προθεσμίες αυτές απορρέουν από τα βασικά φυσικά φαινόμενα που λαμβάνουν χώρα στο σύστημα υπό έλεγχο. Δηλαδή, σε μια οθόνη υπολογιστή η εικόνα πρέπει να ανανεώνεται τουλάχιστον κατά 30 πλαίσια το δευτερόλεπτο (fps) για να προκύψει η ψευδαίσθηση της συνεχόμενης

κίνησης, γεγονός που βασίζεται στην ικανότητα του ανθρώπινου οφθαλμού να ανανεώνει τις εικόνες που δέχεται σε αργότερο ρυθμό από τα 30 fps. Στα συστήματα πλοήγησης για παράδειγμα, οι επιταχύνσεις διαβάζονται με ρυθμό βασισμένο στην μέγιστη ταχύτητα του οχήματος. Αν οι προθεσμίες αυτές βασίζονται σε αβάσιμες υποθέσεις ή παρωχημένες απαιτήσεις μπορεί να οδηγήσουν σε περιττούς ή λανθασμένους χρονικούς περιορισμούς σε ένα σύστημα πραγματικού χρόνου. Είναι σημαντικό λοιπόν να υπάρχει κατανόηση της φύσης του χρόνου κατά τον σχεδιασμό τέτοιων συστημάτων.

Εξίσου σημαντική είναι και η μοντελοποίηση και αναπαράσταση του χρόνου ώστε να μπορέσει να χρησιμεύσει σαν μία από τις μεταβλητές εισόδου του συστήματος. Πολλά συστήματα πραγματικού χρόνου χρησιμοποιούν τεχνικές χρονοσφράγισης (timestamping) και παγκόσμια ρολόγια (global clocks) για συγχρονισμό και εκκίνηση διεργασιών, πρακτικές που μπορεί να είναι αναξιόπιστες, αφού κάθε ρολόι απαιτεί συντονισμό και συχνές ρυθμίσεις. [2] Χρησιμοποιώντας χρονιστές (timers), οι οποίοι με βάση συγκεκριμένα μοντέλα του χρόνου μετρούν το πέρασμα του και παρακολουθούν την εξέλιξη των γεγονότων οδηγείται η συμπεριφορά του συστήματος. Όταν συμβεί το αναμενόμενο γεγονός, ο χρονιστής σταματάει και λαμβάνει χώρα η προγραμματισμένη αντίδραση του συστήματος ελέγχου. Αν το γεγονός δεν συμβεί την αναμενόμενη στιγμή, ο χρονιστής μηδενίζεται και το σύστημα αναλαμβάνει δράσεις επανόρθωσης και ενημέρωσης των πιθανόν χειριστών του συστήματος. [6] Ένα ακόμα πρόβλημα που δημιουργείται με τη χρήση του χρόνου σαν μεταβλητή εισόδου είναι οι συνθήκες κούρσας. Συνθήκη κούρσας συμβαίνει όταν μια είσοδος εξαρτάται από απρόβλεπτους χρονικούς παράγοντες και είναι συνήθως δύσκολη η ανίχνευση της. Παραδείγματος χάριν, σε μια αμφίδρομη τηλεφωνική ζεύξη με βάση τον απρόβλεπτο παράγοντα του χρόνου που τοποθετείται μια κλήση, είναι δυνατόν να δεσμευθεί και για μια εξερχόμενη και για μια εισερχόμενη κλήση αν αυτές συμβούν ταυτόχρονα. [6]

- Γεγονότα:

Σημαντικός θεωρείται στον σχεδιασμό συστημάτων πραγματικού χρόνου και ο ορισμός των γεγονότων. Τα γεγονότα διακρίνονται σε διακριτά και συνεχή και σημαίνουν μια αλλαγή στην φυσική κατάσταση του συστήματος που βρίσκεται υπό έλεγχο. [7] Αντιστοιχούν σε αλλαγή της ροής ελέγχου στο πρόγραμμα ελέγχου. Τα συστήματα πραγματικού χρόνου είναι αιτιατά, δηλαδή οι έξοδοι και οι επόμενες καταστάσεις του συστήματος είναι γνωστές για τις ομάδες εισόδων που προκαλούν κάποιο γεγονός. [2] Οι αλλαγές στην φυσική κατάσταση του συστήματος ή της διεργασίας υπό έλεγχο, γίνονται γνωστές μέσω διαφόρων αισθητήρων, οι έξοδοι των οποίων αποτελούν τις εισόδους στο σύστημα πραγματικού χρόνου που ελέγχει την διεργασία. Τα γεγονότα κοινοποιούνται συνεπώς μέσω της διεπαφής οργάνων.

2.2.2 Έλεγχος και Αξιολόγηση Συστημάτων Πραγματικού Χρόνου

Για να εξασφαλιστεί η ασφάλεια και πιστότητα του συστήματος πραγματικού χρόνου, κατά τον σχεδιασμό εφαρμόζονται διάφορες τεχνικές εξομοίωσης (simulation), ελέγχου (testing) μοντελοποίησης (modelling) και πιστοποίησης (verification).

- Εξομοίωση:

Όπως αναφέρεται στο προηγούμενο κεφάλαιο, η εξομοίωση ενός συστήματος αφορά την εφαρμογή ενός σχεδίου ελέγχου στο μοντέλο του συστήματος με ζητούμενο την ανάλυση της απόκρισης του

κυκλώματος. [8] Εφόσον τα συστήματα πραγματικού χρόνου είναι κατεξοχήν ενσωματωμένα, ισχύουν τα δεδομένα της παραγράφου 1.3.4 για την εξομοίωση. Υπάρχουν δυο σημαντικές κατηγορίες εξομοίωσης που εφαρμόζονται στα συστήματα πραγματικού χρόνου [3]:

- Εξομοίωση γεγονότων πραγματικού χρόνου (real-time-event simulation): Εδώ οι αντιδράσεις και τα γεγονότα συμβαίνουν και καταγράφονται σε πραγματικό χρόνο. Σαν ένα παράδειγμα αναφέρεται οι δοκιμές σύγκρουσης αυτοκινήτου.

- Εξομοίωση διακριτών γεγονότων (discrete-event simulation): Κάνει χρήση λογικών ρολογιών σε λογισμικό για να αναπαραστήσει τα γεγονότα και τις αποκρίσεις. Αυτά δεν συμβαίνουν σε πραγματικό χρόνο, αλλά εξαρτώνται από την ταχύτητα του υλικού του εξομοιωτή ο οποίος είναι ουσιαστικά ένα πρόγραμμα, και των εντολών που εκτελούνται. Οι λογικοί χρόνοι που λαμβάνουν χώρα τα διάφορα γεγονότα και οι αντιδράσεις, εξαρτώνται από προηγούμενα και παροντικά σύνολα γεγονότων και αντιδράσεων που έχουν εξομοιωθεί. Έτσι στο σύστημα-στόχος (target system) αναπαριστούν γεγονότα πραγματικού χρόνου.

Η εξομοίωση για συστήματα πραγματικού χρόνου αλλά και για οποιοδήποτε σύστημα, μειονεκτεί στο ότι ακόμα και αν τα σύνολα των δυνατών γεγονότων είναι πραγματικά και όχι άπειρα, ο αριθμός των πιθανών γεγονότων είναι τόσο μεγάλος που δύσκολα ένα πρόγραμμα εξομοίωσης μπορεί να υπολογίσει όλες τις πιθανές ακολουθίες αυτών.

➢ Έλεγχος:

Κάθε σύστημα ελέγχεται κατά τη διάρκεια και μετά την ολοκλήρωση του κύκλου σχεδιασμού του. Ένας έλεγχος σημαίνει ότι το σύστημα λειτουργεί με περιορισμένο σύνολο εισόδων και οι αποκρίσεις στις εισόδους αυτές καθώς και η γενικότερη συμπεριφορά του συστήματος ελέγχεται αν ακολουθεί τις προδιαγραφές που τέθηκαν κατά τον σχεδιασμό για το σύστημα αυτό. [3] Τα συστήματα πραγματικού χρόνου ιδιαίτερα, λόγω των υψηλών απαιτήσεων για ασφάλεια και πιστότητα αυτών, υπόκεινται σε αυστηρούς ελέγχους. Η δυσκολία στον έλεγχο τέτοιων συστημάτων, βασίζεται στο γεγονός ότι τα πιο ανεξιχνίαστα σφάλματα προκύπτουν κατά την ανταλλαγή πληροφοριών μεταξύ διεργασιών. Συνήθως εξαρτώνται από το χρόνο και εκδηλώνονται σε σπάνιες καταστάσεις του συστήματος. [4]

Για παράδειγμα, τρεις από τις πιο κοινές μεθόδους ελέγχου λογισμικού για συστήματα πραγματικού χρόνου αναφέρονται στο [3]:

- Λειτουργικός έλεγχος (functional testing): Ακολουθεί προσέγγιση μαύρου κουτιού. Δημιουργούνται δεδομένα ελέγχου από τις προδιαγραφές του συστήματος με διάφορες τεχνικές, κατόπιν αυτά εφαρμόζονται σαν είσοδος στο σύστημα και ελέγχεται αν η έξοδος αποκλίνει από τις προδιαγραφές.

- Έλεγχος διαχωρισμού (partition testing): Ο τομέας εισόδου δεδομένων του συστήματος χωρίζεται σε υποσύνολα και δεδομένα ελέγχου που αντιπροσωπεύουν κάθε υποσύνολο εφαρμόζονται σαν είσοδοι στο συνολικό σύστημα.

- Δομικός έλεγχος (structural testing): Ακολουθεί προσέγγιση λευκού κουτιού, κατά την οποία τα δεδομένα ελέγχου προκύπτουν από μελέτη του πηγαίου κώδικα που αποτελεί το σύστημα.

➢ Μοντελοποίηση:

Για είναι δυνατό να πιστοποιηθεί ένα σύστημα, αρχικά πρέπει αυτό να μετατραπεί σε ένα εύχρηστο μοντέλο μέσω μιας τυπικής γλώσσας περιγραφής με σκοπό την αναπαράσταση των ιδιοτήτων του

συστήματος με μαθηματικές φόρμες. Οι ιδιότητες αυτές χωρίζονται σε ιδιότητες ασφαλείας που εγγυώνται ότι δεν θα συμβεί σφάλμα και σε ιδιότητες βιωσιμότητας που εγγυώνται την καλή λειτουργία του συστήματος. Στη συνέχεια το σύστημα μπορεί να πιστοποιηθεί. [9]

Η μοντελοποίηση παίζει σημαντικό ρόλο στον σχεδιασμό συστημάτων αφού μπορεί να αντικαταστήσει την εφαρμογή πειραματικών ελέγχων στο πραγματικό σύστημα αποφέροντας διάφορα πλεονεκτήματα. Το μοντέλο και οι παράμετροι του, είναι ευέλικτα και εύκολα προσαρμόσιμα ενώ επιτρέπεται ο συνδυασμός μοντέλων ετερογενών συνιστωσών. Παράλληλα, η λειτουργία του συστήματος μπορεί να παρατηρηθεί διεξοδικά και η μελέτη γενικεύεται μέσω της αφαιρετικότητας που προσφέρει το μοντέλο. Συνήθως διάφορες τεχνικές μοντελοποίησης εφαρμόζονται στα αρχικά στάδια της ανάπτυξης ενός συστήματος με υψηλό επίπεδο αφαιρετικότητας. [10]

- Πιστοποίηση:

Στα συστήματα πραγματικού χρόνου, απαραίτητη είναι η πιστοποίηση των χρονικών συνθηκών λειτουργίας τους. Η διαδικασία της πιστοποίησης, εγγυάται ότι το σύστημα συμφωνεί με τις προδιαγραφές του μετά τη διαδικασία του ελέγχου, μέσω μεθόδων όπως model checking, language inclusion algorithm, bisimulation. [7] Η τυπική πιστοποίηση απαιτεί αναπαράσταση των ιδιοτήτων που προσδίδουν οι προδιαγραφές στο σύστημα και των ιδιοτήτων του συστήματος με μη αμφισβητήσιμες γλώσσες όπως οι μαθηματικές σύμφωνα με τη διαδικασία της μοντελοποίησης του συστήματος.

Η πιστοποίηση των συστημάτων υπόκειται σε περιορισμούς. Κατά τις σημειώσεις στο [8]:

- Δεν είναι απόλυτη εγγύηση ότι ένα σχέδιο είναι απαλλαγμένο από σφάλματα.
- Δεν μπορεί να βρει που βρίσκεται το λάθος, απλώς επιβεβαιώνει ότι αυτό υπάρχει.
- Δεν μπορεί να είναι απόλυτα ολοκληρωμένη και να καλύψει όλες τις πιθανές περιπτώσεις χρήσης.
- Εξαρτάται από το κόστος της εύρεσης του επομένου σφάλματος συγκριτικά με το κόστος του παρόντος σφάλματος.

Διάφοροι μέθοδοι πιστοποίησης (μέσω εξομοίωσης) που μπορούν να χρησιμοποιηθούν ξεχωριστά ή και σε συνδυασμό αναφέρονται στο [8].

- Block level simulation: Πιστοποιεί την κύρια λειτουργία των βασικών μονάδων που αποτελούν το σύστημα εφαρμόζοντας ένα απλό λειτουργικό σχέδιο ελέγχου. Είναι συνήθως η πρώτη που εφαρμόζεται και ελέγχει την απόκριση των κυκλωμάτων σε γενικές γραμμές.
- Simple top level simulation: Πιστοποιεί την κύρια λειτουργία του ολοκληρωμένου σχεδίου. Εφαρμόζεται μετά των σχεδιασμό όλων των μονάδων και πιστοποιεί τις επιμέρους διασυνδέσεις.
- System oriented simulation: Ελέγχει την η συμπεριφορά του συστήματος σε σχέση με τις προδιαγραφές του σε όλους τους τρόπους λειτουργίας.
- Regress simulation: Αυτοματοποιημένη εξομοίωση για τον έλεγχο του ήδη πιστοποιημένου συστήματος.
- Timing simulation: Υψηλού επιπέδου εξομοίωση που δίνει σαν έξοδο δεδομένα χρονισμού του συστήματος.

- Fault simulation: Εφαρμόζει κάθε πιθανό σφάλμα και εξομοιώνει την εσφαλμένη συμπεριφορά του συστήματος.
- Static timing analysis: Δεν είναι εξομοίωση. Υπολογίζονται οι καθυστερήσεις σε όλα τα μονοπάτια μεταξύ των καταχωρητών που αποτελούν τα ολοκληρωμένα κυκλώματα. Ολοκληρώνεται σε σύντομο χρόνο και δεν απαιτεί δεδομένα ελέγχου (test pattern).
- Board level simulation: Πιστοποιεί ολόκληρο το σύστημα σε επίπεδο πλακέτας (board level). 'Ολες οι επιμέρους συσκευές που απαρτίζουν το σύστημα εξομοιώνονται μαζί. Λόγω της αυξημένης λεπτομέρειας, είναι η πιο απαιτητική μέθοδος σε υπολογιστική ισχύ.

2.3 Προκλήσεις στον Προγραμματισμό Συστημάτων Πραγματικού Χρόνου

Στα συστήματα πραγματικού χρόνου απαιτείται οι γλώσσες προγραμματισμού που χρησιμοποιούνται να είναι απολύτως ντετερμινιστικές. Μη αιτιατά χαρακτηριστικά, όπως για παράδειγμα ένας ενσωματωμένος συλλέκτης απορριμάτων, όπως αυτός της Java, που μπορεί να ενεργοποιηθεί σε τυχαίες στιγμές προκαλώντας τυχαίες καθυστερήσεις είναι μη αποδεκτά. Επιπλέον, οι προγραμματιστές τέτοιων συστημάτων πρέπει να έχουν γνώση του υλικού στο οποίο αναπτύσσουν το λογισμικό τους και των υπηρεσιών του λειτουργικού συστήματος αν αυτό υπάρχει. Χρησιμοποιούν χαμηλού επιπέδου προγραμματισμό και διαχειρίζονται λειτουργίες που αποκρύπτονται από τις γλώσσες προγραμματισμού υψηλού επιπέδου. Σύμφωνα με το [11] τέτοιες λειτουργίες αφορούν δομές δεδομένων όπως είναι οι σημαφόροι, οι μακροεντολές, οι διασωληνώσεις, η δημιουργία script και οι θύρες επικοινωνίας.

2.3.1 Δομές Δεδομένων που Χρησιμέυουν στον Προγραμματισμό Συστημάτων Πραγματικού Χρόνου

Ο σημαφόρος, μια έννοια που δημιουργήθηκε από τον Edsger Dijkstra, χρησιμοποιείται από τους προγραμματιστές για τον συντονισμό των διαφόρων διεργασιών και της χρήσης των κοινών πόρων. Οι δύο τύποι σημαφόρων, δυαδικός (binary) και μέτρησης (counting), προσφέρουν αμοιβαίο αποκλεισμό και έλεγχο πρόσβασης πολλών διεργασιών σε έναν κοινό πόρο αντιστοίχως.

Μια μακροεντολή μπορεί να αντικαταστήσει ένα μπλοκ εντολών σε ένα πρόγραμμα. Μοιάζει με υπορουτίνα, αλλά διαφέρει στον τρόπο κλήσης. Μια υπορουτίνα λειτουργεί με άλμα στην θέση μνήμης όπου είναι αποθηκευμένες οι εντολές, ενώ μια μακροεντολή λειτουργεί με αντικατάσταση κειμένου, σαφώς πιο γρήγορα, με το μειονέκτημα ότι καταλαμβάνει περισσότερη μνήμη.

Η διασωλήνωση είναι μια ροή δεδομένων μεταξύ διεργασιών με σκοπό είτε την διασύνδεση τους είτε την ανταλλαγή μηνυμάτων μεταξύ τους. Μπορεί να υλοποιηθεί με μια First In First Out (FIFO) ουρά και εξαλείφει την ανάγκη χρήσης προσωρινών αρχείων για αποθήκευση αποτελεσμάτων μειώνοντας τον απαιτούμενο χώρο αποθήκευσης.

Ένα script είναι ένα αρχείο των χαρακτήρων που χρησιμοποιείται σαν είσοδος ή για παροχή οδηγιών σε ένα πρόγραμμα. Μπορεί να προσομοιώσει την διαδραστική διεπαφή με έναν χρήστη ή άλλες συσκευές εισόδου-εξόδου. Ένα αρχείο script θα μπορούσε να είναι μια λίστα εντολών για ένα διερμηνέα όπως ένα batch file.

Μια θύρα επικοινωνίας αποτελείται από μια ουρά για την αποθήκευση μηνυμάτων και δύο σημαφόρους. Ένας σημαφόρος ελέγχει τους παραγωγούς, τις διαδικασίες που παράγουν τα μηνύματα, ενώ ο άλλος ελέγχει τους καταναλωτές, δηλαδή τις διαδικασίες που χρησιμοποιούν τα μηνύματα.

2.3.2 Δομές Ελέγχου που Χρησιμεύουν στον Προγραμματισμό Συστημάτων Πραγματικού Χρόνου

Πέρα από τις δομές δεδομένων που προαναφέρθηκαν, συχνά γίνεται χρήση δύο βασικών δομών ελέγχου λογισμικού, του βρόχου polling και των συστημάτων οδηγούμενων από γεγονότα (event driven systems). [11]

Σε έναν βρόχο polling, το πρόγραμμα εξετάζει κάθε είσοδο με τη σειρά για να δει αν ένα συμβάν έχει συμβεί. Η δομή του προγράμματος είναι ένας βρόχος, ενώ οι είσοδοι που πρέπει να εξεταστούν είναι προκαθορισμένες. Στην περίπτωση ανίχνευσης συμβάντος, η διαδικασία ελέγχου σταματάει, λαμβάνεται κάποια δράση και μετά συνεχίζεται. Για παράδειγμα, ο έλεγχος της θερμοκρασίας ενός ψυγείου μπορεί να χρησιμοποιεί μια τέτοια δομή ελέγχου. Η θερμοκρασία θα πρέπει να διαβάζεται σαν είσοδος και με βάση την ένδειξη ενεργοποιείται και απενεργοποιείται ο συμπιεστής. Αν η θερμοκρασία είναι μέσα στα προκαθορισμένα όρια, δεν γίνεται καμία ενέργεια.

Σε συστήματα οδηγούμενα από γεγονότα, το πρόγραμμα εκτελείται συνεχώς σε βρόχο μέχρι να συμβεί διακοπή οπότε η εκτέλεση του σταματάει για να εξυπηρετηθεί η διακοπή. Η καθυστέρηση διακοπής (interrupt latency) ορίζεται ως ο χρόνος που περνάει από την στιγμή εμφάνισης της διακοπής μέχρι την εκτέλεση της υπορουτίνας εξυπηρέτησης αυτής. Το αίτημα εξυπηρέτησης της διακοπής δεν είναι απαραίτητο να έχει άμεση ανταπόκριση αν το πρόγραμμα εξυπηρετεί μια άλλη λειτουργία. Ο χρόνος καθυστέρησης διακοπής και ο χρόνος εξυπηρέτησης της, μελετούνται σε βάθος από ένα σχεδιαστή συστημάτων πραγματικού χρόνου. [12]

Υπάρχουν 3 είδη συστημάτων οδηγούμενων από γεγονότα [11]:

- Προσκήνιο/Παρασκήνιο (Foreground/Background), που αποτελεί βασικά ένα βρόχο polling με ενεργοποιημένες τις διακοπές. Ο βρόχος τρέχει στο παρασκήνιο και από τις διακοπές εξυπηρετούνται μόνο κριτικής σημασίας συμβάντα.
- Multitasking, πού είναι μια τεχνική για την κατανομή του χρόνου επεξεργασίας της CPU μεταξύ διαφόρων εργασιών. Όσο η εκτέλεση μιας διεργασίας δεσμεύει τους φυσικούς πόρους του επεξεργαστή, οι άλλες διεργασίες έχουν τους πόρους τους αποθηκευμένους στη μνήμη του συστήματος. Τα συστήματα αυτά χαρακτηρίζονται ως preemtive ή non-preemptive ανάλογα με το αν μπορούν να διακόψουν μια τρέχουσα εργασία. Σε ένα preemtive σύστημα, σε κάθε διεργασία δίνεται ένα προκαθορισμένο χρονικό διάστημα.
- Multiprocessor συστήματα που διαθέτουν παραπάνω από ένα επεξεργαστή. Αυτά τα συστήματα, καθώς και τα προηγούμενα, προϋποθέτουν την ύπαρξη λειτουργικού συστήματος που αναλαμβάνει τον συγχρονισμό των διεργασιών και την ανταλλαγή πληροφοριών μεταξύ τους.

2.3.3 Προκλήσεις κατά τον Προγραμματισμό Συστημάτων Πραγματικού Χρόνου

Κατά το [13] υπάρχουν αρκετές παγίδες κατά τη συγγραφή προγραμμάτων για συστήματα πραγματικού χρόνου. Αυτά αποτελούν κυρίως τη χρήση ασαφών ως προς τον χρόνο εκτέλεσης δομών που προκαλούν τυχαίες καθυστερήσεις και προσβάλλουν την αξιοπιστία του συστήματος.

Αναλυτικά αυτές είναι:

- Πολλές και μεγάλες δομές τύπου if-then-else ή case. Οι δομές αυτές εισάγουν στον κώδικα πολλά μονοπάτια διαφορετικού μήκους που έχουν διαφορετικούς χρόνους εκτέλεσης.
- Καθυστερήσεις υλοποιημένες με κενούς βρόχους (dummy loops), οι οποίες οδηγούν σε ανακριβείς καθυστερήσεις που εξαρτώνται από το υλικό. Στη θέση αυτών προτιμώνται οι μηχανισμοί των λειτουργικών συστημάτων πραγματικού χρόνου για την υλοποίηση καθυστερήσεων.
- Η αδιάκριτη χρήση διακοπών μειώνει την αξιοπιστία του συστήματος, αφού συνήθως οι διακοπές έχουν υψηλή προτεραιότητα εξυπηρέτησης και συμβαίνουν εκτός της χρονοδρομολόγησης. Επιπλέον δυσχεραίνουν την αποσφαλμάτωση του προγράμματος και τρέχουν στον χώρο του πυρήνα του λειτουργικού προγράμματος (kernel space). Ενώ σε ένα μικρό σύστημα πραγματικού χρόνου, βασισμένο για παράδειγμα σε έναν μικροεπεξεργαστή, ο μηχανισμός των διακοπών εξοικονομεί το χρόνο της κεντρικής μονάδας επεξεργασίας, σε ένα πολύπλοκο σύστημα πραγματικού χρόνου είναι προτιμότερη η χρήση non-preemtive περιοδικών διεργασιών ώστε οι χρόνοι εκτέλεσης να είναι προκαθορισμένοι.
- Υλοποίηση με ένα μεγάλο μοναδικό βρόχο. Με αυτόν τον τρόπο ο συγχρονισμός των διεργασιών και η ρύθμιση του χρόνου εκτέλεσης δυσκολεύει.
- Η χρήση της τεχνικής περάσματος μηνυμάτων σαν μόνος τρόπος επικοινωνίας μεταξύ των διεργασιών. Η τεχνική αυτή μειώνει κατά πολύ την προβλεψιμότητα του συστήματος και επιβαρύνει λόγω των μη περιοδικών αιτημάτων, ενώ παράλληλα μπορεί να οδηγήσει σε deadlocks κατά την εκτέλεση του προγράμματος. Στη θέση του μπορεί να χρησιμοποιηθεί κοινός χώρος στη μνήμη σε συνδυασμό με τεχνικές συγχρονισμού και αναστροφή προτεραιοτήτων.
- Σχεδιασμός χωρίς μελέτη χρήσης χειρότερης περίπτωσης και ανάλυση του χρόνου εκτέλεσης. Συχνά ένα πρόγραμμα θεωρείται αυθαίρετα αρκετά μικρό με επαρκή χρόνο εκτέλεσης, γεγονός που συχνά οδηγεί σε απρόοπτα σφάλματα. Το σύστημα πρέπει να σχεδιάζεται με τέτοιο τρόπο ώστε ο χρόνος εκτέλεσης μιας εντολής να είναι μετρήσιμος ανά πάσα στιγμή.
- Η μεγαλύτερη παγίδα στην δημιουργία προγραμμάτων για συστήματα πραγματικού χρόνου είναι η παράβλεψη κάποιου σφάλματος κατά την αποσφαλμάτωση. Ο κώδικας πρέπει να ελέγχεται διεξοδικά για σφάλματα πριν το τελικό στάδιο χρήσης του γιατί σφάλματα που είναι καλά κρυμμένα μπορεί να προκαλέσουν πρόβλημα κατά την εκτέλεση του προγράμματος. Σε τέτοια συστήματα όπου η ασφάλεια και η αξιοπιστία πρέπει να είναι δεδομένες, αυτό είναι μη αποδεκτό.

Στις γλώσσες προγραμματισμού που είναι διαδεδομένες για συστήματα πραγματικού χρόνου περιλαμβάνονται συμβολικές γλώσσες, η C και η C++, η Forth, η Ada και η real time Java. Η Java, λόγω του αυτόματου μηχανισμού σύλλεξης απορριμάτων μειώνει τον ντετερμινισμό του συστήματος και δεν συνιστάται. Ο προγραμματισμός σε πραγματικό χρόνο περιλαμβάνει την παρακολούθηση του χρόνου, το συντονισμό των διεργασιών, και την προβλεψιμότητα των γεγονότων εντός των ορίων που αυτό είναι εφικτό. Η κατανόηση της επίδρασης του υλικού στον χρόνο εκτέλεσης και των εννοιών του λειτουργικού συστήματος είναι απαραίτητη προϋπόθεση.

2.4 Λειτουργικά Συστήματα Πραγματικού Χρόνου

2.4.1 Εισαγωγή

Η περιγραφή των λειτουργικών συστημάτων που ακολουθεί βασίζεται κυρίως στα [14], [15].

Το λειτουργικό σύστημα είναι ένα λογισμικό το οποίο παρέχει λειτουργικότητα και ένα στρώμα αφαιρετικότητας στο υλικό. Προσδίδει χαρακτηριστικά και λειτουργίες με το πιο σημαντικό να είναι ο πολυπρογραμματισμός (multiprogramming), ο οποίος αναφέρεται στην συνύπαρξη πολλών προγραμμάτων στη μνήμη ενός υπολογιστή που χρησιμοποιούν το στοιχείο επεξεργασίας με τη σειρά. Το κομμάτι του λειτουργικού συστήματος το οποίο παρέχει τη δυνατότητα του πολυπρογραμματισμού είναι ο πυρήνας (kernel), ο οποίος είναι υπεύθυνος για την δημιουργία νέων διεργασιών, τον μηχανισμό λήψης αποφάσεων για το ποια διεργασία θα αναλάβει ο επεξεργαστής αλλά και για να γίνει η μετάβαση από τη μία στην άλλη, ενώ παράλληλα καθορίζει τη χρήση των κοινών πόρων που έχουν οι διάφορες διεργασίες.

Επιπλέον παρέχει την διεπαφή μεταξύ των συσκευών εισόδου/εξόδου και του υπολογιστικού συστήματος. Αυτό σημαίνει ότι το λειτουργικό σύστημα παρέχει ένα στρώμα αφαιρετικότητας, τέτοιο ώστε οι εφαρμογές χρήστη εκτελούνται και αλληλεπιδρούν με το υλικό χωρίς να έχουν γνώση του τύπου του υλικού ή και του πως αυτό λειτουργεί. Ελέγχει την πρόσβαση στο υλικό, εάν πολλές διεργασίες προσπαθούν να έχουν ταυτόχρονη πρόσβαση στην ίδια συσκευή εισόδου/εξόδου, την πρόσβαση στην μνήμη καθώς και την διαχείριση της.

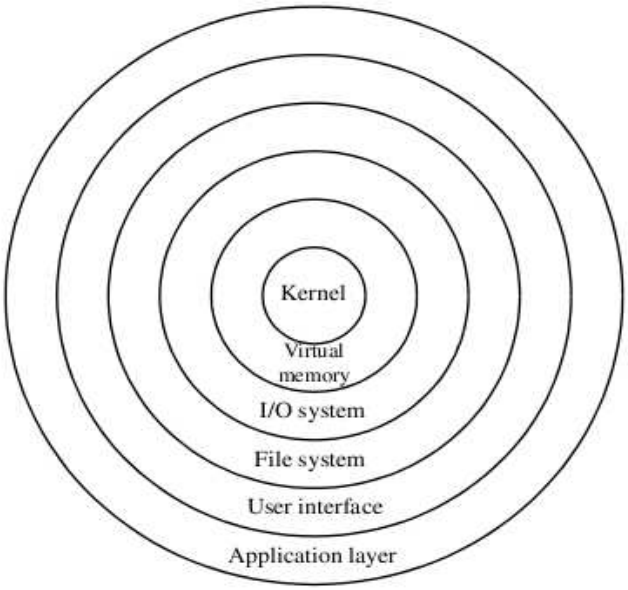
Τέλος, διαθέτει σύστημα διαχείρισης αρχείων το οποίο επιτρέπει στον χρήστη και τις εφαρμογές του διαδικασίες διαγραφής, μεταφοράς και αποθήκευσης αρχείων στη μνήμη ή σε εξωτερικές συσκευές αποκρύπτοντας την κατανομή των μπλοκ φυσικής μνήμης στα αρχεία αυτά. Συνοψίζοντας, το λειτουργικό σύστημα παρέχει τον μηχανισμό για την κατανομή των πόρων και ελέγχει την πρόσβαση στους πόρους.

Τα λειτουργικά συστήματα κατηγοριοποιούνται ανάλογα με τις εφαρμογές που υποστηρίζουν σε πραγματικού χρόνου και μη πραγματικού χρόνου. Ακόμη, χωρίζονται σε στατικά και δυναμικά. Σε στατικά λειτουργικά συστήματα ο αριθμός των διεργασιών πρέπει να καθορίζεται σαφώς πριν το χρόνο λειτουργίας (run time) ώστε αυτό να δημιουργεί τις διεργασίες κατά την εκκίνηση του, ενώ σε δυναμικά συστήματα ο αριθμός των διεργασιών που θα αναλάβουν δεν είναι προκαθορισμένος και αυτές δημιουργούνται και καταστρέφονται όσο αυτό λειτουργεί.

Η διάταξη των σύγχρονων λειτουργικών συστημάτων μπορεί να είναι μονολιθική (monolithic), πολυεπίπεδη (layered) ή στη μορφή εικονικών μηχανών (virtual machines), εξωπυρήνων (exokernels) και συστημάτων πελάτη-εξυπηρετητή (client-server).

Ένα μονολιθικό λειτουργικό σύστημα είναι γραμμένο σαν συλλογή διαδικασιών χωρίς ιδιαίτερη δομή πέρα από το κύριο πρόγραμμα, ένα σύνολο διαδικασιών εξυπηρέτησης και βοηθητικές διαδικασίες. Κάθε διαδικασία έχει καλά καθορισμένες παραμέτρους, καθορισμένη διασύνδεση στα αποτελέσματα που παράγει και είναι ελεύθερη να καλέσει κάποια από τις άλλες διαδικασίες.

Το λειτουργικό σύστημα σε πολυεπίπεδη διάταξη αποτελείται από στρώματα, με το κάθε στρώμα να έχει σαφώς καθορισμένο ρόλο και διασύνδεση με τα στρώματα πάνω και κάτω από αυτό. Στην πολυεπίπεδη δομή, κάθε στρώμα μπορεί να αλλάξει σημαντικά με μικρή ή και καθόλου επιρροή στα άλλα στρώματα του συστήματος. Επιπλέον, επιβάλλει μια λογική δομή η οποία καθιστά το συνολικό σύστημα πιο κατανοητό, και συνεπώς διευκολύνει την συντήρηση του. Είναι και η πιο διαδεδομένη δομή λειτουργικών συστημάτων. Τα στρώματα της δομής αυτής φαίνονται στην Εικόνα 2.



Εικόνα 2: Δομή Πολυεπίπεδου Λειτουργικού Συστήματος

Ο σκοπός του πυρήνα όπως προαναφέρθηκε, είναι να παρέχει τη δυνατότητα multitasking στο σύστημα. Με άλλα λόγια, είναι το μέρος του συστήματος που μπορεί να κάνει την αλλαγή από μια διεργασία σε μια άλλη. Είναι συνήθως ένα σχετικά μικρό μέρος του κώδικα του λογισμικού, και συχνά είναι γραμμένο σε συμβολική γλώσσα, δεδομένου ότι απαιτεί πρόσβαση σε πολύ χαμηλό επίπεδο λειτουργικότητας στον επεξεργαστή. Μια σύγχρονη τάση στα λειτουργικά συστήματα είναι να έχουμε ένα μικρο-πυρήνα (microkernel) που είναι ένας πυρήνας με πολύ μικρό μέγεθος. Η τάση αυτή αναπτύχθηκε λόγω της φορητότητας που προσφέρει ένας πυρήνας μικρού μεγέθους. Τα στρώματα που λειτουργούν πάνω από τον πυρήνα είναι συνήθως γραμμένα σε κάποια υψηλή γλώσσα προγραμματισμού. Το επόμενο στρώμα της εικονικής μνήμης διαχειρίζεται την κατανομή και

προστασία της μνήμης. Τα εξωτερικά στρώματα αποτελούνται συνήθως από μεγάλα ποσά κώδικα και παρέχουν δυνατότητες όπως διαχείριση εισόδου/εξόδου, σύστημα αρχείων, περιβάλλον εργασίας χρήστη και υποστήριξη εφαρμογών. Τα στρώματα χρήστη και εφαρμογών είναι τα λιγότερο προνομιούχα, με τον μικρότερο βαθμό πρόσβασης στο υλικό.

Οι εικονικές μηχανές παρέχουν πολυπρογραμματισμό ενώ εκτελούνται απευθείας στο υλικό και παρέχουν πολλές εικονικές μηχανές στο επόμενο επίπεδο. Σε αντίθεση με τους άλλους τύπους λειτουργικών συστημάτων, δεν αποτελούνται από αρχεία και διαδικασίες, αλλά είναι πανομοιότυπες με το πραγματικό υλικό και μπορούν να εκτελέσουν οποιοδήποτε λειτουργικό σύστημα.

Ο εξωπυρήνας είναι ένα πρόγραμμα που κατανέμει τους πόρους ενός συστήματος σε εικονικές μηχανές και ελέγχει τις απόπειρες χρήσης τους ώστε να εξασφαλιστεί ότι μια μηχανή δεν προσπαθεί να έχει πρόσβαση σε πόρους που ανήκουν σε μια άλλη μηχανή. Έχει το πλεονέκτημα ότι διαχωρίζει τον πολυπρογραμματισμό από τον κώδικα του λειτουργικού συστήματος του χρήστη με μικρή επιβάρυνση.

Στο μοντέλο πελάτη-εξυπηρετητή, το λειτουργικό σύστημα χωρίζεται σε τμήματα καθένα από τα οποία χειρίζεται μια πλευρά του συστήματος μόνο, πχ διαχείριση αρχείων, που διατηρούν με τις διαδικασίες εξυπηρέτησης επικοινωνία πελάτη-εξυπηρετητή. Για παράδειγμα, η ανάγνωση ενός αρχείου περιλαμβάνει την διαδικασία-πελάτη που είναι η διεργασία που κάλεσε ο χρήστης για να αναγνώσει το αρχείο και την διαδικασία-εξυπηρετητή στην πλευρά του λειτουργικού που εκτελεί το αίτημα και αποστέλλει την απάντηση. Ο πυρήνας στο μοντέλο αυτό απλά διαχειρίζεται την επικοινωνία αυτή.

2.4.2 Απαιτήσεις Λειτουργικών Συστημάτων Πραγματικού Χρόνου

Το multitasking πραγματικού χρόνου μπορεί να επιτευχθεί χωρίς την χρήση διακοπών, ακόμα και χωρίς λειτουργικό σύστημα αυτό καθεαυτό. Για ένα μικρής πολυπλοκότητας σύστημα τέτοιες προσεγγίσεις είναι ιδανικές, αφού η ανάλυση τους είναι απλή και σύντομη. Ψευδοπυρήνες (pseudokernel) που χρησιμοποιούν βρόχους polling μπορούν να χρησιμοποιηθούν σε μικρά συστήματα πραγματικού χρόνου. [2]

Με την τάση να χρησιμοποιούνται ολοένα και περισσότερα ενσωματωμένα συστήματα πραγματικού χρόνου και την ολοένα αυξανόμενη πολυπλοκότητα τους, αναπτύχθηκε και η ανάγκη για λειτουργικά συστήματα που να καλύπτουν τις απαιτήσεις των συστημάτων αυτών. Τα λειτουργικά συστήματα πραγματικού χρόνου (Real Time Operating System-RTOS) αυξάνουν την ευελιξία και τις δυνατότητες στον προγραμματισμό συστημάτων πραγματικού χρόνου. Τα RTOS μοιράζονται κάποια χαρακτηριστικά με τα λειτουργικά συστήματα γενικού σκοπού (General Purpose Operating System-GPOS) [16]:

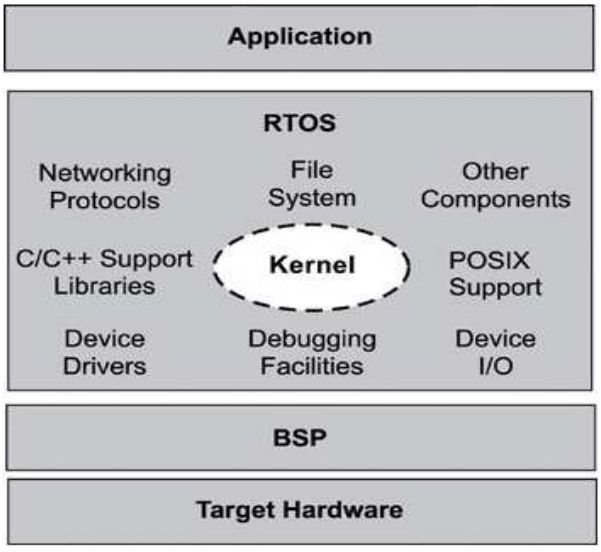
- Διαθέτουν σε κάποιο βαθμό multitasking.
- Έχουν την δυνατότητα να διαχειρίζονται πόρους υλικού και λογισμικού.
- Παρέχουν σημαντικές υπηρεσίες του λειτουργικού συστήματος στις εφαρμογές.
- Θέτουν ένα επίπεδο αφαιρετικότητας αποκρύπτοντας το υλικό από το λογισμικό των εφαρμογών.

Πέρα από τις ομοιότητες τους, διαφέρουν σε σημεία όπως τα παρακάτω:

- Προσφέρουν αυξημένη αξιοπιστία στα πλαίσια των εφαρμογών ενσωματωμένων συστημάτων.
- Έχουν την ικανότητα να αναβαθμιστούν ή να υποβαθμιστούν ανάλογα με τις ανάγκες της εφαρμογής.
- Αποδίδουν ταχύτερα.
- Έχουν μειωμένες απαιτήσεις σε μνήμη.
- Χρησιμοποιούν τεχνικές χρονοδρομολόγησης σχεδιασμένες για ενσωματωμένα συστήματα πραγματικού χρόνου.
- Υποστηρίζουν συστήματα χωρίς δίσκους επιτρέποντας την εκκίνηση εκτελέσιμων από μνήμες τυχαίας πρόσβασης και μνήμες μόνο ανάγνωσης.
- Διαθέτουν αυξημένο βαθμό φορητότητας για διαφορετικές πλατφόρμες υλικού.

Ένα λειτουργικό σύστημα πραγματικού χρόνου (RTOS) είναι ένα πρόγραμμα που προγραμματίζει την εκτέλεση των εφαρμογών σε εύθετο χρόνο, διαχειρίζεται τους πόρους του συστήματος και παρέχει τη βάση για την ανάπτυξη κώδικα εφαρμογών. Οι εφαρμογές που σχεδιάζονται σε RTOS μπορεί να κυμαίνονται από μια απλή εφαρμογή ψηφιακού χρονομέτρου σε μια πολύ πιο σύνθετη εφαρμογή πλοήγησης αεροσκαφών. [16] Γενικότερα, ένα λειτουργικό σύστημα θεωρείται ντετερμινιστικό, όταν ο χρόνος χειρότερης εκτέλεσης είναι υπολογίσιμος.

Για παράδειγμα, σε ορισμένες εφαρμογές, ένα RTOS μπορεί να περιλαμβάνει μόνο ένα πυρήνα ο οποίος είναι το εποπτικό λογισμικό που παρέχει την ελάχιστη λογική, χρονοδρομολόγηση και αλγορίθμους διαχείρισης των πόρων. Από την άλλη πλευρά, ένα RTOS μπορεί να είναι ένας συνδυασμός από διάφορα στοιχεία συμπεριλαμβανομένου του πυρήνα, όπως ένα σύστημα αρχείων, στοίβες πρωτοκόλλων δικτύου όπως φαίνεται και σε υψηλό επίπεδο στην Εικόνα 3.



Εικόνα 3: Στοιχεία Λειτουργικού Συστήματος Πραγματικού Χρόνου

Ο πυρήνας ενός RTOS, περιλαμβάνει το χρονοδρομολογητή (scheduler), αντικείμενα (objects) και υπηρεσίες (services). Ο χροόνοδρομολογητής αναλαμβάνει με βάση διάφορους αλγορίθμους τον προγραμματισμό της εκτέλεσης των διεργασιών, δηλαδή το ποια διεργασία θα αναλάβει ο επεξεργαστής και πότε θα συμβεί αυτό. Παράδειγμα αλγορίθμων χρονοδρομολόγησης είναι ο round robin και ο preemtive χρονοδρομολογητής. Τα αντικείμενα είναι δομές του πυρήνα που επιτρέπουν την ανάπτυξη εφαρμογών για ενσωματωμένα συστήματα πραγματικού χρόνου από προγραμματιστές, όπως είναι οι σημαφόροι και οι ουρές. Οι υπηρεσίες είναι λειτουργίες που επιτελεί ο πυρήνας σε σχέση με τα αντικείμενα, αλλά και λειτουργίες όπως ο χρονισμός, η διαχείριση πόρων και διακοπών. [16]

Τα RTOS πρέπει να παρέχουν τεσσερις συγκεκριμένες λειτουργίες όσον αφορά τις διεργασίες: χρονοδρομολόγηση (scheduling), αποστολή (dispatching), επικοινωνία (intercommunication) και συγχρονισμό (synchronization).[2] Ακόμη, πρέπει να διαθέτουν λειτουργίες διαχείρισης μνήμης και υποστήριξη για δίκτυα.

Το βασικό μέρος του κώδικα ενός RTOS είναι ο διαχειριστής διεργασιών (task manager). Αυτός αποτελείται από τον χρονοδρομολογητή (scheduler) και τον dispatcher. [13]

Ο χρονοδρομολογητής αναλαμβάνει να επιλέξει την διεργασία η οποία θα χρησιμοποιήσει το στοιχείο επεξεργασίας. Οι αλγόριθμοι που χρησιμοποιούνται για την επιλογή αυτή, είναι μια από τις κυριότερες διαφορές μεταξύ GPOS και RTOS. Τα συστήματα πραγματικού χρόνου απαιτούν ιδιαίτερους αλγορίθμους χρονοδρομολόγησης για να διατηρήσουν την αξιοπιστία τους ακόμα και στα χειρότερα σενάρια. Οι αλγόριθμοι χρονοδρομολόγησης που συναντώνται μπορεί να είναι στατικοί ή δυναμικοί. Η στατική χρονοδρομολόγηση απαιτεί οι πληροφορίες (ο αριθμός των διεργασιών, οι προθεσμίες, οι προτεραιότητες, οι περίοδοι εκτέλεσης) να είναι γνωστές εκ των προτέρων. Αν οι πληροφορίες αυτές μπορούν να αλλάξουν κατά τη διάρκεια του χρόνου εκτέλεσης, τότε μιλάμε για δυναμική χρονοδρομολόγηση.

Ο dispatcher αναλαμβάνει την αλλαγή περιεχομένου (context switch), το να σώσει δηλαδή τις παραμέτρους για την διεργασία της οποίας η εκτέλεση τερματίζεται ενώ πρέπει να φορτώσει τις παραμέτρους της διεργασίας προς εκτέλεση, και το να αναθέσει την κεντρική μονάδα επεξεργασίας στην διεργασία προς εκτέλεση.

Για την διατήρηση της αξιοπιστίας του συστήματος, αυστηρά καθορισμένη πρέπει να είναι και η ανταλλαγή πληροφοριών μεταξύ των διεργασιών (inter-task communication), καθώς επίσης και ο συγχρονισμός τους (synchronization).

Είναι απαραίτητο οι μηχανισμοί επικοινωνίας που χρησιμοποιούνται να παρέχουν ντετερμινιστική συμπεριφορά που μπορεί να επιτευχθεί κατασκευάζοντας ένα πρωτόκολλο επικοινωνίας με ντετερμινιστικά χαρακτηριστικά, όπως οριοθετημένη καθυστέρηση πρόσβασης καναλιού και οριοθετημένη καθυστέρηση μηνύματος. Η καθυστέρηση πρόσβασης καναλιού ορίζεται ως η χρονική απόσταση μεταξύ της στιγμής που μια διεργασία θέτει το αίτημα για χρήση του επικοινωνιακού διαύλου και της στιγμής που όντως ξεκινάει να μεταδίδει πληροφορίες στον δίαυλο. Η καθυστέρηση μηνύματος ορίζεται ως ο χρόνος που χρειάζεται από τη στιγμή που μια διεργασία ζητάει να στείλει ένα μήνυμα έως την στιγμή που αυτό το μήνυμα θα παραδοθεί στον προορισμό του. Εάν ξεπεραστεί το όριο που έχει τεθεί για αυτές τις καθυστερήσεις, η επικοινωνία θεωρείται ανεπιτυχής. [17] Διάφορα πρωτόκολλα έχουν αναπτυχθεί με σκοπό την αξιόπιστη επικοινωνία μεταξύ διεργασιών. Η ανάγκη για αξιόπιστη επικοινωνία αυξάνει με την χρήση κατανεμημένων συστημάτων πραγματικού χρόνου, η λειτουργία των οποίων βασίζεται στην αξιόπιστη και ασφαλή ανταλλαγή μηνυμάτων μεταξύ τους.

Ο συγχρονισμός των διεργασιών είναι ζωτικής σημασίας, αφού κάποιες διεργασίες μπορεί να αντιμετωπίσουν απρόβλεπτες καθυστερήσεις λόγω της δέσμευσης σε κοινούς πόρους που απαιτούν αποκλειστική πρόσβαση (π.χ. A/D και D/A converters) και επιπλέον, η εκτέλεση ορισμένων διεργασιών εξαρτάται από τα αποτελέσματα της εκτέλεσης άλλων διεργασιών. Σημαφόροι, προτεραιότητες και αντιστροφή προτεραιοτήτων, είναι μερικοί από τους μηχανισμούς που χρησιμοποιούνται για την επίτευξη του συγχρονισμού και την αποφυγή των αδιεξόδων. [13]

Η δυναμική κατανομή μνήμης, είναι σημαντική από την άποψη τόσο της χρήσης της μνήμης on demand από τις εφαρμογές όσο και για τις απαιτήσεις του λειτουργικού συστήματος. Οι εφαρμογές χρησιμοποιούν τη μνήμη άμεσα, για παράδειγμα μέσω αιτήσεων για τη μνήμη σωρού, και έμμεσα, μέσω της διατήρησης της μνήμης κατά τον χρόνο εκτέλεσης που απαιτείται για την υποστήριξη υψηλού επιπέδου γλωσσών προγραμματισμού. Το λειτουργικό σύστημα πρέπει να εκτελεί εκτενή διαχείριση της μνήμης, ώστε οι διεργασίες να παραμένουν απομονωμένες. Επικίνδυνη δυναμική κατανομή της μνήμης είναι οποιαδήποτε κατανομή που βλάπτει τον ντετερμινισμό του συστήματος. Μια επικίνδυνη κατανομή μπορεί να καταστρέψει τον ντετερμινισμό των γεγονότων, για παράδειγμα υπερχειλίζοντας την στοίβα, ή να καταστρέψει τον χρονικό ντετερμινισμό προκαλώντας αδιέξοδο. Ως εκ τούτου, είναι σημαντικό να αποφεύγεται η επικίνδυνη κατανομή της μνήμης, ενώ ταυτόχρονα μειώνεται η επιβάρυνση που προκαλείται από την κατανομή της μνήμης. Αυτή η επιβάρυνση είναι ένα τυπικό στοιχείο του χρόνου αλλαγής περιεχομένου και πρέπει να ελαχιστοποιηθεί. [2]

Σύμφωνα με το [2], τα RTOS μπορεί να ακολουθούν τα εξής μοντέλα απόκρισης στα αιτήματα διεργασιών:

- Συστήματα οδηγούμενα από διακοπές (interrupt driven systems), στα οποία το κύριο πρόγραμμα είναι μια απλή εντολή άλματος στον εαυτό του ώστε να έχουμε έναν συνεχιζόμενο βρόχο. Οι διεργασίες που επιτελούνται στο σύστημα προγραμματίζονται από διακοπές υλικού ή λογισμικού και η απόκριση του συστήματος είναι με τη μορφή υπορουτίνων εξυπηρέτησης διακοπών.

- Συστήματα διακοπτόμενης προτεραιότητας (preemtive priority systems). Στα συστήματα αυτά μια διεργασία υψηλής προτεραιότητας μπορεί να διακόψει την εκτέλεση μιας διεργασίας χαμηλότερης προτεραιότητας. Ο βαθμός της προτεραιότητας μιας διεργασίας καθορίζεται από το πόσο επείγουσα για την αξιοπιστία και ασφάλεια του συστήματος είναι η εξυπηρέτηση της και μπορεί να καθορίζεται στατικά ή δυναμικά.

- Υβριδικά συστήματα, όπου γίνεται χρήση συνδυασμού αλγορίθμων χρονοδρομολόγησης. Για παράδειγμα, ένα σύστημα μπορεί να είναι διακοπτόμενης προτεραιότητας, αλλά όταν δύο διεργασίες με τον ίδιο βαθμό προτεραιότητας επιχειρούν ταυτόχρονη εκτέλεση, εξυπηρετούνται με τον round-robin αλγόριθμο εκτέλεσης.

Μερικά παραδείγματα συστημάτων πραγματικού χρόνου είναι τα εξής: QNX, RTLinux, Windows CE, AMX (KADAK), C Executive (JMI Software), RTX (CMX Systems), eCos (Red Hat), LynxOS (LynuxWorks), µC/OS-II (Micrium), Nucleus (Mentor Graphics), Quadros (RTXC), ThreadX (Express Logic), Linux/RT (TimeSys), VRTX (Mentor Graphics), VxWorks (Wind River).

Αναφορές

[1] H. Kopetz, *Real-time systems*, Springer, 1997.

[2] P.A. Laplante, *Real-time systems design and analysis*, Wiley-IEEE, 2004.

[3] A.M.K. Cheng, *Real-time systems*, John Wiley and Sons, 2002.

[4] A. Burns and A.J. Wellings, *Real-time systems and programming languages*, Pearson Education, 2001.

[5] Donglin Liu, Xiaobo Sharon Hu, Senior Member, IEEE, Michael D. Lemmon, and Qiang Ling, "Firm Real-Time System Scheduling Based on a Novel QoS Constraint," 2006.

[6] "Issues in Embedded and Realtime System Design"; http://www.eventhelix.com/RealtimeMantra/IssuesInRealtimeSystemDesign.htm.

[7] D. Ionescu and A. Cornell, *Real-time systems*, World Scientific, 2007.

[8] "ASIC Design Home Page"; http://www.ict.tuwien.ac.at/asicdesign/lehre/384089.html.

[9] Stephan Merz Nicolas Navet, *Modeling and Verification of Real-Time Systems : Formalisms and Software Tools*, Great Britain: ISTE Ltd and John Wiley & Sons, Inc., 2008.

[10] J. Sifakis, *Modelling Real-Time Systems: Challenges and Work Directions*, VERIMAG, 2 rue Vignate, 38610, Gieres, France: .

[11] "STSC CrossTalk - An Introduction to Real-Time Programming - Nov 2003"; http://www.stsc.hill.af.mil/crosstalk/2003/11/0311ludwig.html.

[12] S.R. Savitzky, *Real-time microprocessor systems*, Van Nostrand Reinhold, 1985.

[13] Gambier A, "Real-time Control Systems: A Tutorial," Automation Laboratory, B6 23-29, EG. Bauteil C University of Mannheim, 68131 Mannheim, Germany: .

[14] A.S. Tanenbaum, *Modern operating systems*, Prentice Hall, 1992.

[15] R. Betz, *Introduction to Real Time Operating Systems*, Australia: 2001.

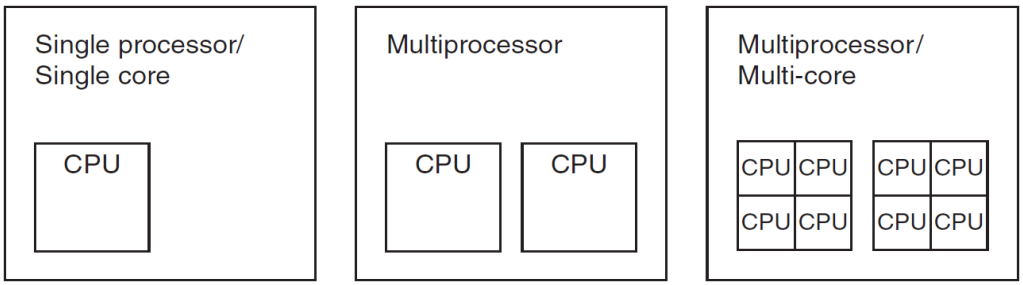
[16] Qing Li and Carolyn Yao , *Real-Time Concepts for Embedded Systems*, CMP Books, 2003.

[17] F. Panzieri R. Davoli, *Real Time Systems: A Tutorial*, Piazza di Porta S. Donato, 5 40127 Bologna (Italy): Laboratory for Computer Science University of Bologna , 1993.

3 Συστήματα Πολυεπεξεργασίας

3.1 Πολυεπεξεργασία (Multiprocessing)

Η έννοια της πολυεπεξεργασίας αναφέρεται είτε στην ύπαρξη δύο ή και παραπάνω κεντρικών μονάδων επεξεργασίας σε ένα σύστημα με την ικανότητα της κατανομής διεργασιών σε αυτές, είτε στην ταυτόχρονη εκτέλεση πολλών διαδικασιών λογισμικού, οπότε ονομάζεται πολυπρογραμματισμός (multiprogramming, multitasking). Ένα σύστημα μπορεί να έχει είτε μόνο την ιδιότητα της πολυεπεξεργασίας είτε μόνο του πολυπρογραμματισμού ή και τις δύο ή καμία από τις δύο. Η πολυεπεξεργασία υλοποιείται είτε με αυτόνομες μονάδες εργασίας, ή όπως εξελίχθηκε, με πολλούς μικροεπεξεργαστές (πυρήνες) σε ένα chip, δηλαδή με πολυπύρηνους (multicore) επεξεργαστές. Η τεχνική αυτή ονομάζεται και Chip Multiprocessing ή CMP και η εξέλιξη σε αυτήν ήταν κατά κάποιο τρόπο απαραίτητη, μιας και η αύξηση της συχνότητας ρολογιού ενός επεξεργαστή προκαλεί την αύξηση της θερμοκρασίας, ενώ απαιτεί την μείωση του μεγέθους τρανζίστορ με αποτέλεσμα την μείωση της ικανότητας λειτουργίας σε αυξημένες θερμοκρασίες.[1] Στην Εικόνα 1 φαίνονται οι διάφορες αρχιτεκτονικές επεξεργασίας. [2]



Εικόνα 1: Μονός Επεξεργαστής, Πολυεπεξεργαστής, Πολυπύρηνος Επεξεργαστής

Η υλοποίηση λογισμικού υψηλών απαιτήσεων σε υπολογιστική ισχύ όπως οι εφαρμογές πολυμέσων, οι εφαρμογές τηλεδιάσκεψης ή κινητών τηλεφώνων είναι πολύ ελκυστική λόγω της ευελιξίας, της επεκτασιμότητας, και της εν δυνάμει φορητότητας των υλοποιήσεων. Ωστόσο, οι ρυθμοί δεδομένων που εμπλέκονται σε πολλές από αυτές τις εφαρμογές τείνουν να είναι πολύ υψηλοί, με αποτέλεσμα σχετικά λίγους κύκλους ρολογιού επεξεργαστή διαθέσιμους ανά δείγμα. Η χρήση πολλών επεξεργαστών είναι συνήθως το μέσο για την επίτευξη της απαιτούμενης υπολογιστικής ισχύος χωρίς μετάβαση σε κυκλώματα ASIC. Με τα επίπεδα της ολοκλήρωσης που είναι δυνατά σήμερα, μπορεί κανείς να υλοποιήσει εύκολα εκατοντάδες ψηφιακούς επεξεργαστές σήματος σε ένα μόνο ολοκληρωμένο κύκλωμα. Η στρατηγική πολλαπλών επεξεργαστών είναι μια προσέγγιση για την αντιμετώπιση της πολυπλοκότητας που συνδέονται με τα SoC. Ωστόσο, η ανάπτυξη λογισμικού για την αποτελεσματική εκμετάλλευση συστημάτων πολυεπεξεργασίας παραμένει μια σημαντική πρόκληση. [3]

3.1.1 Εξέλιξη Συστημάτων Πολυεπεξεργασίας

Ο παραλληλισμός στους μικροεπεξεργαστές ξεκίνησε με την τεχνική της διασωλήνωσης (pipelining) των εντολών τη δεκαετία του 1960 με τα project ILLIAC II IBM Stretch. [4], [5] Η τεχνική της διασωλήνωσης αφορά στην εκκίνηση της εκτέλεσης του κύκλου εντολής ενός μικροεπεξεργαστή πριν ολοκληρωθεί ο προηγούμενος κύκλος εντολής με παράλληλο τρόπο. Για παράδειγμα αν ο κύκλος εντολής ενός μικροεπεξεργαστή αποτελείται από τέσσερα βήματα, λήψης (fetch), αποκωδικοποίησης (decode), εκτέλεσης (execute), εγγραφής (write-back), κατά την εκτέλεση του βήματος αποκωδικοποίησης του κύκλου μιας εντολής εκτελείται παράλληλα το βήμα λήψης του κύκλου της επόμενης εντολής κ.ο.κ. [6]

Στη συνέχεια εφαρμόστηκε η τεχνική του παραλληλισμού επιπέδου εντολών (Instruction Level Parallelism-ILP), η ταυτόχρονη δηλαδή εκτέλεση εντολών που δεν εξαρτάται η μία από τα αποτελέσματα εκτέλεσης της άλλης. Υλοποιείται σε superscalar υπολογιστές στους οποίους εκδίδονται πολλαπλές ανεξάρτητες εντολές στον ίδιο κύκλο, σε superpipelined υπολογιστές με χρόνο κύκλου μικρότερο από τον τυπικό χρόνο καθυστέρησης εντολής, σε υπολογιστές με εκτέλεση εντολών εκτός σειράς (out-of-order execution) στους οποίους οι εντολές εκτελούνται όχι ακολουθιακά αλλά με βάση ποιά δεδομένα είναι άμεσα διαθέσιμα. [7]

Παράλληλα σε εξέλιξη βρισκόταν και η τεχνική του multithreading, της εκτέλεσης δηλαδή πολλών νημάτων τα οποία μοιράζονται τους πόρους του συστήματος και τον επεξεργαστή. Τα νήματα προέρχονται από τον χωρισμό ενός προγράμματος σε δύο ή παραπάνω διεργασίες που τρέχουν ταυτόχρονα. Σύμφωνα με το [8] και με βάση το [9], μερικοί τύποι multithreading είναι οι κάτωθι:

- Πολλαπλή ακολουθία (multiple sequence) με χρήση πολλαπλών καταχωρητών προγράμματος.
- Σαφές (explicit multithreading). Εδώ ανήκουν τα παρακάτω είδη multithreading:
  - Λεπτής υφής (fine grain multithreading), κατά το οποίο εναλλάσσεται η εκτέλεση των νημάτων γρήγορα και σε κάθε κύκλο εντολών.
  - Χονδρής υφής (coarse grain multithreading), στο οποίο τα νήματα εκτελούνται περιοδικά κάθε έναν ορισμένο αριθμό κύκλων ρολογιού.
  - Ταυτόχρονο (Simultaneous Multithreading-SMT): Το SMT είναι μια αρχιτεκτονική επεξεργαστή που συνδυάζει το multithreading του υλικού με superscalar επεξεργαστές ώστε να επιτρέψει σε πολλαπλά νήματα να εκδίδουν εντολές ταυτόχρονα σε κάθε κύκλο ρολογιού. Επιτρέπει στο περιεχόμενο όλων των νημάτων να ανταγωνίζονται και να μοιράζονται τους πόρους του επεξεργαστή. Σε αντίθεση με τους συμβατικούς superscalar επεξεργαστές που υποφέρουν από έλλειψη παραλληλισμού ανά νήμα σε επίπεδο εντολών, χρησιμοποιεί πολλαπλά νήματα εκτέλεσης για να αποζημιώσει για το χαμηλό ILP ενός νήματος. [10]
- Σιωπηρό (implicit multithreading) στο οποίο τα νήματα δημιουργούνται και καταστρέφονται δυναμικά με βάση το υλικό και τον επεξεργαστή.
- Δυναμικό (dynamic multithreading). Εδώ η αλλαγή περιεχομένου γίνεται με βάση την εμφάνιση κάποιου γεγονότος δυναμικά.

Στη συνέχεια η πολυεπεξεργασία εξελίχθηκε στην πολυεπεξεργασία σε chip (Chip Multi Processing-CMP) όπως προαναφέρθηκε. Συγκρίνοντας το ταυτόχρονο multithreading με το CMP, οι SMT επεξεργαστές διαπλέκουν την εκτέλεση των εντολών διαφορετικών νημάτων στην ίδια διασωλήνωση. Έτσι πολλοί καταχωρητές προγράμματος είναι διαθέσιμοι και τα πολλαπλά περιεχόμενα των νημάτων αποθηκεύονται σε διαφορετικά σετ καταχωρητών στο chip. Οι μονάδες εκτέλεσης πολυπλέκονται μεταξύ αυτών των περιεχομένων νημάτων. Οι καθυστερήσεις που προκύπτουν από τον υπολογισμό μιας μεγάλης ροής εντολών καλύπτονται από τον υπολογισμό εντολών άλλων νημάτων. Σε αντίθεση, ένας CMP χρησιμοποιεί την κάθε μονάδα επεξεργασίας ανεξάρτητα από τις άλλες και όλα τα χαρακτηριστικά των μονάδων επεξεργασίας είναι πολλαπλάσια ανάλογα με τον αριθμό τους. [9] Οι CMP επεξεργαστές θα ζητηθούν αργότερα στο υποκεφάλαιο των πολυπύρηνων συστημάτων.

3.1.2 Κατηγοριοποίηση Συστημάτων Πολυεπεξεργασίας

Τα συστήματα πολυεπεξεργασίας κατηγοριοποιούνται με βάση διάφορα κριτήρια όπως τον τρόπο κατανομής των πόρων του συστήματος. Σε αυτή την περίπτωση διακρίνονται δύο μεγάλες κατηγορίες:

- Συμμετρική πολυεπεξεργασία (Symmetric Multiprocessing-SMP).

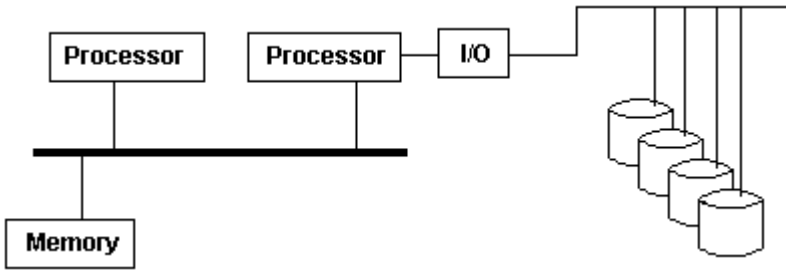
Τα στοιχεία επεξεργασίας είναι πανομοιότυπα, έχουν ίσα δικαιώματα χρήσης των πόρων του συστήματος, συνδέονται σε μία κοινή μνήμη και ελέγχονται από ένα στιγμιότυπο του λειτουργικού συστήματος. Οι επεξεργαστές μπορεί να επικοινωνούν μεταξύ τους μέσω διαύλων και να δρουν ανεξάρτητα, για παράδειγμα όσο μία CPU κάνει αναζήτηση σε μια βάση δεδομένων, η άλλη ή άλλες CPU μπορούν να ενημερώνουν την βάση δεδομένων ή να εκτελούν άλλες διεργασίες. Συνδυάζεται ιδανικά με λειτουργικό σύστημα τύπου μικροπυρήνα λόγω των μικρών αλλαγών που απαιτούνται για την υποστήριξη του SMP. Προβλήματα παρουσιάζονται όταν μπλοκάρονται οι επεξεργαστές προσπαθώντας να έχουν πρόσβαση ταυτόχρονα στον κοινό δίαυλο μνήμης (bottleneck). [11]

- Ασύμμετρη πολυεπεξεργασία (Asymmetric Multiprocessing-ASMP).

Στο ASMP, ξεχωριστοί επεξεργαστές χρησιμοποιούνται για συγκεκριμένες ομάδες διεργασιών που συνήθως παίρνουν τους ρόλους master ή slave επεξεργαστών. Οι επεξεργαστές που χρησιμοποιούνται μπορούν να είναι πανομοιότυποι, ή να διαφέρουν εντελώς μεταξύ τους στην αρχιτεκτονική που χρησιμοποιούν, στην λειτουργικότητα τους και τις διασυνδέσεις τους. Ένας μόνο επεξεργαστής μπορεί να είναι υπεύθυνος για την εξυπηρέτηση όλων των διακοπών του συστήματος ή ίσως ακόμη και την εκτέλεση όλων των λειτουργιών εισόδου/εξόδου στο σύστημα. Μια κάρτα γραφικών, λόγου χάρη, που λειτουργεί κάτω από τον έλεγχο μιας CPU σε έναν υπολογιστή μπορεί να θεωρηθεί μια μορφή ASMP. Όταν αναφερόμαστε σε multicore ή multiprocessor συστήματα, είναι πιο συχνά ASMP συστήματα. Η διαφορά του με το SMP είναι ότι οι επεξεργαστές δεν είναι πανομοιότυποι και δεν έχουν ισάξια πρόσβαση στους πόρους του συστήματος. Για παράδειγμα, μπορεί ένας επεξεργαστής να συνδέεται με την μνήμη και ένας άλλος με το σύστημα εισόδου/εξόδου. [12] Στην Εικόνα 2 φαίνεται ένα σύστημα ASMP με δύο CPU όπου ενώ έχουν και οι δύο πρόσβαση στη μνήμη, μόνο ο ένας χειρίζεται τις λειτουργίες εισόδου/εξόδου.

Όπως προαναφέρθηκε, ο όρος πολυεπεξεργασία μπορεί να αναφέρεται και σε λογισμικό πέρα από υλικό. Σε αυτή την περίπτωση ένα σύστημα SMP είναι σε θέση να αναθέσει οποιαδήποτε διεργασία σε οποιαδήποτε από τις κεντρικές μονάδες επεξεργασίας. Επειδή τα συστήματα SMP δεν καθορίζουν master ή slave επεξεργαστές, κάθε CPU δύναται να ολοκληρώσει ένα δεδομένο στόχο. Σε ένα σύστημα

ASMP αντίθετα, κάποιος επεξεργαστής μπορεί να μην είναι σε θέση να ολοκληρώσει μια διεργασία για λόγους όπως η αδυναμία πρόσβασης στο σύνολο της μνήμης και ως εκ τούτου τα καθήκοντα πρέπει να ανατίθενται στους επεξεργαστές από τον master επεξεργαστή. Έτσι, ο προγραμματιστής πρέπει να βεβαιώσει ότι οι επεξεργαστές χρησιμοποιούνται στο μέγιστο των δυνατοτήτων τους και ότι ένας επεξεργαστής δύναται να ολοκληρώσει ένα συγκεκριμένο έργο.

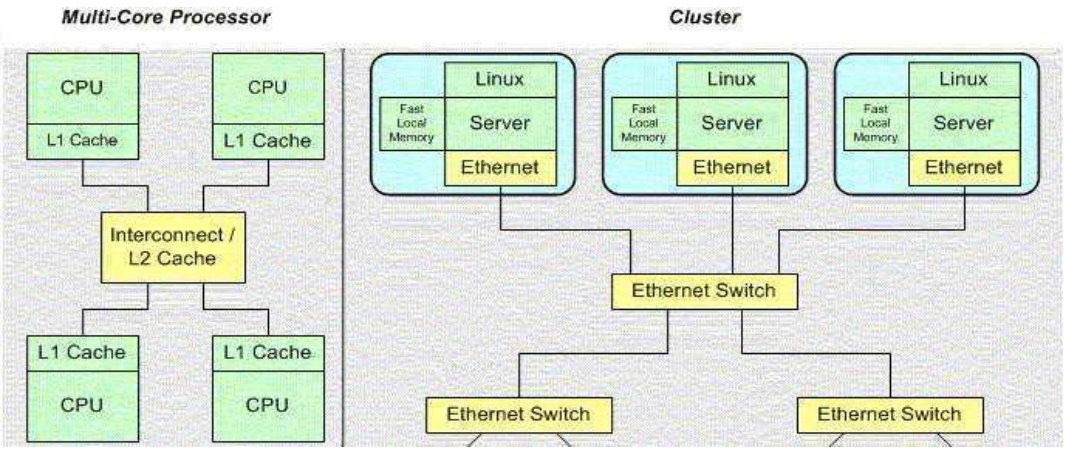


Εικόνα 2: Παράδειγμα ASMP Συστοιχίας Επεξεργαστών

Ανάλογα με την συνδεσμολογία των μονάδων που συμμετέχουν σε ένα σύστημα πολυεπεξεργασίας διακρίνονται σε [13]:

- Στενά συνδεδεμένη πολυεπεξεργασία (tightly coupled multiprocessing).

Οι επεξεργαστές είναι συνδεδεμένοι σε επίπεδο διαύλου και μπορούν να έχουν πρόσβαση σε μια κεντρική κοινόχρηστη μνήμη (SMP ή Unified Memory Access-UMA), ή μπορεί να συμμετέχουν σε μια ιεραρχία μνήμης με τοπική μνήμη για κάθε επεξεργαστή αλλά και κοινόχρηστη μνήμη (Non Unified Memory Access-NUMA). Συνήθως αναφέρεται στην πολυεπεξεργασία επιπέδου chip (Chip Level Multiprocessing-CMP), δύο ή και παραπάνω μονάδες επεξεργασίας σε ένα μόνο chip που ονομάζεται και multicore επεξεργαστής.



Εικόνα 3: Στενά και Χαλαρά Συνδεδεμένη Πολυεπεξεργασία

Στην Εικόνα 3 φαίνεται η αρχιτεκτονική ενός πολυπύρηνου (multicore) επεξεργαστή, ή αλλιώς μια στενά συνδεδεμένη αρχιτεκτονική πολυεπεξεργασίας και μία συστοιχία (cluster) επεξεργαστών σε μια χαλαρά συνδεδεμένη αρχιτεκτονική. [13]

- Χαλαρά συνδεδεμένη πολυεπεξεργασία (loosely coupled multiprocessing).

Τα συστήματα αυτά που συχνά ονομάζονται και συστοιχίες επεξεργαστών, βασίζονται σε πολλαπλά αυτόνομα συστήματα μονών ή διπλών επεξεργαστών που συνδέονται μέσω ενός συστήματος υψηλής ταχύτητας επικοινωνίας όπως έναν δίαυλο Ethernet. Ενώ η κατασκευή χαλαρά συνδεδεμένων αρχιτεκτονικών είναι απλή, το να κατασκευάσει κανείς ένα μεγάλο δίκτυο πολυεπεξεργασίας με αυτόν τον τρόπο απαιτεί χώρο και ισχύ. Αυτό συμβαίνει γιατί τέτοια δίκτυα αποτελούνται από κοινό υλικό, όπως προσωπικούς υπολογιστές με αποτέλεσμα το δίκτυο να περιλαμβάνει μεγάλη ποσότητα υλικού που δεν χρειάζεται. Προβλήματα παρουσιάζονται και με την επεκτασιμότητα του συστήματος.

Συνοψίζοντας, τα στενά συνδεδεμένα συστήματα αποδίδουν καλύτερα και είναι μικρότερα από τα χαλαρά συνδεδεμένα, αλλά απαιτούν ένα μεγάλο αρχικό ποσό επένδυσης και μπορεί να υποτιμηθούν γρήγορα. Από την άλλη, οι κόμβοι ενός χαλαρά συνδεδεμένου συστήματος είναι συνήθως χαμηλού κόστους υπολογιστές και μπορούν να επαναχρησιμοποιηθούν σαν ανεξάρτητες μηχανές εάν αφαιρεθούν από τη συστοιχία. Η κατανάλωση ρεύματος λαμβάνεται επίσης υπόψιν. Τα στενά συνδεδεμένα συστήματα τείνουν να είναι πολύ πιο αποδοτικά ενεργειακά από τις συστοιχίες. Αυτό οφείλεται στο γεγονός ότι με το σχεδιασμό στοιχείων να εργάζονται εξαρχής από κοινού σε στενά συνδεδεμένα συστήματα μειώνεται η ενεργειακή κατανάλωση, ενώ τα χαλαρά συνδεδεμένα συστήματα χρησιμοποιούν συστατικά στοιχεία που δεν προορίζονται αναγκαστικά για χρήση μόνο σε τέτοια συστήματα.

Ακόμα, ανάλογα με την ροή των εντολών και δεδομένων τα υπολογιστικά συστήματα χωρίζονται στις παρακάτω κατηγορίες οι οποίες πρώτα καθορίστηκαν το 1972 από τον Michael J. Flynn. [14]

Single Instruction Single Data (SISD) όπου ένα στοιχείο επεξεργασίας δέχεται μια ροή εντολών και επεξεργάζεται δεδομένα που είναι αποθηκευμένα σε μια μοναδική μνήμη. Σε αυτό το σύστημα δεν υφίσταται η έννοια της πολυεπεξεργασίας.

Single Instruction Multiple Data (SIMD). Εδώ έχουμε μοναδική ροή εντολών και πολλά δεδομένα, έναν επεξεργαστή δηλαδή που εκτελεί την ίδια πράξη σε διάφορα δεδομένα ταυτόχρονα. Έτσι υλοποιείται για παράδειγμα ο παραλληλισμός επιπέδου δεδομένων.

Multiple Instruction Single Data (MISD) είναι ένας τύπος παράλληλης αρχιτεκτονικής υπολογιστών όπου πολλές μονάδες επεξεργασίας εκτελούν διάφορες πράξεις στα ίδια δεδομένα. Έχουμε δηλαδή πολλαπλές ροές εντολών και μοναδικά δεδομένα.

Multiple Instruction Multiple Data (MIMD). Εδώ πολλά αυτόνομα στοιχεία επεξεργασίας εκτελούν διάφορες πράξεις σε πολλά δεδομένα επιτυγχάνοντας παραλληλισμό. Τα κατανεμημένα συστήματα είναι αυτής της αρχιτεκτονικής.

3.1.3 Πολυπύρηνα Συστήματα

Η συγγραφή βασίζεται στα [2], [15], [16], [17].

Ένας πολυπύρηνος (multicore) επεξεργαστής αποτελείται από πολλαπλές κεντρικές μονάδες επεξεργασίας (CPU) που υλοποιούνται σε ένα ενιαίο φυσικό πακέτο (chip) και διαθέτουν διεπαφές με μια μητρική κάρτα. Οι πολυπύρηνοι επεξεργαστές έχουν εισαχθεί από τους κατασκευαστές σε πολλά τμήματα της αγοράς. Το βασικό κίνητρο είναι αυτό της απόδοσης. Η χρήση πολυπύρηνων επεξεργαστών μπορεί να οδηγήσει σε αύξηση της ταχύτητας εκτέλεσης, αυξημένη απόδοση και μείωση της χρήσης ενέργειας για τις ενσωματωμένες εφαρμογές. Η πολυπύρηνη επεξεργασία αλλιώς ονομάζεται και πολυεπεξεργασία επιπέδου chip (Chip level Multi Processing-CMP).

Οι πολυπύρηνοι επεξεργαστές προσφέρουν ότι μεγάλα συστήματα πολυεπεξεργασίας σε επίπεδο chip. Έτσι, με αυτό το βαθμό παραλληλισμού μέχρι το επίπεδο chip, ο τρόπος κλιμάκωσης του συστήματος αλλάζει δραστικά. Κλιμάκωση του συστήματος ονομάζεται ο βαθμός στον οποίο αυτό επωφελείται από την αύξηση του αριθμού των πυρήνων με αντίστοιχη αύξηση της απόδοσης. Δεδομένου ότι το κόστος ενός συστήματος CMP είναι κοντά σε συστήματα που φέρουν μονούς επεξεργαστές (uniprocessor), είναι αναπόφευκτο ότι οι εφαρμογές θα λειτουργούν με υψηλό επίπεδο παραλληλισμού των νημάτων των διεργασιών.

Οι multicore επεξεργαστές προσφέρουν μια λύση στην ανάγκη για την ανάμειξη νέων χαρακτηριστικών γνωρισμάτων με ήδη υπάρχοντες κώδικες προγραμμάτων και επιτρέπουν τον συνδυασμό πολλαπλών περιβαλλόντων λειτουργίας στο ίδιο σύστημα. Συγκρινόμενο με τα παραδοσιακά ενσωματωμένα συστήματα που αποτελούνται από πολλά υποσυστήματα, ένα σύστημα υψηλής ολοκλήρωσης μπορεί να κατασκευαστεί με λογισμικό πραγματικού χρόνου και λειτουργίες του χρήστη του συστήματος να εκτελούνται σε ξεχωριστούς πυρήνες σε ένα ενιαίο σύστημα επεξεργασίας. Με τον τρόπο αυτό, μειώνεται το κόστος κατασκευής και συντήρησης του συστήματος μέσω της κατάργησης άσκοπης χρήσης υλικού.

Ένα βήμα μετά την πολυεπεξεργασία σε chip, είναι το Chip Multithreading (CMT). [18] Το CMT είναι ένας συνδυασμός του CMP και του SMT. Υπάρχουν πολλαπλοί πυρήνες ανά chip αλλά και πολλαπλά νήματα ανά πυρήνα. Ο συνδυασμός συμμετρικής πολυεπεξεργασίας και SMT είναι προαιρετικός. Μια απλοϊκή άποψη ενός συστήματος CMP είναι ότι φαίνεται στο λογισμικό σαν ένα σύστημα SMP που έχει τόσους επεξεργαστές όσους ο αριθμός των νημάτων στο chip, ο καθένας με ελαφρώς μειωμένη ικανότητα επεξεργασίας. Δεδομένου ότι κάθε νήμα του υλικού μοιράζεται τους πόρους ενός πυρήνα του επεξεργαστή, κάθε νήμα έχει κάποιο κλάσμα της συνολικής απόδοσης του πυρήνα του.

Η πολυπύρηνη επεξεργασία χωρίζεται σε [17]:

- Ομοιογενή (homogeneous) όπου οι επεξεργαστές που υλοποιούνται στο ίδιο chip είναι πανομοιότυποι με το ίδιο σετ εντολών. Ένας ομοιογενής επεξεργαστής πολλαπλών πυρήνων έχει δύο ή περισσότερες ταυτόσημους πυρήνες, που μοιράζονται τα περιφερειακά και τη μνήμη.

- Ετερογενή (heterogeneous) όπου οι επεξεργαστές διαθέτουν διαφορετικά σετ εντολών, για παράδειγμα μπορεί ο ένας να λειτουργεί σαν κεντρική μονάδα επεξεργασίας και ένας άλλος σαν ψηφιακός επεξεργαστής σήματος (Digital Signal Processor-DSP). Ένας ετερογενής επεξεργαστής διαθέτει πολλαπλά μοναδικά στοιχεία επεξεργασίας κάθε ένα από τα οποία είναι προσαρμοσμένα σε μια συγκεκριμένη λειτουργία. Επιπλέον, κάθε πυρήνας μπορεί να έχει μόνο επιλεκτική πρόσβαση στα περιφερειακά και τη μνήμη.

Και οι δύο τύποι των επεξεργαστών πολλαπλών πυρήνων μπορούν να αυξήσουν δραματικά την απόδοση. Οι σχεδιαστές μπορούν να προσαρμόσουν το σχεδιασμό με βάση τις ανάγκες της εφαρμογής και να επιτευχθεί μια ισορροπία ανάμεσα στις επιδόσεις, την ευελιξία και την κατανάλωση ισχύος. Οι προγραμματιστές που εργάζονται είτε με ομοιογενή ή ετερογενή συστήματα πολλαπλών πυρήνων θα αντιμετωπίσουν γενικά δύο προβλήματα: τον συγχρονισμό και χρονισμό καθώς και την κατανόηση της απόδοσης.

Όπως και τα συστήματα πολυεπεξεργασίας, έτσι και η πολυπύρηνη επεξεργασία χωρίζεται σε δύο κατηγορίες με βάση την πρόσβαση στη μνήμη, το λειτουργικό σύστημα που χρησιμοποιείται, την κατανομή του βάρους και τη συγγένεια των επεξεργαστών όπως επίσης και την επικοινωνία μεταξύ των πυρήνων [2]:

- Συμμετρική: Οι πυρήνες έχουν παρόμοια οπτική του συστήματος και μοιράζονται την κύρια μνήμη με αποτέλεσμα να μπορούν όλοι οι πυρήνες να έχουν πρόσβαση σε και να αλλάζουν δεδομένα που βρίσκονται στην κύρια μνήμη. Συνήθως υπάρχει ένα λειτουργικό σύστημα που ελέγχει τους πυρήνες. Η επικοινωνία των πυρήνων βασίζεται στην κύρια μνήμη και ελέγχεται από το λειτουργικό σύστημα που παρέχει μηχανισμούς συγχρονισμού και κλειδώματος. Η εξισορρόπηση φορτίου ολοκληρώνεται με το λειτουργικό σύστημα και καθορίζεται από τον χρονοδρομολογητή. Για παράδειγμα, μια υλοποίηση Linux για ένα τέτοιο σύστημα θα αντιμετωπίζει τους πυρήνες του επεξεργαστή σαν την συλλογή των πόρων της διεργασίας και θα χρονοδρομολογεί τα νήματα στους πυρήνες με βάση το υφιστάμενο φορτίο λειτουργίας για κάθε πυρήνα. Λειτουργικά συστήματα με ενεργοποιημένο το SMP υποστηρίζουν την συγγένεια των επεξεργαστών (processor affinity), γεγονός που επιτρέπει την εκχώρηση διεργασιών σε συγκεκριμένους πυρήνες.

- Ασύμμετρη: Οι πυρήνες διαθέτουν διαφορετική οπτική ο καθένας για το σύστημα. Έτσι οι περιοχές μνήμης είναι ασυνεχείς, γεγονός που υποδηλώνει ότι δύο πυρήνες δεν μπορούν να έχουν άμεση πρόσβαση και να τροποποιήσουν την ίδια θέση μνήμης. Ένα τέτοιο σύστημα μπορεί να έχει διαφορετικά λειτουργικά συστήματα εγκατεστημένα στους διάφορους πυρήνες Σε ASMP συστήματα με μνήμη χωρισμένη σε ασυνεχείς περιοχές, η επικοινωνία επιτυγχάνεται με ένα μήνυμα που διέρχεται εγκατάσταση που εξυπηρετεί να συγκεντρώσει δεδομένα και να περνούν από το ένα λειτουργικό σύστημα στο άλλο για χρήση. Το Multiple Communication Application Programming Interface (MCAPI) είναι ένα παράδειγμα πρότυπου περάσματος μηνυμάτων που να υποστηρίζει την επικοινωνία μεταξύ των πυρήνων ενός επεξεργαστή. Μια άλλη τεχνική για την ανταλλαγή δεδομένων είναι η παροχή ενός χάρτη μνήμης που χρησιμοποιεί εξειδικευμένο υλικό για να επιτρέψει σε δύο πυρήνες να μοιράζονται διακριτούς χώρους διευθύνσεων.

3.2 Προκλήσεις κατά την Ανάπτυξη Συστημάτων Πολυεπεξεργασίας

Η συγγραφή βασίζεται στα [2], [15], [16], [17].

Κάθε στοιχείο επεξεργασίας ενός πολυπύρηνου συστήματος εκτελεί ένα μέρος της λειτουργικότητας του συστήματος. Σε συνδυασμό με τη βελτίωση της κωδικοποίησης, της συσκευασίας, και των πρακτικών ολοκλήρωσης, οι παραδοσιακές τεχνικές εντοπισμού σφαλμάτων, όπως το σύγχρονο άλμα, βήμα, και halt είναι αποτελεσματικά σε πολυπύρηνα συστήματα. Ωστόσο, όταν τα νήματα των διεργασιών αλληλεπιδρούν, προκύπτουν προβλήματα συγχρονισμού και χρονισμού.

Τα πολυπύρηνα συστήματα παρουσιάζουν πάντα εξαρτήσεις μεταξύ των στοιχείων επεξεργασίας. Πολλές φορές, ένα στοιχείο επεξεργασίας πρέπει να περιμένει τα αποτελέσματα ενός άλλου, και αν είναι εκτός συγχρονισμού, η ανταλλαγή δεδομένων μεταξύ των δύο νημάτων θα είναι ανακριβής ή αναποτελεσματική. Για παράδειγμα, σε ένα σύστημα επεξεργασίας πλαισίων, ένα νέο πλαίσιο θα μπορούσε να χαθεί επειδή η επεξεργασία δεν ολοκληρώθηκε εγκαίρως. Αν και αυτό δεν προκαλεί απαραιτήτως τον αναπάντεχο τερματισμό της εκτέλεσης του προγράμματος, θα μπορούσε να υποβαθμίσει την ποιότητα των αποτελεσμάτων της εφαρμογής. Τα θέματα αυτά είναι πολύ δύσκολο να απομονωθούν εφόσον προκύπτουν από την αλληλεπίδραση των πυρήνων, τη χρήση των διαύλων και τα κοινόχρηστα περιφερειακά.

Οι κοινόχρηστοι πόροι οδηγούν σε καταστάσεις όπως bottlenecks, spin locks και συνθήκες κούρσας. Bottleneck (μποτιλιάρισμα) προκαλείται όταν πολλοί πυρήνες προσπαθούν να μεταβάλλουν δεδομένα στην κοινή τους μνήμη από έναν δίαυλο, γεγονός που οδηγεί σε “μποτιλιάρισμα” των δεδομένων που πρέπει να μεταφερθούν από τον δίαυλο. Ένα spin lock από την άλλη, είναι η περιγραφή ενός νήματος μιας διεργασίας που αναγκάζεται να περιμένει σε βρόχο για να έχει το “κλείδωμα” ενός πόρου που είναι κλειδωμένος από άλλα νήματα. [19]

Αν τα νήματα ενός επεξεργαστή CMP ανταγωνίζονται για σημαντικούς πόρους του συστήματος, τότε μπορεί να επηρεαστούν οι επιδόσεις των άλλων νημάτων. Για παράδειγμα, όταν πολλαπλά νήματα βρίσκονται σε έναν πυρήνα και ως εκ τούτου μοιράζονται την κρυφή μνήμη του πρώτου επίπεδου, τότε η απόδοση ενός δεδομένου νήματος μπορεί να επηρεάζεται από τις πράξεις των άλλων νημάτων του ίδιου πυρήνα στην κρυφή μνήμη με τα δεδομένα του πρώτου. Σε μια άλλη περίπτωση, ένα νήμα μπορεί να επωφεληθεί εάν τα άλλα νήματα που μοιράζονται την κρυφή μνήμη εισάγουν χρήσιμα για το πρώτο νήμα δεδομένα πριν αυτό τα χρειαστεί.

Στα ενσωματωμένα συστήματα, η παύση λειτουργίας ενός στοιχείου επεξεργασίας δεν μειώνει μόνο τις επιδόσεις, αλλά επηρεάζει και την κατανάλωση ενέργειας. Για την μεγιστοποίηση της απόδοσης και την μείωση της κατανάλωσης ενέργειας, ο προγραμματιστής πρέπει να ελαχιστοποιεί το χρόνο αναμονής δεδομένων του κάθε στοιχείου επεξεργασίας. Κατά την αδράνεια, μπορεί να μειώνεται η τροφοδοσία του αδρανούς στοιχείου επεξεργασίας. Οι πολυπύρηνοι επεξεργαστές μπορούν να προσφέρουν περισσότερους πόρους του επεξεργαστή σε ένα πρόγραμμα. Κάθε εξυπηρετητής θα μπορούσε να ανταποκριθεί σε μεγαλύτερο αριθμό συναλλαγών στον ίδιο χρόνο απόκρισης που θα είχε με έναν μονό επεξεργαστή. Οι πόροι του συστήματος μπορούν να εκμεταλλευτούν για να προσφέρουν δύο πλεονεκτήματα: αύξηση της διαπερατότητας (throughput) και βελτίωση του χρόνου ολοκλήρωσης μέσω του καθορισμού της σωστής τροφοδοσίας τους. Ένας προϋπολογισμός τροφοδοσίας μπορεί να είναι προκαθορισμένος για το σύστημα στην πλευρά του πελάτη ή τη συλλογή των υποστηρικτών εξυπηρετητών (back-end servers). Η πρόσθετη απόδοση μπορεί να οδηγήσει σε οικονομία χώρου, ισχύος και ψύξης.

Στην ανάπτυξη λογισμικού και υλικού, πρόκληση αποτελεί η επικοινωνία μεταξύ των εφαρμογών που τρέχουν σε διαφορετικούς πυρήνες, γνωστή και ως Inter Process Communication (IPC). Η επικοινωνία αυτή συχνά βασίζεται στην χρήση κοινής μνήμης μεταξύ των πυρήνων. Για multithreaded εφαρμογές, αυτό μπορεί να σημαίνει την κοινόχρηστη μνήμη μεταξύ των νημάτων εκτέλεσης σε διαφορετικούς πυρήνες. Για άλλες εφαρμογές, μπορεί να σημαίνει ρητό καθορισμό κοινών περιοχών μνήμης ή κοινόχρηστη μνήμη η οποία χρησιμοποιείται για το πέρασμα μηνυμάτων μεταξύ των διεργασιών. Τα θέματα της αρχιτεκτονικής μνήμης, και ειδικότερα η συνοχή της και η κρυφή μνήμη, είναι το κλειδί για την ορθότητα και την απόδοση σε συστήματα πολυεπεξεργασίας. Επίσης σημαντικό σε συστήματα με πολλούς επεξεργαστές είναι ο μηχανισμός διακοπών μεταξύ διεργασιών (Inter-Process Interrupt-IPI), που επιτρέπει σε επεξεργαστές παράγουν κοινοποιήσεις προς άλλους επεξεργαστές, όπως να ενημερώσουν έναν άλλο πυρήνα για την ανάγκη να ακυρώσει TLB εγγραφές σε κοινόχρηστη περιοχή μνήμης. [20]

Ένα συχνό πρόβλημα είναι και το ψευδές μοίρασμα (false sharing), σύμφωνα με το οποίο όταν ένας πυρήνας προσπαθεί να έχει πρόσβαση κατά περιόδους σε στοιχεία που ποτέ δεν θα μεταβληθούν από έναν άλλο πυρήνα, αλλά τα δεδομένα αυτά βρίσκονται σε ένα μπλοκ μνήμης με δεδομένα που μεταβάλλονται, το πρωτόκολλο που χρησιμοποιεί η κρυφή μνήμη μπορεί να αναγκάσει τον πρώτο πυρήνα να φορτώσει εκ νέου το μπλοκ, παρότι δεν χρειάζεται. Το πρωτόκολλο δεν γνωρίζει την έλλειψη δραστηριότητας σε αυτό το μπλοκ και ο πυρήνας αναγκάζεται να υποστεί την επιβάρυνση που προκαλεί μια πραγματική κοινή πρόσβαση ενός πόρου. [21]

Η συνοχή της κρυφής μνήμης (cache coherency) είναι επίσης σημαντική. Ενώ οι μονάδες επεξεργασίας έχουν λογικά πρόσβαση στην ίδια μνήμη, οι on-chip ιεραρχίες της μνήμης cache είναι ζωτικής σημασίας για την επίτευξη γρήγορης απόδοσης για την πλειονότητα των αναφορών μνήμης που γίνονται από τους πυρήνες. Έτσι, ένα βασικό πρόβλημα της κοινής μνήμης των πολυεπεξεργαστικών συστημάτων είναι η παροχή συνεκτικής άποψης της κρυφής μνήμης που έχει διάφορα επίπεδα ή ιεραρχίες. Οι μηχανισμοί συνοχής μνήμης όχι μόνο διοικούν την επικοινωνία σε έναν πολυεπεξεργαστή κοινής μνήμης, αλλά και καθορίζουν τον τρόπο που το σύστημα μεταφέρει δεδομένα μεταξύ των επεξεργαστών/πυρήνων, των κρυφών μνημών και τη μνήμη. [22]

Στην αύξηση της απόδοσης συμβάλλει και η εξισορρόπηση φορτίου (load balancing) μεταξύ των πυρήνων, μια τεχνική για τη κατανομή του φόρτου εργασίας ομοιόμορφα σε δύο ή περισσότερους υπολογιστές, συνδέσεις δικτύου, επεξεργαστές, σκληρούς δίσκους, ή άλλους πόρους, προκειμένου να αποκτηθεί η βέλτιστη αξιοποίηση των πόρων, η μεγιστοποίηση της απόδοσης, η ελαχιστοποίηση του χρόνου απόκρισης και η αποφυγή υπερφόρτωσης κάποιου από τους πυρήνες.

Τα διάφορα λειτουργικά συστήματα επίσης αντιμετωπίζουν προβλήματα πλήρους εκμετάλλευσης των δυνατοτήτων των πολυπύρηνων επεξεργαστών. Τα τρέχοντα λειτουργικά συστήματα θεωρούν τον κάθε πυρήνα μια ανεξάρτητη CPU και τις αρχιτεκτονικές των πυρήνων όπως και τα συστήματα μνήμης ομοιόμορφα και σταθερά. Παραβλέπεται από το λειτουργικό σύστημα η ύπαρξη κοινών πηγών, όπως κρυφή μνήμη και ο δίαυλος δεδομένων ενώ δεν λαμβάνουν υπόψιν τα χαρακτηριστικά του υλικού των πυρήνων, αδυνατώντας να βελτιώσουν το ισοζύγιο της απόδοσης με την κατανάλωση ισχύος. Τα προβλήματα επιδεινώνονται για heterogeneous multicore chips για τα οποία παραβλέπεται και η ανομοιογένεια των πυρήνων.

Συνοπτικά οι προκλήσεις που παρουσιάζονται σε πολυπύρηνα συστήματα ακολουθούν:

- Κοινοί πόροι του συστήματος όπως κρυφές μνήμες και δίαυλοι που προκαλούν bottlenecks.
- Καθυστερήσεις στην προσπέλαση της μνήμης και στις ταχύτητες προσβάσεις στους διαύλους.
- Ο σχεδιασμός του λογισμικού που πρέπει να ακολουθεί μοντέλα παράλληλου προγραμματισμού για να υπάρχει πλήρη εκμετάλλευση των δυνατοτήτων του συστήματος.
- Τα λειτουργικά συστήματα που δεν εκμεταλλεύονται την ύπαρξη των πυρήνων σε ένα chip με κοινούς πόρους.
- Η μετανάστευση κώδικα που σχεδιάστηκε για μονοπύρηνα συστήματα.

3.3 Προγραμματισμός Συστημάτων Πολυεπεξεργασίας

Η συγγραφή βασίζεται στα [23], [24], [3].

Η εξάπλωση της αρχιτεκτονικής πολυεπεξεργασίας έχει επίδραση στον τρόπο που αναπτύσσεται το λογισμικό. Μέχρι πρόσφατα, η πρόοδος στην τεχνολογία σήμαινε αύξηση της ταχύτητας ρολογιού, με αποτέλεσμα η εκτέλεση του λογισμικού να μπορεί στην πράξη να επιταχύνει από την ίδια την πάροδο του χρόνου. Τώρα όμως η πρόοδος στην τεχνολογία οδηγεί στην ολοένα μεγαλύτερη αξιοποίηση του παραλληλισμού παρά σε αύξηση της ταχύτητας.

Στην πληροφορική ο φυσικός διαχωρισμός των μονάδων επεξεργασίας αναφέρεται ως παραλληλισμός (parallelism) και η λογική ιδιότητα πολλών γεγονότων να συμβαίνουν ταυτόχρονα ως συγχρονισμός (concurrency). Ο παραλληλισμός υπονοεί τον συγχρονισμό αλλά το αντίστροφο δεν ισχύει απαραιτήτως. Σχεδόν όλα τα λειτουργικά συστήματα ασχολούνται με συγχρονισμό τον οποίο διαχειρίζονται πολυπλέκοντας πολλές διεργασίες ή νήματα σε ένα μεμονωμένο επεξεργαστή. Λίγα ασχολούνται με τον παραλληλισμό, αναθέτοντας για παράδειγμα νήματα σε φυσικά διαχωρισμένους επεξεργαστές. Τα τυπικά ενσωματωμένα συστήματα παρουσιάζουν τόσο συγχρονισμό όσο και παραλληλισμό αλλά σε διαφορετικό πλαίσιο από εκείνο των λειτουργικών συστημάτων γενικού σκοπού.

Στα ενσωματωμένα συστήματα, οι ταυτόχρονες διεργασίες συχνά ορίζονται στατικά και παραμένουν σε μεγάλο βαθμό αμετάβλητες κατά τη διάρκεια ζωής του συστήματος. Ένα κινητό τηλέφωνο, για παράδειγμα, έχει σχετικά λίγους διακριτούς τρόπους λειτουργίας (κλήση, ομιλία, αναμονή, κλπ.) και για κάθε τρόπο λειτουργίας, ένα καλά καθορισμένο σετ διεργασιών είναι ταυτόχρονα ενεργό. Η στατική δομή του συγχρονισμού προσφέρει ευκαιρίες για πολύ πιο λεπτομερή ανάλυση και βελτιστοποίηση από ότι ένα πιο δυναμικό περιβάλλον.

Προκλήσεις προκύπτουν στον προγραμματισμό των συστημάτων πολυεπεξεργασίας όλων των μεγεθών. Σε πολύ μικρή κλίμακα, οι επεξεργαστές σε ένα ενιαίο chip καλούνται να συντονίσουν την πρόσβαση σε μια κοινόχρηστη θέση μνήμης, και σε μεγάλη κλίμακα, οι επεξεργαστές σε έναν υπερυπολογιστή πρέπει να συντονίσουν τη δρομολόγηση των δεδομένων. Ο προγραμματισμός συστημάτων πολυεπεξεργασίας αποτελεί πρόκληση γιατί τα σύγχρονα συστήματα ηλεκτρονικών υπολογιστών είναι εγγενώς ασύγχρονα, αφού οι δραστηριότητες μπορούν να σταματήσουν ή να καθυστερήσουν χωρίς προειδοποίηση από διακοπές, έχουμε τυχαίες αποτυχίες κρυφής μνήμης,

αστοχίες και άλλα συμβάντα. Οι καθυστερήσεις αυτές είναι απρόβλεπτες, και μπορεί να ποικίλλουν σε μεγάλο βαθμό. Μια αστοχία κρυφής μνήμης θα μπορούσε να καθυστερήσει έναν επεξεργαστή για λιγότερες από δέκα εντολές, ένα σφάλμα σελίδας για μερικά εκατομμύρια εντολές, και η ικανότητα preemption του λειτουργικού συστήματος για εκατοντάδες εκατομμύρια εντολές.

Η ορθότητα των προγραμμάτων πολλαπλών επεξεργαστών, από τη φύση τους, είναι πιο περίπλοκη από εκείνη των προγραμμάτων για απλούς επεξεργαστές που δεν απαιτούν παράλληλο προγραμματισμό και απαιτεί ένα διαφορετικό σύνολο εργαλείων. Η διαδοχική ορθότητα συνήθως αφορά τις ιδιότητες ασφαλείας. Η ιδιότητα ασφάλειας εγγυάται ότι ένα μη επιθυμητό γεγονός δεν συμβαίνει ποτέ. Φυσικά, η ορθότητα του συγχρονισμού αφορά επίσης την ασφάλεια, αλλά το πρόβλημα είναι πολύ πιο δύσκολο, γιατί πρέπει να εξασφαλιστεί η ιδιότητα της ασφαλείας παρά το μεγάλο αριθμό των τρόπων που τα νήματα μπορούν να διαπλεκούν. Η ορθότητα του συγχρονισμού προσθέτει την ιδιότητα της βιωσιμότητας που εγγυάται ότι ένα επιθυμητό γεγονός θα συμβεί κάποια στιγμή.

Το λογισμικό για ενσωματωμένα συστήματα έχει περισσότερα κοινά με το υλικό σε σύγκριση με το παραδοσιακό λογισμικό. Το υλικό είναι υλοποιημένο συνήθως με ιδιαίτερα υψηλό βαθμό παραλληλισμού. Εννοιολογικά, το υλικό είναι ένα σύνολο από στοιχεία που λειτουργούν συνεχώς ή διακριτά σε σχέση με το χρόνο και αλληλεπιδρούν μέσω σύγχρονης ή ασύγχρονης επικοινωνίας. Το λογισμικό είναι μια συνάθροιση στοιχείων που ανταγωνίζονται για τη χρήση ενός επεξεργαστή, που λειτουργούν διαδοχικά αφήνοντας ίχνη της επικοινωνίας τους σε μια στοίβα ή στη μνήμη. Ο συγχρονισμός του λογισμικού αποσκοπεί στην οργάνωση ακολουθιών, ενώ ο συγχρονισμός του υλικού είναι αποσκοπεί στην οργάνωση σημάτων.

Ο κύριος μηχανισμός αφαιρετικότητας στο λογισμικό είναι η διαδικασία ή η μέθοδος στον αντικειμενοστραφή προγραμματισμό. Οι διαδικασίες είναι ουσιαστικά ολοκληρωμένοι υπολογισμοί. Ο κύριος μηχανισμός αφαιρετικότητας στο υλικό είναι πολύ διαφορετικός και αφορά μια μονάδα που λειτουργεί παράλληλα με τις άλλες συνιστώσες. Οι μονάδες αυτές αντιπροσωπεύουν υπολογισμούς που δεν ολοκληρώνονται, δεν εκκινούν, δεν εκτελούνται, δεν περατώνονται και δεν επιστρέφουν. Απλώς υπάρχουν όπως και σε ενσωματωμένα συστήματα, τα στοιχεία του λογισμικού έχουν συχνά την ίδια ιδιότητα. Η διάκριση μεταξύ υλικού και λογισμικού, από την άποψη του υπολογισμού, έχει να κάνει μόνο με το βαθμό συγχρονισμού και το ρόλο του χρόνου.

Λόγω της μεγάλης πολυπλοκότητας των εφαρμογών λογισμικού πολλαπλών επεξεργαστών και των σοβαρών περιορισμών των επιδόσεων των εφαρμογών πολυμέσων, η ανάπτυξη αυτόματων εργαλείων για τη χαρτογράφηση (mapping) των προδιαγραφών των εφαρμογών πολυμέσων σε αποδοτικές υλοποιήσεις είναι δύσκολη. Η χαρτογράφηση μιας εφαρμογής σε ένα σύστημα πολλαπλών επεξεργαστών περιλαμβάνει τρεις βασικές λειτουργίες [3]:

- Την ανάθεση διεργασιών στους επεξεργαστές.
- Την “παραγγελία” διεργασιών από κάθε επεξεργαστή.
- Τον καθορισμό του χρόνου κατά το οποίο κάθε εργασία ξεκινά την εκτέλεση της.

Οι πράξεις αυτές αναφέρονται συλλογικά ως χρονοδρομολόγηση της εφαρμογής στην δεδομένη αρχιτεκτονική. Ένα βασικό θέμα της χρονοδρομολόγησης πολλαπλών επεξεργαστών για εφαρμογές πολυμέσων που διαφέρει από το τυπικό περιβάλλον προγραμματισμού είναι ο κεντρικός ρόλος της επικοινωνίας μεταξύ των επεξεργαστών, η αποτελεσματική διαχείριση δηλαδή της μεταφοράς δεδομένων μεταξύ των διεργασιών που έχουν ανατεθεί σε διαφορετικούς επεξεργαστές. Δεδομένου ότι

το συνολικό κόστος της επικοινωνίας μεταξύ των επεξεργαστών μπορεί να έχει δραματικές επιπτώσεις στην ταχύτητα εκτέλεσης και την κατανάλωση ενέργειας, η αποτελεσματική διαχείριση της επικοινωνίας αυτής είναι ζωτικής σημασίας για την ανάπτυξη οικονομικά αποδοτικών υλοποιήσεων με πολλούς επεξεργαστές.

Από την άποψη του προγραμματιστή εφαρμογών, ένα σύστημα πολυεπεξεργασίας παρέχει μια παράλληλη αρχιτεκτονική η οποία αποτελείται από επεξεργαστές που συνδέονται μεταξύ τους μέσω ενός δικτύου επικοινωνίας. Για την αξιοποίηση αυτής της παράλληλης αρχιτεκτονικής, απαιτείται παράλληλος προγραμματισμός. Γενικά, υπάρχουν δύο τύποι μοντέλων παράλληλου προγραμματισμού: προγραμματισμού κοινόχρηστης μνήμης (shared memory programming) και προγραμματισμού περάσματος μηνυμάτων (message passing programming). Η πλατφόρμες προγραμματισμού OpenMP και MPI (Message Passing Interface) είναι κάποια παραδείγματα αντιστοίχως.

Τέλος, περιγράφονται τέσσερα προγραμματιστικά μοντέλα για την εφαρμογή του παραλληλισμού, το μοντέλο των νημάτων POSIX και οι πλατφόρμες OpenCL, OpenMP και CUDA.

- POSIX Threads

Οι βιβλιοθήκες νημάτων POSIX αποτελούν ένα πρότυπο Application Programming Interface (API) νημάτων για C/C++. Επιτρέπει σε ένα νήμα να αναπαράγει μια νέα ταυτόχρονη ροή της διεργασίας. Είναι πιο αποτελεσματικό για πολλαπλούς επεξεργαστές ή multicore συστήματα, όπου η ροή της διαδικασίας μπορεί να προγραμματιστεί να τρέχει σε ένα άλλο επεξεργαστή κερδίζοντας ταχύτητα μέσω παράλληλης ή κατανεμημένης επεξεργασίας. Τα νήματα προκαλούν μικρότερη επιβάρυνση από ότι το "forking" ή η αναπαραγωγή μιας νέας διεργασίας, επειδή το σύστημα δεν αναγκάζεται να δεσμεύσει νέο εικονικό χώρο μνήμης και περιβάλλον για τη νέα διεργασία. Ενώ η χρήση νημάτων είναι πιο αποτελεσματική σε ένα σύστημα πολυεπεξεργασίας, ακόμα και τα μονοπύρηνα συστήματα επωφελούνται, λόγου χάρη όταν υπάρχουν καθυστερήσεις στις λειτουργίες εισόδου/εξόδου που μπορεί να οδηγήσει στην παύση εκτέλεσης της διεργασίας. Ένα νήμα μπορεί να εκτελείται ενώ ένα άλλο είναι σε αναμονή για κάποια είσοδο ή έξοδο. Τεχνικές παράλληλου προγραμματισμού όπως το MPI χρησιμοποιείται σε ένα κατανεμημένο υπολογιστικό περιβάλλον, ενώ τα νήματα χρησιμοποιούνται κυρίως σε ένα ενιαίο σύστημα πληροφορικής. Όλα τα νήματα σε μια διεργασία μοιράζονται τον ίδιο χώρο διευθύνσεων. Ένα νήμα γίνεται spawned, καθορίζοντας μια συνάρτηση και τα ορίσματα της τα οποία θα επεξεργαστούν από το νήμα. Σκοπός της χρήσης της βιβλιοθήκης νημάτων POSIX στο λογισμικό είναι η αύξηση στην ταχύτητα εκτέλεσης. [25]

- OpenCL

Η πλατφόρμα OpenCL (Open Computing Language) αποτελεί ένα πλαίσιο για τη σύνταξη προγραμμάτων που εκτελούνται σε ετερογενή επεξεργαστικά στοιχεία που μπορεί να είναι CPUs, GPUs (Graphic Processing Unit) και άλλοι επεξεργαστές. Περιλαμβάνει μια γλώσσα (με βάση την C99) για τη γραφή πυρήνων (kernels), καθώς και APIs που χρησιμοποιούνται για τον καθορισμό και τον έλεγχο της πλατφόρμας. Παρέχει παράλληλη υπολογιστική που βασίζεται είτε σε διεργασίες είτε σε δεδομένα. Δίνει πρόσβαση σε κάθε εφαρμογή στην GPU ώστε να την χρησιμοποιήσει σαν επιπλέον στοιχείο επεξεργασίας. Η GPU ήταν προηγουμένως διαθέσιμη για εφαρμογές γραφικών μόνο. Η μνήμη της GPU γίνεται διαθέσιμη στο λειτουργικό σύστημα ή και τις εφαρμογές σαν ταχύτερη μνήμη από την κύρια μνήμη του συστήματος. Κατά συνέπεια, η OpenCL επεκτείνει την ισχύ των καρτών γραφικών πέρα από την απλή επεξεργασία γραφικών (γενικής χρήσης υπολογιστών σε γραφικά μονάδες επεξεργασίας). Η OpenCL είναι ανάλογη με τα ανοικτά πρότυπα βιομηχανίας OpenGL και OpenAL, για 3D γραφικά και ήχο υπολογιστών, αντίστοιχα. [26]

- OpenMP

Η πλατφόρμα OpenMP (Open Multi-Processing) είναι μια διεπαφή προγραμματισμού εφαρμογών (API) που υποστηρίζει πολλαπλές πλατφόρμες κοινής μνήμης πολυπρογραμματισμού σε C, C++ και Fortran για πολλές αρχιτεκτονικές, συμπεριλαμβανομένου και των λειτουργικών συστημάτων Unix και Microsoft Windows. Αποτελείται από ένα σύνολο οδηγιών compiler, ρουτίνες βιβλιοθήκης και μεταβλητές περιβάλλοντος που επηρεάζουν τη συμπεριφορά του χρόνου εκτέλεσης. Είναι ένα φορητό, επεκτάσιμο μοντέλο που δίνει στους προγραμματιστές ένα απλό και ευέλικτο περιβάλλον για την ανάπτυξη παράλληλων εφαρμογών για όλες τις κλίμακες, από την επιφάνεια εργασίας μέχρι έναν υπερυπολογιστή. Μια εφαρμογή σχεδιασμένη με το υβριδικό μοντέλο του παράλληλου προγραμματισμού, μπορεί να τρέξει σε ένα σύμπλεγμα υπολογιστή χρησιμοποιώντας τόσο την OpenMP όσο και τεχνικές περάσματος μηνυμάτων, ή μέσω της χρήσης επεκτάσεων της OpenMP για συστήματα μη μεριζόμενης μνήμης. Είναι μια υλοποίηση του multithreading, μια μέθοδος παραλληλοποίησης με την οποία το κύριο νήμα (master thread, μια σειρά δηλαδή από οδηγίες που εκτελούνται διαδοχικά, κάνει fork ένα συγκεκριμένο αριθμό νημάτων σκλάβων (slave threads) και μια διεργασία διαιρείται μεταξύ τους. Τα νήματα στη συνέχεια τρέχουν ταυτόχρονα, με το περιβάλλον του χρόνου εκτέλεσης να τα κατανέμει σε διαφορετικούς επεξεργαστές. Το τμήμα του κώδικα που έχει ως στόχο να τρέξει παράλληλα φέρει την ανάλογη σήμανση, με μια οδηγία προεπεξεργαστή που θα προκαλέσει τα νήματα για να σχηματίσουν πριν από την εκτέλεση του τμήματος. Κάθε νήμα έχει ένα αναγνωριστικό που είναι προσαρμοσμένο σε αυτό που μπορεί να αποκτηθεί μέσω μιας συνάρτησης όπως για παράδειγμα η omp_get_thread_num() σε C/C++. Το id του νήματος είναι ένας ακέραιος αριθμός, και το κύριο νήμα έχει αναγνωριστικό το "0". Μετά την εκτέλεση του παράλληλου κώδικα, τα νήματα επιστρέφουν στο κύριο νήμα, το οποίο συνεχίζει την εκτέλεση του μέχρι το τέλος του προγράμματος. Κάθε νήμα εκτελεί το τμήμα του παράλληλου κώδικα ανεξάρτητα. Το περιβάλλον χρόνου εκτέλεσης αναθέτει νήματα στους επεξεργαστές ανάλογα με το βαθμό χρήσης τους, το φορτίο και άλλους παράγοντες. Ο αριθμός των νημάτων που μπορεί να ανατεθεί από το περιβάλλον χρόνου εκτέλεσης βασίζεται σε μεταβλητές περιβάλλοντος ή σε κώδικα μέσω συναρτήσεων. Οι λειτουργίες OpenMP περιλαμβάνονται σε να αρχείο επικεφαλίδας με το όνομα "omp.h" στην C/C++. [27]

- CUDA

Η πλατφόρμα CUDA (Compute Unified Device Architecture) αποτελεί μια παράλληλη αρχιτεκτονική υπολογιστών που αναπτύχθηκε από την NVIDIA. CUDA είναι ο μηχανισμός υπολογιστικής στις κάρτες γραφικών NVIDIA (GPUs) που είναι προσβάσιμη σε προγραμματιστές λογισμικού, μέσω πρότυπων γλωσσών προγραμματισμού. Παρέχει πρόσβαση στους προγραμματιστές στο σετ εντολών και τη μνήμη των παράλληλων υπολογιστικών στοιχείων σε CUDA GPUs. Χρησιμοποιώντας CUDA, οι NVIDIA GPUs χρησιμοποιούνται αποτελεσματικά σαν ανοικτές αρχιτεκτονικές όπως οι επεξεργαστές. Σε αντίθεση με τους επεξεργαστές ωστόσο, οι GPUs έχουν συνήθως πολυπύρηνη αρχιτεκτονική με τον κάθε πυρήνα ναι μπορεί να τρέχει χιλιάδες νήματα ταυτοχρόνα. Αν μια εφαρμογή είναι κατάλληλη για αυτό το είδος της αρχιτεκτονικής, η GPU μπορεί να ωφελήσει σημαντικά την απόδοση. [28]

Αναφορές

[1] “Chip Multithreading and Multiprocessing”; http://developers.sun.com/solaris/articles/cmtmp.html.

[2] M. Domeika, *Software development for embedded multi-core systems*, Newnes, 2008.

[3] S. Sriram and S.S. Bhattacharyya, *Embedded multiprocessors*, CRC Press, 2009.

[4] “Computer History Museum | IBM Stretch Collection”; http://www.computerhistory.org/collections/ibmstretch/.

[5] “Index of /pdf/univOfIllinoisUrbana/illiac/ILLIAC_II”; http://www.bitsavers.org/pdf/univOfIllinoisUrbana/illiac/ILLIAC_II/.

[6] B.B. Brey, *The Intel microprocessors*, Merrill, 1991.

[7] David W. Wall, *Limits of Instruction-Level Parallelism* , Western Research Laboratory 250 University Avenue Palo Alto, California 94301 USA: 1993.

[8] “Multithreading -- Mark Smotherman”; http://www.cs.clemson.edu/~mark/multithreading.html.

[9] Theo Ungerer, Borut Robic and Jurij Silc, “Multithreaded Processors,” *British Computer Society*, vol. Vol. 45, 2002, p. 29.

[10] “Simultaneous Multithreading home page”; http://www.cs.washington.edu/research/smt/.

[11] “SMP boosts scalability for net elements”; http://www.networkworld.com/news/tech/2001/1029tech.html.

[12] “Who’s Afraid of Asymmetric Multiprocessing? | RTC Magazine”; http://www.rtcmagazine.com/articles/view/100404#.

[13] “Linux and symmetric multiprocessing,” Mar. 2007; http://www.ibm.com/developerworks/library/l-linux-smp/.

[14] M. Flynn, “Some Computer Organizations and Their Effectiveness,” *IEEE Trans. Comput.*, vol. C-21, 1972, p. 948; http://en.wikipedia.org/wiki/Flynn's_taxonomy.

[15] “White Paper: How Multicore Enables the Fast and Efficient Deployment of Multi-OS Systems – Embedded Computing Design”; http://embedded-computing.com/white-enables-fast-efficient-deployment-multi-os-systems.

[16] “Multicore processors: Providing opportunities for embedded systems designers – Embedded Computing Design”; http://embedded-computing.com/multicore-opportunities-embedded-systems-designers.

[17] “Multicore vs. multiprocessing vs. multicore multiprocessing”; http://www.vmecritical.com/articles/id/?3573.

[18] “Chip Level Multiprocessing - Siwiki”; http://www.solarisinternals.com/wiki/index.php/Chip_Level_Multiprocessing.

[19] &. Silberschatz, *Operating System Concepts*, Addison-Wesley, 1994.

[20] Robert N. M. Watson, *Introduction to Multithreading and Multiprocessing in the FreeBSD*

SMPng Network Stack, Computer Laboratory : University of Cambridge, .

[21] “C H A P T E R 7 - Performance Considerations”; http://docs.sun.com/source/819-0501/7_tuning.html.

[22] CACHE COHERENCE TECHNIQUES FOR MULTICORE PROCESSORS, “CACHE COHERENCE TECHNIQUES FOR MULTICORE PROCESSORS,” phd, UNIVERSITY OF WISCONSIN - MADISON, 2008.

[23] M. Herlihy and N. Shavit, *The art of multiprocessor programming*, Morgan Kaufmann, 2008.

[24] W. Wolf and A.A. Jerraya, *Multiprocessor systems-on-chips*, Elsevier, 2005.

[25] “POSIX Threads Programming”; https://computing.llnl.gov/tutorials/pthreads/.

[26] “OpenCL”; http://www.khronos.org/opencl/.

[27] “OpenMP.org”; http://openmp.org/wp/.

[28] “CUDA Zone”; http://www.nvidia.com/object/cuda_home_new.html.

4 Αξιοποίηση του Λειτουργικού Συστήματος Linux σε Ενσωματωμένα Συστήματα Πραγματικού Χρόνου

4.1 Εισαγωγή

Το Linux είναι ένα ελεύθερο τύπου-Unix λειτουργικό σύστημα που αρχικά δημιουργήθηκε από τον Linus Torvalds στο Πανεπιστήμιο του Ελσίνκι το 1991. Είναι μια ανεξάρτητη POSIX υλοποίηση και στα χαρακτηριστικά του συμπεριλαμβάνει πραγματικό multitasking, πραγματικό πολυχρηστικό περιβάλλον, virtual memory, shared libraries, demand loading, TCP/IP networking και πολλά άλλα χαρακτηριστικά. Ο Linus Torvalds αρχικά δημιούργησε το Linux ως hobby. Αποφάσισε να δημιουργήσει ένα σύστημα που θα είχε περισσότερες δυνατότητες από το Minix, ένα μικρό λειτουργικό σύστημα βασισμένο σε Unix. Ξεκίνησε την δημιουργία του Linux το 1991, ανακοινώνοντας την έκδοση 0.02 και δουλεύοντας σταθερά έφτασε το 1994 στην έκδοση 1.0 του Linux Kernel. Το Linux μπορεί να χρησιμοποιηθεί για πολλούς σκοπούς, όπως το networking, η ανάπτυξη προγραμμάτων, ακόμα και για πλατφόρμα για απλούς χρήστες. Λόγω της φύσης και της ευελιξίας του, το Linux έχει γίνει πολύ διάσημο παγκοσμίως και ένας μεγάλος αριθμός προγραμματιστών έχει επικεντρώσει το ενδιαφέρον του πάνω σε αυτό.

Ένας από τους βασικούς λόγους για την επιτυχία του Linux και την ανάπτυξή του είναι η έννοια του ανοιχτού κώδικα (opensource), μέσω της άδειας GNU General Public License (GPL). Αυτή η άδεια επιτρέπει σε όλους τους κώδικες που αναπτύχθηκαν για τον πυρήνα του Linux να χρησιμοποιούνται ελεύθερα από άλλους, για προσωπική ή εμπορική χρήση και μάλιστα απαγορεύει την εμπορία του συστήματος χωρίς να έχει συνοδευτικό πηγαίο κώδικα, συμπεριλαμβανομένων όλων των τροποποιήσεων του πυρήνα. Το χαρακτηριστικό του ανοιχτού κώδικα έχει προσελκύσει συνεισφορές από προγραμματιστές από όλο τον κόσμο για τις νέες τεχνολογίες που σχετίζονται με σύγχρονες ανάγκες πληροφορικής. Επιτρέποντας σε οποιονδήποτε να μελετήσει υπάρχουσες υλοποιήσεις και να δέχεται βελτιώσεις ως εισφορές, ο δημιουργός του Linux και άλλοι, έχουν δημιουργήσει ένα λειτουργικό σύστημα που διαφορετικά θα απαιτούσε πολλούς εμπορικούς πόρους σε σύντομο χρονικό διάστημα. Η ανάπτυξη του ελεύθερου λογισμικού οδηγεί στην υψηλή αξιοπιστία και απόδοση του λειτουργικού συστήματος Linux. [1]

Η αρχική κοινότητα ανάπτυξης γύρω από τον πυρήνα του Linux έχει πλέον εξελιχθεί σε μια ακόμα μεγαλύτερη κοινότητα υποστηρίζοντας μεγάλο, πολυποίκιλο σύνολο εφαρμογών και τεχνολογιών για τα προγράμματα που τρέχει σε Linux. Μία από τις πιο πρόσφατες εξελίξεις έχει η ταχεία προσαρμογή του Linux για την καταλληλότητα σε ενσωματωμένα συστήματα. Αυτό ξεκίνησε με τον πυρήνα και την υποστήριξη compiler για όλες τους δημοφιλείς 32-bit μικροεπεξεργαστές που έχουν σχεδιαστεί για ενσωματωμένα συστήματα σήμερα, συμπεριλαμβανομένων των Intel x86, ARM, Motorola/IBM PowerPC, MIPS NEC και Hitachi SH. Πολλές εμπορικές εταιρίες άρχισαν να αναπτύσσουν διανομές Linux με υποστήριξη για τα χαρακτηριστικά που απαιτούνται στον σχεδιασμό ενσωματωμένων συστημάτων. Οι ενσωματωμένες διανομές Linux παρέχουν κατά κανόνα πολλαπλή στήριξη προγραμμάτων ανάπτυξης για πλατφόρμα-στόχο (target platform) διαφορετική από το περιβάλλον υποδοχής (host environment). [2]

Ορισμένα χαρακτηριστικά του Linux και άλλων λειτουργικών συστημάτων τύπου Unix ακολουθούν. [3]

- Multitasking:

Ο χρονοδρομολογητής του Linux υλοποιεί αληθινό, preemtive multitasking, με την έννοια ότι μια διεργασία υψηλότερης προτεραιότητας που προκαλείται από ένα ασύγχρονο γεγονός μπορεί να διακόψει την εκτέλεση της τρέχουσας διαδικασίας. Ωστόσο, οι διεργασίες του Linux kernel δεν είναι preemptible. Έτσι, μια διαδικασία δεν μπορεί να διακοπεί ενώ εκτελεί μια υπηρεσία πυρήνα Ορισμένες υπηρεσίες του πυρήνα μπορεί να απαιτούν μεγάλα χρονικά διαστήματα για την ολοκλήρωση τους με αποτέλεσμα την πρόκληση καθυστερήσεων (latencies). Για τον λόγο αυτό η βασική έκδοση του Linux καθίσταται γενικά ακατάλληλη για εφαρμογές πραγματικού χρόνου.

- Περιβάλλον πολλών χρηστών:

Το Unix είχε εξελιχθεί ως λειτουργικό σύστημα καταμερισμού χρόνου που επέτρεπε σε πολλαπλούς χρήστες να μοιράζονται ένα ακριβό (την εποχή εκείνη) υπολογιστή. Έτσι, υπάρχουν μια σειρά από χαρακτηριστικά για την υποστήριξη και προστασία της ιδιωτικότητας και των δεδομένων των χρηστών. Το Linux παρέχει αντίστοιχα και πιο εξελιγμένα χαρακτηριστικά, ιδιαιτέρως χρήσιμα σε περιβάλλοντα εξυπηρετητών.

- Πολυεπεξεργασία:

Το Linux παρέχει εκτεταμένη υποστήριξη για αληθινή συμμετρική πολυεπεξεργασία.

- Λειτουργία προστατευμένης μνήμης:

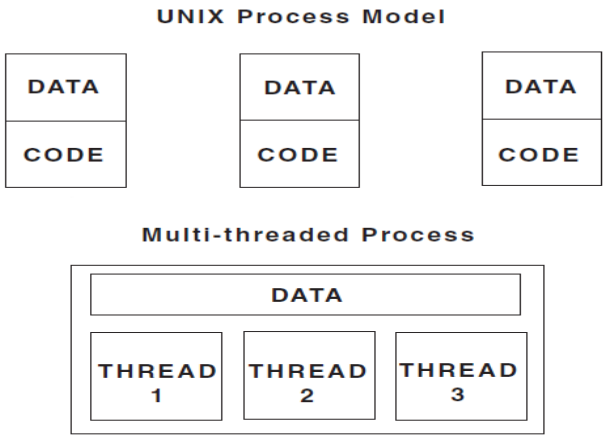
Κάθε διαδικασία Linux λειτουργεί στο δικό της ιδιωτικό χώρο μνήμης και δεν επιτρέπεται να υπάρχει άμεση πρόσβαση στο χώρο μνήμης από μια άλλη διαδικασία. Αυτό αποτρέπει για παράδειγμα έναν δείκτη σε μια διαδικασία από το να καταστρέψει το χώρο στη μνήμη μιας άλλης διαδικασίας. Η ανεπιθύμητη πρόσβαση παγιδεύεται από το υλικό προστασίας μνήμης του επεξεργαστή και η διεργασία τερματίζεται με σχετική ανακοίνωση.

- Ιεραρχικά δομημένο σύστημα αρχείων:

Όλα τα μοντέρνα λειτουργικά συστήματα διαθέτουν ιεραρχικό μοντέλο αρχείων. Το Linux προσθέτει σε αυτά τους συνδέσμους (links), που δείχνουν σε ένα αρχείο χωρίς να είναι αρχείο οι ίδιοι, και τον μηχανισμό εισόδου/εξόδου ανεξαρτήτως συσκευής. Το τελευταίο σημαίνει ότι όλα τα περιφερειακά αντιμετωπίζονται σαν είσοδοι στο σύστημα αρχείων με αποτέλεσμα να μην υπάρχει διαφορά για μια εφαρμογή αν θα καταγράψει την έξοδο της σε ένα αρχείο στον σκληρό δίσκο ή σε έναν εκτυπωτή.

Το μοντέλο διεργασιών του Linux (process model) έχει σαν βασικό δομικό στοιχείο την διεργασία που αποτελείται από τον εκτελέσιμο κώδικα και μια συλλογή από πόρους, όπως δεδομένα, περιγραφείς αρχείων και ούτω καθεξής. Αυτοί οι πόροι προστατεύονται πλήρως, έτσι ώστε μια διαδικασία δεν μπορεί να έχει άμεσα πρόσβαση στους πόρους μιας άλλης. Προκειμένου δύο διεργασίες να επικοινωνούν μεταξύ τους, πρέπει να χρησιμοποιήσουν τους μηχανισμούς επικοινωνίας μεταξύ διεργασιών που ορίζονται από το Linux, όπως οι κοινές περιοχές μνήμης ή οι διασωληνώσεις. Αυτό δημιουργεί ένα υψηλό βαθμό προστασίας στο σύστημα. Μια λανθασμένη διαδικασία είναι πολύ πιθανό να ανιχνευτεί από το σύστημα και να ακυρωθεί πριν μπορέσει να κάνει οποιαδήποτε ζημιά σε άλλες διαδικασίες. Το τίμημα που πρέπει να καταβληθεί είναι η υπερβολική επιβάρυνση στη δημιουργία διαδικασιών και στη χρήση των μηχανισμών επικοινωνίας στο εσωτερικό των διαδικασιών.

Ένα νήμα (thread) από την άλλη πλευρά είναι μόνο κώδικας. Τα νήματα υπάρχουν μόνο στο πλαίσιο μιας διεργασίας και όλα τα νήματα μίας διεργασίας μοιράζονται τους πόρους της έχοντας ίση πρόσβαση στα δεδομένα της μνήμης και τους περιγραφείς αρχείων. Αυτό το μοντέλο μερικές φορές ονομάζεται ελαφρύ multitasking για να ξεχωρίζει από το Unix/Linux μοντέλο διεργασίας. Το πλεονέκτημα του ελαφριού multitasking είναι ότι μεταξύ των νημάτων η επικοινωνία είναι πιο αποτελεσματική. Το μειονέκτημα, βέβαια, είναι ότι κάθε νήμα μπορεί να μονοπωλεί δεδομένα άλλων νημάτων. Τα περισσότερα λειτουργικά συστήματα πραγματικού χρόνου διαρθρώνονται γύρω από το ελαφρύ μοντέλο. Πρόσφατα το κόστος της προστασίας μνήμης έχει μειωθεί δραματικά και έτσι πολλές εκδόσεις RTOS προσφέρουν πλέον προστατευόμενη λειτουργία των συστημάτων τους που μοιάζει με το μοντέλο διεργασίας του Linux. Στην Εικόνα 1 [3] φαίνεται σχηματικά η σύνδεση δεδομένων και κώδικα στο μοντέλο διεργασιών Unix και μιας multitasking διεργασίας.



Εικόνα 1: Μοντέλο Διεργασιών και Μοντέλο Νημάτων

4.2 Περιγραφή των Δομικών Στοιχείων και των Λειτουργιών του Πυρήνα του Linux

Η συγγραφή των επόμενων παραγράφων βασίζεται στα [4], [5].

Ο πυρήνας είναι το πιο θεμελιώδες συστατικό του λογισμικού όλων των συστημάτων Linux. Είναι υπεύθυνος για τη διαχείριση του υλικού εντός του συστήματος στόχου και βάζει τάξη σε αυτό που διαφορετικά θα ήταν ένας χαοτικός αγώνας μεταξύ των διαφόρων συστατικών στοιχείων λογισμικού σε ένα τυπικό σύστημα. Στην ουσία, αυτό σημαίνει ότι ο πυρήνας φροντίζει για τον προγραμματισμό χρήσης και την πρόσβαση στους διαθέσιμους πόρους του υλικού. Οι πόροι που διαχειρίζονται από τον πυρήνα περιλαμβάνουν τον χρόνο επεξεργαστή που διατίθεται στα προγράμματα, της χρήση της διαθέσιμης μνήμης RAM, και την έμμεση πρόσβαση σε ένα πλήθος συσκευών υλικού. Ο πυρήνας παρέχει μια ποικιλία στρωμάτων αφαιρετικότητας του λογισμικού, μέσω των οποίων τα προγράμματα εφαρμογής μπορούν να ζητήσουν την πρόσβαση σε πόρους του συστήματος, χωρίς επικοινωνία με το υλικό του υπολογιστή.

Οι ακριβείς δυνατότητες που παρέχονται από κάποιο συγκεκριμένο στήσιμο (build) του πυρήνα του Linux είναι διαμορφώσιμες κατά την ρύθμιση (configuration) του. Η ρύθμιση του πυρήνα επιτρέπει να αφαιρεθεί η υποστήριξη για περιττές δυνατότητες ή περιφερειακά. Για παράδειγμα, είναι δυνατόν να αφαιρέσετε την υποστήριξη για πολλά διαφορετικά συστήματα αρχείων δικτύου από ένα ενσωματωμένο σύστημα που δεν συνδέεται σε δίκτυο. Είναι δυνατή η προσθήκη υποστήριξης για μια συγκεκριμένη περιφερειακή συσκευή μοναδική για ένα επιλεγμένο σύστημα στόχο. Ανάλογα με τη λειτουργία του συστήματος, πολλές ιδιότητες μπορούν επίσης να ενσωματωθούν σε προαιρετικό φόρτωμα κατά το runtime. Αυτές μπορούν να φορτωθούν αργότερα, όταν μια συγκεκριμένη δυνατότητα χρειαστεί.

Πολλές εταιρίες προσφέρουν ήδη στημένους πυρήνες με τις διανομές τους που να παρέχουν υποστήριξη για όλες τις πιθανές απαιτήσεις και χρήσεις ενός υπολογιστικού συστήματος ή ενός εξυπηρετητή για παράδειγμα. Αντίθετα, τα ενσωματωμένα συστήματα με λειτουργικό σύστημα Linux συνήθως δεν κάνουν χρήση των έτοιμων διανομών γενικής χρήσης και των πυρήνων τους. Ο λόγοι για αυτό ποικίλουν και περιλαμβάνουν τόσο την ανικανότητα των έτοιμων πυρήνων να χειριστούν ορισμένες ειδικές προσαρμογές και λειτουργίες που απαιτεί ένα ενσωματωμένο σύστημα όσο και τον στόχο η ρύθμιση του πυρήνα να διατηρείται απλή. Μια απλούστερη ρύθμιση παραμέτρων είναι πιο εύκολη να αποσφαλματωθεί και συνήθως προκαλεί μειωμένη επιβάρυνση στους πόρους.

Η αρχιτεκτονική ενός τυπικού πυρήνα Linux αποτελείται από το στρώμα αφαιρετικότητας υλικού (Hardware Abstraction Layer-HAL), τη διαχείριση μνήμης (memory manager), το χρονοδρομολογητή (scheduler), το σύστημα αρχείων (filesystem), το υποσύστημα εισόδου/εξόδου (I/O subsystem), το υποσύστημα δικτύου (networking subsystem) και τον μηχανισμό επικοινωνίας μεταξύ διεργασιών (InterProcess Communication-IPC). [6]

- Στρώμα Αφαιρετικότητας Υλικού:

Το στρώμα αφαιρετικότητας υλικού (HAL) εικονοποιεί (virtualize) την πλατφόρμα του υλικού, ώστε οι διάφοροι οδηγοί να μεταφέρονται εύκολα σε οποιοδήποτε υλικό. Το HAL είναι ισοδυναμεί με το πακέτο υποστήριξης πλακέτας (Board Support Package-BSP) που παρέχεται από τα περισσότερα RTOS με τη διαφορά ότι το BSP των εμπορικών RTOS έχει πρότυπες διεπαφές προγραμματισμού εφαρμογών (API) που επιτρέπουν την εύκολη μεταφορά. Το Linux δεν διαθέτει API γιατί αρχικά προοριζόταν για συστήματα επιφάνειας εργασίας αρχιτεκτονικής x86 και η υποστήριξη για άλλες πλατφόρμες προστέθηκε στην πορεία. Έτσι οι προγραμματιστές δεν ασχολήθηκαν με την τυποποίηση των HAL. Ωστόσο, οι πρόσφατες εκδόσεις του πυρήνα περιέχουν κάποια πρότυπα API. Δύο κοινές αρχιτεκτονικές επεξεργαστών, οι ARM και PowerPC, διαθέτουν σαφώς καθορισμένη σήμανση των δομών δεδομένων και των API. Από την έκδοση 2.6 του πυρήνα και μετά, οι κάτωθι αρχιτεκτονικές υποστηρίζονται από τα HAL: MIPS, M68K, CRIS, V850, SuperH. Το HAL προσφέρει την υποστήριξη για τα εξής [6]:

- Επεξεργαστής, κρυφή μνήμη και μονάδα διαχείρισης μνήμης (Memory Management Unit-MMU)
- Καθορισμός του χάρτη μνήμης.
- Χειρισμός εξαιρέσεων και διακοπών.
- Απευθείας πρόσβαση στη μνήμη (Direct Memory Access-DMA)
- Χρονιστές (timers).

- Το τερματικό του συστήματος.
- Διαχείριση διαύλων.
- Διαχείριση τροφοδοσίας.

➢ Διαχειριστής Μνήμης:

Ο διαχειριστής μνήμης του Linux είναι υπεύθυνος για τον έλεγχο της πρόσβασης στους πόρους μνήμης του υλικού και για την παροχή δυναμικής μνήμης για υποσυστήματα του πυρήνα, όπως οι οδηγοί συσκευών, τα συστήματα αρχείων και η στοίβα δικτύου. Υλοποιεί επίσης το λογισμικό που απαιτείται για την παροχή εικονικής μνήμης για τις εφαρμογές του χρήστη. Κάθε διαδικασία λειτουργεί σε χωριστό χώρο διευθύνσεων που ονομάζεται εικονική διεύθυνση. Με τη χρήση εικονικών διευθύνσεων, μια διαδικασία δεν μπορεί να καταστρέψει τη μνήμη του συστήματος ή μιας άλλης διαδικασίας. Ο πυρήνας του Linux διαιρεί τη συνολική μνήμη σε σελίδες των 4 KB. Αν και όλες οι σελίδες είναι προσβάσιμες από τον πυρήνα, αυτός κάνει χρήση μερικών από αυτές και οι υπόλοιπες χρησιμοποιούνται από τις εκάστοτε εφαρμογές. Επιπλέον, οι σελίδες του πυρήνα δεν φορτώνονται στην μνήμη κατά απαίτηση σε αντίθεση με αυτές των εφαρμογών. Η παρουσία της ξεχωριστής μνήμης για χρήστες και πυρήνα είναι σημαντική αλλαγή από ένα εμπορικό RTOS αφού γι'αυτό όλες οι εφαρμογές αποτελούν μέρος της ίδιας εικόνας που περιέχει το λειτουργικό σύστημα. Έτσι, όταν αυτή η εικόνα φορτωθεί, αντιγράφονται και οι εφαρμογές στη μνήμη. Στο Linux, ωστόσο, το λειτουργικό σύστημα και οι εφαρμογές γίνονται compile και build χωριστά. Κάθε εφαρμογή έχει το δικό της στιγμιότυπο αποθήκευσης που συχνά αναφέρεται ως το πρόγραμμα.

➢ Χρονοδρομολογητής:

Ο χρονοδρομολογητής του Linux παρέχει τις δυνατότητες multitasking και εξελίσσεται στις νέες εκδόσεις του πυρήνα, με στόχο την παροχή ντετερμινιστικής πολιτικής προγραμματισμού. Τα στιγμιότυπα εκτέλεσης που αναγνωρίζει ο χρονοδρομολογητής είναι το νήμα πυρήνα, διεργασίες δηλαδή που εκτελούνται στον χώρο του πυρήνα (kernel space) κατά τη διάρκεια ζωής τους, η διαδικασία χρήστη και το νήμα χρήστη. Κάθε διαδικασία χρήστη έχει το δικό της χώρο διευθύνσεων, χάρη στην εικονική μνήμη. Μπαίνουν σε κατάσταση λειτουργίας πυρήνα (kernel mode), όταν μια διακοπή, εξαίρεση, ή μια κλήση συστήματος εκτελείται. Σημειώστε ότι όταν μια διεργασία εισέρχεται κατάσταση λειτουργίας πυρήνα, χρησιμοποιεί μια εντελώς διαφορετική στοίβα. Αυτό αναφέρεται ως στοίβα πυρήνα και κάθε διεργασία έχει τη δική της πυρήνα στοίβα. Ο χρονοδρομολογητής του Linux αναλύεται διεξοδικά σε επόμενο κεφάλαιο.

➢ Σύστημα Αρχειοθέτησης:

Το σύστημα αρχείων του Linux είναι ιεραρχικό, μπορεί δηλαδή να υπάρχει αριθμός υποκαταλόγων κάτω από τους καταλόγους που δηλώνονται με “/” και υποστηρίζει μεγάλα ονόματα αρχείων αλλά σε αντίθεση με τα Windows, το επίθεμα των αρχείων μετά την τελεία “.” δεν είναι κατ'ανάγκη δηλωτικό της μορφής του αρχείου και η ύπαρξη του δεν είναι απαραίτητη. Το περιεχόμενο των αρχείων δηλώνεται σε επικεφαλίδα του αρχείου αντί για συγκεκριμένο επίθεμα. Τα ονόματα των αρχείων είναι ευαίσθητα σε κεφαλαία και πεζή γραφή και τα αρχεία των οποίων το όνομα ξεκινάει με τελεία αποκρύπτονται από τον χρήστη, εκτός αν ζητηθεί διαφορετικά. Επιπλέον, λόγω του περιβάλλοντος πολλών χρηστών κάθε αρχείο συνοδεύεται με ένα σετ αδειών που δηλώνουν ποιος μπορεί να αναγνώσει ,να μεταβάλλει ή να εκτελέσει το εν λόγω αρχείο, όπως στο Unix. Ο χρήστης ρίζας (root user) μπορεί να χειριστεί με οποιονδήποτε τρόπο όλα τα αρχεία ανεξάρτητα από τις σημαίες αδειών που αυτά έχουν.

Η περιγραφή των βασικών υποκαταλόγων που περιέχονται στις περισσότερες διανομές Linux κάτω από τον κατάλογο ρίζας “/”, ακολουθεί. [7]

- /proc : Το σύστημα αρχείων /proc είναι ένα ενδιαφέρον χαρακτηριστικό του Linux. Λειτουργεί ακριβώς όπως ένα σύνηθες σύστημα αρχείων. Μπορεί κάποιος να δει λίστα των αρχείων στον /proc κατάλογο, να διαβάσει και να γράψει τα αρχεία, αλλά αυτά δεν υπάρχουν πραγματικά. Οι πληροφορίες σε ένα /proc κατάλογο παράγονται επιτόπου, όταν διαβάζεται ένα αρχείο. Το module του πυρήνα που είναι υπεύθυνο για ένα δεδομένο /proc αρχείο περιέχει τις λειτουργίες που παράγουν ανάγνωση δεδομένων και δέχονται εγγραφή δεδομένων. Τα αρχεία /proc είναι ένα παράθυρο στον πυρήνα. Παρέχουν δυναμικά πληροφορίες σχετικά με την κατάσταση του συστήματος κατά τρόπο που να υπάρχει σε αυτές εύκολη πρόσβαση από το επίπεδο χρήστη και από την κονσόλα.
- /bin : Περιέχει βασικά δυαδικά δεδομένα εκτελέσιμων εντολών.
- /boot : Στατικά, μη μεταβλητά δηλαδή, αρχεία του φορτωτή εκκίνησης του συστήματος (boot loader).
- /etc : Ειδικά αρχεία ρυθμίσεων (configuration) του συστήματος υποδοχής.
- /dev : Αρχεία περιφερειακών και συσκευών (devices).
- /home : Προαιρετικός κατάλογος αποθήκευσης αρχείων των χρηστών.
- /lib : Βασικές κοινόχρηστες βιβλιοθήκες και modules.
- /mnt : Σημείο προσωρινής φόρτωσης και ενσωμάτωσης (mount) εξωτερικού συστήματος αρχείων.
- /opt : Πακέτα λογισμικού επιπλέον εφαρμογών.
- /root : Προαιρετικός κατάλογος αποθήκευσης αρχείων του χρήστη ρίζας.
- /sbin : Περιέχει βασικά δυαδικά δεδομένα συστήματος.
- /tmp : Εδώ βρίσκονται τα προσωρινά αρχεία.
- /usr : Επιπλέον προγράμματα και βιβλιοθήκες.
- /var : Περιέχει τα αρχεία που αλλάζουν κατά την διάρκεια εκτέλεσης, όπως κλειδώματα (locks) και αρχεία καταγραφής (logs).

➢ Υποσύστημα Εισόδου/Εξόδου:

Το υποσύστημα εισόδου/εξόδου του Linux παρέχει μια απλή και ενιαία διεπαφή για τις onboard συσκευές. Τα τρία είδη των συσκευών που υποστηρίζονται από το υποσύστημα I/O ακολουθούν:

- Συσκευές χαρακτήρων για την υποστήριξη συσκευών διαδοχής.
- Συσκευές μπλοκ για την υποστήριξη τυχαία προσβάσιμων συσκευών οι οποίες είναι ουσιαστικής σημασίας για την υλοποίηση συστημάτων αρχείων.
- Συσκευές δικτύου.

- Υποσύστημα Δικτύου:

Το Linux υποστηρίζει ποικίλα πρωτόκολλα δικτύωσης για την υλοποίηση των οποίων είναι υπεύθυνο το υποσύστημα δικτύου.

- Μηχανισμός Επικοινωνίας μεταξύ Διεργασιών:

Οι μηχανισμοί IPC του Linux περιλαμβάνουν σήματα (signals) για ασύγχρονη επικοινωνία, διασωληνώσεις (pipes), υποδοχές (sockets), καθώς και το μηχανισμούς IPC συστήματος V όπως είναι η κοινή μνήμη, οι ουρές μηνυμάτων και οι σημαφόροι. Ο πυρήνας έκδοσης 2,6 έχει την πρόσθετη υποστήριξη για τύπου POSIX ουρών μηνυμάτων.

Ένα σήμα είναι ένα πολύ σύντομο μήνυμα που μπορεί να σταλθεί σε μια διαδικασία ή μια ομάδα διεργασιών. Η μόνη πληροφορία που δίνεται στη διαδικασία είναι συνήθως ένα αριθμό που προσδιορίζει το σήμα αφού δεν υπάρχει χώρος σε τυποποιημένα σήματα για arguments, ένα μήνυμα, ή άλλες πληροφορίες για να το συνοδεύσουν. Τα σήματα εξυπηρετούν δύο βασικούς σκοπούς: γνωστοποιούν σε μια διαδικασία ένα συγκεκριμένο συμβάν και προκαλούν μια διαδικασία για να εκτελέσει μια συνάρτηση εξυπηρέτησης σήματος που περιλαμβάνεται στον κώδικα της. [4]

Η διασωλήνωση στο Linux και φυσικά στο Unix προσφέρει έναν μονόδρομο ροής δεδομένων. Το “|” σύμβολο είναι το σύμβολο της διασωλήνωσης που χρησιμοποιείται στη γραμμή εντολών. Αυτό που σημαίνει είναι ότι η τυπική έξοδο της εντολής από τα αριστερά του σωλήνα αποστέλλεται ως είσοδος της εντολής από τα δεξιά του σωλήνα. [8]

Μια υποδοχή domain Unix ή IPC υποδοχή είναι ένας τελικό σημείο επικοινωνίας δεδομένων παρόμοιο με Internet socket, αλλά χωρίς να χρησιμοποιεί πρωτόκολλο δικτύου για την επικοινωνία. Χρησιμοποιείται σε POSIX λειτουργικά συστήματα για την επικοινωνία μεταξύ διεργασιών. Οι υποδοχές Unix εμφανίζονται σαν ροές byte όπως και οι δικτυακές συνδέσεις, αλλά όλα τα δεδομένα παραμένουν εντός του υπολογιστή. Χρησιμοποιούν το σύστημα αρχείων σαν χώρο ονομάτων διευθύνσεων, δηλαδή αυτά αναφέρονται από διεργασίες ως inodes του συστήματος αρχείων. Αυτό επιτρέπει σε δύο διαφορετικές διαδικασίες να επικοινωνήσουν μέσω της ίδιας υποδοχής. Ωστόσο, η πραγματική επικοινωνία, η ανταλλαγή δεδομένων δηλαδή, δεν χρησιμοποιεί το σύστημα αρχείων αλλά προσωρινούς χώρους αποθήκευσης (buffers) στη μνήμη του πυρήνα. Εκτός από την αποστολή των στοιχείων, οι διαδικασίες μπορούν να στείλουν περιγραφείς αρχείων μέσω μιας σύνδεσης υποδοχής Unix χρησιμοποιώντας tiw sendmsg() και recvmsg() κλήσεις συστήματος. Αυτό σημαίνει ότι οι υποδοχές τομέα μπορούν να χρησιμοποιηθούν ως ένα σύστημα επικοινωνίας με ικανότητα αντικειμένων. [9]

Οι System V IPC μηχανισμοί μοιράζονται κοινές μεθόδους ελέγχου ταυτότητας. Οι διεργασίες μπορούν να έχουν πρόσβαση στους πόρους μόνο δίνοντας ένα μοναδικό αναγνωριστικό ως αναφορά στον πυρήνα μέσω κλήσης συστήματος. Η πρόσβαση σε αυτά τα System V IPC αντικείμενα ελέγχεται χρησιμοποιώντας τα δικαιώματα πρόσβασης και μοιάζει πολύ με προσβάσεις σε ελεγχόμενα αρχεία. Τα δικαιώματα πρόσβασης στο αντικείμενο IPC συστήματος V έχει οριστεί από το δημιουργό του αντικειμένου μέσω κλήσης συστήματος. Το αναγνωριστικό αναφοράς του αντικείμενου χρησιμοποιείται από κάθε μηχανισμό σαν δείκτης σε έναν πίνακα των πόρων. [10]

Ο χώρος χρήστη (user space) του Linux βασίζεται στις έννοιες του προγράμματος, της εικονικής μνήμης και της κλήσης συστήματος [6].

- Πρόγραμμα: Αυτή είναι η εικόνα μιας εφαρμογής και βρίσκεται σε ένα σύστημα αρχείων. Όταν μια εφαρμογή χρειάζεται να τρέξει, η εικόνα φορτώνεται και τρέχει στη μνήμη. Λόγω της εικονικής μνήμης δεν φορτώνεται στη μνήμη εικόνα της όλης διαδικασίας, μόνο οι απαιτούμενες σελίδες μνήμης.
- Εικονική μνήμη: Αυτό επιτρέπει σε κάθε διαδικασία να έχει το δικό της χώρο διεύθυνσεων. Η εικονική μνήμη επιτρέπει την ύπαρξη χαρακτηριστικών όπως οι κοινόχρηστες βιβλιοθήκες. Κάθε διεργασία έχει το δικό της χάρτη μνήμης στον εικονικό χώρο διευθύνσεων. Αυτό είναι μοναδικό για κάθε διαδικασία και είναι εντελώς ανεξάρτητο από το χάρτη μνήμης του πυρήνα.
- Κλήση συστήματος: Αυτές είναι τα σημεία εισόδου στον πυρήνα, έτσι ώστε ο πυρήνας να μπορεί να εκτελεί υπηρεσίες για λογαριασμό των εφαρμογών.

4.3 Ανάλυση του Χρονοδρομολογητή του Linux

Η συγγραφή των επομένων βασίζεται στα [11] [12] [13].

Ο χρονοδρομολογητής (scheduler) επιτρέπει την εκτέλεση πολλαπλών προγραμμάτων την ίδια στιγμή ενώ ένας σημαντικός στόχος του είναι το αποτελεσματικό μοίρασμα των φετών χρόνου (time slice) της κεντρικής μονάδας επεξεργασίας. Μπορεί επίσης να αντιμετωπίσει προβλήματα όπως η ελαχιστοποίηση των χρόνων απόκρισης για καθήκοντα πραγματικού χρόνου με παράλληλη μεγιστοποίηση της συνολικής χρήσης της CPU.

Μέχρι την έκδοση 2.6 του πυρήνα του Linux, ο χρονοδρομολογητής δεν διέθετε δυνατότητες πραγματικού χρόνου, ήταν O(n) με αποτέλεσμα οι καθυστερήσεις να μεγαλώνουν με την πρόσθεση νέων εργασιών και δεν υποστήριζε preemption. Επιπλέον είχε μόνο μια ουρά διαδικασιών που τρέχουν και μια ουρά κλειδώματος. Στον πυρήνα 2.6 αναπτύχθηκε νέος χρονοδρομολογητής τάξης O(1) ο οποίος λειτουργεί σε συνεχή χρόνο ανεξάρτητα από την είσοδο. Παρέχει δυναμική κατανομή του φόρτου εργασιών και δυναμική παροχή προτεραιότητας σε διεργασίες, αλλά μόνο για διεργασίες χρηστών.

Οι στόχοι και τα ζητήματα που ελήφθησαν υπόψιν κατά την ανάπτυξη του νέου χρονοδρομολογητή είναι με βάση το [14]:

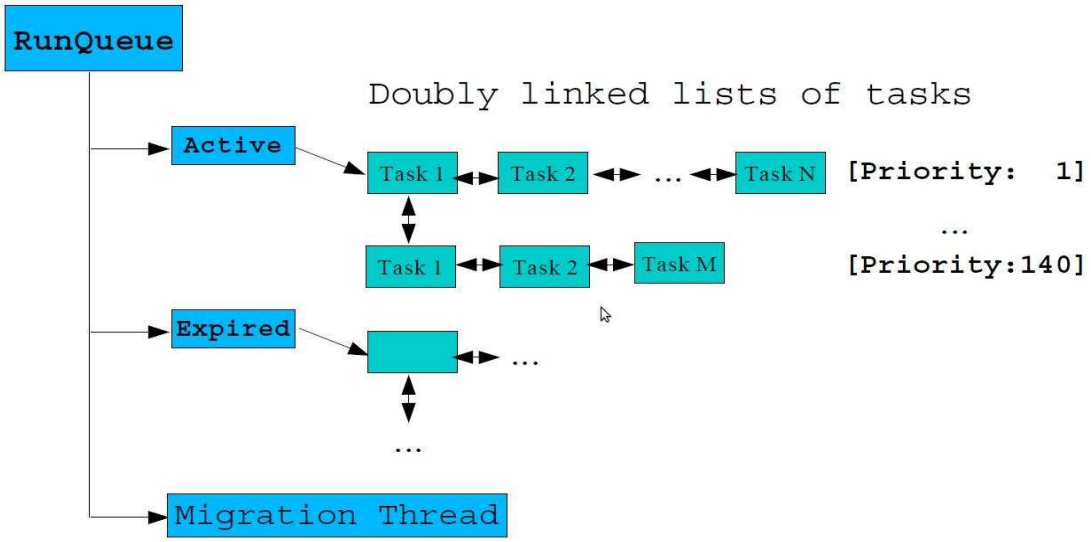
- Η δικαιοσύνη και συνεπώς η αποφυγή της “λιμοκτονίας” κάποιας διεργασίας.
- Ο χρόνοπρογραμματισμός των καθυστερήσεων μέσω της μείωσης του χρόνου μεταξύ του ξυπνήματος μιας διεργασίας και της εκτέλεσης της και του χρόνου που απαιτείται για τη λήψη των αποφάσεων από τον χρονοδρομολογητή.
- Οι καθυστερήσεις που εμφανίζονται κατά την εξυπηρέτηση μιας διακοπής ειδικότερα μιας διακοπής υλικού.
- Οι αποφάσεις του χρονοδρομολογητή.

➢ Οι στόχοι προς επίτευξη:

- Καλή διαδραστική απόδοση κατά τη λειτουργία με υψηλό φόρτο εργασίας.
- Οι προτεραιότητες.
- Αποδοτική συμμετρική πολυεπεξεργασία με τέλεια κλιμάκωση. Χρήση ουρών και κλειδωμάτων ανά επεξεργαστή και παράλληλη εκτέλεση λειτουργιών όπως αλλαγή περιεχομένου, ξύπνημα διεργασιών κλπ με προσοχή στην ιδιότητα συγγένειας ώστε να μην υπάρχουν τυχαίες αναπηδήσεις διεργασιών από επεξεργαστή σε επεξεργαστή.
- Χρονοδρομολόγηση πραγματικού χρόνου.
- Πλήρης O(1) χρονοδρομολόγηση ακόμα και σε πραγματικό χρόνο.

Κάθε στοιχείο επεξεργασίας διαθέτει μια ουρά εκτέλεσης (runqueue) που αποτελείται από 140 λίστες προτεραιοτήτων που εξυπηρετούνται με μορφή FIFO. Τα καθήκοντα που έχουν προγραμματιστεί για εκτέλεση προστίθεται στο τέλος της λίστας προτεραιότητας των αντίστοιχων runqueue τους. Κάθε διαδικασία έχει ένα κομμάτι του χρόνου που καθορίζει για πόσο χρόνο επιτρέπεται να εκτελεστεί. Οι πρώτες 100 λίστες προτεραιότητας του runqueue προορίζονται για καθήκοντα πραγματικού χρόνου και οι τελευταίες 40 χρησιμοποιούνται για καθήκοντα χρήστη.

Εκτός από τη runqueue της CPU, η οποία ονομάζεται ενεργός (active) runqueue, υπάρχει επίσης μια “ληγμένη” (expired) runqueue. Όταν μια εργασία που ανήκει στην ενεργό runqueue χρησιμοποιεί όλες τις φέτες χρόνου που της αντιστοιχούν, μετακινείται στη λήγμένη runqueue. Κατά τη διάρκεια της μετακίνησης, η φέτα χρόνου και συνεπώς και η προτεραιότητα της εργασίας υπολογίζεται εκ νέου. Εάν δεν υπάρχουν άλλα καθήκοντα στην ενεργό runqueue για κάθε συγκεκριμένη προτεραιότητα, οι δείκτες της ενεργής και ληγμένης runqueue ανταλλάσονται, καθιστώντας έτσι την ληγμένη runque ενεργή. Οι ουρές φαίνονται στην Εικόνα 2 [14].



Εικόνα 2: Ουρές και Λίστες Προτεραιοτήτων

Η δουλειά του χρονοδρομολογητή είναι η εξής: επιλογή της διεργασίας στην υψηλότερη λίστα προτεραιοτήτων ώστε να εκτελεστεί. Ένας χάρτης bit χρησιμοποιείται για να καθορίσει πότε μια διαδικασία βρίσκεται σε μια συγκεκριμένη λίστα προτεραιότητας. Ως εκ τούτου, στις περισσότερες αρχιτεκτονικές μια find-first-bit-set εντολή χρησιμοποιείται για να βρεθεί το σετ bit υψηλότερης προτεραιότητας σε μία από τις πέντε λέξεις 32-bit (για τις 140 προτεραιότητες). Ο χρόνος που χρειάζεται για να βρει μια εργασία για εκτέλεση δεν εξαρτάται από τον αριθμό των ενεργών εργασιών, αλλά από τον αριθμός των προτεραιοτήτων. Αυτό κάνει τον scheduler 2.6 μια διαδικασία O(1) αφού ο χρόνος για τον προγραμματισμό είναι τόσο σταθερός, όσο και ντετερμινιστικός ανεξάρτητα από τον αριθμό των ενεργών εργασιών.

Αποδεικνύοντας ότι ο νέος χρονοδρομολογητής είναι τάξης O(1) έχουμε [14]:

- Επιλέγεται η λίστα υψηλότερης προτεραιότητας που περιέχει τουλάχιστον μια διεργασία. Αυτό διαρκεί κάποιον σταθερό χρόνο “a”.
- Επιλέγεται από την λίστα υψηλότερης προτεραιότητας η διεργασία που βρίσκεται στην κεφαλή της λίστας. Αυτό διαρκεί κάποιον σταθερό χρόνο “b”.
- Συνεπώς ο συνολικός χρόνος για την επιλογή της επόμενης διεργασίας προς εκτέλεση είναι σταθερός και ίσος με t=a+b.

Έτσι, επιτυγχάνεται καλή λειτουργία συμμετρικής πολυεπεξεργασίας. Ακόμη και αν η προηγούμενη έκδοση χρονοπρογραμματιστή μπορούσε να εργαστεί σε συστήματα SMP, η αρχιτεκτονική της η οποία χρησιμοποιούσε μεγάλα κλειδώματα και global λίστες είχε σαν αποτέλεσμα ενώ το κλείδωμα της runqueue από την CPU που εκτελούσε μια εργασία. Κατά συνέπεια οι υπόλοιπες CPU του συστήματος βρισκόντουσαν αναγκαστικά σε αδράνεια και αναμονή απελευθέρωσης του κλειδώματος. Ο χρονοδρομολογητής 2.6 δεν χρησιμοποιεί ενιαίο κλείδωμα για τον προγραμματισμό. Αντίθετα, διαθέτει μηχανισμό κλειδώματος για κάθε runqueue. Αυτό επιτρέπει σε όλους τους επεξεργαστές να προγραμματίζουν εργασίες χωρίς εξάρτηση από τους άλλους επεξεργαστές. Επιπλέον, με runqueue ανά επεξεργαστή, μια εργασία μοιράζεται συγγένεια με τη CPU και κρυφή μνήμη της CPU χρησιμοποιείται πιο αποτελεσματικά.

Ο χρονοδρομολογητής υποστηρίζει τον χρόνο προγραμματισμό πολλαπλών νημάτων σε ένα ενιαίο Symmetric Multi-Threading (SMT) chip. Σε κάθε SMT chip ορίζονται περισσότεροι από έναν εικονικοί επεξεργαστές με την επιφύλαξη ότι μοιράζονται ορισμένους πόρους που δεν θα μοιραζόντουσαν εάν ήταν αυτόνομοι επεξεργαστές (πχ την κρυφή μνήμη). Επειδή ορισμένοι πόροι είναι κοινόχρηστοι, οι εικονικοί επεξεργαστές δεν αντιμετωπίζονται με τον ίδιο τρόπο που αντιμετωπίζονται οι πολλαπλές μονάδες επεξεργασίας.

Ένα άλλο πλεονέκτημα του νέου χρονοδρομολογητή είναι ότι επιτρέπει preemption. Αυτό σημαίνει μια εργασία χαμηλότερης προτεραιότητας δεν θα εκτελεστεί ενώ μια υψηλότερης προτεραιότητας εργασία είναι έτοιμη να τρέξει. Ο χρονοδρομολογητής διακόπτει την διαδικασίας χαμηλότερης προτεραιότητας, τοποθετεί τη διαδικασία πίσω στη λίστα των προτεραιοτήτων της, και στη συνέχεια ξανά-χρονοδρομολογεί.

Επιπλέον παρέχει την δυνατότητα δυναμικής παροχής προτεραιοτήτων σε διεργασίες για να αποτρέψει καθήκοντα από το να μονοπωλήσουν τον επεξεργαστή και έτσι να προκαλέσουν την λιμοκτονία άλλων εργασιών που χρειάζονται πρόσβαση στη CPU. Αυτό επιτυγχάνεται μέσω “τιμωρίας” των διαδικασιών που δεσμεύουν για μεγάλο χρονικό διάστημα έναν επεξεργαστή και επιβραβεύοντας τις διαδικασίες που είναι δεσμευμένες σε λειτουργίες εισόδου/εξόδου.

Τα καθήκοντα δεσμευμένα σε είσοδο/έξοδο συνήθως χρησιμοποιούν τον επεξεργαστή για το στήσιμο μιας εισόδου ή εξόδου και στη συνέχεια περνάνε σε αναμονή της ολοκλήρωσης της λειτουργίας I/O. Λόγω του “ύπνου” των καθηκόντων αυτών, ελευθερώνεται ο επεξεργαστής ώστε να αποκτήσουν πρόσβαση σε αυτών και άλλα καθήκοντα. Επειδή τα δεσμευμένα σε I/O καθήκοντα θεωρούνται ως αλτρουιστικά για την πρόσβαση στην CPU, η προτεραιότητά τους είναι μειωμένη (ανταμοιβή) κατά πέντε επίπεδα προτεραιότητας ενώ τα δεσμευμένα στην CPU καθήκοντα έχουν αυξημένη κατά πέντε επίπεδα προτεραιότητα. Είναι σημαντικό να σημειωθεί ότι οι προσαρμογές προτεραιότητας εκτελούνται μόνο σε εργασίες χρηστών και όχι σε καθήκοντα πραγματικού χρόνου.

Ο νέος χρονοδρομολογητής στηρίζει και την ανακατανομή του φόρτου εργασίας σε συστήματα συμμετρικής πολυεπεξεργασίας. Όταν τα καθήκοντα δημιουργούνται σε ένα σύστημα τοποθετούνται στην runqueue μιας δεδομένης CPU. Γενικά, δεν μπορεί να είναι γνωστό πότε ένα έργο θα είναι βραχύβιο ή πότε η εκτέλεση του θα διαρκέσει για πολύ καιρό. Ως εκ τούτου, η αρχική κατανομή των καθηκόντων στους επεξεργαστές είναι πιθανό να μην αντιστοιχεί στην θεωρητικώς βέλτιστη κατανομή. Για να διατηρηθεί ένας ισορροπημένος φόρτος εργασίας σε όλους τους επεξεργαστές, η εργασία μπορεί να αναδιανεμηθεί, λαμβάνοντας εργασία από κάποια υπερφορτωμένη CPU και δίνοντας εργασία σε κάποια πιο ελεύθερη. Κάθε 200ms, ένας επεξεργαστής ελέγχει για να διαπιστωθεί κατά πόσον ασύμμετρα είναι τα φορτία των μονάδων επεξεργασίας. Αν ισχύει κάτι τέτοιο, ο επεξεργαστής εκτελεί μια διασταυρωμένη εξισορρόπηση των καθηκόντων.

Μια αρνητική πτυχή αυτής της διαδικασίας είναι ότι η μνήμη cache της CPU στην οποία μεταναστεύει μια διαδικασία είναι κρύα, χρειάζεται δηλαδή να τραβήξει τα δεδομένα της διαδικασίας από τη μνήμη του συστήματος. Εάν κατά την εκτέλεση μιας διεργασίας σε μια CPU τα δεδομένα της εισάγονται στην τοπική κρυφή μνήμη της CPU, η κρυφή μνήμη θεωρείται καυτή (hot). Εάν δεν υπάρχουν δεδομένα για μια διεργασία στην τοπική μνήμη cache του επεξεργαστή, τότε για αυτό το έργο η μνήμη cache θεωρείται κρύα (cold). Αυτό μπορεί να θεωρηθεί σαν μειονέκτημα της εξισορρόπησης εργασίας αλλά δεν συγκρίνεται με τα πλεονεκτήματα που αυτή προσφέρει.

Ο πυρήνας του Linux υποστηρίζει μη ενιαία πρόσβαση μνήμης (NUMA), το οποίο σημαίνει ότι μπορεί να τρέξει ένα σύστημα εικόνας σε περισσότερους από έναν κόμβο αν υπάρχουν. Σαν κόμβος θεωρείται μια μητρική πλακέτα. Σε επίπεδο hardware, ένας κόμβος είναι κάτι σαν ένας παραδοσιακός uniprocessor ή ένα σύστημα πολλαπλών επεξεργαστών με την έννοια ότι έχει την δική του μνήμη και επεξεργαστή. Ωστόσο, τα συστήματα NUMA αντιμετωπίζουν τους πολλαπλούς κόμβους ως μέρη ενός ενιαίου συστήματος που τρέχει μια ενιαία εικόνα του συστήματος (δηλαδή ένα στιγμιότυπο του πυρήνα του Linux). Αυτό συνήθως επιτυγχάνεται μέσω μιας διασύνδεσης υψηλής ταχύτητας η οποία συνδέει τους κόμβους περισσότερο σε επίπεδο μητρικής πλακέτας παρά σε επίπεδο δικτύωσης. Αυτό σημαίνει ότι όλοι οι επεξεργαστές είναι ικανοί να εκτελέσουν οποιαδήποτε νήμα, καθώς επίσης και ότι το σύνολο της μνήμης σε όλους τους κόμβους είναι προσβάσιμο μέσω του ίδιου χώρου διευθύνσεων, δηλαδή κάθε CPU μπορεί να εκχωρήσει μνήμη για κάθε κόμβο του συστήματος.

Η υποστήριξη NUMA έχει σαν αποτέλεσμα την αντιμετώπιση προβλημάτων που αφορούν τις κρυφές μνήμες παρόμοια με τα SMP συστήματα και έχουν επιπλέον προβλήματα με την τοπικότητα της μνήμης. Για παράδειγμα αν μια CPU εκτελεί ένα νήμα που εκχωρεί μνήμη από μια τοπική μνήμη, θα ήταν αναποτελεσματικό να μετακινηθεί το νήμα σε όλο το σύστημα καθώς οι αιτήσεις για εκχώρηση μνήμης θα απαιτούσαν περισσότερο χρόνο για την εκπλήρωση τους. Ένα από τα μεγαλύτερα ζητήματα που πρέπει να αντιμετωπίσει ένας χρονοδρομολογητής με υποστήριξη NUMA, είναι η ύπαρξη πολλών περισσότερων κεντρικών μονάδων επεξεργασίας σε σχέση με συστήματα πολυεπεξεργασίας προς χρονοπρογραμματισμό. Τα κοινά συστήματα SMP μπορεί να έχουν από 2 μέχρι 8 επεξεργαστές ενώ τα NUMA συστήματα μπορεί να διαθέτουν εκατοντάδες επεξεργαστές. [11]

Το χρονοδιάγραμμα Linux υποστηρίζει χρονοδρομολόγηση σε ελαστικό πραγματικό χρόνο (soft real time). Παρόλο που είναι σε θέση να ελέγξει την τήρηση πολύ αυστηρών προθεσμιών, δεν εγγυάται ότι οι προθεσμίες θα τηρηθούν όντως. Τα καθήκοντα πραγματικού χρόνου αντιμετωπίζονται διαφορετικά από τα απλά καθήκοντα και ο χρόνοδρομολογητής τους ίνει προτεραιότητα σε σχέση με οποιοδήποτε άλλο καθήκον στο σύστημα.

Υπάρχουν δύο τρόποι χρονοπρογραμματισμού πραγματικού χρόνου [14]

- First-In-First-Out (FIFO), κατά τον οποίο δίνεται στατική προτεραιότητα στις διεργασίες πραγματικού χρόνου ενώ απαγορεύεται η διακοπή κάποιας διεργασίας, δεν είναι preemtive δηλαδή, και η κεντρική μονάδα επεξεργασίας αναλαμβάνει άλλο καθήκον όταν την απελευθερώσει το τρέχον.
- Round-Robin (RR), κατά τον οποίο σε κάθε καθήκον πραγματικού χρόνου ανατίθεται μια φέτα χρόνου και αυτό τρέχει μέχρι ο χρόνος του να εξαντληθεί. Όταν όλες η διεργασίες ενός ορισμένου επιπέδου προτεραιότητας εξαντλήσουν τις φέτες χρόνου που τις αντιστοιχούν, οι φέτες χρόνου ξαναγεμίζουν και συνεχίζεται η εκτέλεση τους. Τα επίπεδα προτεραιότητας των καθηκόντων διατηρούνται αμετάβλητα.
-

4.4 Το Linux σαν Λειτουργικό Σύστημα Πραγματικού Χρόνου για Ενσωματωμένα Συστήματα

Τα χαρακτηριστικά ενός λειτουργικού συστήματος πραγματικού χρόνου συνοψίζονται [15]:

- Υποστήριξη πολυεπεξεργασίας.
- Preemtive χρονοδρομολογητής.
- Χρονοδρομολογητής οδηγούμενος με βάση προτεραιότητες και ύπαρξη μέτρων για την αποφυγή λιμοκτονίας διεργασιών.
- Υποστήριξη προβλέψιμων μηχανισμών συγχρονισμού, όπως οι σημαφόροι και τα σήματα.
- Κληρονομικότητα των ιδιοτήτων από διεργασίες γονείς σε διεργασίες παιδιά.

Με βάση τα χαρακτηριστικά αυτά και τις παραπάνω παραγράφους, φαίνεται ότι το Linux μπορεί να χρησιμοποιηθεί σαν λειτουργικό σύστημα πραγματικού χρόνου με κάποιες επιπλέον τροποποιήσεις. Οι λόγοι για τις τροποποιήσεις αυτές ακολουθούν.

Το μέγεθος της μνήμης που απαιτείται για να τρέξει το Linux, καθώς επίσης και το αποτύπωμα του στη μνήμη πρέπει να ληφθούν υπόψιν. Από προεπιλογή, το Linux έχει μεγάλο αποτύπωμα μνήμης, ίσως πολύ μεγάλο για τα περισσότερα ενσωματωμένα συστήματα. Μια διανομή Linux είναι μεγέθους εκατοντάδων megabytes. Ένας τυπικός πυρήνας του Linux έχει μέγεθος 1,5 MB ασυμπίεστος και οι απαιτήσεις μνήμης RAM από τον πυρήνα για μια αρχική ρύθμιση είναι πάνω από 4 MB. Το Linux απαιτεί επίσης ένα σύστημα αρχείων για τη λειτουργία του, και εάν το ενσωματωμένο σύστημα δεν έχει δίσκο, αυτό σημαίνει ότι θα πρέπει να χρησιμοποιηθεί περισσότερη μνήμη με τη μορφή μνήμης τυχαίας προσπέλασης.

Η λύση στο ζήτημα απαιτήσεων για μνήμη του Linux είναι μέσω της προσεκτικής ρύθμισης του πυρήνα. Ο πυρήνας του Linux είναι οικοδομήσιμος (modular) και σε κάποιο βαθμό οι άχρηστες δυνατότητες που προσφέρει μπορούν να μην ενσωματωθούν στον πυρήνα. Ακόμα και οι πόροι του συστήματος, όπως ο ανώτατος αριθμός των διαδικασιών ή το άνοιγμα αρχείων στο σύστημα, μπορούν να ρυθμιστούν. Μέσω της ρύθμιση του πυρήνα και των δομικών στοιχείων του λειτουργικού συστήματος, το μέγεθος της ROM και το ίχνος της RAM μπορούν να μειωθούν. Είναι δυνατόν να ρυθμιστεί ένας πυρήνας του Linux για αρχιτεκτονική x86 με ασυμπίεστο μέγεθος 259 KB. Ένα ελάχιστο μέγεθος μνήμης ROM για την παροχή συστήματος αρχείων για τον πυρήνα αυτό μπορεί να είναι μεγέθους 102 KB. Η συνολική χρήση μνήμης RAM μπορεί να παραμείνει κάτω από 4 MB. [16]

Το Linux σχεδιάστηκε αρχικά ως ένα desktop λειτουργικό σύστημα και ακολουθεί το μοντέλο του Unix. Μια κονσόλα κειμένου χρησιμοποιείται για την αλληλεπίδραση με τον χειριστή του συστήματος. Η κονσόλα χρησιμοποιείται επίσης για τον έλεγχο και την εκκίνηση του λειτουργικού συστήματος αποτελώντας ένα μέσο για την παροχή εντολών αλλά και για την παρακολούθηση της δραστηριότητας του συστήματος.

Λόγω της φύσης χρήσης των ενσωματωμένων συστημάτων και των συστημάτων πραγματικού χρόνου, τα συστήματα αυτά αρκετά συχνά καλούνται να εκκινήσουν και να λειτουργήσουν χωρίς την ύπαρξη και την αλληλεπίδραση με τον χειριστή του συστήματος. Επιπλέον, αρκετά συχνά είναι συστήματα χωρίς κάποιον σκληρό δίσκο για να υποστηρίξουν σύστημα αρχείων. Ως εκ τούτου, ένα ενσωματωμένο σύστημα με Linux πρέπει να συγκροτηθεί έτσι ώστε να μην βασίζεται σε ένα χειριστή. Το Linux, όπως και τα άλλα λειτουργικά συστήματα βασισμένα σε Unix, χρησιμοποιούν το σύστημα αρχείων για την αποθήκευση και τον εντοπισμό εκτελέσιμων προγραμμάτων και μόνιμων δεδομένων. Ακόμη και οι εξωτερικές συσκευές του συστήματος αναπαριστώνται από ειδικά αρχεία στο σύστημα αρχείων. Ένα σύστημα αρχείων σε κάποιο είδος δίσκου αποτελεί αναπόσπαστο μέρος του περιβάλλοντος εκτέλεσης Linux.

Λύσεις υπάρχουν στα προηγούμενα προβλήματα. Η κονσόλα μπορεί απλά να είναι μια μηδενική (null) διάταξη. Δεν απαιτείται αλληλεπίδραση με χειριστή, αν τα script εκκίνησης αρχίζουν κατευθείαν τα προγράμματα εφαρμογής αντί για τις διαδικασίες σύνδεσης. Η παρακολούθηση μπορεί να γίνει απομακρυσμένα, εάν χρειάζεται, με τον καθορισμό ενός ψευδό-τερματικού ως κονσόλα με την εξασφάλιση και επιβεβαίωση της απομακρυσμένης σύνδεσης. Μπορεί το Linux να υποστηρίζει γραφικό περιβάλλον λειτουργίας, όμως είναι προαιρετικό και δεν απαιτείται για να τρέξει το λειτουργικό σύστημα.

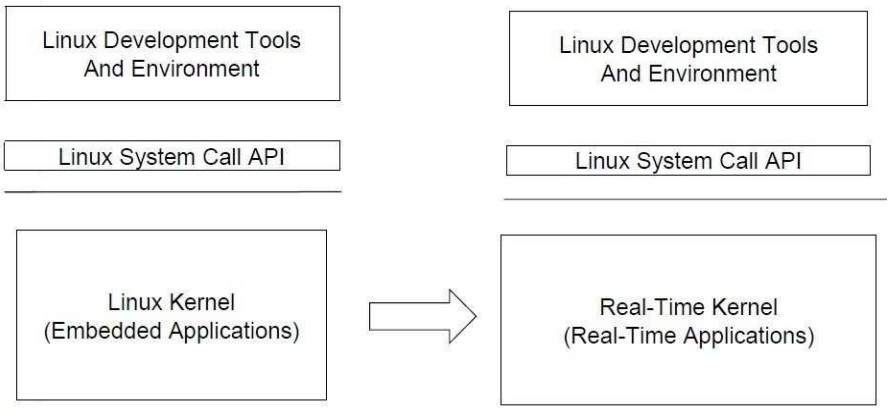
Το πρόβλημα της λειτουργίας χωρίς δίσκο μπορεί να επιλυθεί με ορισμένους τρόπους. Υπάρχουν συσκευές μνήμης flash που μιμούνται σκληρούς δίσκους. Αυτές οι συσκευές μπορούν να έχουν πολύ μικρό φυσικό μέγεθος, όχι πολύ μεγαλύτερο από το βύσμα που απαιτείται για την διασύνδεση ενός σκληρού δίσκου. Linux περιλαμβάνει επίσης ένα πρόγραμμα οδήγησης δίσκου RAM. Πρόκειται για ένα πρόγραμμα οδήγησης συσκευής που χρησιμοποιεί κάποιο ποσό της διαθέσιμης μνήμης τυχαίας προσπέλασης του συστήματος για να μιμηθεί μια μονάδα δίσκου. Ένας δίσκος RAM μπορεί να χρησιμοποιηθεί για αρχεία δεδομένων που δεν πρέπει να είναι ανθεκτικά κατά την επανεκκίνηση του συστήματος. Είναι επίσης δυνατό να χρησιμοποιηθεί ο δίσκος RAM για την πρόσβαση σε ένα σύστημα αρχείων στη μνήμη μόνο ανάγνωσης. Αν το ενσωματωμένο σύστημα μνήμης flash δεν μπορεί να μιμηθεί μια μονάδα σκληρού δίσκου, ένα σύστημα αρχείων flash μπορεί να χρησιμοποιηθεί για να είναι εφικτή η χρήση της μνήμης flash σαν σκληρός δίσκος. [16]

Η νέα έκδοση του χρονοδρομολογητή προσφέρει σημαντικά καλύτερο preemptability κατά μέσο όρο. Από την άλλη πλευρά, ο συγχρονισμός μεταξύ των καθηκόντων που εξακολουθεί να είναι μην είναι ο βέλτιστος, διότι η ικανότητα διακοπής του πυρήνα παραμένει απενεργοποιημένη όσο το σύστημα κατέχει κάποια αποκλειστική πηγή, ακόμη και εάν δεν υπάρχει άλλο καθήκον που να ανταγωνίζεται για τον ίδιο πόρο. Σαν συνέπεια αυτού, μόνο η μέση καθυστέρηση βελτιώνεται, αλλά η χειρότερη περίπτωση καθυστέρησης εξακολουθεί να μη συμβιβάζεται με τις απαιτήσεις αυστηρού πραγματικού χρόνου. Ένας τρόπος κατάλυσης αυτού του περιορισμού είναι να καταστεί η απόκτηση και η αποδέσμευση κάθε αποκλειστικού πόρου αληθινά ανεξάρτητου από τους άλλους πόρους.

Ακόμη και με τον preemptible πυρήνα που συνεπάγεται ο χρονοδρομολογητής, ένα έργο χαμηλής προτεραιότητας που έχει διακοπεί (preempted) θα μπορούσε να κρατήσει αποκλειστική έναν πόρο για αρκετό καιρό ώστε να καθυστερήσει καταστροφικά την εκτέλεση ενός έργου υψηλής προτεραιότητας που απαιτεί πρόσβαση στον ίδιο πόρο. Το να συμβεί κάτι τέτοιο είναι απλώς θέμα μιας απροσδόκητης καθυστέρησης εκτέλεσης του έργου χαμηλής προτεραιότητας ενώ κατέχει τον πόρο. Αυτό είναι ένα παράδειγμα της αναστροφής προτεραιοτήτων. Επειδή οι αποκλειστικοί πόροι είναι διάσπαρτοι στον πυρήνα του Linux, οι πιθανότητες να συμβεί η αναστροφή προτεραιοτήτων είναι πολύ υψηλές. Αυτό το θέμα μπορεί να αντιμετωπιστεί με συγκεκριμένες τεχνικές όπως η κληρονομικότητα προτεραιότητας ή το ανώτατο όριο προτεραιότητας, το οποίο εγγυάται ότι μια τέτοια αντιστροφή θα παραμείνει βραχυπρόθεσμη και χρονικά οριοθετημένη, ή δεν θα συμβεί καθόλου. [17]

Φαίνεται ότι ακόμα και στον τομέα του πραγματικού χρόνου απαιτούνται μεταρρυθμίσεις στον τρόπο λειτουργίας του πυρήνα ώστε να ανταποκρίνεται σε αυστηρά χρονικά όρια και εργασίες με απαιτήσεις μικρών καθυστερήσεων. Τέσσερις προσεγγίσεις στην μετατροπή του λειτουργικού συστήματος Linux σε λειτουργικό σύστημα πραγματικού χρόνου δίνονται στις παρουσιάσεις στο [15].

- Compliant Kernel

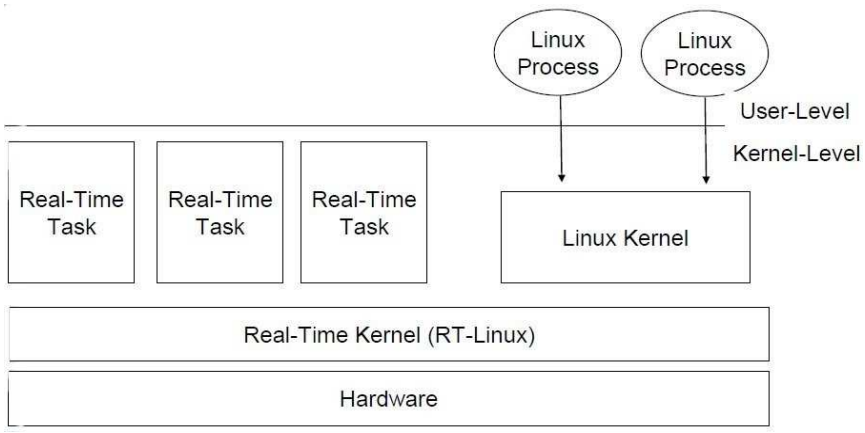


Εικόνα 3: Αρχιτεκτονική Compliant Kernel

Σαν κεντρική ιδέα της αρχιτεκτονικής συμπληρωματικού πυρήνα (compliant kernel) βρίσκεται ο καθορισμός του Linux από τις διεπαφές προγραμματισμού εφαρμογών που παρέχει και όχι από την εσωτερική υλοποίηση του ενώ ο πυρήνας πραγματικού χρόνου που χρησιμοποιείται δεν είναι Linux.

Τα μειονεκτήματα αυτής της προσέγγισης περιλαμβάνουν την παραμονή της κεντρική ροής του Linux ενώ οι τελευταίες βελτιώσεις του πυρήνα δεν περιλαμβάνονται αυτόματα. Η προσθήκη νέου υλικού απαιτεί νέα μεταφορά (porting) των προγραμμάτων οδήγησης και τα προγράμματα οδήγησης του Linux δεν μπορούν να χρησιμοποιηθούν. Σαν πλεονεκτήματα της υλοποίησης αυτής αναφέρεται η παροχή του πραγματικό API του Linux. Οποιαδήποτε εφαρμογή Linux μπορεί να τρέξει στον πυρήνα πραγματικού χρόνου και επιπλέον όλες οι βιβλιοθήκες του Linux είναι διαθέσιμες στον πυρήνα πραγματικού χρόνου.

- Dual Kernel



Εικόνα 4: Αρχιτεκτονική Dual Kernel

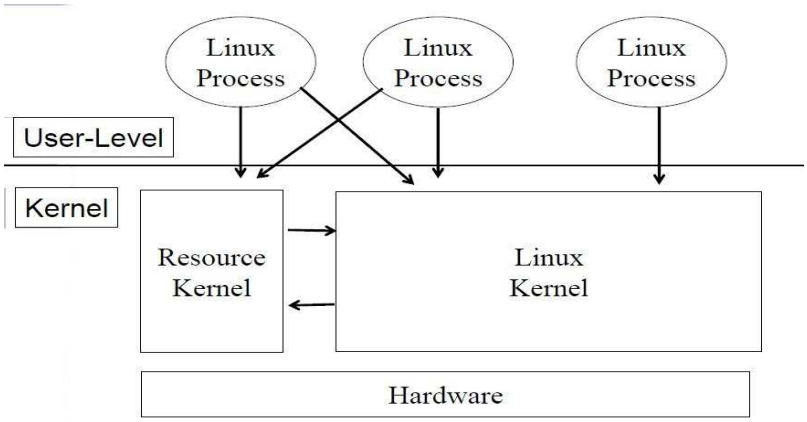
Σαν ιδέα της συγκεκριμένης υλοποίησης βρίσκεται το γεγονός ότι τα καθήκοντα πραγματικού χρόνου δεν χρησιμοποιούν τη διεπαφή προγραμματισμού του Linux ή τις δυνατότητες που αυτό παρέχει. Το μεγάλο μειονέκτημα της αρχιτεκτονικής διπλού πυρήνα είναι ότι η αποτυχίας μιας διεργασίας πραγματικού χρόνου οδηγεί σε αποτυχία ολόκληρου του συστήματος.

Η κύρια χρήση του πυρήνα πραγματικού χρόνου (ή λεπτού πυρήνα), εκτός από τον πυρήνα που αναλαμβάνει τις διεργασίες πραγματικού χρόνου, είναι η διαχείριση διακοπών. Ο λεπτός πυρήνας παρακολουθεί τις διακοπές για να εξασφαλίσει ότι μη-πραγματικού χρόνου πυρήνας δεν μπορεί να παρέμβει στη λειτουργία πραγματικού χρόνου. Αυτό επιτρέπει την παροχή υποστήριξης αυστηρού πραγματικού χρόνου.

Το πλεονέκτημα της μεθόδου είναι η παροχή αυστηρού πραγματικού χρόνου σε συνύπαρξη με έναν πρότυπο πυρήνα Linux. Παρουσιάζει όμως και μειονεκτήματα. Λόγω της ανεξαρτητοποίησης των λειτουργιών πραγματικού χρόνου και μη-πραγματικού χρόνου δυσχεραίνει η αποσφαλμάτωση του συστήματος. Επίσης, οι διεργασίες μη-πραγματικού χρόνου δεν έχουν την πλήρη υποστήριξη της πλατφόρμας Linux λόγω του λεπτού πυρήνα.

Παραδείγματα αυτής της προσέγγισης περιλαμβάνουν το RTLinux, το Application Interface (RTAI), και το Xenomai. [18]

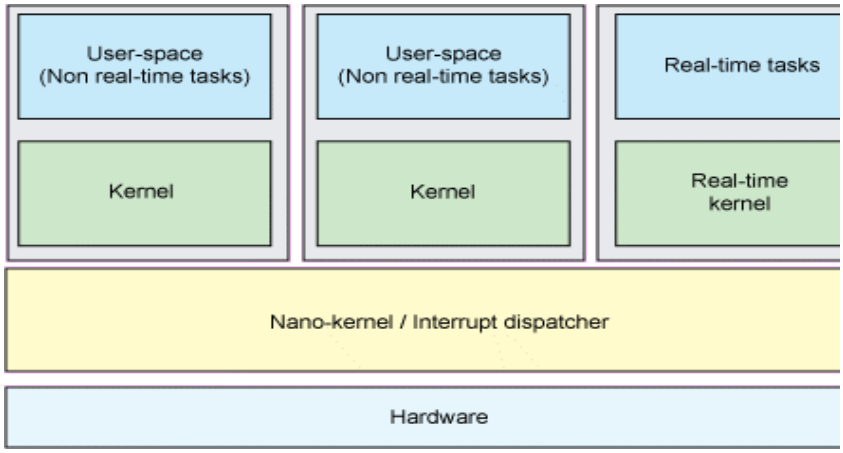
➢ Resource Kernel (Πυρήνας Πόρων)



Εικόνα 5: Αρχιτεκτονική Πυρήνα Πόρων

Η τεχνική αυτή περιλαμβάνει έναν πυρήνα ο οποίος εικονοποιεί (virtualizes) το υλικό με τρόπο ώστε όλες οι εφαρμογές να έχουν πρόσβαση στους πόρους που προσφέρει το υλικό. Οι εφαρμογές εκδίδουν αιτήσεις για χρήση πόρων και ο πυρήνας ικανοποιεί τις απαιτήσεις αυτές χρησιμοποιώντας διαχειριστικά σχήματα που αποκρύπτονται από τις εφαρμογές. Η προσέγγισή του πυρήνα πόρων προσθέτει μια δομική μονάδα (module) σε ένα πυρήνα που φυλάει τα διάφορα είδη των πόρων. Οι επιφυλάξεις εγγυώνται την πρόσβαση στους πόρους του συστήματος με πολύπλεξη χρόνου. Οι πόροι αυτοί έχουν διάφορες παραμέτρους φύλαξης, όπως το χρονικό διάστημα της επανάληψης, ο απαιτούμενος χρόνος επεξεργασίας καθώς και την προθεσμία της διεργασίας που κατέχει τον πόρο. Ο πυρήνας των πόρων παρέχει ένα σύνολο από API για να διευκολύνει τα καθήκοντα να απαιτήσουν τέτοια κλειδώματα σε πόρους. Ο πυρήνας πόρων συγχωνεύει στη συνέχεια τα αιτήματα και καθορίζει ένα χρονοδιάγραμμα για την παροχή εγγυημένης πρόσβασης μέσω των περιορισμών που καθορίζουν οι διεργασίες (ή να επιστρέψει ένα λάθος, αν δεν είναι εξασφαλισμένη). Χρησιμοποιώντας έναν αλγόριθμο προγραμματισμού, όπως Earliest-Define-First (EDF), ο πυρήνας μπορεί στη συνέχεια να χρησιμοποιηθεί για να χειριστεί το δυναμικού προγραμματισμού φόρτο εργασίας. [18]

➢ Nano Kernel (Νανοπυρήνας)



Εικόνα 6: Αρχιτεκτονική Νανοπυρήνα

Η προσέγγιση του νανοπυρήνα προχωρά ένα βήμα παραπέρα με την ελαχιστοποίηση του πυρήνα ακόμη περισσότερο. Στην πραγματικότητα είναι λιγότερο πυρήνας και περισσότερο ένα στρώμα αφαιρετικότητας υλικού (HAL). Ο νανοπυρήνας παρέχει κοινή χρήση στους πόρους του υλικού σε πολλαπλά λειτουργικά συστήματα που λειτουργούν σε υψηλότερο επίπεδο όπως φαίνεται στην Εικόνα 6 [18]. Επειδή ο νανοπυρήνας αποκρύπτει το υλικό, μπορεί να δώσει προτεραιότητα στο τρίτο στρώμα με τα λειτουργικά συστήματα και να υποστηρίζει συνεπώς αυστηρό πραγματικό χρόνο.

Είναι παρόμοιο με την τεχνική της εικονοποίησης (virtualization) για την εκτέλεση πολλαπλών λειτουργικών συστημάτων. Ο νανοπυρήνας αποκρύπτει το υλικό από τους πυρήνες πραγματικού και μη πραγματικού χρόνου όπως οι hypervisors αποκρύπτουν το υλικό από τα φιλοξενούμενα (guest) λειτουργικά συστήματα.

Ένα παράδειγμα είναι το Adaptive Domain Environment for Operating Systems (ADEOS). Το ADEOS υποστηρίζει πολλαπλά συγχρονισμένα λειτουργικά συστήματα που εκτελούνται ταυτόχρονα. Όταν συμβαίνουν γεγονότα υλικού, το ADEOS απευθύνει αίτηση σε κάθε λειτουργικό σύστημα σε μορφή αλυσίδας για να αποφασιστεί ποιο θα χειριστεί την εκδήλωση του συμβάντος.

- Core Kernel

Η ιδέα πίσω από την τεχνική αυτή είναι η βελτίωση του καθεαυτού πυρήνα του Linux ώστε να γίνει καταλληλότερος για εφαρμογές πραγματικού χρόνου μέσω της σωστής ρύθμισης του. Προσοχή απαιτείται στις δυνατότητες αναβάθμισης του συστήματος η οποία πρέπει να γίνεται εύκολα και χωρίς να συμβιβάζει την αξιοπιστία και κλιμάκωση του λειτουργικού συστήματος.

Αναφορές

[1] “The Linux Home Page at Linux Online”; http://www.linux.org/.

[2] “Century Embedded Technologies : EMBEDDEDAPPDEV”; http://embedded.censoft.com/white_papers/EmbeddedAppDev.php.

[3] D. Abbott, *Linux for embedded and real-time appplications*, Newnes, 2006.

[4] D.P. Bovet and M. Cesatí, *Understanding the Linux Kernel*, O'Reilly Media, Inc., 2005.

[5] “The Linux Kernel”; http://tldp.org/LDP/tlk/tlk.html.

[6] P. Raghavan, A. Lad, and S. Neelakandan, *Embedded Linux system design and development*, CRC Press, 2005.

[7] C. Hollabaugh, *Embedded Linux*, Addison-Wesley, 2002.

[8] “Intro to Unix: Pipes and Filters”; http://www.december.com/unix/tutor/pipesfilters.html.

[9] “Unix Sockets tutorial | Unix Tutorial”; http://www.unixtutorial.org/2008/01/unix-sockets-tutorial/.

[10] “System V IPC Mechanisms”; http://sunsite.nus.sg/LDP/LDP/tlk/node56.html.

[11] Josh Aas, “Understanding the Linux 2.6.8.1 CPU Scheduler,” 2005.

[12] “Kernel Korner - What's New in the 2.6 Scheduler? | Linux Journal”; http://www.linuxjournal.com/article/7178.

[13] “Inside the Linux scheduler”; http://www.ibm.com/developerworks/linux/library/l-scheduler/#N10078.

[14] Mahendra M, “Scheduling in Kernel 2.6”; http://www.infosys.com.

[15] “ASIC Design Home Page”; http://www.ict.tuwien.ac.at/asicdesign/lehre/384089.html.

[16] “Successfully Using Linux and Open Software in an Embedded System Design: Embedded Linux white paper”; http://www.lynuxworks.com/products/whitepapers/usinglinux.php3.

[17] K. Yaghmour, J. Masters, G. Ben-Yossef, and P. Gerum, *Building Embedded Linux Systems*, O'Reilly Media, Inc., 2008.

[18] “Anatomy of real-time Linux architectures”; http://www.ibm.com/developerworks/linux/library/l-real-time-linux/.

5 Αξιοποίηση του Λειτουργικού Συστήματος Πραγματικού Χρόνου ASMP-Linux

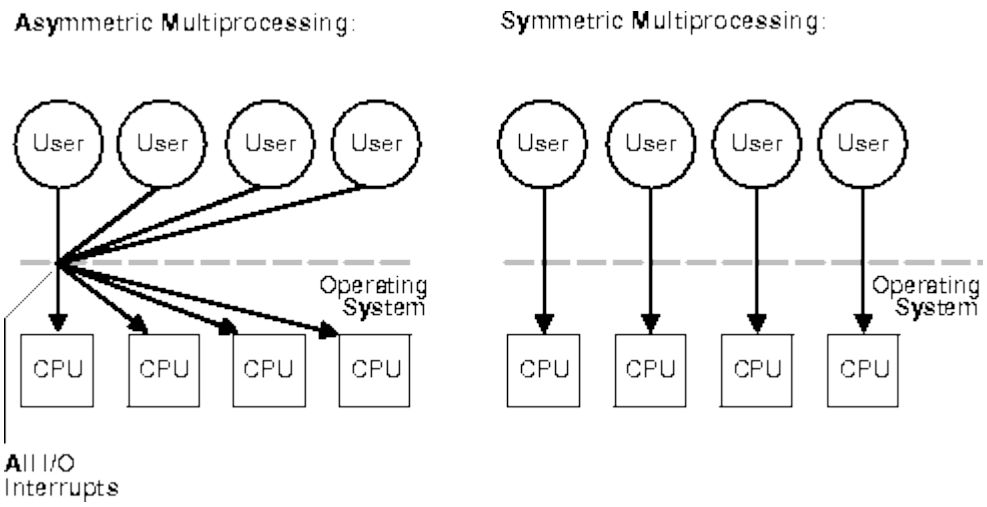
5.1 Εισαγωγή

Το ASMP-Linux (Asymmetric Multiprocessor Linux) [1] αποτελεί μια τροποποιημένη εκδοχή του πυρήνα του Linux που μπορεί να να χρησιμοποιηθεί σε συστήματα πολυεπεξεργασίας, καθώς και σε ενσωματωμένα συστήματα με απαιτήσεις πραγματικού χρόνου. Αναπτύχθηκε το 2007 από την ομάδα προγραμματισμού συστημάτων του πανεπιστήμιου της Ρώμης “Tor Vergata”. [2] Το ASMP-Linux είναι ένα λειτουργικό σύστημα ανοιχτού κώδικα που χρησιμοποιεί μια ασύμμετρη προσέγγιση του πυρήνα.

Σε ένα συμβατικό συμμετρικό πυρήνα, οι συσκευές εισόδου/εξόδου και οι επεξεργαστές θεωρούνται ομοειδή, εφόσον το φορτίο του συστήματος δεν λαμβάνεται υπόψιν. Οι ασύμμετροι πυρήνες αντίθετα, θεωρούν προνομιακές τις διαδικασίες πραγματικού χρόνου και τις συσκευές που χρησιμοποιούν ή χρειάζονται και τις προφυλάσσουν από τις υπόλοιπες δραστηριότητες του συστήματος.

Το ASMP-Linux πλεονεκτεί όσον αφορά την αιτιότητα του χρόνου εκτέλεσης των εφαρμογών, διατηρώντας παράλληλα χαμηλή την επιβάρυνση του συστήματος με υψηλές αποδόσεις. Μέσω της διατήρησης της απλότητας του κώδικα δεν επηρεάζει τις επιδόσεις του υπόλοιπου συστήματος. Επιπλέον, δεν απαιτείται re-compiling και re-linking των ήδη υπαρχόντων εφαρμογών. Απαιτείται όμως η παροχή τουλάχιστον ενός από τους επεξεργαστές για να αφιερωθεί στην εκτέλεση καθηκόντων πραγματικού χρόνου. [3]

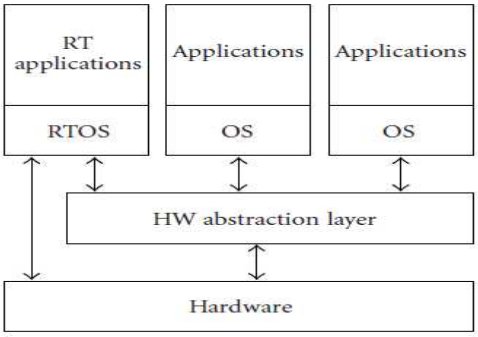
Από το [4] βλέπουμε στην Εικόνα 1 τη διαφορά ενός συμμετρικά και ενός ασύμμετρα δομημένου λειτουργικού συστήματος:



Εικόνα 1: Ασύμμετρη και Συμμετρική Δομή Λειτουργικού Συστήματος

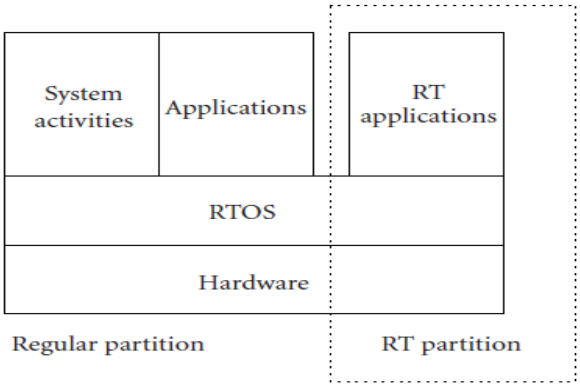
Επιπλέον, η υλοποίηση ενός λειτουργικού συστήματος πραγματικού χρόνου ανάλογα με την κατανομή των διεργασιών μπορεί να είναι οριζόντια (horizontal partitioning) ή κάθετη (vertical partitioning). Ο καταμερισμός αυτός έχει να κάνει με τον παραλληλισμό των διεργασιών που παρατηρείται σε ένα σύστημα, αφού, λόγω της αυξανόμενης πολυπλοκότητας των ενσωματωμένων συστημάτων, ένα RTOS πρέπει να διαχειρίζεται εξωτερικά γεγονότα και κρίσιμες για το σύστημα δραστηριότητες του λειτουργικού, πέρα από την επίτευξη αυστηρών προδιαγραφών για τα καθήκοντα πραγματικού χρόνου.

Τα οριζόντια καταμερισμένα λειτουργικά συστήματα έχουν ένα κατώτατο στρώμα (που ονομάζεται στρώμα αφαιρετικότητας υλικού, Hardware Abstraction Layer-HAL) που εικονοποιεί (virtualize) το πραγματικό υλικό. Το στρώμα αυτό παρουσιάζει στο λειτουργικό σύστημα τους πόρους σαν πολλές εικονικές μηχανές η καθεμία από τις οποίες τρέχει ένα πρότυπο ή τροποποιημένο λειτουργικό σύστημα, ένα για την κάθε περιοχή της εκάστοτε εφαρμογής. Έτσι επιτυγχάνεται η εκτέλεση εφαρμογών στο δικό τους τομέα, σαν να έτρεχαν σε ένα αφιερωμένο για αυτές σύστημα. Σχηματικά αυτό παρουσιάζεται στην Εικόνα 2. [5]



Εικόνα 2: Οριζόντια Καταμερισμένο Λειτουργικό Σύστημα

Στην περίπτωση του κάθετα καταμερισμένου λειτουργικού συστήματος πραγματικού χρόνου, οι πόροι που είναι κρίσιμοι για την εκτέλεση εφαρμογών πραγματικού χρόνου ανατίθενται απευθείας στις εφαρμογές αυτές χωρίς να παρεμβάλλεται στρώμα αφαιρετικότητας λογισμικού. Οι μη-κρίσιμες εφαρμογές και οι διεργασίες του λειτουργικού δεν επιτρέπεται να χρησιμοποιήσουν τους πόρους αυτούς. Μέσω αυτής της προσέγγισης, που ακολουθεί και το ASMP-Linux, οι συνιστώσες πραγματικού χρόνου του λειτουργικού συστήματος διατηρούν την απλότητας τους, σαν συνέπεια της έλλειψης εικονικών μηχανών. Ακόμα παρέχονται δυνητικά υψηλότερες αποδόσεις σε σύγκριση με ένα οριζόντια καταμερισμένο λειτουργικό σύστημα σαν συνέπεια της έλλειψης της επιβάρυνσης που προκαλεί το στρώμα αφαιρετικότητας του υλικού. Επιπλέον, τα απλά καθήκοντα του συστήματος δεν επιβραδύνονται αφού διαθέτουν αφιερωμένους σε αυτά πόρους, διαφορετικούς από αυτούς των καθηκόντων πραγματικού χρόνου. Σχηματικά η δομή αυτή παρουσιάζεται στην Εικόνα 3. [5]



Εικόνα 3: Κάθετα Καταμερισμένο Λειτουργικό Σύστημα

5.2 Χαρακτηριστικά του ASMP-Linux

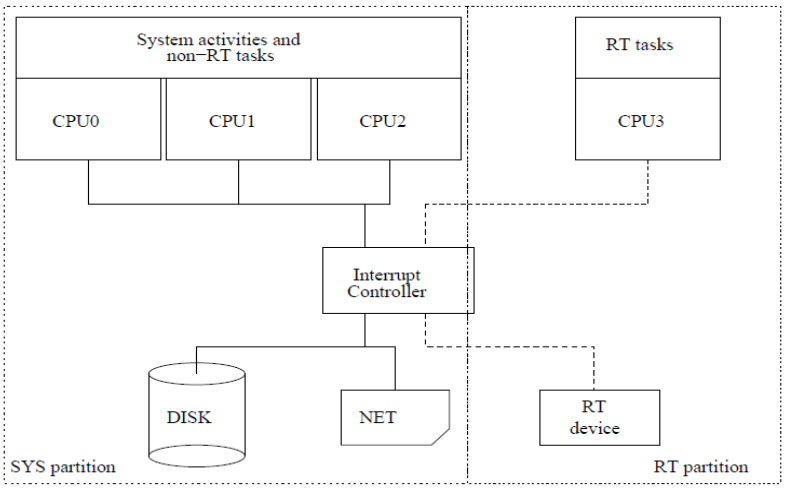
Σύμφωνα με τα [3], [5] και την έρευνα στην παρούσα διπλωματική εργασία παρουσιάζονται τα χαρακτηριστικά του ASMP-Linux.

5.2.1 Διαμερίσματα του Λειτουργικού Συστήματος

Το ASMP-Linux χωρίζει το λειτουργικό σε δύο διαμερίσματα (partitions), το διαμέρισμα του συστήματος και το διαμέρισμα πραγματικού χρόνου. Στο διαμέρισμα συστήματος εκτελούνται όλες οι διεργασίες που δεν είναι πραγματικού χρόνου, όπως οι οδηγοί μη-κρίσιμων συσκευών και η διαχείριση διακοπών μη-κρίσιμων διεργασιών. Το διαμέρισμα πραγματικού χρόνου χειρίζεται τα καθήκοντα πραγματικού χρόνου καθώς και κάθε συσκευή υλικού και τους οδηγούς τους, που είναι ζωτικής σημασίας για τις επιδόσεις των εν λόγω εργασιών. Υπάρχει πάντα ένα διαμέρισμα συστήματος και τουλάχιστον ένα διαμέρισμα πραγματικού χρόνου. Μπορούν να υπάρχουν περισσότερα διαμερίσματα πραγματικού χρόνου καθένα από τα οποία διαχειρίζεται μια συγκεκριμένη διεργασία πραγματικού χρόνου. Όπως φαίνεται στην Εικόνα 4 [3] κάθε διαμέρισμα πραγματικού χρόνου αποτελείται από:

- Ένα επεξεργαστικό στοιχείο (είτε ένα πυρήνα πολυπύρηνου επεξεργαστή, είτε μια κεντρική μονάδα επεξεργασίας), που ονομάζεται θωρακισμένος (shielded).
- Μηδέν ή περισσότερα διανύσματα διακοπών (IRQ) και αντιστοιχούν στις κρίσιμες συσκευές υλικού που διαχειρίζεται το διαμέρισμα.
- Μηδέν ή περισσότερες διαδικασίες πραγματικού χρόνου. Είναι δυνατό να μην υπάρχουν διεργασίες πραγματικού χρόνου στο διαμέρισμα. Αυτό συμβαίνει όταν ο αλγόριθμος πραγματικού χρόνου είναι κωδικοποιημένος μέσα σε ένα πρόγραμμα χειρισμού διακοπών. Τότε η συσκευή που διαχειρίζεται το διαμέρισμα πραγματικού χρόνου μέσω αυτού του διανύσματος διακοπών μπορεί να θεωρηθεί ότι διαθέτει την υπολογιστική δύναμη ενός πρότυπου επεξεργαστή.

Κάθε διαμέρισμα πραγματικού χρόνου προστατεύεται από κάθε εξωτερικό γεγονός ή δραστηριότητα που δεν ανήκουν στην ή στις διεργασίες πραγματικού χρόνου που εκτελούνται σε αυτό το διαμέρισμα. Έτσι δεν υπάρχει η δυνατότητα μια συμβατική διαδικασία να προγραμματιστεί ή μια συμβατική διακοπή να παραδοθεί σε ένα θωρακισμένο επεξεργαστή.



Εικόνα 4: Δομή Λειτουργικού Συστήματος ASMP-Linux

5.2.2 Χειρισμός Διεργασιών

Όσον αφορά την εκτέλεση των διεργασιών το ASMP-Linux ορίζει ότι: μια διεργασία που έχει ανατεθεί σε ένα διαμέρισμα πραγματικού χρόνου, μπορεί να εκτελεστεί μόνο στο συγκεκριμένο διαμέρισμα και επιπλέον κάποια διεργασία που δεν ανήκει σε ένα διαμέρισμα πραγματικού χρόνου, δεν μπορεί να εκτελεστεί σε αυτό.

Ένα διαμέρισμα πραγματικού χρόνου μπορεί να περιέχει μία ή περισσότερες διεργασίες (πραγματικού χρόνου). Για την χρονοδρομολόγηση μεταξύ αυτών χρησιμοποιείται ο κανονικός χρονοδρομολογητής του πυρήνα. Τότε η θέσπιση των προτεραιοτήτων για της διεργασίες είναι ευθύνη του προγραμματιστή. Ο αλγόριθμος του χρονοδρομολογητή του πυρήνα τροποποιείται όμως από το ASMP-Linux σε σημεία που αφορούν την κατανομή φορτίου μεταξύ των επεξεργαστικών στοιχείων ώστε οι θωρακισμένοι επεξεργαστές να αγνοούνται.

Υπάρχουν και μη-κρίσιμες διεργασίες σε ένα διαμέρισμα πραγματικού χρόνου λόγω του σχεδιασμού του πυρήνα του Linux. Κάποιες διεργασίες που ονομάζονται νήματα πυρήνα, εκτελούνται στην περιοχή του πυρήνα και σχετίζονται με καθήκοντα του συστήματος, όπως η διεργασία αναμονής (idle process) και τα στιγμιότυπα νημάτων πυρήνα όπως το ksoftirqd που βρίσκονται σε κάθε CPU του συστήματος. [6]

Το patch ASMP-Linux δεν είναι παρεμβατικό, επειδή ο πυρήνας του Linux ήδη παρέχει τις προδιαγραφές και τους μηχανισμούς (processor affinity) για να την επιλογή των διαδικασιών που μπορεί να εκτελέσει ο κάθε επεξεργαστής. Ο περιγραφέας κάθε διαδικασίας περιλαμβάνει ένα πεδίο

που ονομάζεται cpus_allowed. Αυτό είναι ένας χάρτης bit που δηλώνει ποιοι επεξεργαστές επιτρέπεται να εκτελέσουν την συγκεκριμένη διεργασία. [7] Το ASMP-Linux τροποποιεί τους χάρτες των real time διεργασιών θέτοντας τα bit μόνο των θωρακισμένων επεξεργαστών και τους χάρτες των απλών διεργασιών καθαρίζοντας τα bit των θωρακισμένων επεξεργαστών.

5.2.3 Χειρισμός Διακοπών

Οι διακοπές είναι από τις κυριότερες αιτίες δημιουργίας καθυστερήσεων στην εκτέλεση μιας διεργασίες γιατί προέρχονται από τις συσκευές που είναι συνδεδεμένες στο σύστημα με τρόπο απρόβλεπτο και ασύγχρονο. Το ASMP-Linux αναγκάζει τα ολοκληρωμένα κυκλώματα ελέγχου διακοπών να στέλνουν τις διακοπές μόνο σε αθωράκιστους επεξεργαστές, ενώ τα διανύσματα διακοπών που έχουν ανατεθεί σε ένα διαμέρισμα πραγματικού χρόνου, αποστέλλονται υποχρεωτικά στον επεξεργαστή εκείνου του διαμερίσματος.

Λόγω της ύπαρξης μηχανισμών επικοινωνίας μεταξύ των επεξεργαστών ενός συστήματος πολυεπεξεργασίας, όταν ορισμένοι επεξεργαστές χρησιμοποιούν διακοπές για την υλοποίηση της επικοινωνίας αυτής (Inter-Processor Interrupts-IPI), υπάρχει η πιθανότητα να απευθυνθούν στο διαμέρισμα πραγματικού χρόνου διακοπές που δεν έχουν ανατεθεί σε αυτό. Στο Linux, αυτός ο μηχανισμός διακοπών χρησιμοποιείται συχνά για τον συγχρονισμό και την κατανομή φορτίου μεταξύ των επεξεργαστών. Οι θωρακισμένοι επεξεργαστές δεν επηρεάζονται από αυτή την κατανομή.

Ακόμα, ορισμένοι επεξεργαστές διαθέτουν ένα ρολόι υλοποιημένο με λογισμικό (local timer) που ανά τακτά χρονικά διαστήματα προκαλεί αυτό-διακοπή. Ο τοπικός χρονιστής ενός διαμερίσματος είναι δυνατό να απενεργοποιηθεί μέσω των διεπαφών του ASMP-Linux, καθιστώντας όμως αδύνατο τον χρονοπρογραμματισμό των διεργασιών πραγματικού χρόνου στο διαμέρισμα αυτό.

5.2.4 Κληρονομικότητα Πραγματικού Χρόνου

Μια διεργασία σε ένα λειτουργικό σύστημα τύπου Unix μπορεί να χρησιμοποιήσει τις υπηρεσίες του πυρήνα με τη βοήθεια κλήσεων συστήματος. Το ASMP-Linux τροποποιεί τις ρουτίνες εξυπηρέτησης των ακόλουθων κλήσεων συστήματος που σχετίζονται με την δημιουργία και τον τερματισμό μιας διαδικασίας: `fork()`, `clone()` και `exit()`. Οι ρουτίνες εξυπηρέτησης αυτών των κλήσεων επηρεάζουν δομές δεδομένων που κάνει χρήση το ASMP-Linux που σχετίζονται με τους περιγραφείς μια διαδικασίας.

Μια διεργασία που εκτελείται σε ένα διαμέρισμα πραγματικού χρόνου κληρονομεί στο ASMP-Linux την ιδιότητα του πραγματικού χρόνου στις διεργασίες-παιδιά της, τα οποία διατηρούν την ιδιότητα αυτή ακόμα και όταν εκτελούνται μέσω της κλήσης συστήματος `exec()`. Αν η διεργασία-παιδί δεν είναι αναγκαίο να βρίσκεται στο διαμέρισμα πραγματικού χρόνου, μπορεί μέσω των διεπαφών του ASMP-Linux να μετακινηθεί στο διαμέρισμα συστήματος.

5.2.5 Διεπαφές Χρήστη

Το ASMP-Linux παρέχει τους μηχανισμούς διεπαφής του για την επιλογή των θωρακισμένων επεξεργαστών, των διεργασιών και των διανυσμάτων διακοπών που ανατίθενται στο κάθε διαμέρισμα, όπως επίσης και για την ρύθμιση του τοπικού χρονιστή του επεξεργαστή στο σύστημα αρχείων */proc*.

Συγκεκριμένα μέσα στο κατάλογο */proc*, δημιουργείται σε έναν τροποποιημένο πυρήνα (μετά την φόρτωση των απαραίτητων modules όπως θα συζητηθεί παρακάτω) ένας επιπλέον κατάλογος με την ονομασία asmp που περιέχει τόσους υποκαταλόγους όσα είναι τα επεξεργαστικά στοιχεία του συστήματος. Οι υποκατάλογοι ονομάζονται cpuN, όπου N ο αριθμός του επεξεργαστή με την αρίθμηση των επεξεργαστών να ξεκινάει από το μηδέν.

Σχηματίζεται δηλαδή για τον N επεξεργαστή ο κατάλογος */proc/asmp/cpuN* που περιέχει τα κάτωθι αρχεία:

- *shielded*

Η τιμή του είναι *'0'* ή *'1'*.Όταν είναι *'1'*, αυτό σημαίνει ότι ο συγκεκριμένος επεξεργαστής υλοποιεί ένα διαμέρισμα πραγματικού χρόνου. Για παράδειγμα στην Εικόνα 4, αυτός είναι ο επεξεργαστής cpu3, δηλαδή ο τέταρτος επεξεργαστής του συγκεκριμένου συστήματος. Όταν το αρχείο έχει την τιμη *'0'*, ο επεξεργαστής είναι μέρος του διαμερίσματος συστήματος.

- *pids*

Στο αρχείο αυτό γράφονται ο αριθμός ταυτότητας της διεργασίας (process id-pid), ή οι αριθμοί ταυτότητας των διεργασιών που έχουν ανατεθεί στο συγκεκριμένο διαμέρισμα πραγματικού χρόνου. Όταν ο επεξεργαστής ανήκει στο διαμέρισμα συστήματος, το αρχείο έχει την τιμη *'(unshielded)'*.

- *irqs*

Εδώ δηλώνονται ποιες διακοπές σχετίζονται με το διαμέρισμα πραγματικού χρόνου. Μετά την ετικέτα *'promoted:'* ακολουθούν οι αριθμοι αυτών των διακοπών που ανατέθηκαν στο διαμέρισμα ή χρησιμοποιούνται από κάποια διεργασία που έχει ανατεθεί στο διαμέρισμα αυτό. Μετά την ετικέτα *'moved:'* ακολουθούν οι αριθμοί των διακοπών που αφαιρέθηκαν από το διαμέρισμα αυτό επειδή δεν σχετίζονται με κάποια κρίσιμη διεργασία και έχουν μεταφερθεί στο διαμέρισμα συστήματος.

- *localtimer*

Στο αρχείο αυτό δίνεται η δυνατότητα καθορισμού του τοπικού χρονιστή του επεξεργαστεί. Όταν η τιμή είναι *'1'*, ο τοπικός χρονιστής χρησιμοποιείται κανονικά. Όταν η τιμή του αρχείου είναι *'0'*, ο τοπικός χρονιστής είναι απενεργοποιημένος. Η τιμή αυτή στην ουσία αντιπροσωπεύει πόσα “ticks” πρέπει να περάσουν μέχρι την έκδοση της διακοπής του τοπικού χρονιστή. Τα “ticks” αυτά είναι διακοπές του γενικού χρονιστή (global timer) που παράγονται από το προγραμματιζόμενο chip διαστημάτων χρονιστή (Programmable Interval Timer Chip-PITC).Αν η τιμή αυτή είναι για παράδειγμα *'10'*, η συχνότητα παραγωγής διακοπών από τον local timer είναι το 1/10 της συχνότητας παραγωγής διακοπών από τον global timer.

Υπάρχουν δύο τρόποι αλληλεπίδρασης του χρήστη με της διεπαφές του ASMP-Linux. Ο πρώτος είναι μέσω της κονσόλας του Linux χρησιμοποιώντας την εντολή `echo`. Αν για παράδειγμα θέλουμε να αναθέσουμε μια διεργασία με pid 1234 στον πρώτο επεξεργαστή (cpu0), μπορούμε να δώσουμε την εντολή:

- `echo 1234 > /proc/asmp/cpu0/pids`

Η εντολή αυτή αναθέτοντας την διεργασία, δημιουργεί ένα διαμέρισμα πραγματικού χρόνου στον επεξεργαστή cpu0 κάνοντας με αυτό τον τρόπο την τιμή του αρχείου *shielded '1'* ενώ κατά την διάρκεια εκτέλεσης τις διεργασίας μεταβάλλονται ακόμα οι τιμές στο αρχείο *irqs* όπως περιγράφηκε προηγουμένως.

Αντίστοιχα μπορούμε να αναθέσουμε μια ρουτίνα εξυπηρέτησης διακοπής σε ένα διαμέρισμα πραγματικού χρόνου ή να μεταβάλουμε τη λειτουργία του τοπικού χρονιστή μέσω της εντολής `echo`. Ακόμα μπορούμε να δημιουργήσουμε ένα διαμέρισμα πραγματικού χρόνου χωρίς να του αναθέσουμε διεργασίες γράφοντας την τιμη *'1'* στο αρχείο *shielded*.

Ο δεύτερος τρόπος είναι μέσω των κλήσεων συστήματος `open()` και `write()` μέσα στον κώδικα της διεργασίας. Στο προηγούμενο παράδειγμα θα μπορούσαμε να χρησιμοποιήσουμε το ακόλουθο κομμάτι κώδικα γλώσσας προγραμματισμού C για να αναθέσουμε μια διεργασία στον επεξεργαστή cpu0:

```
#include <linux/fcntl.h>

size_t fd, wr;

fd = open("/proc/asmp/cpu0/pids", O_WRONLY | O_APPEND);
if (fd == -1){
            perror("File error!");
            return EXIT_FAILURE;
            }

int *pid = (int *)getpid();

wr = write(fd, pid, sizeof(pid));
if(wr == -1){
            perror(“Write error!”);
            return EXIT_FAILURE;
            }

close(fd);
```

Το αρχείο επικεφαλίδας `fcntl.h` περιέχει ορισμούς απαραίτητους για την κλήση συστήματος `open()` και πρέπει να συμπεριλαμβάνεται σε περίπτωση χρήσης της.

Ο περιγραφέας αρχείου `fd` είναι ένας ακέραιος αριθμός, θετικός στην περίπτωση επιτυχούς ανοίγματος του αρχείου, ή *'-1'* σε περίπτωση σφάλματος. Η ετικέτα `O_WRONLY` δηλώνοει ότι το αρχείο ανοίγεται μόνο για εγγραφή και η ετικέτα `O_APPEND` δηλώνει διατήρηση των δεδομένων που ήδη υπάρχουν στο αρχείο και εγγραφή μετά από αυτά. Στη συνέχεια αποθηκεύεται ο αριθμός ταυτότητας της διεργασίας στη θέση που δείχνει ο δείκτης pid και γίνεται η εγγραφή αυτού στο αρχείο. Όπως και η `open()`, η `write()` επιστρέφει *'-1'* σε περίπτωση σφάλματος. Τέλος κλείνουμε τον περιγραφέα αρχείου για να γίνει το αρχείο διαθέσιμο.

Με τον ίδιο τρόπο θα μπορούσε μια διεργασία να μετατεθεί από το διαμέρισμα πραγματικού χρόνου σε αυτό του συστήματος απλά γράφοντας στο αρχείο που αντιστοιχεί στον επεξεργαστή που έχει ανατεθεί σαν τιμή το pid της με αρνητικό πρόσημο, ή να καταργώντας το διαμέρισμα πραγματικού χρόνου στο οποίο εκτελείται εάν δεν μας χρειάζεται πλέον, γράφοντας την τιμή *'0'* στο αρχείο *shielded* του αντίστοιχου επεξεργαστή.

Όπως φαίνεται, το ASMP-Linux παρέχει ένα περιβάλλον ευέλικτο, πλήρως προσαρμόσιμο κατά το χρόνο εκτέλεσης, όσον αφορά τα διαμερίσματα συστήματος και πραγματικού χρόνου αλλά και την ανάθεση και μετανάστευση των εργασιών ή και διακοπών από το ένα διαμέρισμα στο άλλο.

5.3 Εγκατάσταση του Λειτουργικού Συστήματος ASMP-Linux

Για την εγκατάσταση του ASMP-Linux είναι απαραίτητο να γίνει compile ένας “καθαρός” πυρήνας Linux, ή αλλιώς vanilla kernel [8] έκδοσης 2.6.19-1 με το patch ASMP-Linux που μπορεί να βρεθεί εδώ [1]. Με τον όρο “καθαρό” πυρήνα, εννοούμε πυρήνα που δεν προέρχεται από κάποια διανομή Linux. Η εγκατάσταση του λειτουργικού αυτού συστήματος έγινε στον εξυπηρετητή IBM x3455.

5.3.1 Χαρακτηριστικά του Εξυπηρετητή IBM x3455

Ο εξυπηρετητής x3455 διαθέτει τα ακόλουθα χαρακτηριστικά:

- Υλικό:
  - Μητρική κάρτα IBM 7948
  - Δύο επεξεργαστές AMD Opteron 2218 Dual-Core
  - Host Bridge AMD K8 [Athlon64/Opteron]
  - Μνήμη τυχαίας ανάγνωσης 2 GB
  - Γέφυρα PCI/PCI-X Broadcom HT1000
  - Γέφυρα PCI-Express Broadcom HT2100
  - Διεπαφή δικτύου Broadcom Ethernet NetXtreme BCM5704
  - Ελεγκτής διαύλου RAID Broadcom BCM5785 (HT1000) SATA Native SATA Mode

- Σκληρός δίσκος 80 GB Western Digital WD800ABJS-23
- Κάρτα γραφικών ATI ES1000

➢ Λογισμικό πριν την εγκατάσταση του ASMP-Linux:

Λειτουργικό σύστημα Red Hat Enterprise Linux ES Release 4 Nahant Update 2. Οι σημειώσεις έκδοσης του μπορούν να βρεθούν εδώ [9].

Αφού αποκτήσαμε τον πηγαίο κώδικα του πυρήνα Linux 2.6.19-1 που είναι ένα αρχείο της μορφής *linux-2.6.19.1.tar.gz* και το patch ASMP-Linux (*patch-2.6.19.1-asmp-linux.tar.gz*), έπρεπε να λάβουμε ακριβείς πληροφορίες για το υλικό του εξυπηρετητή ώστε να προκύψει ένας λειτουργικός πυρήνας από το compilation. Οι πληροφορίες αυτές παρουσιάζονται στην σελίδα 74 και προέκυψαν μέσω των ακόλουθων εργαλείων για λειτουργικό σύστημα Linux.

➢ `lshw`: Στα πλαίσια της διπλωματικής εργασίας χρειάστηκε να εγκατασταθεί το πρόγραμμα αυτό που τυπώνει λεπτομερείς πληροφορίες σχετικά με τη ρύθμιση παραμέτρων του υλικού του συστήματος. Μπορεί να αναφέρει ακριβή διάρθρωση μνήμης, έκδοση firmware, μοντέλο μητρικής κάρτας και επεξεργαστή, αλλά και την ταχύτητα του, το μέγεθος της μνήμης cache και την ταχύτητα διαύλου.

➢ `dmidecode`: Αναφέρει πληροφορίες παρόμοιες με το προηγούμενο πρόγραμμα, όπως αυτές καθορίζονται στο BIOS του συστήματος ανάλογα με το πρότυπο SMBIOS/DMI.

➢ `lspci`: Η εντολή αυτή τυπώνει πληροφορίες για όλες τις κάρτες διαύλου PCI του συστήματος.

➢ `cat /proc/cpuinfo`: Μέσω του συστήματος αρχείων του Linux μπορούμε να πάρουμε δυναμικά πληροφορίες για το ή τα στοιχεία επεξεργασίας του συστήματος, όπως τον αριθμό των πυρήνων, τα επίπεδα και το μέγεθος της κρυφής μνήμης κ.ο.κ.

➢ `lsmod`: Τυπώνονται πληροφορίες για τα modules που είναι φορτωμένα στον πυρήνα που είναι ήδη σε χρήση. Είναι βοηθητικές πληροφορίες ώστε να γνωρίζουμε κατά την διάρκεια ρύθμισης (configuration) του πυρήνα ποια στοιχεία να γίνουν compile ως modules.

5.3.2 Compilation ενός Πυρήνα με το patch ASMP-Linux

Συνοπτικά τα βήματα εγκατάστασης και οι αντίστοιχες εντολές που χρησιμοποιήθηκαν για το λειτουργικό σύστημα ASMP-Linux είναι τα ακόλουθα:

Αποσυμπίεση του πηγαίου κώδικα του πυρήνα:

◆ `tar –xvzf linux–2.6.19.1.tar.gz`

Αντιγραφή στο φάκελο που περιέχει τους πηγαίους κώδικες των πυρήνων στο Redhat Enterprise Linux 4 και προαιρετικά πρόσθεση του αναγνωριστικού asmp (ή οποιουδήποτε άλλου αναγνωριστικού):

◆ `cp –rl linux–2.6.19.1 /usr/src/kernels/linux–2.6.19.1–asmp`

Αποσυμπίεση του patch ASMP-Linux:

- `tar -xvzf  patch-2.6.19.1-asmp-linux.tar.gz`

Μεταφορά στο νέο κατάλογο:

- `cd /usr/src/kernels/linux-2.6.19.1-asmp`

Εφαρμογή του patch ASMP-Linux:

- `patch -p1 -i /<path to patch>/patch-2.6.19.1-asmp-linux`

Αντιγραφή του αρχείου ρύθμισης του σε χρήση πυρήνα:

- `cp /boot/config-<kernel in use> /usr/src/kernel/linux-2.6.19.1/.config`

Ρύθμιση του πυρήνα:

- `make menuconfig`

Εκεί επιλέγουμε 'Load an alternate configuration file.' και δίνουμε το όνομα '.config'. Στη συνέχεια επελέγησαν τα χαρακτηριστικά του πυρήνα που μας ενδιαφέρουν να είναι ενσωματωμένα στον πυρήνα (built-in) ή να φορτώνονται ως modules. Προσοχή χρειάστηκε στην επιλογή των χαρακτηριστικών που προσδίδει το patch ASMP-Linux και βρίσκονται στην κατηγορία 'Processor types and features':

- Symmetric Multi-processing support : επελέγη να είναι built-in.
- ASymmetric MultiProcessor (ASMP) : επελέγη να είναι built-in.
- Proc interface for ASMP : επελέγη να είναι module.
- Localtimer test for ASMP : επελέγη να είναι module.

Σημαντικό ήταν να επιλεγεί σαν module το 'kconfig' στην κατηγορία 'drivers' και μετά 'ata', το οποίο παρέχει την υποστήριξη για σκληρούς δίσκους τύπου SATA. Ακόμα, με την καθοδήγηση και βοήθεια της ομάδας που ανέπτυξε το ASMP-Linux [2] καθορίστηκαν και τα ακόλουθα χαρακτηριστικά.

- Συχνότητα ρολογιού HZ=1000 .
- Επιλογή Kernel Preemption (Low-Latency Desktop)
- Επιλογή Big Kernel Lock Preemptive
- Επιλογή IRQ use register argument
- Επιλογή Multicore support
- Επιλογή IRQ balancing
- Επιλογή Generic IRQ pending
- Όχι ACPI γιατί μπορεί να μεταβάλει τη συχνότητα ρολογιού.
- Όχι MCE (Machine Check Exception) γιατί αλλιώς αποστέλλεται περιοδικά IPI.

Compilation του πυρήνα και των modules:

- `make`
- `make modules`

Εγκατάσταση του πυρήνα και των modules, δημιουργία δηλαδή των αρχείων *vmlinuz-<kernel>*, *config-<kernel>*, *System.map-<kernel>* στον κατάλογο boot μετά την απόκτηση δικαιωμάτων superuser (εντολή `su`):

- `make modules_install`
- `make install`

Δημιουργία εικόνας initrd που χρειάζεται για τον boot-loader του συστήματος:

- `cd /boot`
- `mkinitrd –o initrd.img–2.6.19.1–asmp 2.6.19.1–asmp`

Για να εκκινήσει ο εξυπηρετητής με τον νέο πυρήνα προστέθηκε στο αρχείο του boot-loader */boot/grub/menu.lst* το ακόλουθο κομμάτι

```
 title ASMP-Linux (2.6.19.1–asmp)
      root (hd0,0)
      kernel /vmlinuz–2.6.19.1–asmp ro root=/dev/VolGroup00/LogVol00
acpi=off rhgb quiet
      initrd /initrd–2.6.19.1–asmp.img
```

και επελέγη σαν default ώστε μετά την επανεκκίνηση να τρέχει ο νέος προσαρμοσμένος πυρήνας με το λειτουργικό ASMP-Linux. Μπορεί να επιβεβαιωθεί ότι τρέχει ο νέος πυρήνας με την εντολή `uname`.

Για να ενεργοποιηθούν οι διεπαφές χρήστη του λειτουργικού πρέπει να φορτωθούν τα απαραίτητα modules με τις εντολές (μετά την εντολή `su`):

- `modprobe asmp_proc`
- `modprobe asmp_localtimer_test`

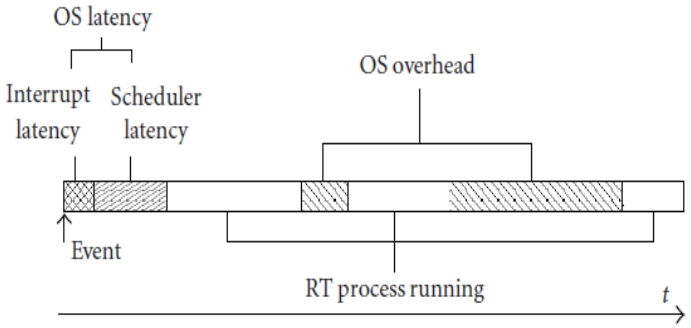
Με την εντολή `lsmod | grep asmp` επιβεβαιώνεται η επιτυχία φόρτωσης των modules.

5.4 Αξιολόγηση του Λειτουργικού Συστήματος ASMP-Linux

5.4.1 Εισαγωγή

Για τον έλεγχο της αποτελεσματικότητας του ASMP-Linux σε συνθήκες πραγματικού χρόνου, υλοποιήθηκε ένα πρόγραμμα σε γλώσσα προγραμματισμού C το οποίο μετράει τον πραγματικό χρόνο που κάνει `sleep()` μια διεργασία. Η θεωρητική διάρκεια της `sleep()` δηλώνεται σαν όρισμα της από τον χρήστη. Μεταξύ των δύο αυτών τιμών, τη θεωρητική και την πραγματική διάρκεια υπάρχει διαφορά που μελετήθηκε στατιστικά με διαγράμματα σε διάφορες συνθήκες στον εξυπηρετητή IBM x3455.

Η διαφορά αυτή, οφείλεται στον θόρυβο (jitter), τις καθυστερήσεις δηλαδή που προκαλούνται από το λειτουργικό σύστημα, τα στρώματα λογισμικού μεταξύ της συσκευής υπό έλεγχο και του προγράμματος που την ελέγχει, την ύπαρξη άλλων συσκευών που ζητούν και αυτά να τρέξουν σε πραγματικό χρόνο και σε παράγοντες όπως το υλικό του συστήματος. Η Εικόνα 5 [3] δείχνει σχηματικά τις αιτίες θορύβου σε μια διεργασία πραγματικού χρόνου.

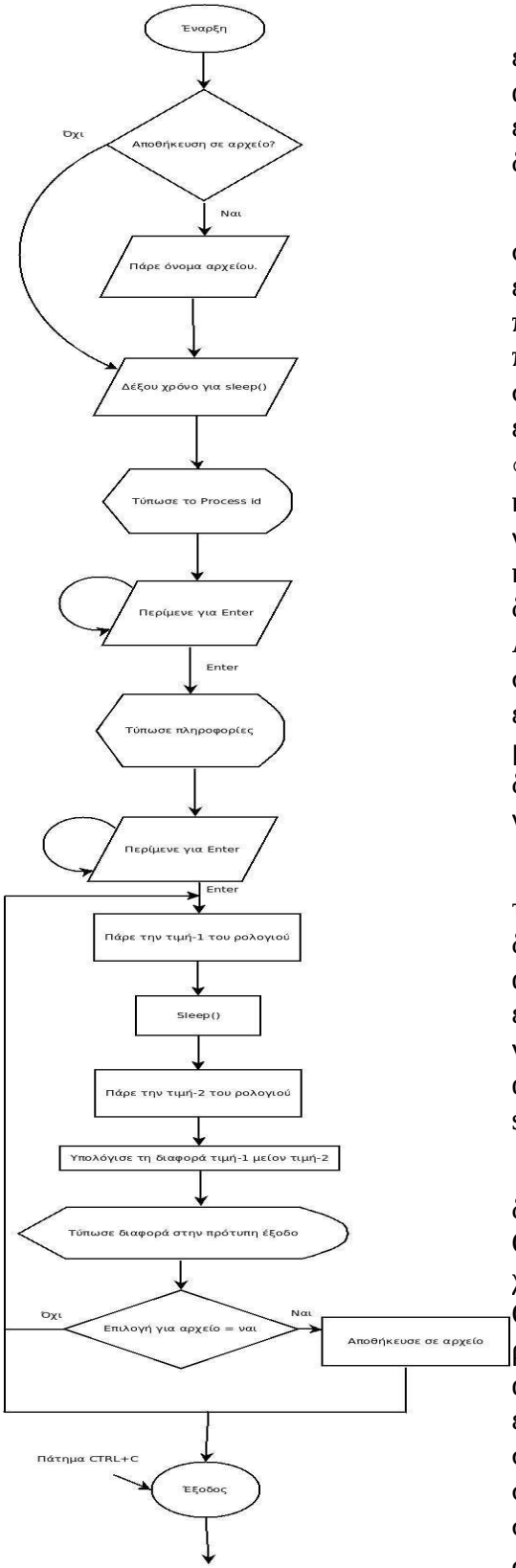


Εικόνα 5: Σχηματική Απεικόνιση των Αιτιών Θορύβου

Αυτές διαχωρίζονται σύμφωνα με το [5] σε δύο σκέλη, την καθυστέρηση και την επιβάρυνση του λειτουργικού συστήματος. Η καθυστέρηση του λειτουργικού συστήματος, είναι ο χρόνος που περνάει από τη στιγμή που συμβαίνει ένα γεγονός που απαιτεί εξυπηρέτηση μέχρι το χρόνο που του παρέχεται η εξυπηρέτηση αυτή. Ως επιβάρυνση λειτουργικού συστήματος ορίζεται ο χρόνος που ο επεξεργαστής ενός συστήματος ξοδεύει εκτελώντας κώδικα συστήματος, όπως για παράδειγμα το να διαχειρίζεται τη μνήμη.

5.4.2 Πρόγραμμα Εφαρμογής Ελέγχου

Το πρόγραμμα ελέγχου ακολουθεί το διάγραμμα ροής που παρουσιάζεται στο Διάγραμμα 1 στην επόμενη σελίδα.



Διάγραμμα 1: Διάγραμμα Ροής Προγράμματος Ελέγχου

Το πρόγραμμα ουσιαστικά ζητάει από το χρήστη την εισαγωγή του ονόματος του αρχείου που θα αποθηκευτούν τα αποτελέσματα, εφόσον αυτό είναι επιθυμητό και το χρόνο που είναι επιθυμητό να γίνει sleep σε δευτερόλεπτα και νάνο-δευτερόλεπτα.

Στη συνέχεια τυπώνει το process id (pid) και δίνει το χρόνο στο χρήστη, εάν επιθυμεί, να αναθέσει τη διεργασία αυτή σε έναν επεξεργαστή και κατά συνέπεια σε ένα διαμέρισμα πραγματικού χρόνου. Αυτό δεν συμβαίνει μέσω του προγράμματος αλλά μέσω νέας κονσόλας. Η επιλογή αυτή στο σχεδιασμό έγινε για δύο λόγους. Ο ένας λόγος είναι η ελαχιστοποίηση της χρήσης κλήσεων συστήματος όπως η `open()` και `write()` μέσα στο πρόγραμμα για την αποφυγή καθυστερήσεων. Ο δεύτερος λόγος έχει να κάνει με το γεγονός ότι για να γράψει ο χρήστης σε ένα αρχείο στον κατάλογο */proc*, άρα και για να αναθέσει κάποιος μια διεργασία σε έναν επεξεργαστή μέσω των διεπαφών του ASMP-Linux, χρειάζεται δικαιώματα superuser. Εάν αυτό συνέβαινε μέσω του προγράμματος, αυτό θα μπορούσε να εκτελεστεί μόνο από κάποιον με δικαιώματα superuser, ενώ με τη σχεδιαστική επιλογή που έγινε, ο χρήστης χρειάζεται τα δικαιώματα αυτά μόνο για την ανάθεση της διεργασίας και όχι για την εκτέλεση του προγράμματος.

Έπειτα αφού τυπώσει πληροφορίες για τους επεξεργαστές του συστήματος, δηλαδή το αν αυτοί λειτουργούν σε διαμέρισμα συστήματος ή σε πραγματικού χρόνου και τους αριθμούς των διεργασιών που έχουν ανατεθεί στον κάθε επεξεργαστή, εάν αυτό συμβαίνει, περιμένει από τον χρήστη να πατήσει το πλήκτρο αλλαγής γραμμής (Enter) για να αρχίσει η μέτρηση του πραγματικού χρόνου που διαρκεί η sleep της διεργασίας.

Η μέτρηση αυτή συμβαίνει μέσα σε ένα while loop που διακόπτεται από τον χρήστη με τη χρήση των πλήκτρων CTRL+C. Ακριβώς πριν την sleep αποθηκεύεται η τιμή του χρόνου που δείχνει το συμβατό με το πρότυπο POSIX ρολόι CLOCK_PROCESS_CPUTIME_ID. Το ρολόι αυτό, είναι ένα ρολόι υψηλής ανάλυσης (high resolution) που είναι αποκλειστικό για όλα τα νήματα της διεργασίας που το επικαλείται και εκκινεί ξεχωριστά σε κάθε επεξεργαστή του συστήματος. Αντιπροσωπεύει τον CPU-χρόνο (χρήστη και συστήματος) της καλούσας διεργασίας μέσω των συναρτήσεων `clock_*()` ή `timer_*()` (ο αστερίσκος σημαίνει όλες τις συναρτήσεις με το πρόθεμα “clock_” ή “timer_”), δηλαδή το ποσοστό του χρόνου εκτέλεσης της διεργασίας που συνδέεται με το ρολόι. [10], [11]

Για την αποθήκευση της τιμής αυτής χρησιμοποιήθηκε η συνάρτηση `clock_gettime()` [12] η οποία ορίζεται ως:

- `int clock_gettime(clockid_t clk_id, struct timespec *tp);`

και είναι συμβατή με POSIX. Η συνάρτηση παίρνει την τιμή του ρολογιού που ορίζεται από το `clk_id` και την αποθηκεύει σε μια δομή τύπου `timespec`:

- ```
  struct timespec {
              time_t   tv_sec;        /* seconds */
              long     tv_nsec;       /* nanoseconds */
              };
  ```

Καλείται στη συνέχεια η συνάρτηση `clock_nanosleep()`[13] για να κάνει sleep η διεργασία, η οποία είναι και αυτή υψηλής ανάλυσης και συμβατή με το πρότυπο POSIX και ορίζεται ως:

- ```
  int clock_nanosleep(clockid_t clock_id, int flags, const struct
                              timespec *rqtp, struct timespec *rmtp);
  ```

Στην `clock_nanosleep()` το `clock_id` δεν μπορεί να είναι το CLOCK_PROCESS_CPUTIME_ID, επομένως έγινε χρήση του CLOCK_REALTIME που είναι το πρότυπο POSIX ρολόι για πραγματικό χρόνο. Το πεδίο `flags` μπορεί να είναι είτε TIMER_ABSTIME είτε να έχει την τιμή μηδέν. Όταν η τιμή του πεδίου είναι μηδέν, η συνάρτηση πρέπει προκαλεί την αναστολή της εκτέλεσης της διεργασίας που την κάλεσε μέχρι είτε το χρονικό διάστημα που ορίζεται στο πεδίο `rqtp` να παρέλθει, ή μέχρι η διεργασία να διακοπεί από ένα σήμα ή να τερματιστεί. Αν η τιμή του πεδίου είναι TIMER_ABSTIME, η συνάρτηση προκαλεί την αναστολή της εκτέλεσης της διεργασίας έως ότου είτε η χρονική τιμή του ρολογιού που καθορίζεται από το `clock_id` φτάσει τον απόλυτο χρόνο που ορίζεται από το επιχείρημα `rqtp`, ή μέχρι η διεργασία να διακοπεί από ένα σήμα ή να τερματιστεί. Εάν, κατά την ώρα της κλήσης, η αξία που καθορίζεται από το πεδίο `rqtp` είναι μικρότερη ή ίση με την αξία του χρόνου του συγκεκριμένου ρολογιού, τότε η συνάρτηση επιστρέφει αμέσως και η διεργασία δεν αναστέλλεται. Στην υλοποίηση του προγράμματος επιλέχθηκε η τιμή του πεδίου να είναι TIMER_ABSTIME για να υπάρχει απόλυτος έλεγχος και ακρίβεια στις μετρήσεις. Το πεδίο `rmtp` δείχνει σε περίπτωση διακοπής της sleep τον χρόνο που υπολειπόταν για αναστολή τη στιγμή της διακοπής. Στο πρόγραμμα προστίθεται με την συνάρτηση `timespec_add()` στην τιμή του CLOCK_REALTIME πριν την sleep η ορισμένη από το χρήστη καθυστέρηση ώστε η διεργασία να ξυπνήσει ακριβώς μετά τη στιγμή 'χρόνος τώρα+καθυστέρηση'.

Τέλος, αμέσως μετά τη `clock_nanosleep()` καλείται η `clock_gettime()` και λαμβάνεται η τιμή του CLOCK_PROCESS_CPUTIME_ID. Η προηγούμενη τιμή του αφαιρείται από αυτή με την συνάρτηση `timepec_diff_micro()` που δίνει το αποτέλεσμα σε μικρο-δευτερόλεπτα και έτσι προκύπτει ο πραγματικός χρόνος χρήστη και συστήματος που διήρκεσε η sleep.

Το πρόγραμμα στη συνέχεια τυπώνει στην οθόνη το χρόνο που προκύπτει απο την αφαίρεση των δύο τιμών και εάν έχει επιλεγεί από το χρήστη γράφει το αποτέλεσμα στο ορισμένο από το χρήστη αρχείο. Ο κώδικας του προγράμματος ακολουθεί.

```
/*ΑΡΧΗ ΠΡΟΓΡΑΜΜΑΤΟΣ-----------------------------------------------------------------*/
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>
#include <errno.h>

/*Δήλωση συναρτήσεων-----------------------------------------------------------------*/

void acquire_data(void);
int display_information(void);
double timespec_diff_micro (const struct timespec *, const struct timespec *);
struct timespec timespec_add (const struct timespec *, const struct timespec *);
int print_to_file (double, char *);
void stdin_flush (void);

/*Δήλωση Global Μεταβλητών-----------------------------------------------------------*/

int delay_sec=0, delay_nsec=0;
char filename[20], choice;

/*-----------------------------------------------------------------------------------*/
/*Συνάρτηση main()-------------------------------------------------------------------*/
int main (void)
{
	//Δήλωση και αρχικοποίηση μεταβλητών-----------------------------------------
	int result=0, buf=0, i=0;
	double difference=0;
	struct timespec interval, remainder, start, end, next, delay;
	acquire_data();
```

```
      if (display_information()==EXIT_FAILURE){
                        printf("\nERROR: Please press Enter next time!\n");
                        return EXIT_FAILURE;
                        }
      delay.tv_sec =  delay_sec;
      delay.tv_nsec = delay_nsec;
      while(1){
/*Παίρνουμε την τιμή του CLOCK_REALTIME ώστε να της προσθέσουμε το χρόνο που θα κάνει
sleep η διεργασία.*/
                    clock_gettime(CLOCK_REALTIME, &next);
/*Υπολογίζεται ο απόλυτος χρόνος που θα ξυπνήσει η διεργασία*/
                    interval = timespec_add(&next, &delay);
/*Παίρνουμε την τιμή του CLOCK_PROCESS_CPUTIME_ID πριν ξεκινήσει η αναστολή εκτέλεσης της
διεργασίας και την αποθηκεύουμε στη δομή start.*/
                    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);
                    if (clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &interval,
                                        &remainder) == -1){
                          if (errno == EINTR) {
                                printf("ERROR: Nanosleep interrupted\n");
                                printf("Remaining secs: %u\n", remainder.tv_sec);
                                printf("Remaining nsecs: %ld\n", remainder.tv_nsec);
                                }
                                else perror("ERROR in nanosleep!");
                          }
/*Παίρνουμε την τιμή του CLOCK_PROCESS_CPUTIME_ID όταν ξυπνήσει η διεργασία και την
αποθηκεύουμε στη δομή end.*/
                    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
/*Υπολογίζουμε τη διαφορά.*/
                    difference = timespec_diff_micro(&start,&end);
/*Τυπώνουμε τη διαφορά στην οθόνη.*/
                    printf("%d\t%lf\tmicroseconds\n",i++, difference);
/*Αποθηκεύουμε εφόσον έχει επιλεγεί την διαφορά σε αρχείο.*/
                    if(choice == 'y') print_to_file(difference, filename);
                    }
```

```
	return EXIT_SUCCESS;
}

/*Τέλος main()-------------------------------------------------------------------------*/

/*Συνάρτηση acquire_data()-------------------------------------------------------*/

/*Παίρνει από το χρήστη την επιθυμητή διάρκεια sleep και το όνομα του αρχείου αποθήκευσης εφόσον ο χρήστης επιλέξει την αποθήκευση των αποτελεσμάτων σε αρχείο.*/

void acquire_data(void)

{

	printf("\n-----------PROGRAM TO MEASURE TRUE NANOSLEEP DELAY--------------\n");

	printf("\nDo you wish to log results in a file? type y (yes) or n (no):\t");

	 scanf("%c", &choice);

	 if (choice == 'y'){

			printf("Please type the desired filename (maximum 20 characters):\t");

			scanf("%20s",filename);

			}

     printf("For how long would you like process to sleep? (specify integer number)\n");


	  printf("Seconds:\t");

	  scanf("%d", &delay_sec);

	  printf("Nanoseconds:\t");

	  scanf("%d", &delay_nsec);

}

/*Συνάρτηση display_information()-----------------------------------------------------*/

/*Προσφέρει πληροφορίες για τις διεπαφές του ASMP-Linux και δίνει το χρόνο στον χρήστη να αναθέσει μια διεργασία σε έναν επεξεργαστή δηλαδή σε ένα διαμέρισμα πραγματικού χρόνου.*/

int display_information (void)

{

	int c=0;


	printf("\n---------------------------------INFO-----------------------------\n");

	printf("\nProcess id is %d\n", (int)getpid());	//Τυπώνει το pid

	printf("\nTime to assign process to cpu via ASMP-Linux interface..\n");
	printf("Please press Enter to show cpu info!\n");

	stdin_flush();
```

```
	if ((c=getchar())!='\n'){                          //Περιμένει για Enter
                                printf("Waiting for Enter...\n");
                                return EXIT_FAILURE;
                                }
	printf("\nCpu status: 1 implies shielded, 0 implies unshielded.\n");
	printf("\nCpu0 is: \n");
	system("cat /proc/asmp/cpu0/shielded");
	printf("Cpu0 has the followind pid(s) assigned: \n");
	system("cat /proc/asmp/cpu0/pids");
	printf("\nCpu1 is: \n");
	system("cat /proc/asmp/cpu1/shielded");
	printf("Cpu1 has the followind pid(s) assigned: \n");
	system("cat /proc/asmp/cpu1/pids");
	printf("\nCpu2 is: \n");
	system("cat /proc/asmp/cpu2/shielded");
	printf("Cpu2 has the followind pid(s) assigned: \n");
	system("cat /proc/asmp/cpu2/pids");
	printf("\nCpu3 is: \n");
	system("cat /proc/asmp/cpu3/shielded");
	printf("Cpu3 has the followind pid(s) assigned: \n");
	system("cat /proc/asmp/cpu3/pids");

	printf("\n-------------------------END OF INFO-----------------------------\n");
	printf("\nPress Enter again to take measurements!\n");
	stdin_flush();
	 if ((c=getchar())!='\n') {
                                printf("Waiting for Enter...\n");
                                return EXIT_FAILURE;
                                }
	return EXIT_SUCCESS;
}
/*Τέλος display_information()-------------------------------------------------------------*/
```

```
/*Συνάρτηση timespec_diff_micro()-----------------------------------------------------*/
/*Υπολογίζει συνολικά τη διαφορά σε μsec των δευτερολέπτων και nsec δύο δομών τύπου timespec.*/
double timespec_diff_micro (const struct timespec *start, const struct timespec *end)
{
	return (double)(end->tv_sec - start->tv_sec)*1000000 + (double)(end->tv_nsec -
                                                    start ->tv_nsec)/1000.0;
}
/*Τέλος timespec_diff_micro()---------------------------------------------------------*/

/*Συνάρτηση timespec_add()------------------------------------------------------------*/
/*Υπολογίζει το άθροισμα των sec και nsec δύο δομών τύπου timespec.*/
struct timespec timespec_add (const struct timespec *time1, const struct timespec *time2)
{
	struct timespec result;
	result.tv_sec = time1->tv_sec + time2->tv_sec ;
	result.tv_nsec = time1->tv_nsec + time2->tv_nsec ;
	if (result.tv_nsec >= 1000000000L){                    //Υπολογίζει carry
                                    result.tv_sec++ ;
                                    result.tv_nsec = result.tv_nsec - 1000000000L ;
                                    }
	return (result) ;
}
/*Τέλος timespec_add()----------------------------------------------------------------*/

/*Συνάρτηση print_to_file()-----------------------------------------------------------*/
/*Τυπώνει σε αρχείο ονόματος name ένα αριθμό τύπου double.*/
int print_to_file (double value, char *name)
{
	FILE *fp = NULL;
	fp = fopen (name, "a");
```

```
	if(fp == NULL){
			perror("ERROR while opening log file!");
			return EXIT_FAILURE;
			}
	fprintf(fp, "%lf\n", value);
	fclose(fp);
	return EXIT_SUCCESS;
}
/*Τέλος print_to_file()-----------------------------------------------------------------*/

/*Συνάρτηση stdin_flush()---------------------------------------------------------------*/
/*Καθαρίζει τον προσωρινό χώρο αποθήκευσης πρότυπης εισόδου.*/
void stdin_flush (void)
{
	int ch=0;
	do
		ch = getchar();
	while (ch != EOF && ch != '\n');
	clearerr(stdin);
}
/*Τέλος stdin_flush()-------------------------------------------------------------------*/
/*ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ--------------------------------------------------------------------*/
```

Για να γίνει compile το πρόγραμμα, απαραίτητη είναι η χρήση των βιβλιοθηκών πραγματικού χρόνου με χρήση της επιλογής -lrt στον gcc.

5.4.3 Μελέτη και Σχολιασμός της Πραγματικής Διάρκειας sleep

Το πρόγραμμα εφαρμόστηκε στο λειτουργικό ASMP-Linux στο διαμέρισμα συστήματος και πραγματικού χρόνου, για sleep χρονικού διαστήματος 1 msec και 100 msec, χωρίς φορτίο και με φορτίο. Ακόμα εφαρμόστηκε χωρίς το patch ASMP-Linux στον πυρήνα 2.6.9-22 EL του Redhat Enterprise Linux. Η sleep του 1 msec μετρήθηκε για διάστημα πέντε λεπτών, και αυτή των 100 msec για διάστημα δέκα λεπτών. Τα αποτελέσματα αποθηκεύτηκαν σε αρχείο και με βάση αυτά προέκυψαν οι ακόλουθες εικόνες που τοποθετούνται ανά δύο ανάλογα με τη διάρκεια sleep και το φορτίο, ένα για το διαμέρισμα συστήματος και ένα για το διαμέρισμα πραγματικού χρόνου. Σαν φορτίο κατά την αξιολόγηση χρησιμοποιήθηκαν τα εξής: αναπαραγωγή βίντεο, ping μιας IP, χρήση πλοηγού διαδικτύου και απλή χρήση του συστήματος, όπως επεξεργασία κειμένου κλπ.

Υπολογίστηκαν η μέση τιμή (MT), η τυπική απόκλιση (ΤΑ), και η μέση απόκλιση (ΜΑ) των μετρήσεων. Ακόμα, υπολογίστηκε η μέση τιμή της καθυστέρησης (MTK), δηλαδή κατά μέσο όρο πόσο καθυστερεί να ξυπνήσει η διεργασία από τη θεωρητική τιμή της sleep. Η μέση τιμή υπολογίζεται από το άθροισμα των τιμών και την διαίρεση με τον αριθμό των δειγμάτων. Ο Τύπος 1 εκφράζει την μέση τιμή ενός πλήθους δειγμάτων.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{n}(x_1 + \cdots + x_n)$$

Τύπος 1: Μέση Τιμή (MT)

Ο Τύπος 2 εκφράζει την τυπική απόκλιση ενός πλήθους δειγμάτων. Δείχνει πόσο μεγάλη διακύμανση υπάρχει από τη μέση τιμή. Μια χαμηλή τυπική απόκλιση υποδηλώνει τα δεδομένα τείνουν να είναι πολύ κοντά στη μέση, ενώ υψηλή τυπική απόκλιση σημαίνει ότι τα δεδομένα είναι απλωμένα σε ένα μεγάλο εύρος τιμών.

$$s_N = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}.$$

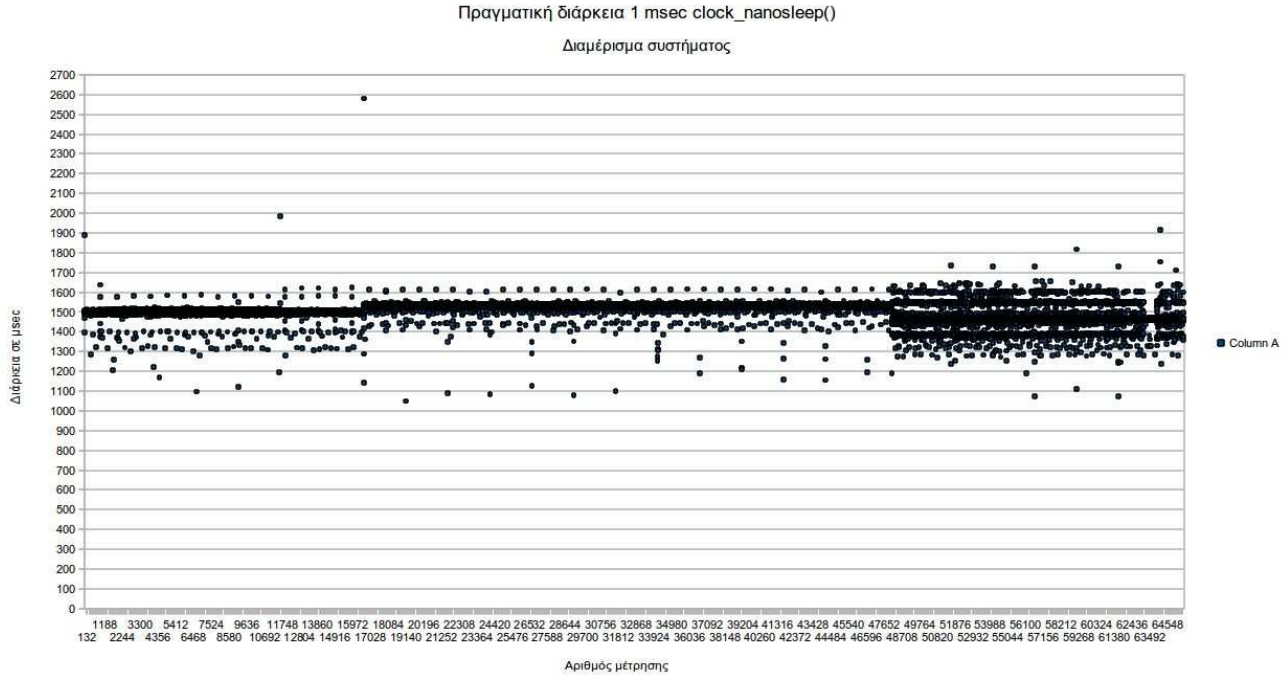
Τύπος 2: Τυπική Απόκλιση (ΤΑ)

Η μέση απόκλιση ή η μέση απόλυτη απόκλιση (Τύπος 3) είναι ο μέσος όρος των απόλυτων αποκλίσεων των επιμέρους τιμών από τη μέση τιμή των δειγμάτων. Μικρή τιμή της μέσης απόκλισης δηλώνει ότι τα αποτελέσματα βρίσκονται κοντά στη μέση τιμή, ενώ μεγάλη τιμή ότι είναι πιο διεσπαρμένα.

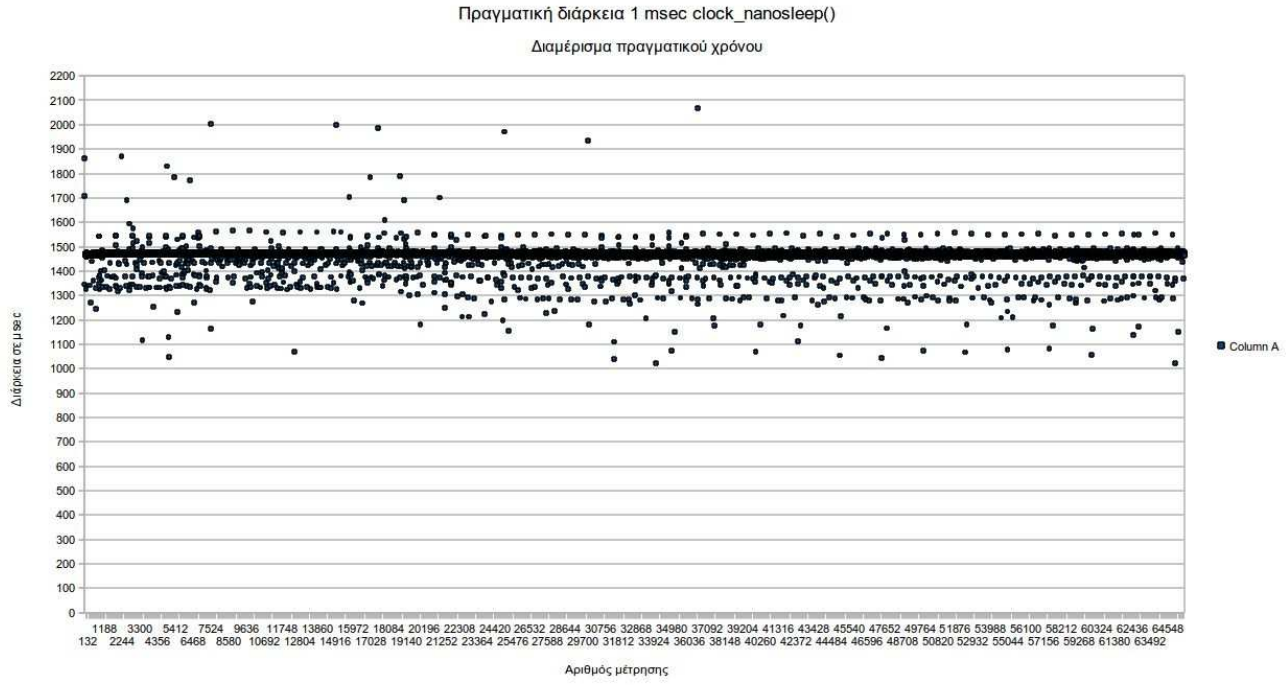
$$\frac{1}{n}\sum_{i=1}^{n} |x_i - m(X)|.$$

Τύπος 3: Μέση Απόκλιση (ΜΑ)

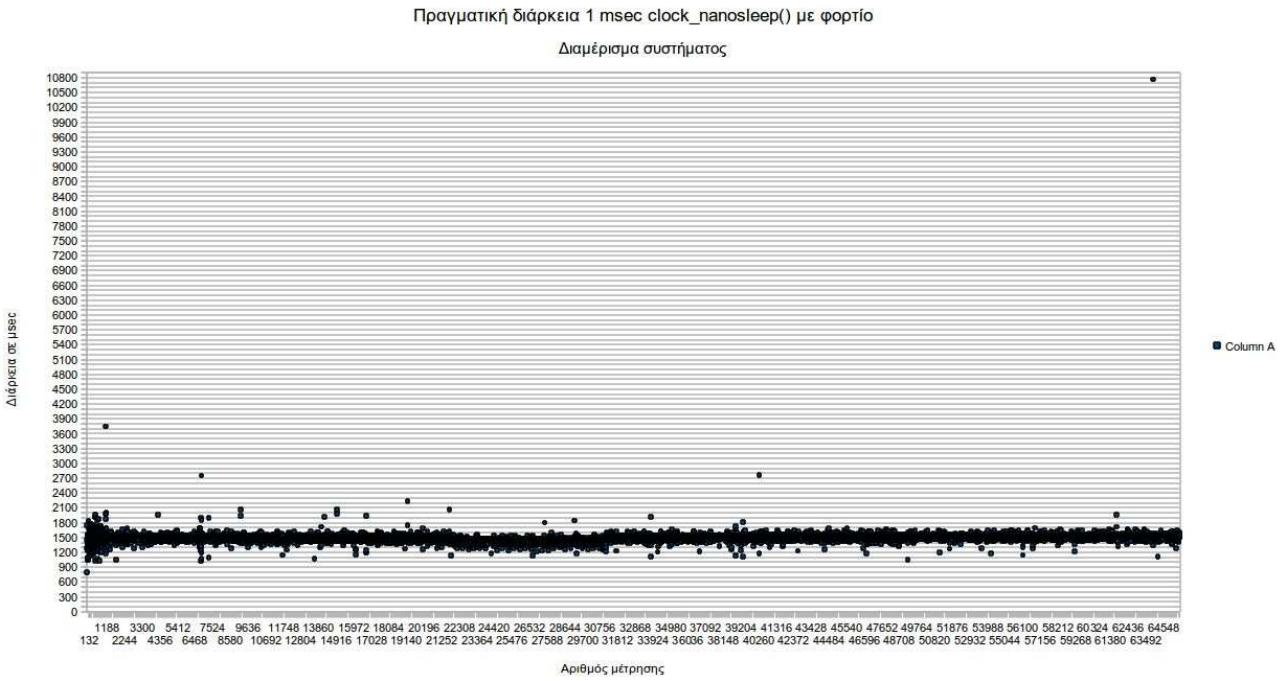
Η μέση τιμή της καθυστέρησης υπολογίζεται αφαιρώντας τη θεωρητική τιμή της sleep από τη μέση τιμή του πλήθους των δειγμάτων. Οι τιμές αυτές παρουσιάζονται σε πίνακες κάτω από τις αντίστοιχες εικόνες.



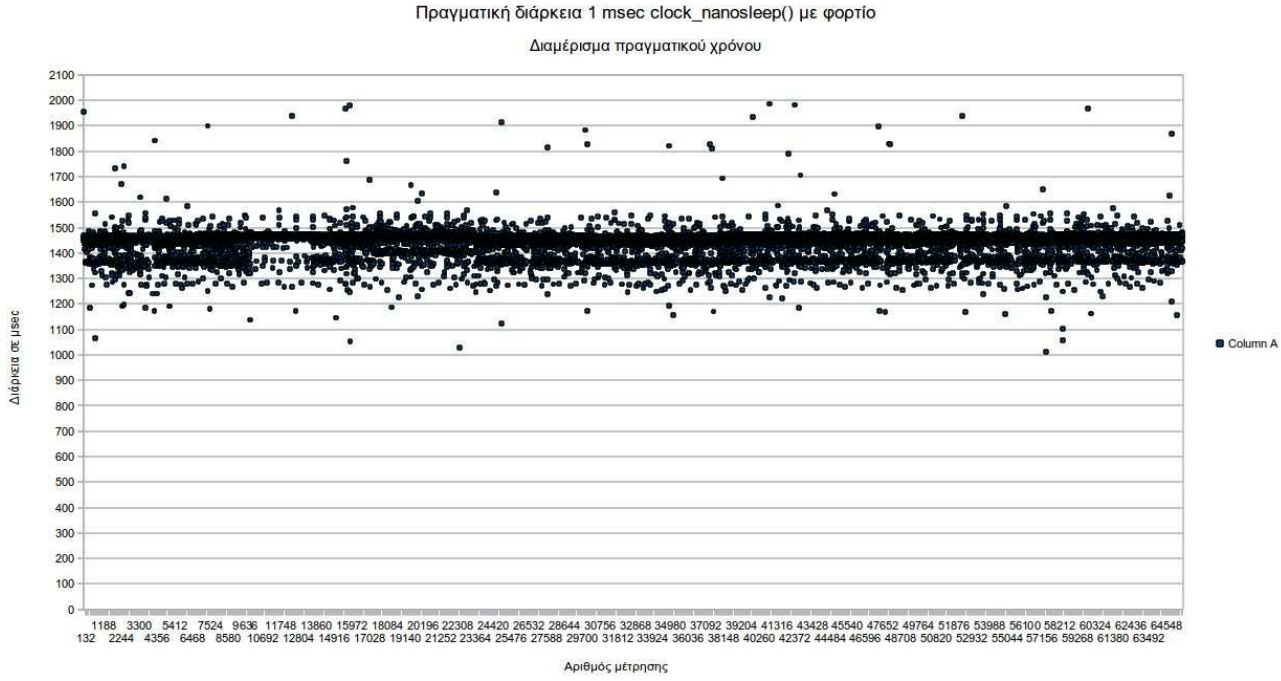
Εικόνα 6: 1 ms sleep, Διαμέρισμα Συστήματος, Χωρίς Φορτίο



Εικόνα 7: 1 ms sleep, Διαμέρισμα Πραγματικού Χρόνου, Χωρίς Φορτίο



Εικόνα 8: 1 ms sleep, Διαμέρισμα Συστήματος, Με Φορτίο



Εικόνα 9: 1 ms sleep, Διαμέρισμα Πραγματικού Χρόνου, Με Φορτίο

Πίνακας 1

1 msec Sleep ASMP-Linux	Διαμέρισμα Συστήματος				Διαμέρισμα Πραγματικού Χρόνου			
	ΜΤ	ΜΑ	ΤΑ	ΜΤΚ	ΜΤ	ΜΑ	ΤΑ	ΜΤΚ
Χωρίς Φορτίο	1507.42	28.62	36.31	507.42	1471.19	4.23	16.4	471.19
Με Φορτίο	1530.37	29.88	56.27	530.37	1458.05	12.47	25.83	458.05

Στα διαγράμματα στις Εικόνα 6 και Εικόνα 7 βλέπουμε την πραγματική διάρκεια sleep για 1 msec χωρίς φορτίο, στα διαμερίσματα συστήματος και πραγματικού χρόνου.

Στα διαγράμματα στις Εικόνα 8 και Εικόνα 9 βλέπουμε την πραγματική διάρκεια sleep για 1 msec με φορτίο, στα διαμερίσματα συστήματος και πραγματικού χρόνου.

Στον Πίνακα 1 βλέπουμε συγκεντρωτικά τις τιμές: μέση τιμή, μέση απόκλιση, τυπική απόκλιση και μέση τιμή καθυστέρησης για τα παραπάνω αποτελέσματα.

Το 1 msec είναι και η μικρότερη δυνατή διάρκεια που μια διεργασία μπορεί να κάνει sleep, αφού θέσαμε κατά την παραμετροποίηση του πυρήνα τη συχνότητα του timer 1000 Hz.

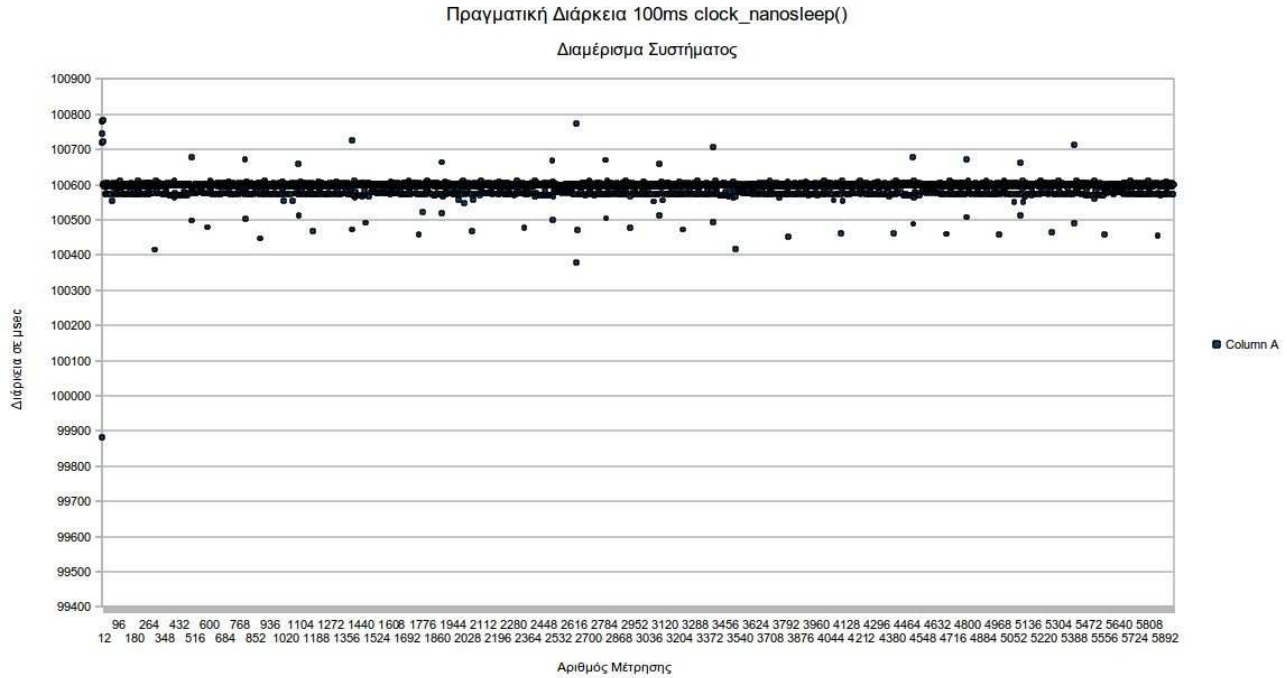
Φαίνεται ότι για την sleep χωρίς φορτίο στο διαμέρισμα συστήματος τα αποτελέσματα είναι λιγότερο συγκεντρωμένα γύρω από μια τιμή απ'ότι στο διαμέρισμα πραγματικού χρόνου. Βλέπουμε ότι με φορτίο η διασπορά των αποτελεσμάτων είναι μεγαλύτερη απ'ότι χωρίς φορτίο, κάτι που είναι λογικό δεδομένου ότι οι επεξεργαστές του συστήματος και το λειτουργικό σύστημα αναλαμβάνου παράλληλα και άλλες διεργασίες.

Η sleep του 1 msec διαρκεί στην πραγματικότητα (μέση τιμή) 1507.42 μsec χωρίς φορτίο και 1530.37 μsec με φορτίο στο διαμέρισμα συστήματος, με μια μέση τιμή καθυστέρησης της τάξεως του 0.5 msec και αρκετά μικρή διασπορά. Στο διαμέρισμα πραγματικού χρόνου, βλέπουμε τη διασπορά να μειώνεται αρκετά κατά 20-30 μsec (λαμβάνουμε υπ'όψιν την τυπική απόκλιση σαν πιο αξιόπιστη), ενώ η πραγματική διάρκεια είναι κατά μέση τιμή 100433.99 μsec χωρίς φορτίο και 100429.2 μsec με φορτίο. Η καθυστέρηση μειώνεται στην τάξη του 0.4 msec.

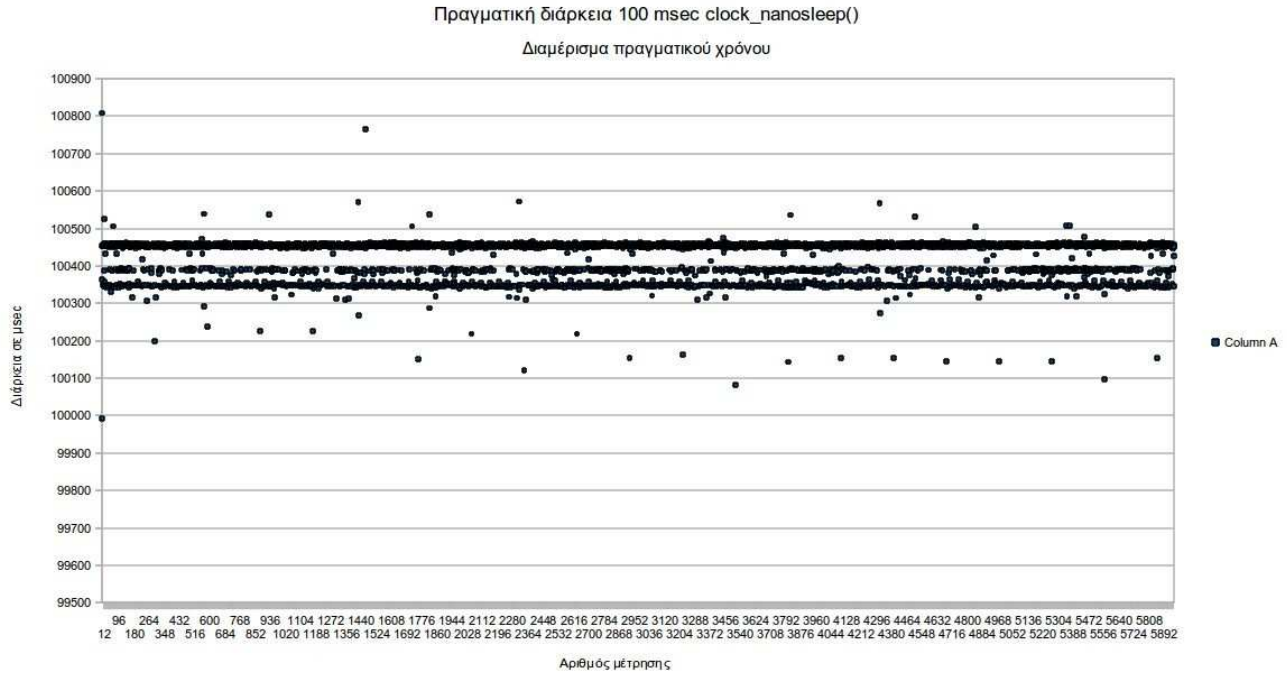
Στη συνέχεια ακολουθούν τα γραφήματα για την sleep των 100 msec.

Στα διαγράμματα στις Εικόνα 10 και Εικόνα 11, βλέπουμε την πραγματική διάρκεια της sleep των 100 msec στο διαμέρισμα συστήματος και στο διαμέρισμα πραγματικού χρόνου χωρίς φορτίο.

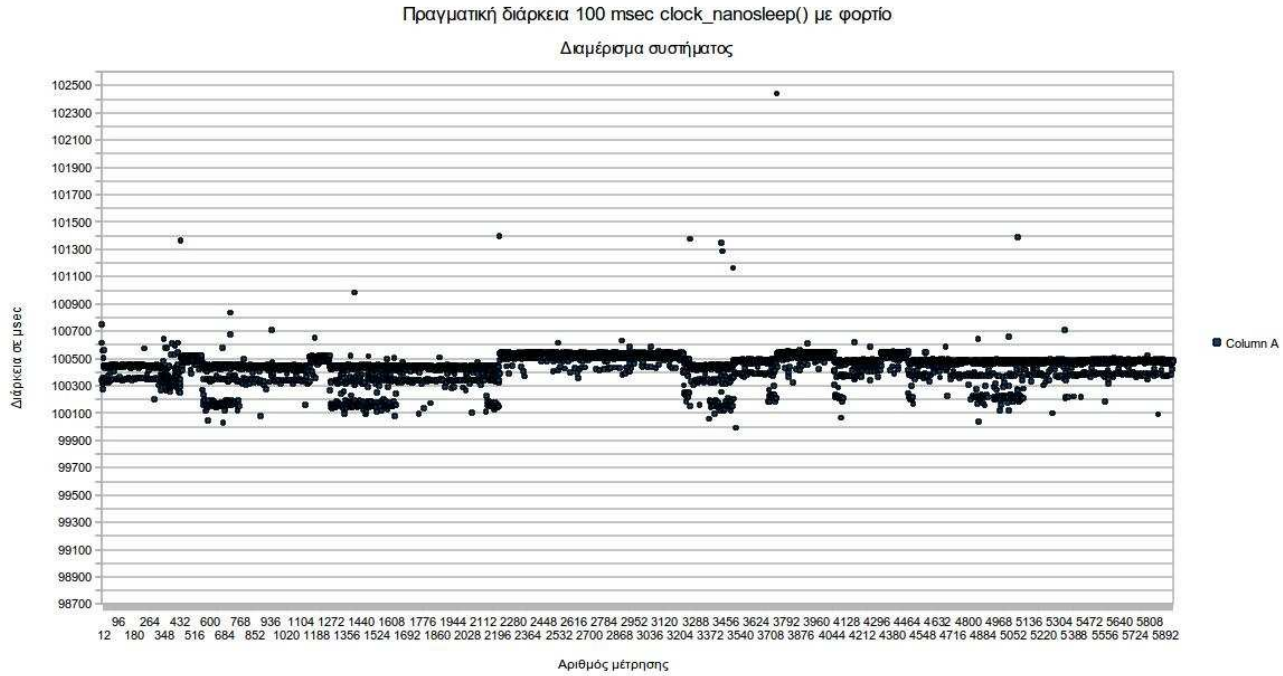
Στα διαγράμματα στις Εικόνα 12 και Εικόνα 13, βλέπουμε την πραγματική διάρκεια της sleep των 100 msec στο διαμέρισμα συστήματος και στο διαμέρισμα πραγματικού χρόνου με φορτίο.



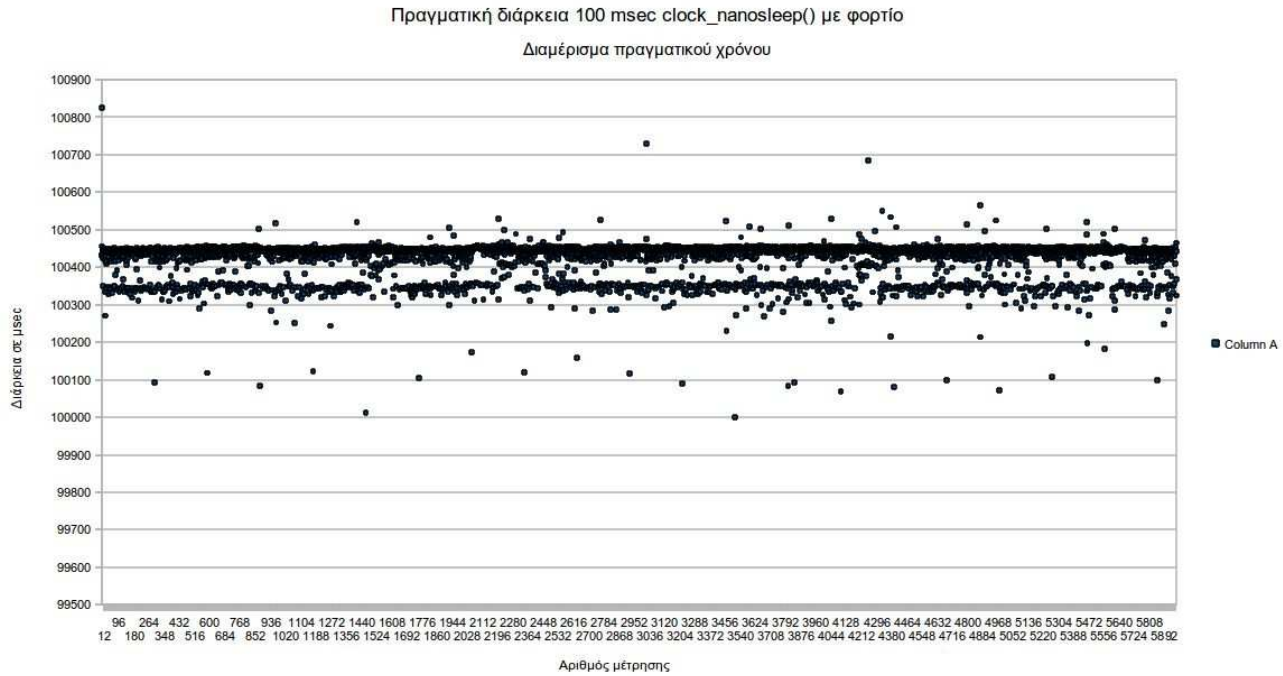
Εικόνα 10: 100 ms sleep, Διαμέρισμα Συστήματος, Χωρίς Φορτίο



Εικόνα 11: 100 ms sleep, Διαμέρισμα Πραγματικού Χρόνου, Χωρίς Φορτίο



Εικόνα 12: 100 ms sleep, Διαμέρισμα Συστήματος, Με Φορτίο



Εικόνα 13: 100 ms sleep, Διαμέρισμα Πραγματικού Χρόνου, Με Φορτίο

Πίνακας 2

100 msec Sleep ASMP-Linux	Διαμέρισμα Συστήματος				Διαμέρισμα Πραγματικού Χρόνου			
	MT	MA	TA	MTK	MT	MA	TA	MTK
Χωρίς Φορτίο	100596.02	7.16	17.35	596.2	100433.99	33.42	43.2	433.99
Με Φορτίο	100453.11	62.27	98.06	453.11	100429.2	28.51	43.41	429.2

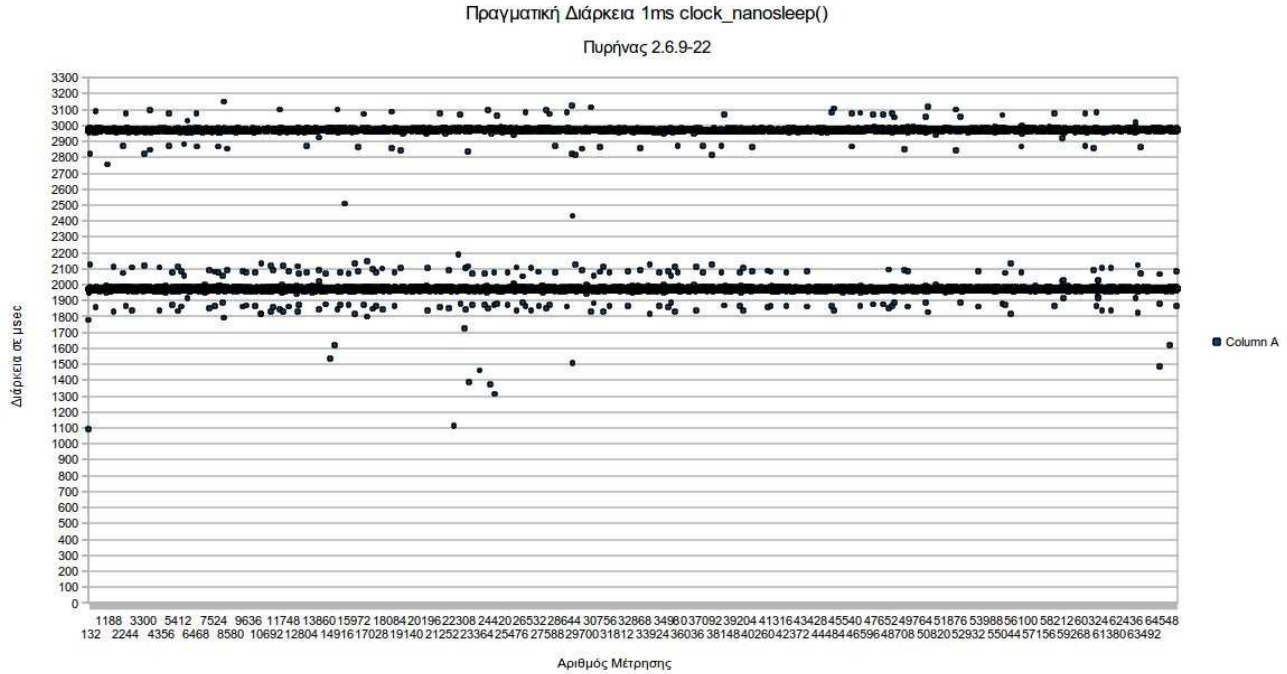
Στον Πίνακα 2 βλέπουμε συγκεντρωτικά τις τιμές: μέση τιμή, μέση απόκλιση, τυπική απόκλιση και μέση τιμή καθυστέρησης για τα παραπάνω αποτελέσματα.

Η μεγαλύτερη βελτίωση φαίνεται να συμβαίνει για λειτουργία με φορτίο, όπου στο διαμέρισμα συστήματος η διασπορά είναι μεγάλη (η τυπική απόκλιση έχει την τιμή 98.06 msec) και η κατανομή των αποτελεσμάτων ανόμοια, ενώ στο διαμέρισμα πραγματικού χρόνου η διασπορά είναι πολύ πιο μικρή (43.31 msec).

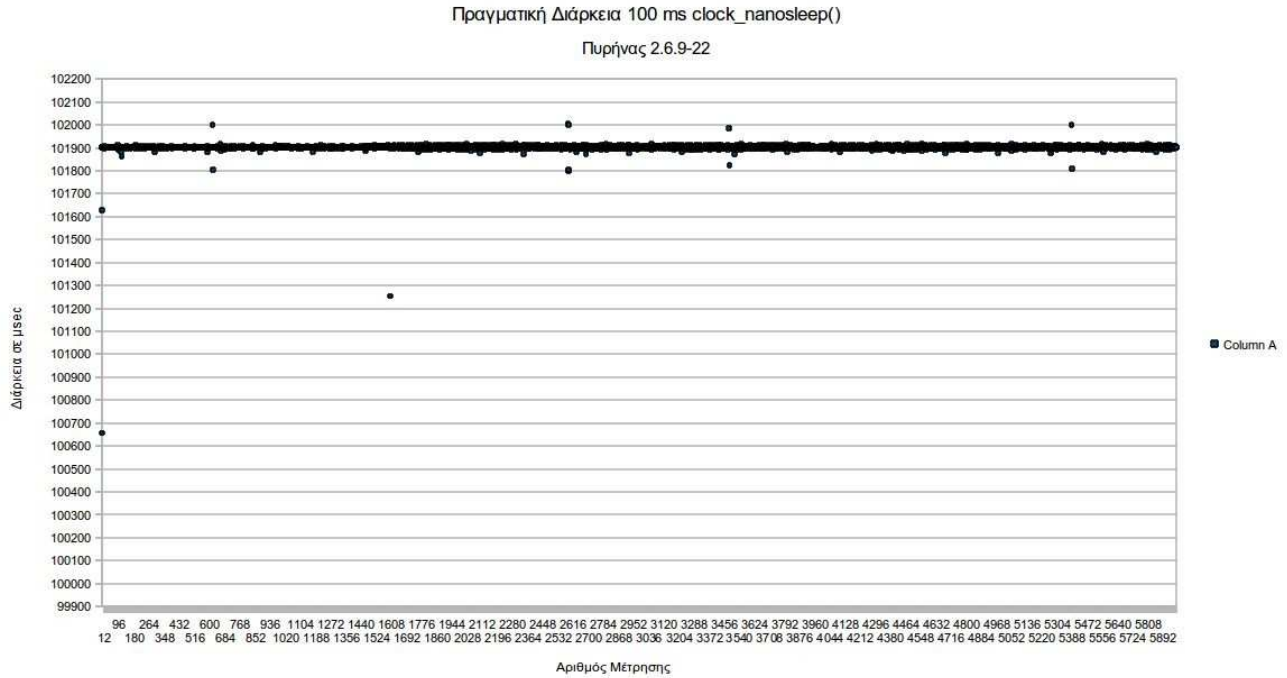
Η sleep των 100 msec διαρκεί στο διαμέρισμα συστήματος 100596.02 μsec χωρίς φορτίο και 100453.11 μsec με φορτίο, έχουμε πάλι δηλαδή μια μέση καθυστέρηση 0.5 msec ενώ στο διαμέρισμα πραγματικού χρόνου η καθυστέρηση αυτή μειώνεται πάλι στο 0.4 msec με την sleep να διαρκεί 100433.99 μsec χωρίς και 100429.2 μsec με φορτίο.

Οι εικόνες στις σελίδες 94 και 95 αφορούν την μέτρηση της πραγματικής χρονικής διάρκειας της sleep μιας διεργασίας σε ένα απλό πυρήνα Linux χωρίς το patch ASMP-Linux. Εφαρμόστηκε sleep 1 msec, 100 msec, με και χωρίς φορτίο.

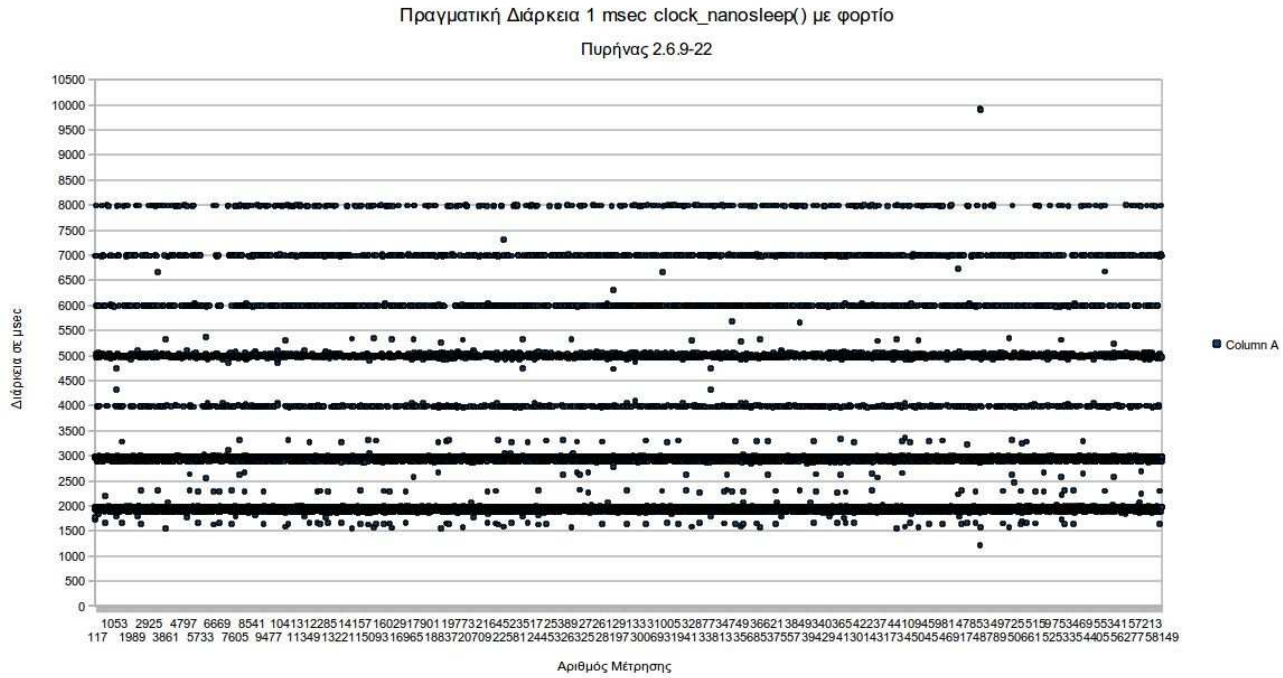
Στο διαγράμμα στην Εικόνα 14 βλέπουμε την πραγματική διάρκεια της sleep του 1 msec χωρίς φορτίο σε έναν απλό πυρήνα Linux 2.6.9-22 ενώ στην Εικόνα 15 βλέπουμε την πραγματική διάρκεια της sleep των 100 msec χωρίς φορτίο σε έναν απλό πυρήνα. Στο διαγράμμα στην Εικόνα 16 βλέπουμε την πραγματική διάρκεια της sleep του 1 msec με φορτίο σε έναν απλό πυρήνα Linux 2.6.9-22 ενώ στην Εικόνα 17 βλέπουμε την πραγματική διάρκεια της sleep των 100 msec με φορτίο σε έναν απλό πυρήνα.



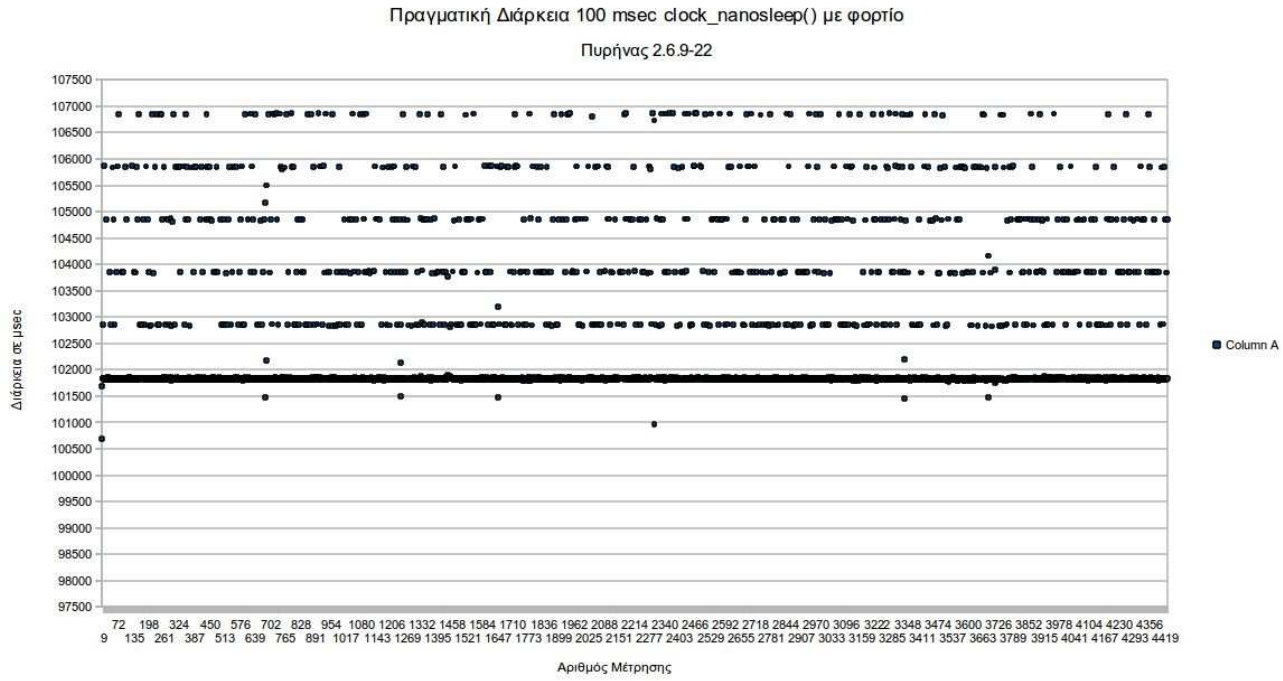
Εικόνα 14: 1 ms sleep, Πυρήνας 2.6.9-22, Χωρίς Φορτίο



Εικόνα 15: 100 ms sleep, Πυρήνας 2.6.9-22, Χωρίς Φορτίο



Εικόνα 16: 1 ms sleep, Πυρήνας 2.6.9-22, Με Φορτίο



Εικόνα 17: 100 ms sleep, Πυρήνας 2.6.9-22, Με Φορτίο

Πίνακας 3

Πυρήνας 2.6.9-22	1 ms sleep				100 ms sleep			
	MT	MA	TA	MTK	MT	MA	TA	MTK
Χωρίς Φορτίο	2247.19	397.3	445.75	1247.19	101903.07	2.42	19.22	1903.07
Με Φορτίο	2651.36	793.33	1108.22	1651.36	102304.29	775.55	1157.61	2304.29

Στον Πίνακα 3 βλέπουμε συγκεντρωτικά τις τιμές: μέση τιμή, μέση απόκλιση, τυπική απόκλιση και μέση τιμή καθυστέρησης για τα παραπάνω αποτελέσματα.

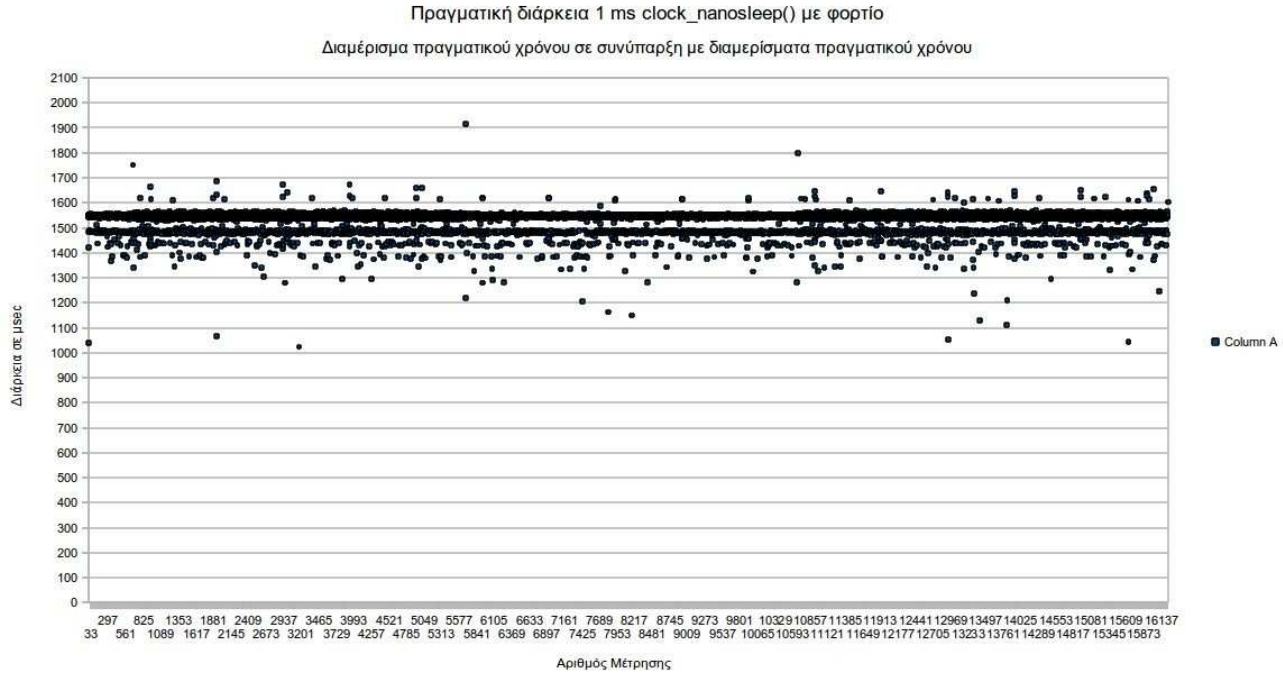
Χωρίς φορτίο, η sleep 1 msec στον απλό πυρήνα διαρκεί πραγματικά 2247.19 μsec κατά μέσο όρο, παρατηρείται δηλαδή μια μέση καθυστέρηση της τάξεως του 1.2 msec, ενώ και η διασπορά των αποτελεσμάτων είναι μεγάλη. Αυτό σημαίνει ότι η προβλεψιμότητα του συστήματος μειώνεται δραστικά, καθιστώντας βέβαια τη λειτουργία αυστηρού πραγματικού χρόνου αδύνατη. Με φορτίο τα αποτελέσματα αυτά χειροτερεύουν με την πρόσθετη καθυστέρηση της τάξεως των 0.4 msec, σύνολο δηλαδή 1.6 msec, περίπου 160% δηλαδή πάνω από την θεωρητική τιμή της sleep, και αύξηση της απόκλισης.

Η sleep των 100 msec σε έναν απλό πυρήνα διαρκεί στην πραγματικότητα 101903.07 μsec κατά μέσο όρο, καθυστερεί δηλαδή κατά 1.9 msec περίπου. Η απόκλιση των αποτελεσμάτων δεν είναι πολύ μεγάλη, χειροτερεύει όμως κατά πολύ στη λειτουργία με φορτίο. Τότε η sleep διαρκεί 102304.29 μsec, καθυστερώντας κατά μέσο όρο 2.3 msec.

Βλέπουμε ότι με φορτίο, δημιουργούνται 7 επίπεδα μέσης χρονικής διάρκειας για τη sleep του 1 msec, ένα επίπεδο στα 2 msec, ένα στα 3 msec ένα στα 4 msec ένα στα 5 msec ένα στα 6 msec, ένα στα 7 msec και ένα στα 8 msec. Θεωρούμε ότι λόγω της έλειψης του patch ASMP-Linux η δημιουργία των επιπέδων αυτών έχει να κάνει με το tick του global timer και την εναλλαγή κατάστασης της διεργασίας. Σύμφωνα με τον κύκλο ζωής της διεργασίας , αυτή μπορεί να βρίσκεται σε μία από τις καταστάσεις TASK_RUNNING (η διεργασία εκτελείται), TASK_INTERRUPTABLE (η διεργασία είναι σε κατάσταση sleep αλλά μπορεί να διακοπεί), TASK_UNINTERRUPTABLE (η διεργασία είναι σε κατάσταση sleep και δεν μπορεί να διακοπεί), TASK_ZOMBIE (η διεργασία διακόπηκε και η κατάσταση της δεν αποκτήθηκε), TASK_STOPPED (η διεργασία διακόπηκε), TASK_SWAPPING (η διεργασία εναλλάσεται με κάποια άλλη). Θεωρούμε ότι είναι πιθανό να επηρεάζεται η καθυστέρηση του ξυπνήματος της διεργασίας με τρόπο ανάλογο της εναλλαγής των καταστάσεων της.

Παρομοίως παρατηρούμε την δημιουργία 6 επίπεδων για τη sleep των 100 msec ένα περίπου στα 102 msec, ένα στα 103 msec, ένα στα 104 msec, ένα στα 105 msec, ένα στα 106 msec και ένα στα 107 msec.

Οι ακόλουθες εικόνες (Εικόνα 18 και Εικόνα 19) προέκυψαν με την παράλληλη συνύπαρξη διαμερισμάτων πραγματικού χρόνου που εκτελούσαν και αυτά το πρόγραμμα ελέγχου με διάφορες διάρκειες sleep (300 msec, 1 sec κ.ο.κ.). Μπορεί να θεωρηθεί ότι ο εξυπηρετητής λειτουργούσε υπό "φορτίο πραγματικού χρόνου".



Εικόνα 18: 1 ms sleep, Διαμέρισμα Πραγματικού Χρόνου, Φορτίο Πραγματικού Χρόνου

Εικόνα 19: 100 ms sleep, Διαμέρισμα Πραγματικού Χρόνου, Φορτίο Πραγματικού Χρόνου

Πίνακας 4

	ASMP-Linux με Φορτίο Πραγματικού Χρόνου			
	MT	MA	TA	MTK
1 ms sleep	1539.91	16.05	31.16	539.91
100 ms sleep	100478.99	35.19	45.56	478.99

Στον Πίνακα 4 βλέπουμε συγκεντρωτικά τις τιμές: μέση τιμή, μέση απόκλιση, τυπική απόκλιση και μέση τιμή καθυστέρησης για τα παραπάνω αποτελέσματα.

Ακόμα και με όλους τους επεξεργαστές του εξυπηρετητή να είναι θωρακισμένοι και να τρέχουν το πρόγραμμα ελέγχου, σε μια ποικιλία τιμών sleep, η πραγματική διάρκεια της sleep είναι κατά πολύ βελτιωμένη σε σχέση με τον απλό πυρήνα Linux. Αυτή του 1 msec διαρκει 1539.91 μsec, και αυτή των 100 msec διαρκεί 100478.99 msec, κρατώντας την καθυστέρηση προβλέψιμη στην τάξη του 0.5 msec.

Από τη μελέτη των γραφημάτων και των αποτελεσμάτων, διαπιστώνουμε ότι σε σχέση με έναν απλό πυρήνα Linux, το patch ASMP-Linux βελτιώνει κατά πολύ την καθυστέρηση ξυπνήματος μιας διεργασίας. Με το ASMP-Linux βελτιώνεται η συμπεριφορά του συστήματος, και όσον αφορά την καθυστέρηση αλλά και όσον αφορά την προβλεψιμότητα του, ακόμα και όταν η διεργασία βρίσκεται στο διαμέρισμα συστήματος.

Καταλήγοντας, σε σύγκριση με έναν απλό πυρήνα Linux, το ASMP-Linux διατηρεί την καθυστέρηση ξυπνήματος της διεργασίας προβλέψιμη περίπου στο 0.5 msec, περίπου κατά 1.2 msec μικρότερη από ότι στον απλό πυρήνα, και μειώνει την διασπορά των αποτελεσμάτων αυξάνοντας την προβλεψιμότητα του συστήματος.

Αναφορές

[1] “ASMP Linux”; http://www.sprg.uniroma2.it/asmplinux/.

[2] “System Programming Research Group”; http://www.sprg.uniroma2.it/index_e.html.

[3] “Hard real-time performances in multiprocessor-embedded systems using ASMP-Linux,” *EURASIP J. Embedded Syst.*, vol. 2008, 2008, pp. 1-16; http://portal.acm.org/citation.cfm?id=1361151.1361161.

[4] “Implementing Parallel Processing”; http://download.oracle.com/docs/cd/F49540_01/DOC/server.815/a67778/ch2_succ.htm.

[5] Emiliano Betti, “Satisfying hard real-time constraints using COTS components,” phd, UNIVERSITÁ DI ROMA "TOR VERGATA", 2009.

[6] “ksoftirqd(9)”; http://www.tin.org/bin/man.cgi?section=9&topic=ksoftirqd.

[7] “Take charge of processor affinity”; http://www.ibm.com/developerworks/linux/library/l-affinity.html.

[8] “The Linux Kernel Archives”; http://kernel.org/.

[9] “Red Hat Enterprise Linux 4 Update 2 Release Notes”; http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/release-notes/as-x86/RELEASE-NOTES-U2-x86-en.html.

[10] “Posix compliant cpu clocks V7 [0/2]: Rationale and test program [LWN.net]”; http://lwn.net/Articles/105582/.

[11] “timer_create(2)”; http://www.kernel.org/doc/man-pages/online/pages/man2/timer_create.2.html.

[12] “clock_gettime(3): clock/time functions - Linux man page”; http://linux.die.net/man/3/clock_gettime.

[13] “clock_nanosleep”; http://www.opengroup.org/onlinepubs/009695399/functions/clock_nanosleep.html.

5 Συμπεράσματα και Περαιτέρω Εργασία

6.1 Συμπεράσματα

Στην παρούσα διπλωματική εργασία μελετήθηκαν διεξοδικά τα ενσωματωμένα συστήματα, η έννοια του πραγματικού χρόνου, τα συστήματα πολυεπεξεργασίας και ο πυρήνας του δημοφιλούς λειτουργικού συστήματος Linux. Ερευνήθηκαν διαφορετικοί τρόποι προγραμματισμού ενσωματωμένων συστημάτων και κατανοήθηκε σε βάθος η λειτουργία του χρονοδρομολογητή του Linux.

Κυρίως αναζητήθηκαν τρόποι εφαρμογής του λειτουργικού συστήματος Linux με περιορισμούς πραγματικού χρόνου σε συστήματα πολυεπεξεργασίας, κάτι που μπορεί να φανεί πολύ χρήσιμο στη νέα γενιά ενσωματωμένων συστημάτων. Όπως περιγράφηκε στη διπλωματική αυτή εργασία, η πολυπλοκότητα των ενσωματωμένων συστημάτων πραγματικού χρόνου, τείνει να αυξάνεται με τη χρήση λειτουργικών συστημάτων για διεύρυνση των δυνατοτήτων τους και την νέα τάση των πολυπύρηνων επεξεργαστών.

Για την αξιολόγηση ενός πυρήνα Linux πραγματικού χρόνου σε σύστημα πολυεπεξεργασίας, χρησιμοποιήθηκε ο εξυπηρετητής IBM x3455 του εργαστηρίου συστημάτων υπολογιστών και το patch ASMP-Linux. Αναπτύχθηκε ένα πρόγραμμα ελέγχου της πραγματικής διάρκειας που κάνει sleep μια διεργασία και εφαρμόστηκε με διαφορετικούς χρόνους σε διαφορετικές περιπτώσεις. Τα αποτελέσματα μελετήθηκαν γραφικά και στατιστικά.

Συμπεραίνουμε ότι η χρήση ενός τροποποιημένου πυρήνα Linux σε τέτοια συστήματα, θα μπορούσε να βελτιώσει την απόδοση τους. Ακόμα, η επιλογή του Linux σαν λειτουργικό σύστημα έχει αρκετά πλεονεκτήματα, όπως η δυνατότητα πολύ μικρού μεγέθους πυρήνα, κάτι που είναι απαραίτητο για την περιορισμένη μνήμη των ενσωματωμένων συστημάτων, ενώ η ιδιότητα του ανοιχτού κώδικα του λειτουργικού το καθιστά οικονομικά συμφέρον και ευρέως τροποποιήσιμο ανάλογα με τις ανάγκες της πλατφόρμας στην οποία εγκαθίσταται.

Όσον αφορά το patch ASMP-Linux, διαπιστώθηκε ότι είναι μια ιδιαιτέρως χρήσιμη τροποποίηση του πυρήνα του Linux, κυρίως γιατί επιτρέπει σε ένα σύστημα πολυεπεξεργασίας να υποστηρίξει τόσα εικονικά λειτουργικά συστήματα πραγματικού χρόνου όσοι και οι επεξεργαστές/πυρήνες του, ενώ παράλληλα μπορεί να διατηρήσει ένα διαμέρισμα συστήματος που να αναλαμβάνει όλες τις απλές λειτουργίες χρήστη και συστήματος. Έτσι οι απλές λειτουργίες του συστήματος δεν επηρεάζουν τις διεργασίες πραγματικού χρόνου που εκτελούνται σε διαμερίσματα πραγματικού χρόνου.

Η πιο σημαντική ιδιότητα του είναι ότι αυτή η ανάθεση διεργασιών και δημιουργία διαμερισμάτων γίνεται δυναμικά κατά το run time από το χρήστη χωρίς επιπλέον τροποποίηση του συστήματος ή διακοπή της λειτουργίας του. Θεωρούμε ότι δεν είναι ακόμα κατάλληλο για λειτουργία αυστηρού πραγματικού χρόνου λόγω της μέσης καθυστέρης του 0.5 msec που παρατηρήθηκε. Πιθανόν μια τέτοια καθυστέρηση να μην είναι ουσιώδης για μεγάλα χρονικά διαστήματα, όμως για χρονικά διαστήματα του 1 msec, θεωρείται σημαντική.

Η δοκιμή σε ένα μη ενσωματωμένο σύστημα θεωρούμε ότι δεν επηρεάζει σημαντικά την εξαγωγή των συμπερασμάτων. Εύκολα θα μπορούσε να γίνει compile ένας πυρήνας με το patch αυτό και για ένα ενσωματωμένο σύστημα, αφού και στο σύστημα μας ο πυρήνας έχει μέγεθος μόλις 1.6 MB παρότι πολλές από τις ιδιότητες που πρέπει να υποστηρίζονται από ένα υπολογιστή γενικού σκοπού ενσωματώθηκαν στον πυρήνα αυτό. Μπορούμε να συμπεράνουμε ότι η μείωση των δυνατοτήτων του πυρήνα για ένα ενσωματωμένο σύστημα, όπως λόγου χάρη η κατάργηση υποστήριξης κάποιων περιφερειακών συσκευών, μπορεί να οδηγήσει σε ένα ακόμα μικρότερο πυρήνα, κατάλληλο για χρήση σε ενσωματωμένα συστήματα.

6.2 Προτάσεις για Περαιτέρω Εργασία

Σαν περαιτέρω εργασία που προκύπτει από τη διπλωματική εργασία αυτή, μπορεί να σημειωθεί η δοκιμή ενός τροποποιημένου ASMP-Linux πυρήνα σε ένα ενσωματωμένο σύστημα, και η διεύρυνση της χρήσης του σε συστήματα 64 bit. Ακόμα, είναι δυνατή η βελτίωση του patch ASMP-Linux ώστε να μην επηρεάζεται λόγου χάρη από διακοπές λόγω της ενδο-επεξεργαστικής επικοινωνίας ή τα νήματα του πυρήνα που υποχρεωτικά παραμένουν στους θωρακισμένους επεξεργαστές. Τέλος, θα μπορούσε να αναπτυχθεί και να δοκιμαστεί μια εφαρμογή σε ένα πραγματικό περιβάλλον που απαιτεί την εργασία του λειτουργικού αυτού συστήματος σε συνθήκες πραγματικού χρόνου.

ΠΑΡΑΡΤΗΜΑ Α

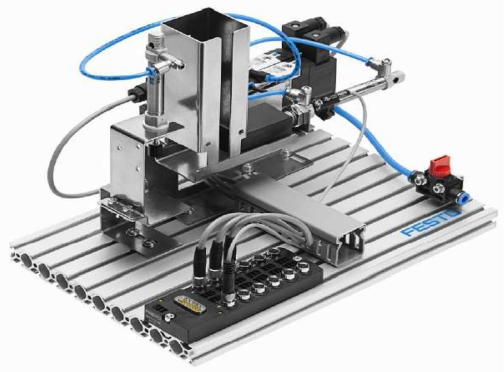
Μελέτη Περίπτωσης Χρήσης: Εκτέλεση Προγράμματος Ελέγχου του Διδακτικού Συστήματος MecLab της Festo στο Λειτουργικό Σύστημα ASMP-Linux

1 Περιγραφή των σταθμών που συνιστούν το MecLab

Το σύστημα εκμάθησης της Festo MecLab, προσομοιώνει τις τρεις βασικές λειτουργίες μιας αυτοματοποιημένης γραμμής παραγωγής: στοίβαξη και διαχωρισμός, μεταφορά, χειρισμός. Αυτό επιτυγχάνεται μέσω των τριών σταθμών του συστήματος.

- Σταθμός στοίβαξης.

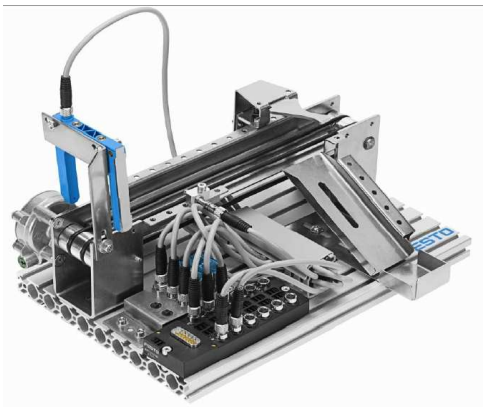
Η στοίβαξη είναι ένας τρόπος αποθήκευσης των τεμαχίων προς επεξεργασία με ένα οργανωμένο τρόπο. Εδώ αποθηκεύονται ή διαχωρίζονται τεμάχια ενώ μπορούν να σφραγιστούν, ή να εφαρμοστεί μια διαδικασία εισαγωγής στην γραμμή παραγωγής. Διαθέτει μαγνητικό αισθητήρα απόστασης και πνευματικές βαλβίδες. Προσφέρει πιεστική λειτουργία, πιέζοντας με πνευματικό κύλινδρο τα τεμάχια. Ο σταθμός στοίβαξης του MecLab φαίνεται στην Εικόνα 1.



Εικόνα 1: Σταθμός Στοίβαξης

- Σταθμός μεταφοράς.

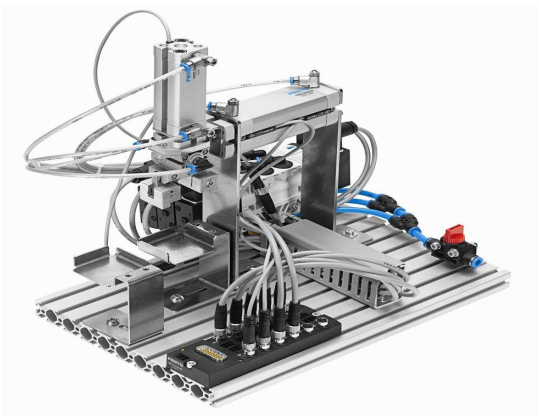
Αναλαμβάνει την μεταφορά τεμαχίων προς επεξεργασία, τα οποία μπορούν να διαχωριστούν μέσω εκτίναξης. Αποτελείται από μια μεταφορική ταινία και έναν κινητήρα συνεχούς ρεύματος που λειτουργεί προς δύο κατευθύνσεις. Ακόμα διαθέτει ένα σωληνοειδές που μπορεί να λειτουργήσει είτε για να δώσει ώθηση σε ένα τεμάχιο, είτε για να σταματήσει την κίνηση ενός τεμαχίου. Οι αισθητήρες που διαθέτει αποτελούνται από έναν επαγωγικό που ανιχνεύει μέταλο, και έναν οπτικό που ανιχνεύει την ύπαρξη αντικειμένου. Ο σταθμός μεταφοράς φαίνεται στην Εικόνα 2.



Εικόνα 2: Σταθμός Μεταφοράς

- Σταθμός χειρισμού.

Επανατοποθετεί τεμάχια απο τον ένα σταθμό στον άλλο ή συναρμολογεί τα τεμάχια. Ακόμα, αναλαμβάνει τον προσανατολισμό τους και την μετακίνηση τους. Αποτελείται από ένα βραχίονα με αρπάγη, τρεις βαλβίδες που ελέγχουν τους ενεργοποιητές, τέσσερις μαγνητικούς αισθητήρες απόστασης και άλλα στοιχεία. Ο σταθμός αυτός φαίνεται στην Εικόνα 3.



Εικόνα 3: Σταθμός Χειρισμού

Η λειτουργία της γραμμής παραγωγής μπορεί να μεταβληθεί μέσω της προσθαφαίρεσης ή μετακίνησης αισθητήρων. Κάθε σταθμός διαθέτει ένα διανομέα βυσμάτων πολλών pin, μέσω του οποίου δίνεται η δυνατότητα ελέγχου από υπολογιστή των ενεργοποιητών και των αισθητήρων. Ο διανομέας αυτός φαίνεται στην Εικόνα 4.

Εικόνα 4: Διανομέας Βυσμάτων

2 Περιγραφή της λειτουργίας της γραμμής παραγωγής

Ο διδακτορικός φοιτητής του τμήματος Ηλεκτρολόγων Μηχανικών και Τέχνολογίας Υπολογιστών, ανέπτυξε στα πλαίσια του διδακτορικού του ένα πρόγραμμα ελέγχου μιας γραμμής παραγωγής που υλοποιήθηκε σε μικρή κλίμακα με το MecLab. Η περιγραφή της λειτουργίας της γραμμής παραγωγής ακολουθεί.

Τα τεμάχια προς επεξεργασία, πλαστικά και μεταλλικά δισκία, τοποθετούνται στον σταθμό στοίβαξης. Μέσω ενός τροφοδότη που ελέγχεται από μια πνευματική βαλβίδα, τα τεμάχια ωθούνται ένα ένα πάνω στην μεταφορική ταινία.

Ο σταθμός μεταφοράς διαθέτει έναν επαγωγικό αισθητήρα στην αρχή της μεταφορικής ταινίας. Όταν ανιχνευθεί μεταλλικό τεμάχιο, ένας βραχίονας στο τέλος της μεταφορικής ταινίας κατεβαίνει ώστε να αποκλίνει την πορεία του αντικειμένου και να το ωθήσει στον χώρο αποθήκευσης των μεταλλικών δισκίων. Μετά από ένα καθορισμένο χρόνο ο βραχίονας ανέρχεται ώστε στην περίπτωση που δεν ανιχνευτεί μεταλλικό τεμάχιο, να μην εμποδιστεί η πορεία ενός πλαστικού τεμαχίου.

Στο τέλος της γραμμής μεταφοράς, μετά τον βραχίονα, υπάρχει ένας οπτικός αισθητήρας. Όταν ανιχνευτεί αντικείμενο από τον οπτικό αισθητήρα, ενεργοποιείται ο σταθμός χειρισμού. Το αντικείμενο προς επεξεργασία που ανιχνεύεται από τον οπτικό αισθητήρα, είναι σίγουρα πλαστικό, αφού όλα τα μεταλλικά έχουν απομακρυνθεί από τη γραμμή παραγωγής από τον βραχίονα. Ο σταθμός χειρισμού αναλαμβάνει να σηκώσει το τεμάχιο και να το τοποθετήσει στο χώρο αποθήκευσης πλαστικών δισκίων. Οι σταθμοί του MecLab συνδέονται με έναν προσωπικό υπολογιστή μέσω θυρών USB.

3 Πρόγραμμα ελέγχου

Το πρόγραμμα ελέγχου που αναπτύχθηκε από τον διδακτορικό φοιτητή του τμήματος Γεώργιο Δούκα και εκτελέστηκε με επιτυχία στο διαμέρισμα πραγματικού χρόνου στο λειτουργικό σύστημα ASMP-Linux ακολουθεί την παρακάτω λογική:

- Ρύθμιση των παραμέτρων επικοινωνίας σε κάθε θύρα USB.
- Αρχικοποίηση των σταθμών με την ανάγνωση της κατάστασης τους από τις θύρες στις οποίες συνδέονται και αποστολή των κατάλληλων αλφαριθμητικών σε κάθε θύρα USB. Ο τροφοδότης πρέπει να επεκταθεί και ο βραχίονας με αρπάγη του σταθμού χειρισμού να ανυψωθεί.
- Ανάγνωση των αλφαριθμητικών που φτάνουν στις θύρες από τους αισθητήρες και αποθήκευση της κατάστασης των αισθητήρων σε μεταβλητές.
- Έλεγχος της κατάστασης του σταθμού στοίβαξης:
  - Αν ο τροφοδότης είναι εκτεταμένος, πρέπει να επιστρέψει στην αρχική του θέση και να εκκινήσει η μεταφορική ταινία.
- Έλεγχος του επαγωγικού και οπτικού αισθητήρα:
  - Αν ανιχνευτεί μεταλλικό αντικείμενο, πρέπει να κατέβει ο βραχίονας με αποτέλεσμα την παρέκλιση της πορείας του αντικειμένου.
  - Αν ανιχνευτεί απο τον οπτικό αισθητήρα αντικείμενο, πρέπει να εκκινήσει η αρπάγη:
    - Χειρισμός της αρπάγης ώστε να τοποθετήσει το αντικείμενο στον χώρο αποθήκευσης.
- Έλεγχος του τροφοδότη: αν δεν είναι εκτεταμένος πρέπει να επεκταθεί, σπρώχνοντας κατά συνέπεια ένα νέο αντικείμενο στην μεταφορική ταινία.
- Έλεγχος του βραχίονα: αν δεν έχει ανιχνευτεί μεταλλικό αντικείμενο και είναι κατεβασμένος πρέπει να σηκωθεί.
- Έλεγχος του τροφοδότη: αν είναι εκτεταμένος πρέπει να επιστρέψει.

ΠΑΡΑΡΤΗΜΑ Β

HowTo: Install and Use the Lauterbach Trace32 System for the Infineon Processor TC1796

Η τεχνική αναφορά που ακολουθεί αποτελεί εργασία της συντάκτριας που εκπονήθηκε στο Τεχνολογικό Πανεπιστήμιο της Βιέννης (Technische Universitaet Wien) ως μέρος της παρούσας διπλωματικής εργασίας, μέσω της συνεργασίας του επιβλέποντος καθηγητή με το προαναφερθέν πανεπιστήμιο στα πλαίσια του προγράμματος Erasmus.

HowTo: Install and Use the Lauterbach Trace32 System for the Infineon Processor TC1796

Angeliki Chrysochou
Matrikelnummer: 0627966
TU-Vienna, Real-Time Systems Group
email: axrysoxoy@upnet.gr, am_sc_system@yahoo.gr

July 26,2007

Contents

1 Introduction

The following article describes the preparation required so as to be able to use the PowerTrace module of the Trace32 system, developed by Lauterbach Datentechnik GmbH (http://www.lauterbach.com), in order to perform trace and timing analysis on a TriBoard with the TC1796 microprocessor by Infineon Technologies AG (http://www.infineon.com). For the analysis, we use in our experiments the OCDS2 (On-Chip Debug Support Level 2) interface of the TriBoard, which is connected to the PowerTrace module via the preprocessor.

2 Hardware Description

2.1 The Infineon TriBoard TC1796

The TriBoard TC1796 is part of the TriCore Starter Kit, and contains a 32-Bit high performance Real-Time microcontroller with various on-chip peripherals.

The TC1796 microcontroller, targets mainly automotive applications, like automotive engine and transmission control and automotive by-wire control systems, while it can be also used in industrial robotic control. It offers real-time performance with DSP (Digital Signal Processing) capabilities combined with fast interrupt response time and a high fault tolerance level [1].

The name of the architecture (TriCore), signifies that three technologies are integrated within one silicon die, in a compact, unified, re-programmable core:

- RISC (Reduced Instruction Set Computing) processor architecture: provides high computational bandwidth with low system cost.
- DSP operations and addressing modes: provide the computational power necessary to efficiently analyze complex real-world signals.
- On-chip memories and peripherals: are designed to support demanding high-bandwidth real-time embedded control-systems tasks.

These features make TriCore a 32-bit microcontroller with DSP architecture optimized for embedded real-time systems [2, 3].

Figure 1 shows the TriBoard TC1796 placement [4].

2.2 The Lauterbach Trace32 System

Trace32 is a set of modular microprocessor development tools. A modular system contains components (modules) that can work separately as well as

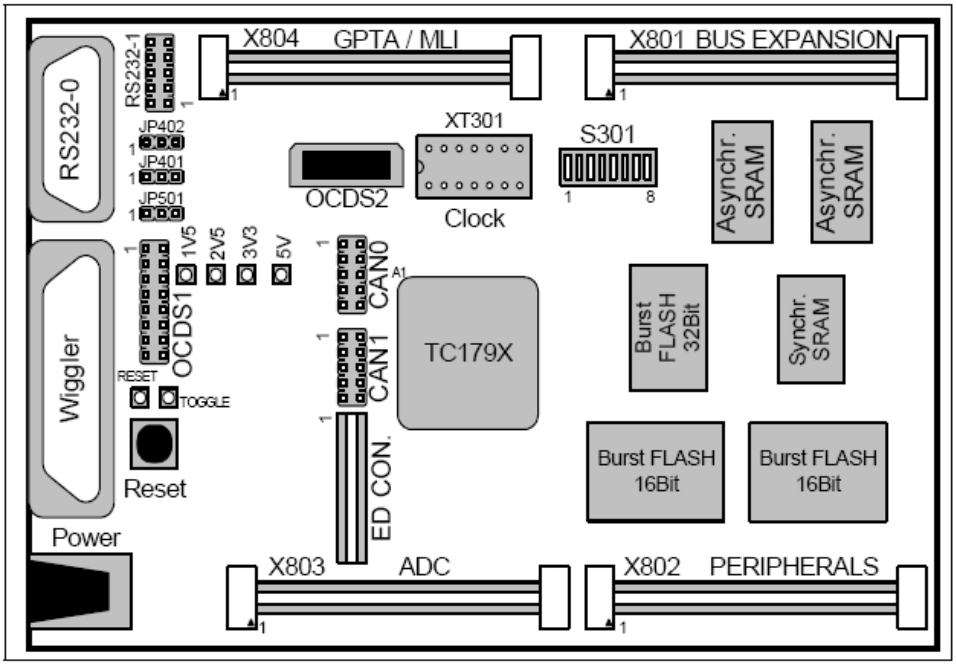


Figure 1: TriBoard placement

combined with each other and other modules. They support many different processors and include debuggers and emulators. Being more specific, the tools included in Trace32 system are:

- Trace32-ICD: In-Circuit Debugger
- Trace32-ICE: In-Circuit Emulator, divided in two major functional units, the Emulator and the Analyzer
- Trace32-FIRE: Fully Integrated RISC Emulator
- Trace32-ICD PowerTools: In-Circuit Debugger and Highspeed Trace, Logic Analyzers, Onchip Trace

For our experiments, we used the Trace32-ICD PowerTrace module (i.e. a trace extension to the Trace32-ICD), the Debug Cable and the Lauterbach Preprocessor module for the TriCore architecture which provides connectivity to the OCDS level 2 interface of the TriBoard (see Figure 4).

An onchip debug system, like the JTAG or the BDM interface, is provided by most CPUs and offers features like read/write memory or CPU registers, single step and real time execution, hardware breakpoints and triggering. The In-Circuit Debugger Trace32-ICD, as well it's PowerTrace module, expand the basic features of the onchip debugging system to provide more powerful debugging capabilities:

- Easy high-level and assembly language debugging.
- Display of internal and external peripherals on a logical level.
- Onchip break and trigger support.
- RTOS (Real-Time Operating System) awareness.
- Flash programming.
- Powerful script language.
- Multiprocessor debugging.

Suggested manuals for first time installation and use of the In-Circuit Debugger module, are the following and they can be found in the Lauterbach TRACE32 POWERVIEW Product Software CD: Trace32 Installation Guide [5], ICD Quick Installation [6], ICD tutorial [7] and Training ICD Basics [8]. For using the PowerTrace module, see Training ICD Trace Extension manual [9]. See Trace32 Online Help file [10].

3 Install the Software

When you install the software from the Lauterbach TRACE32 POWERVIEW Product Software CD, at some point you will be prompted to choose the part of the software for the product type you want to install as shown in Figure 2. We chose ICD (PowerTrace).

Then, you have to choose the connection interface to the PC (here we chose USB), choose License Key not necessary when prompted, your operating system and then, in the ICD list (Figure 3) choose your target (here we chose ICD TriCore).

In Figure 3, you can see the installation directory (C:\T32B). It is possible to choose a different directory in the beginning of the process , e.g. C:\T32 (which will actually be the default option). Continue and finish the installation process. See [5, 11] for more details.

4 Connect the Hardware

In Figure 4 you see the proper connection of the Trace32 modules used for the experiments. The preprocessor is connected to the PowerTrace via cables A and B (make sure to connect them as indicated on the preprocessorcable A in slot A, cable B in slot B), and the debug cable is connected to the preprocessor as well, and *not to our target*. The debug cable without the preprocessor can connect to the target using OCDS level 1 interface, while using the preprocessor, enables us to use OCDS level 2.

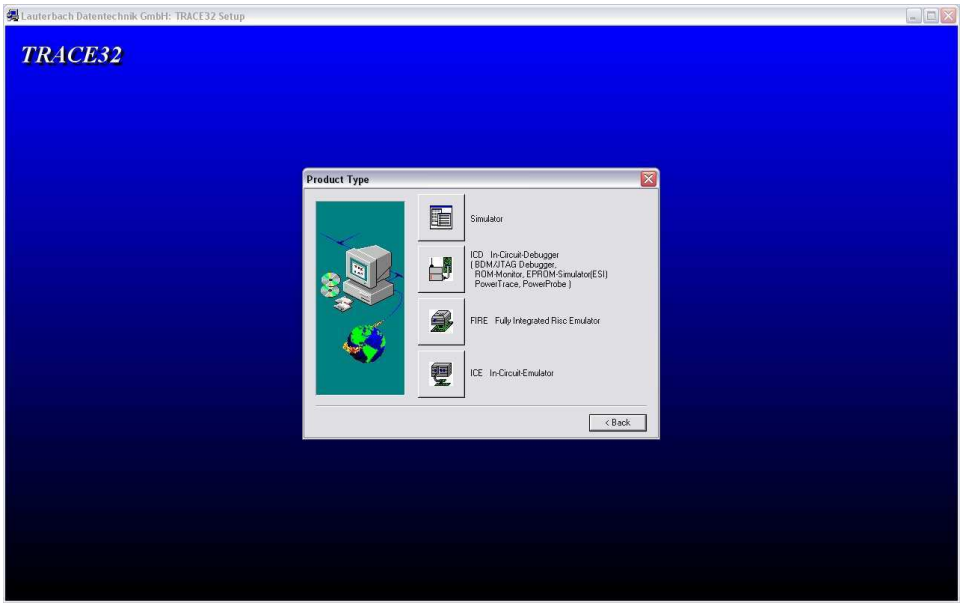


Figure 2: Choose the component to install

To prevent debugger or target from damage, the debug interface (cable or preprocessor) must be connected or disconnected to the target *only when the target power is OFF*! The complete connected system is shown in Figure 5. Notice that our target is not connected directly to the host system, it is connected only to the PowerTrace via the preprocessor (using OCDS2 interface-check Figure 1) and to the host system via the PowerTrace. The black cables shown are for power supply.

5 Software Description

First of all, it must be cleared that to work with the Trace32-ICD and PowerTrace, *a working target is required*! To prevent debugger and target from damage, it is also **very important** to follow carefully the power up and power down sequences as written in the documentation:

- Power up: **debugger-target**
- Power down: **target-debugger**

Moreover, it must be stated that the Trace32 software does *not* assemble, compile, or link your program. This must be done by some other application specific for this purpose. What Trace32 *does* offer to the user, is: download the executable file to the target, run the program, debug it and the rest of the features listed in 5.1.

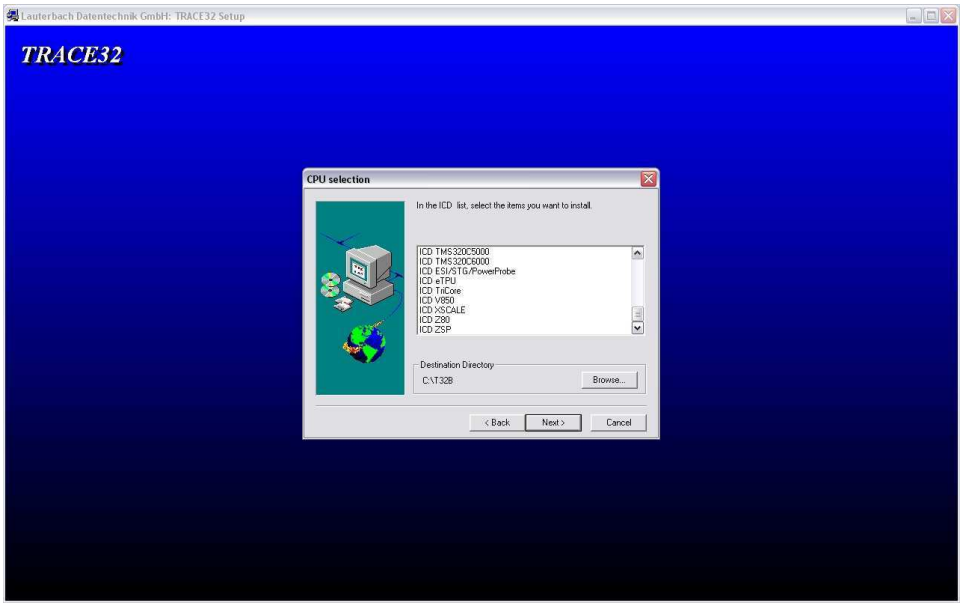


Figure 3: Choose your target

To exit the software *always* use the **exit** option from the pulldown menu "File".

To get familiar with the IDE (Integrated Development Environment), it is suggested to study [12], and for further details [13, 14]. Details on how to use the software are also provided in the specified guides and tutorials for each function offered by the Trace32 tools. See Trace32 online help [10].

In Figure 6 you can see the main window of the Trace32 application.

5.1 Overview of Features

The software supplied with the Trace32 system, provides support for all the functions that the Trace32 system offers:

- Debugger
- Emulator
  - ICE
  - FIRE
- Trace
  - Bus trace, Flow trace, NEXUS trace
  - CTS (Context Tracking System)
  - Cache analysis (PowerTrace)

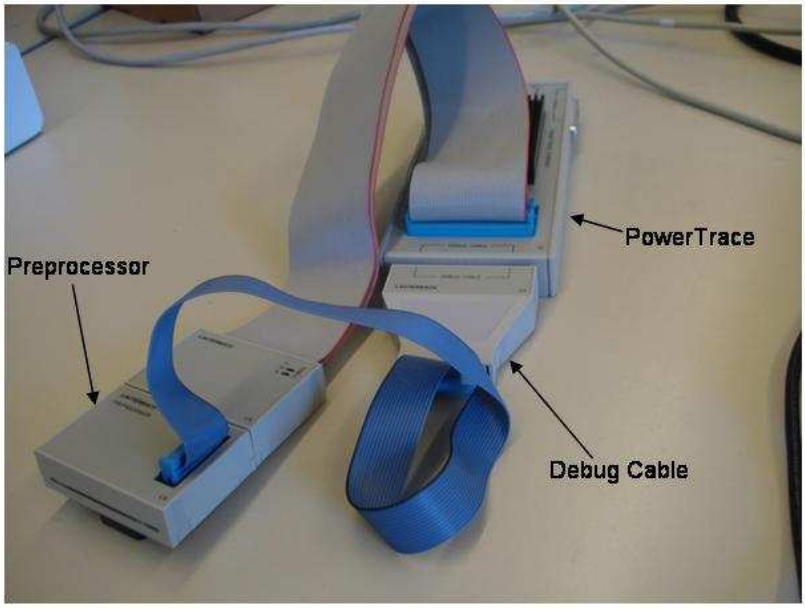


Figure 4: Proper PowerTrace connection

  - Function statistics
  - Performance analysis
  - Code coverage (PowerTrace)
- Timing analyzer
- Port analyzer
- RTOS Debugger
- Flash programming

During our experiments, we focused on the program flow trace including timestamps, which uses the tracing features provided by the PowerTrace module.

5.2 Overview of the PRACTICE Script Language

The PRACTICE script language offers commands to control all the functions of the Trace32 tools. It is *not* case sensitive, and has commands that control program flow, conditional program flow and nesting. It allows creation of block structures, creation and usage of variables, parameter passing and

Figure 5: Complete connection

input/output control. Comments are created using the semicolon character ";", or double forward slash "//". Throughout this document, PRACTICE commands are written in `typewriter` fonts. It is suggested to study [15, 16] for more details.

5.3 How to Use the Trace32 Online Help System

There are three ways to find a help file or a manual in the Trace32 help system:

1. Open the Trace32 Online Help [10] from the pdf folder in the Trace32 installation directory, browse it's contents and use the links available.
2. When filename already known, browse the pdf folder in the Trace32 installation directory and directly open the file wanted.
3. Use help options offered by the software's GUI (Graphical User Interface) [7]:
   - Help topics button in the tool bar.
   - Help contents entry in the menu bar.
   - Enter command `HELP` in the command line.

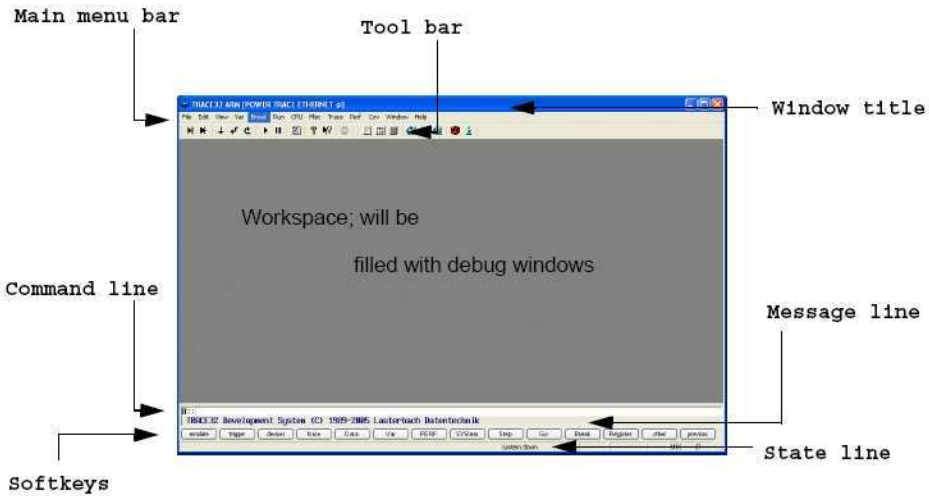


Figure 6: Main window of Trace32 application

5.4 User Interface Description

On the Windows operating system, the user interface can operate in two modes:

1. MDI (Multiple Document Interface) which places all windows inside a surrounding window.
2. MWI (Multiple Window Interface) which uses a command line window and spreads the other windows within the whole screen.

Every command or operation can be executed in three different ways:

1. Using the menu bar and pulldown menus.
2. Using the tool bar and the tool bars with special functions that appear in specific windows.
3. Using the command line and PRACTICE script language.

5.4.1 How to Use PRACTICE commands

In the software GUI, commands can be entered in the command line at the bottom of the main window (see Figure 6). Commands can be shortened using the significant characters for each command entry. Upper case letters are used to indicate significant characters in this document as well as in the Trace32 documentation. However, it is not compulsory to use upper case letters when executing the commands, since the language is not case

sensitive. The soft keys (Figure 6) provide a guided command entry by displaying all possible commands and parameters, so that a command can be assembled by clicking on the soft keys instead of writing in the command line.

5.5 Initialize the Software/Hardware

After the proper connections are made and there is no power supply provided to the debugger or target, the following steps should take place in the order they are mentioned:

1. Turn on debugger.
2. Start Trace32 ICD application.
3. Apply power to the target.
4. Run batchfile.

A batchfile, mentioned in step 4, is a Trace32 startup script, a text file with a ".cmm" extension that contains the necessary initialization and setup commands in PRACTICE language in order for them to be automatically and procedurally executed when it is run. This simplifies the procedure, since initializing and setting up the debug environment needs several commands, while the command line supports a one-command-only execution.

How should one initialize the software/hardware?

- Compulsory:
  - Reset the Trace32 debugger commands to their default state: this is done with the PRACTICE language command "`RESet`".
  - Select the CPU type in order to load specific CPU settings: "`SYStem.CPU <your cpu>`".
  - Enter debug mode - for example, "up" mode: "`SYStem.Up`".
- Optional:
  - Clear previously opened windows: "`WinClear`".
  - Tell the debugger where FLASH is on the target so that Trace32 knows where to use onchip breakpoints: "`MAP.BOnchip <address range>`".
  - Make target specific initializations, for example initializations required to define the interface used (OCDS1, 2 and so on).
  - Make memory initializations, again target specific.
  - Open desired windows.

– Load your program.

By compulsory, we mean the least initializations required in most situations to have a program downloaded to your target, run and debug it. In the optional initializations you might want to make, anything could be added, like setting breakpoints, using commands for bug fixes, enabling/disabling trace, or trigger functions. There are many possibilities, scattered in the specified guides of the functions that Trace32 tools offer. Basically, any PRACTICE command could be added in a batchfile. The programmer must take care of the procedural execution of the commands. It could be possible that some commands can't be executed before others. Furthermore, some of the optional initializations might actually be necessary in some cases in order to make things work.

It is recommended to perform a reset of the Trace32 software in the batchfile, even though you might notice that when you first start the application it performs a reset on it's own. This is recommended, because you might want to run another batchfile while the Trace32 application is already running, and reseting the hardware would have to be done "manually" (i.e. by the command line, menu, or tool bars) if not in the batchfile.

5.5.1 Create a batchfile

The batchfile can be created either within the Trace32 IDE by selecting "Edit Batchfile" from the File menu or by typing "`PEDIT` <filename>.cmm" in the command line [7], either by creating a simple text file in some text editor and provide the ".cmm" extension when naming the file.

An example batchfile/startup script follows in Table 1 [17].

Table 1: Example batchfile

`B::`	;Select the ICD device prompt
`WinClear`	;Clear all windows
`MAP.BOnchip 0xA000000++0x177FFF`	;Specify not writeable memory
`SYStem.CPU TC1766`	;Select CPU
`SYStem.Up`	;Reset the target and enter debug mode
`Data.LOAD.elf myprog.elf`	;Load the application
`Data.List`	;Open disassembly window
`Register`	;Open register window
`Variable.Local`	;Open window with local variables
`PER`	;Open window with peripheral register
`Break.Set main`	;Set breakpoint to function main
`Break.Set 0xD0000200 /Program`	;Set software breakpoint to specified address
`Break.Set 0xA0001000 /Program`	;Set onchip breakpoint to address 0xA0001000
`ENDDO`	;Terminate the PRACTICE program

Some other example batchfiles, along with sample programs for common hardware that supports the TriCore architecture, can be found in the \demo\tricore\hardware folder of the Trace32 software installation directory (for Windows users). The contents of the installation directory change depending on the software component that has been installed. We have installed the ICD software for the TriCore architecture. A batchfile from the specific folder that loads the "demo.elf" program on our TC1796 TriBoard is given in Table 2.

The commands for the TC1796 bug fix that are included in Table 2, must be used, according to Lauterbach, in order to flush the integrated code line buffer that the TriCore architecture uses for caching. In case of stopping the core by the debugger, the buffer will not be flushed correctly at some AUDO-NG processor types (e.g. TC1766, TC1796). The only way to solve this, is the work around included in the batchfile.

Table 2: Another example batchfile

; Script file for TC1796 on TriBoard	TC1796
;	
;Initializes the TriCore, SRAM and Flash,	and loads the Sieve-Demo into memory.
;	
; LastChangedDate: 2006-06-02	17:36:18 +0200 (Fr, 02 Jun 2006)
; LastChangedRevision: 1683	
; LastChangedBy: mobermaier	
;	
; performed tests	
; Dip Switch S301: SW1=ON, SW2=OFF,	SW3=ON, SW4=ON,
; SW5=ON, SW6=ON, SW7=OFF,	SW8=OFF
; TriBoard TC1796.101 TC1796	EES-AB 2005-01-26 MAX
; TriBoard TC1796.101 TC1796	EES-BA 2005-01-26 MAX
; TriBoard TC1796.300 TC1796	EES-BC 2005-01-26 MAX
;	
; memory mappings	
; flash is not declared	- see flash.cmm
; external SRAM mapped to	0xA1000000–0xA10FFFFF 1 MB
; external Flash mapped to	0xA2000000–0xA23FFFFF 4 MB
; processor internal program flash mapped	to 0xA0000000–0xA01FFFFF 2 MB
; processor internal data flash mapped to	0xAFE00000–0xAFE1FFFF 128 KB
;	
; Please note that A-steps can not be	halted immediately after reset.
;	
; A-steps: Since the watchdog is not	suspended in debug mode do not use a low JTAG

```
; clock frequency in case the watchdog        is not already disabled by the target program.
; Otherwise the debugger is possibly too      slow to disable the watchdog in time.
; Minimum JTAG clock depends on the           core clock. Default JTAG clock should be
                                              fine.
;
; Requirements to use OCDS-1:
; Set jumper JP501 to 2-3                     (= disable on-board wiggler)
;
; If onchip flash is not functional:          DIP Switch S401 SW5=ON
; Note: This may disable OCDS-L2              functionality.
;============================ ========================
; initialize and start the debugger
RESet
SYStem.CPU TC1796
//SYStem.Option WATCHDOGFIX ON                ;only required for A-step devices
SYStem.Up
; OCDS L2 (Trace) enable (OCNTRL OSCU         configuration and control register)
Data.Set 0xF000047C %Long 0x0000003           ;trace source TriCore
//Data.Set 0xF000047C %Long 0x000000F         ;trace source PCP
;============================ ========================
; Memory initialization
; initialize external bus unit
Data.Set 0xF8000010 %Long 0x00F9FF68          ;EBU_CON
Data.Set 0xF8000020 %Long 0x001001D0          ;EBU_BFCON

; memory assignment
; AMD AM29BL162CB flash
Data.Set 0xF8000080 %Long 0xA2000051          ;EBU_ADDR_SEL0
Data.Set 0xF80000C0 %Long 0x00820000          ;EBU_BUSCON0
Data.Set 0xF8000100 %Long 0x23FF0100          ;EBU_BUSAP0
; ALLIANCE SRAM
Data.Set 0xF8000088 %Long 0xA1000071          ;EBU_ADDR_SEL1
Data.Set 0xF80000C8 %Long 0x00820000          ;EBU_BUSCON1
Data.Set 0xF8000108 %Long 0x00D80000          ;EBU_BUSAP1
; unused
Data.Set 0xF8000090 %Long 0x00000000          ;EBU_ADDR_SEL2
Data.Set 0xF80000D0 %Long 0x80928000          ;EBU_BUSCON2
Data.Set 0xF8000110 %Long 0xCFFFFFFF          ;EBU_BUSAP2
Data.Set 0xF8000098 %Long 0x00000000          ;EBU_ADDR_SEL3
Data.Set 0xF80000D8 %Long 0x80928000          ;EBU_BUSCON3
Data.Set 0xF8000118 %Long 0xCFFFFFFF          ;EBU_BUSAP3
;============================ ========================
; TC1796 bug fix: PMI line buffer not         invalidated during CPU halt
Data.Assemble 0xA1001000 debug16 nop          nop nop nop nop nop nop nop
```

```
Data.PROLOG.TARGET 0xA1001000++0x1F     0xA1001020++0x1F
Data.PROLOG.ON
Register.Set PC 0xA1001002
Step
;============================ ========================
; load demo program
Data.LOAD.ELF demo.elf
;Run program until breakpoint on function     main
Go main
; open some windows
WinCLEAR                                     ;Clear all windows
WinPOS 0% 0% 100% 50%                        ;Define position of next window to be opened
Data.List                                    ;Open data list window
WinPOS 0% 50% 50% 50%
Var.Frame /Locals /Caller
WinPOS 50% 50% 50% 50%
Var.Watch
Var.AddWatch ast flags

ENDDO
```

For more information on creating your own batchfile, consult your target's documentation on which registers must be initialized and with which values, and for more commands the PRACTICE script language manuals [15, 16]. In addition, the Trace32 documentation, offers target specific guides on using the Trace32 tools. In our case, we consulted the debugger and trace guide for the TriCore architecture [17].

5.6 Tracing Features

Trace32 tools offer many different trace methods, like Analyzer, ART, Logger, SNOOPer etc. as shown in Figure 7. For this set of experiments we used the trace method Analyzer, which means that there is physical trace memory provided by the development tool, in our case, the PowerTrace module.

Apart from the trace methods, there are also different trace technologies supported, like bus trace, flow trace and NEXUS trace. Flow trace, and specifically program flow trace is what we focused on. Program flow tracking of PowerTrace, allows to track the program instruction flow with almost no performance degradation.

To modify the trace setup and control the recording to the trace buffer, use command "`Trace`" or "`Trace.State`". The recording to the trace buffer can also be controlled by filter and trigger features provided either by the Trace32, or by the onchip trace and trigger support of the cpu. The command will open a window like in Figure 7. There, you have various options to control, some of them may vary depending on he target. For example you can change or check:

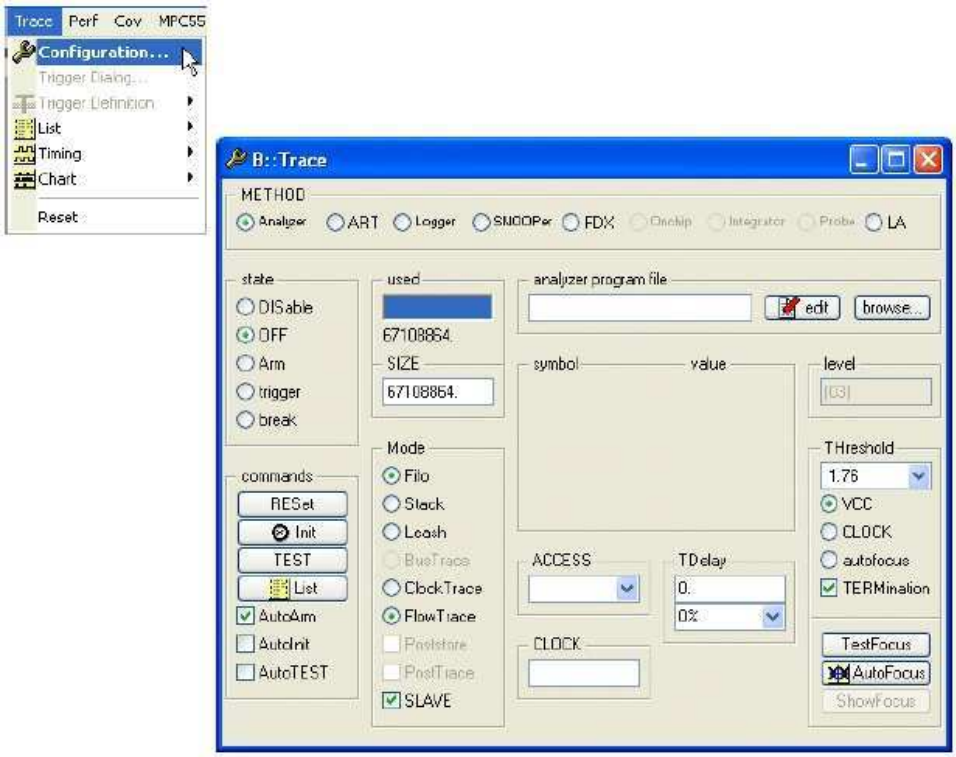


Figure 7: Trace setup window

- The trace **method**:
  - **Analyzer**: Physical trace memory is provided by the tool.
  - **ART** (Advanced Register Tracking): Provides a history for single stepping. For each step the contents of the general purpose registers is recorded. No physical trace memory required. Trace32 reserve memory on the host for ART. Available for all CPUs.
  - **Logger**: A part of the target RAM is used as trace memory. Trace32 provides all commands for trace display and evaluation. No physical trace memory required. Available for all CPUs.
  - **SNOOPer**: TRACE32 can read up to 16 data addresses in a fixed sampling rate and record their contents to the trace. No physical trace memory required. Memory on the host is reserved for the SNOOPer. The CPU has to support memory read while the program execution is running, or the onchip debugging system has to provide communication between the debugger and the application while the program execution is running.
  - **FDX (Fast Data Exchange)**: A small part of the target RAM is used as a ring buffer. Trace32 collects the contents of the ring

buffer permanently while the program is running and builds the trace contents based on this information. No physical trace memory required. Memory on the host is reserved for FDX. Again the CPU has to support memory read while the program execution is running, or the onchip debugging system has to provide communication between the debugger and the application while the program execution is running.

  - **Onchip**: The CPU provides an onchip trace.
  - **Integrator**: A Trace32-PowerIntegrator is set-up to work as a program or/and data flow trace.
  - **Probe**: A Trace32-PowerProbe is set-up to work as a program or/and data flow trace.
  - **LA**: The contents of an external trace device (e.g. from a logic analyzer) are imported to Trace32.

- The **state** of the trace:
  - **DISable**: The trace is disabled.
  - **OFF**: The trace is not sampling and the trace contents can be displayed.
  - **Arm**: The trace is sampling and the trace contents can not be displayed.
  - **trigger**: (not available for all CPUs) A trigger was generated to stop the trace, but the trace is still sampling because a trigger delay was defined (TDelay).
  - **break**: (not available for all CPUs) A trigger was generated to stop the trace and the number of records defined by TDelay was sampled. The trace is not sampling any more and the trace contents can be displayed.

- The trace configuration **commands**:
  - **RESet**: Reset the trace configuration window to its default settings.
  - **Init**: Clear the trace buffer.
  - **TEST**: Uncommonly used.
  - **List**: Display a trace listing.
  - **AutoArm**: The trace is automatically armed, when user program is started. The trace is automatically switched off, when user program stops.
  - **AutoInit**: By default, the trace buffer is not cleared when the user program is restarted by Go or Step. AutoInit automatically clears the trace buffer at every program start.

  - **AutoTEST**: Uncommonly used.

- The **size** of the trace buffer:
  - **used**: Number of sampled records in the trace buffer.
  - **SIZE**: Size of the trace buffer. It can be decreased for faster search or save operations.

- The trace **modes**:
  - **Fifo**: The trace is working in FIFO mode. When the trace buffer is full, the new records will overwrite the older ones. The trace records always the last cycles before the program stopped. When trace is working in FIFO mode, negative record numbers are used.
  - **Stack**: The trace is working in STACK mode. If the trace buffer is full, the trace will stop sampling. The trace records always the first cycles after the program start.
  - **Leash**: (PowerTrace / ETHERNET only) The trace is working in LEASH Mode. As soon as the trace is full, the program execution is stopped.

To display the trace contents, use command "`Trace.List`". The command can contain options and can have a modified format. Consult General Commands Reference Guide T in main file [10]. The default command will open a window like in Figure 8. The trace contents can contain:

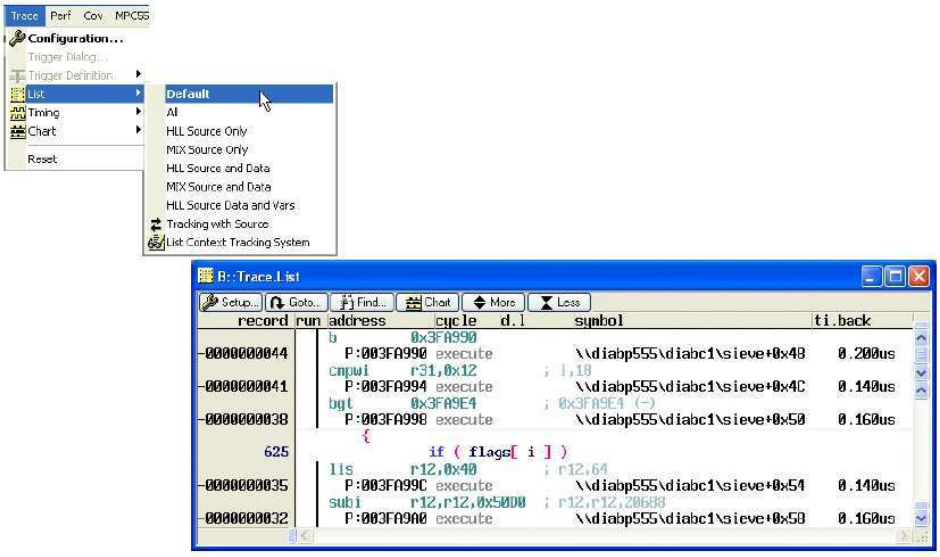


Figure 8: Trace results

- The record number: Positive for STACK and LEASH trace modes, negative for FIFO trace mode.
- CPU signals:
  - **address**: CPU address information.
  - **cycle**: CPU cycle type.
  - **d.b/ d.w/ d.l**: (Not all on the default trace window: must be added as an option to the command.) CPU data information.
  - **symbol**: Symbolic address with path and offset.
- Trace information:
  - **Time.Back**: Time relative to the previous record.
  - **Time.Zero**: (Not on the default trace window: must be added as an option to the command.) Time relative to the global zero point. The global zero point can be set by right clicking on a record and selecting "Set Zero".

A more detailed insight on the subject is offered by [9].

5.6.1 Trigger and Filter Features

Trace32 provides trigger and filter features in order to facilitate analysis, when only a specific part of a program needs to be traced. The trigger and filter operations are specified by setting breakpoints on the instruction we would like to trigger, enable or stop the tracing.

A breakpoint can be set by double clicking on the desired line of the source code in the "`Data.List`" window. Then by right clicking on it and selecting "Breakpoint..." you open a window like the one in Figure 9. This can also be achieved by directly right clicking on the desired line and selecting "Breakpoint..." without having to set the breakpoint first (the breakpoint is set after clicking on the "Ok" button of the window opened). If you click on the arrow of the "action" option, you can choose one of the following that concern tracing: (more options are offered that do not concern tracing)

- **TraceENABLE**: Enable the trace on the specified event.
- **TraceON**: Switch the sampling to the trace ON on the specified event.
- **TraceOFF**: Switch the sampling to the trace OFF on the specified event.
- **TraceTrigger**: Stop the sampling to the trace on the specified event. A trigger delay is possible.

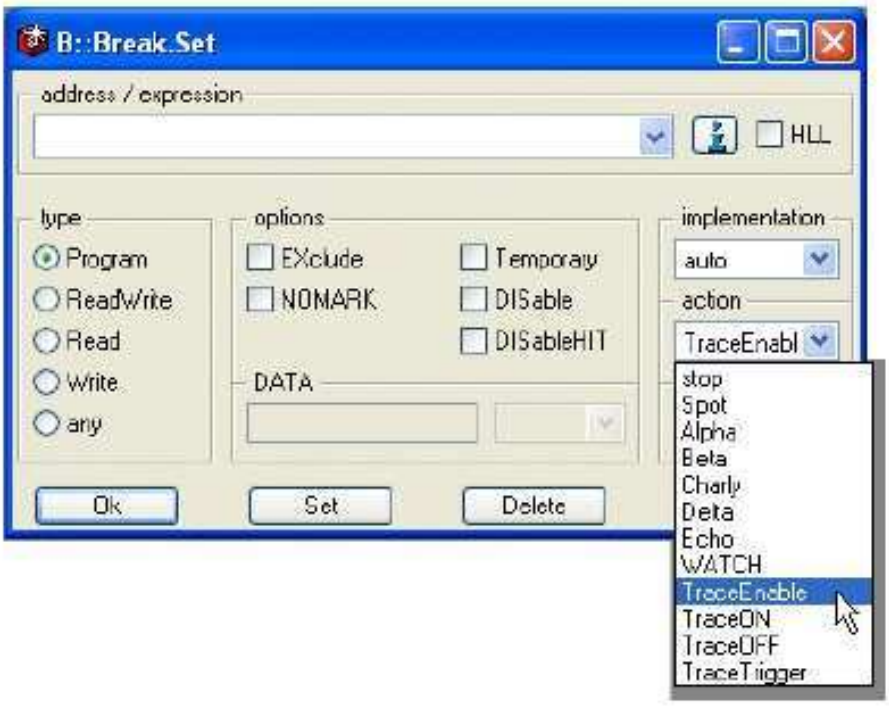


Figure 9: Breakpoint options

A breakpoint can also be set using the PRACTICE command "`Break.Set [<address>|<range>] [/<break>...]] [/<impl>]`" (check General Commands Reference Guide B on the main file of Trace32 help [10]). This means we can predefine our breakpoints and the desired actions that should be performed when they are met, even in the batchfile we create (see Table 1). This can lead to fully automated tracing by running only one file and getting all our results in the windows we select to be opened.

5.6.2 How to Save Trace Results

There are two ways to save your trace results in a readable ASCII file:

1. Using menus:
   (a) Click on "File", then "Print" and then "Printer Selection".
   (b) Select "File" and choose your file type and a file name.
   (c) Write in command line "`WinPrint.Trace.List <options of command Trace.List> (first record number)--(last record number)`".
2. Using commands:

(a) Start redirecting into printer file: "`PRT.FILE <filename>`".

(b) Choose what you want to print: "`WinPrint.Trace.List <options of command Trace.List> (first record number)--(last record number)`".

(c) Close printer file: "`PRT.FILE`".

If you specify a filename with a number in the end (for example test1.txt), every time the command "`WinPrint`" is used, the number of the file name specified is incremented by one and the output is saved to a new file (e.g. test2.txt). If you don't use a number, the file will be overwritten. If you don't specify a file name, the Trace32 software saves the output to file t32.lst. If you use the command again, the output will be saved in file t33.lst, and so on.

Actually, any output of almost any command can be redirected to a printer file. For example, you can save the contents of the data list window using the command "`WinPrint.Data.List`". In the command "`WinPrint.Trace.List (first record number)--(last record number)`", any minus ("-") or plus ("+") signs must also be specified. You can find more details in [8].

6 Trace the TriCore TC1796 Using PowerTrace

In order to perform tracing, one should execute the following steps:

1. Install the necessary software.
2. Create a batchfile for TC1796. The one that was mostly used in our experiments follows in Table 3.
3. Clear the trace buffer: use the command "`Trace.Init`". Clearing the trace buffer, was proven during our experiments to prevent flow errors (see 6.1) from appearing in the trace results.
4. Load the application on the target: the batchfile provided doesn't automatically load the program on the target. This is done using the command "`Data.Load.elf\path_to_application\<application>.elf`".
5. Run the application: you can run the program with the command "`Go`". You can also run the program with the command sequence "`Go main`", that sets the program counter to function main (provided that there is a main function in your program-see example batchfile in Table 2), and then "`Go`". You can stop the execution using the debugger commands "`Break`" or "`Stop`".

Table 3: Batchfile used in experiments

```
WinCLEAR
RESet
SYStem.CPU TC1796
SYStem.Up
Analyzer.Clock 150.0MHz                 ;Specify the CPU clock frequency to improve
                                        accuracy (x4)

; OCDS L2 (Trace) enable (OCNTRL OSCU   configuration and control register)
Data.Set 0xF000047C %Long 0x0000003     ;trace source TriCore

;============================ ========================
; Memory initialization
; initialize external bus unit
Data.Set 0xF8000010 %Long 0x00F9FF68    ;EBU_CON
Data.Set 0xF8000020 %Long 0x001001D0    ;EBU_BFCON

; memory assignment
; AMD AM29BL162CB flash
Data.Set 0xF8000080 %Long 0xA2000051    ;EBU_ADDR_SEL0
Data.Set 0xF80000C0 %Long 0x00820000    ;EBU_BUSCON0
Data.Set 0xF8000100 %Long 0x23FF0100    ;EBU_BUSAP0
; ALLIANCE SRAM
Data.Set 0xF8000088 %Long 0xA1000071    ;EBU_ADDR_SEL1
Data.Set 0xF80000C8 %Long 0x00820000    ;EBU_BUSCON1
Data.Set 0xF8000108 %Long 0x00D80000    ;EBU_BUSAP1
; unused
Data.Set 0xF8000090 %Long 0x00000000    ;EBU_ADDR_SEL2
Data.Set 0xF80000D0 %Long 0x80928000    ;EBU_BUSCON2
Data.Set 0xF8000110 %Long 0xCFFFFFFF    ;EBU_BUSAP2
Data.Set 0xF8000098 %Long 0x00000000    ;EBU_ADDR_SEL3
Data.Set 0xF80000D8 %Long 0x80928000    ;EBU_BUSCON3
Data.Set 0xF8000118 %Long 0xCFFFFFFF    ;EBU_BUSAP3

;============================ ========================
; TC1796 bug fix: PMI line buffer not   invalidated during CPU halt

Data.Assemble 0xA1001000 debug16 nop    nop nop nop nop nop nop nop
Data.PROLOG.TARGET 0xA1001000++0x1F     0xA1001020++0x1F
Data.PROLOG.ON
Register.Set PC 0xA1001002              ;Set program counter to specified address
Step
;============================ ========================

;Open windows
```

```
Data.List                                ;Open window with the source code
Register                                 ;Open register window
Trace                                    ;Open trace setup window
trace.list default ti.zero               ;Open trace results window

ENDDO
```

Now, you should have a window with results that could look like the one in Figure 10. We used the provided batchfile and loaded a sample program get_start.elf on our target. Of course, it is possible to include the commands for all the steps mentioned above in the batchfile.

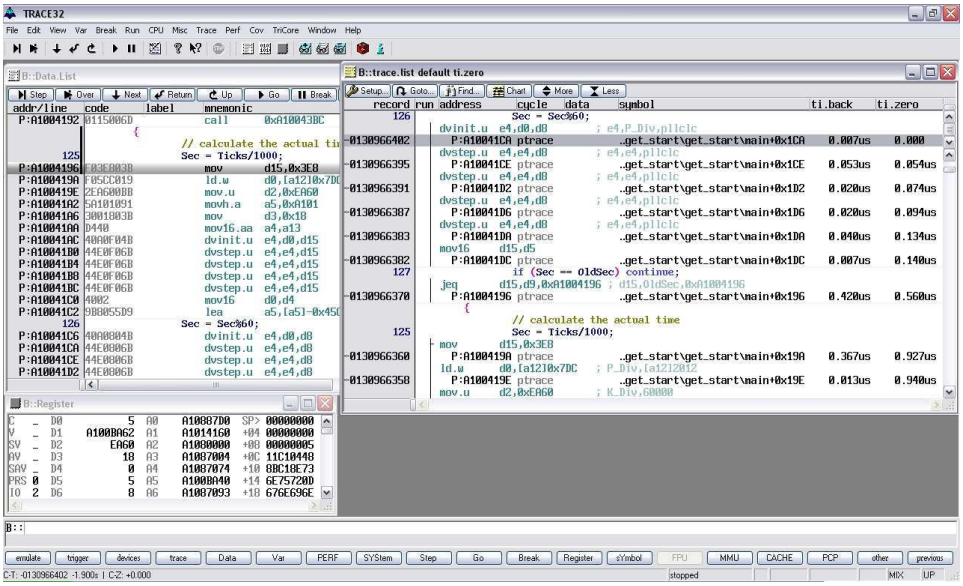


Figure 10: Application window with tracing results

Using the trigger and filter features described in 5.6.1, we can automate the procedure. See the batchfile that follows in Table 4.

Table 4: Automated tracing

```
WinCLEAR
RESet
SYStem.CPU TC1796
SYStem.Up
MAP.BOnchip 0xA0000000--0xA01FFFFF      ;Map target's flash
Analyzer.Clock 150.0MHz                 ;Specify the CPU clock frequency to improve
                                        accuracy (x4)

; OCDS L2 (Trace) enable (OCNTRL OSCU   configuration and control register)
```

```
Data.Set 0xF000047C %Long 0x0000003     ;trace source TriCore

;============================ ========================
; Memory initialization
; initialize external bus unit
Data.Set 0xF8000010 %Long 0x00F9FF68    ;EBU_CON
Data.Set 0xF8000020 %Long 0x001001D0    ;EBU_BFCON

; memory assignment
; AMD AM29BL162CB flash
Data.Set 0xF8000080 %Long 0xA2000051    ;EBU_ADDR_SEL0
Data.Set 0xF80000C0 %Long 0x00820000    ;EBU_BUSCON0
Data.Set 0xF8000100 %Long 0x23FF0100    ;EBU_BUSAP0
; ALLIANCE SRAM
Data.Set 0xF8000088 %Long 0xA1000071    ;EBU_ADDR_SEL1
Data.Set 0xF80000C8 %Long 0x00820000    ;EBU_BUSCON1
Data.Set 0xF8000108 %Long 0x00D80000    ;EBU_BUSAP1
; unused
Data.Set 0xF8000090 %Long 0x00000000    ;EBU_ADDR_SEL2
Data.Set 0xF80000D0 %Long 0x80928000    ;EBU_BUSCON2
Data.Set 0xF8000110 %Long 0xCFFFFFFF    ;EBU_BUSAP2
Data.Set 0xF8000098 %Long 0x00000000    ;EBU_ADDR_SEL3
Data.Set 0xF80000D8 %Long 0x80928000    ;EBU_BUSCON3
Data.Set 0xF8000118 %Long 0xCFFFFFFF    ;EBU_BUSAP3

;============================ ========================
; TC1796 bug fix: PMI line buffer not     invalidated during CPU halt

Data.Assemble 0xA1001000 debug16 nop    nop nop nop nop nop nop nop
Data.PROLOG.TARGET 0xA1001000++0x1F     0xA1001020++0x1F
Data.PROLOG.ON
Register.Set PC 0xA1001002              ;Set program counter to specified address
Step
;============================ ========================

;Open windows
Data.List                               ;Open window with the source code
Register                                ;Open register window
Trace                                   ;Open trace setup window
trace.list default ti.zero              ;Open trace results window

;Next command syntax: data.load.elf     \path_to_file\<filename>.elf
;In one line. In this case there is no space  so it is separated in two lines.
data.load.elf
c:\HighTec\TriCore\examples\TriBoard-TC1796\HelloSerInt\obj\hello.elf
```

```
Trace.Init                                  ;Clear the trace buffer
go main

trace.mode STACK                            ;Set trace mode to STACK
break.set _uart_getchar /TraceON            ;Enable trace in function _uart_getchar
break.set _uart_puts+0x8 /TraceOFF          ;Stop trace in line _uart_puts+0x8 in function
                                            _uart_puts

go                                          ;Begin program execution
ENDDO
```

The batchfile loads the application hello.elf, of the compiled example program HelloSerialInt from the Hightec tools, which were provided with the TC1796 starter kit. It can be found in menu "examples" for the TriBoard-TC1796.

6.1 Flow Errors

In our experiments, several times we encountered flow errors in our trace results. Most of the times, it was a single flow error in the beginning of the trace list.

What a flow error exactly means, is that the traced data are not consistent with the code in the target memories. A flow trace interface like the one TriCore has, requires access to the target memory (with the executed program code), so as to reconstruct the program flow correctly. A reason for a flow error could be that the memory contents have changed (e.g. self modifying code), or wrong trace data (as result of hard errors).

In our case, what was causing the flow error, was not clearing the trace buffer before executing our loaded application with "`Go`". Due to the bug fix for the TC1796 that is included in our batchfile, a single step is performed on the target, before even loading any program. This causes the program counter to change, but the trace disassembler is not informed for this change and it results in an error. If the trace memory is cleared right after the mentioned single step, the trace will not show any error. You can see some more reasons that cause flow errors for the TriCore architecture in [17].

In cases of flow errors the trace flow can be partially invalid until the next synchronization address (e.g. indirect branch). Such errors are important, because any further statistic analysis will probably fail. Single flow errors could be ignored, but they are not desired and should be handled if possible.

However, the Trace32 hardware represents the exact program flow, except in case of flow errors. In case of indirect branches the cpu puts out the full target address, so the program flow can be fully reconstructed.

7 Conclusion

The intention of this article, is help a user to understand the basic function of the Trace32 PowerTrace module and of the Trace32-ICD with trace extension software, so as to make the user able to perform a trace analysis on a TC1796 target. We briefly summarized the tracing function of the tools and provided information necessary, like how to connect the hardware, a very concentrated software user guide, and ways to automate the procedure.

A Appendix: Test the Trace32 System with While Loops

While we were experimenting with the PowerTrace, some questions arose: How accurate is the trace? Are the Trace32 software and hardware, able to trace conditional taken *and* non-taken branches? We decided to test the PowerTrace, using while loops, approaching the problem in the ways described in the following sections.

A.1 C While Loop

First, we wrote a while loop in C programming language, as shown in Table 5, which was correctly traced.

Table 5: C While Loop

```
int main (void){
int i=10;

while(i!=0){i–;}

return 0;}
```

The question that came up next, was whether the assembled code uses a conditional taken or non-taken branch to initiate the while loop. We disassembled the object file, and found out that it uses a conditional taken branch, as shown in Table 6. We then decided to proceed using our own assembly code, in two variations: one with a conditional non taken branch, and one with a conditional taken branch.

A.2 Assembly While Loop with a Conditional Non-Taken Branch

Using the code in Table 7, compiled with HighTec's tricore-gcc toolchain, we performed tracing analysis, and we came up with a flow error. As it turned out after some tests, the flow error was caused by not clearing the trace buffer after running the batchfile. More info in 6.1. After solving this issue, the while loop was traced correctly.

Table 6: Disassembled C While Loop

main.o:	file format elf32-tricore
Disassembly of section .text:	
00000000 <main>:	
0: 20 08	sub.a %sp,8
2: 6d 00 00 00	call 2 ¡main+0x2¿
6: da 0a	mov %d15,10
8: 78 01	st.w [%sp]4,%d15
a: 58 01	ld.w %d15,[%sp]4
c: ee 02	jnz %d15,10 ¡main+0x10¿
e: 3c 05	j 18 ¡main+0x18¿
10: 58 01	ld.w %d15,[%sp]4
12: 92 ff	add %d15,%d15,-1
14: 78 01	st.w [%sp]4,%d15
16: 3c fa	j a ¡main+0xa¿
18: 82 0f	mov %d15,0
1a: 02 f2	mov %d2,%d15
1c: 00 90	ret

A.3 Assembly While Loop with a Conditional Taken Branch

After checking that the PowerTrace can perform successful tracing analysis on C while loops and on assembly while loops that initiate using a non-taken branch, tracing the assembly program in Table 8, that uses a conditional taken branch to initiate like the disassembled C while loop, was also successful.

B Appendix: Save Trace Results in File Automatically

One of the goals of our experiments, was to completely automate the procedure of tracing and saving results in a file, having everything executed from a batchfile, i.e. a startup script. It turned out, given the current version of the Trace32 software, it is not always possible to do so.

The reason for that is, that in order to have the "`WinPrint`" command work correctly, you must specify the range you want to save in the file, as described in 5.6.2. When a batchfile is composed, the numbers that are going to be used for first and last records are not known, since they appear to be used kind of randomly, depending on the contents of the trace buffer. According to Lauterbach, there is still no keyword to specify the first and last record number that could be used in a batchfile. Their suggestion, is to use the following commands in the batchfile after command "`Go`":

Table 7: Assembly While Loop with a Conditional Non-Taken Branch

```
        .file "main.s"
        .section .text,"x",@progbits
        .align 1
        .global main
        .type main,@function
main:
        # arg_overflow_area = 0, pretend = 0
        # frame_size = 8, outgoing_args_size = 0
        # f_size = 8, frame_pointer_needed = 0
        # pool_size = 0

        sub.a %SP, 8                            # allocate space for locals
        call __main
        mov %d15, 10
.L1:
        jz %d15,.L2
        add %d15, %d15, -1
        j .L1
.L2:
        mov %d2, %d15
        ret
.Lfe1:
        .size main,.Lfe1-main
```

- "`Wait 1.s`": Wait for one second.
- "`Break`": Break the CPU.
- "`PRT.FILE <filename>`".
- Depending on the trace mode:
  - For FIFO trace mode:
    - "`LOCAL &nmb`": Declare the name of the variable.
    - "`&nmb = A.RECORDS()`": Set variable with number of records.
    - "`WinPrint.Analyzer.List (-&nmb)--(-1.)`": Use variable to print in file.
  - For STACK or LEASH trace mode:
    - "`LOCAL &nmb`": Declare the name of the variable.
    - "`&nmb = A.RECORDS()`": Set variable with number of records.
    - "`WinPrint.Analyzer.List (+1.)--(+&nmb)`": Use variable to print in file.
- "`PRT.FILE`".

Table 8: Assembly While Loop with a Conditional Taken Branch

```
        .file "main.s"
        .section .text,"x",@progbits
        .align 1
        .global main
        .type main,@function
main:
        # arg_overflow_area = 0, pretend = 0
        # frame_size = 8, outgoing_args_size = 0
        # f_size = 8, frame_pointer_needed = 0
        # pool_size = 0

        sub.a %SP, 8                          # allocate space for locals
        call __main
        mov %d15, 10
.L1:
        jnz %d15, .L2
        j .L3
.L2:
        add %d15, %d15, -1
        j .L1
.L3:
        mov %d2, %d15
        ret
.Lfe1:
        .size main,.Lfe1-main
```

During our experiments, those commands were not fully tested and verified. They worked in random occasions and the reason for successful use of the commands was not known.

References

[1] Product brief: TC1796. by Infineon Technologies AG, January 2004.

[2] Infineon Technologies AG, St.-Martin-Strasse 53, 81669 München, Germany. *TriCore^{TM} 1 32-Bit Unified Processor Core*, User's manual v1.3.5, volume 1(of 2) edition, October 2005. V1.3 Core Architecture.

[3] Infineon Technologies AG, St.-Martin-Strasse 53, 81669 München, Germany. *TC1796 32-Bit Single-Chip Microcontroller*, User's manual v1.0, volume 1(of 2) edition, June 2005. System Units.

[4] Infineon Technologies AG, St.-Martin-Strasse 53, 81669 München, Germany. *TriBoard TC179X Hardware Manual TC179X.300*, User's manual v3.2 edition, September 2006.

[5] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *TRACE32 Installation Guide*, July 2006. filename: installation.pdf.

[6] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *ICD Quick Installation*, November 2006. filename: icd_quick_installation.pdf.

[7] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *ICD Tutorial*, November 2006. filename: icd_tutorial.pdf.

[8] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *Training ICD Basics*, November 2006. filename: traning_icd.pdf.

[9] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *Training ICD Trace Extension*, November 2006. filename: training_icr.pdf.

[10] TRACE32 Online Help: Online Manuals and HELP System. Lauterbach Datentechnik GmbH, November 2006. filename: main.pdf.

[11] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *T32 Start*, November 2006. filename: app_t32start.pdf.

[12] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *IDE User's Guide*, July 2006. filename: ide_user.pdf.

[13] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *IDE Reference Guide*, November 2006. filename: ide_ref.pdf.

[14] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *IDE Functions*, November 2006. filename: ide_func.pdf.

[15] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *PRACTICE Script Language User's Guide*, November 2006. filename: practice_user.pdf.

[16] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *PRACTICE Script Language Reference Guide*, November 2006. filename: practice_ref.pdf.

[17] Lauterbach Datentechnik GmbH, Fichtenstr. 27, D-85649 Hofolding, Germany. *OCDS Debugger and Trace for TriCore*, November 2006. filename: debugger_tricore.pdf.