Putting Fixed Priority Scheduling Theory into Engineering Practice for Safety Critical Applications

N. C. Audsley, I. J. Bate and A. Burns

Department of Computer Science,
University of York,
York, Y01 5DD, UK.

Abstract

This paper describes the approach proposed by the York University Technology Centre (YUTC) for introducing fixed priority scheduling into industrial safety critical hard real-time systems. The work has been performed within the context of a class A (safety-critical) system as defined by civil aircraft software standard DO178B [1]. Traditionally, class A systems have been scheduled by a cyclic executive. However, many such systems can be re-designed using a fixed priority scheduler. This saves time and money, with no significant increase in risk. Also, significant technical benefits are apparent. This paper describes the timing requirements of the system, provides a potential scheduling approach (including appropriate timing analysis), and outlines an approach for gathering the necessary evidence for presentation to certification authorities.

1 Introduction

Recent years have seen an increasing demand for greater complexity / capability (more functionality) while increasing efficiency / effectiveness (better functionality and lower weight) in the aircraft industry. This has lead to avionics software systems of increased complexity and size, requiring more powerful computing platforms. Given that many of the software engineering techniques that have been used in the design process are immature (compared to other related disciplines, e.g. civil engineering), much effort has to be put into verification of the software prior to use within an airbourne system. Indeed, over 50% of the software effort for the Boeing 777 has been in the area of analysis and test for the purpose of verification [2].

Currently, safety-critical systems tend to be produced using cyclic executives. One reason for this is the stringent safety and integrity requirements placed upon software by avionics standards, such as DO178B [1]. Alternative implementation strategies have been proposed, including fixed priority scheduling. The latter has many advantages over the traditionally used cyclic executive scheduling approach [3]:

1. Provides a scheduler mechanism with greater flexibility so that the design process is cheaper and quicker. Indeed, Sha [4] states that:
   "*Under the cyclic approach, meeting the responsiveness, schedulability and stability requirements has become such a difficult job that practitioners often sacrifice program structure to fit the code into the right time slots.*"

2. Provides a more efficient scheduling mechanism. A commonly faced problem with many avionics projects is that of limited resources caused by the cost and availability of military specification components (the operating environment of aircraft require the use of such components) and incorrect system requirements (initial resource budgets are too optimistic).

3. Provides a sufficient and necessary analysis technique to reduce the burden of testing and maintenance.

There are some disadvantages in the adoption of fixed priority scheduling. One that is of interest to control engineers is that of jitter. Potentially, the release jitter of tasks scheduled with a fixed priority scheduler may be worse (than under cyclic scheduling) due to the dynamic nature of the run-time schedule.

Research performed at the University of York Real-Time Systems group into fixed priority scheduling has reached a stage where it was considered mature enough for use in safety critical systems. This paper will justify this claim, showing how the theory can be implemented in practice.

Control Requirement	Timing Constraints	Timing Requirement	Notation
Regular action	Periodic	Iteration rate	T
Event-driven action	Sporadic	Min. inter-arrival time	T
Action within bounded time	Responsiveness	Deadline	D
Variation from periodicity	Output jitter	Output jitter	J
Time of release after action	Offset from action	Task release offset	O
Function Y follows function X	Task Y executes after X	Precedence constraint	P = {X,Y}
Function Y follows function X within bounded time	Task Y executes after X within D	Precedence constraint with end-to-end deadline	P = {X,Y} within D
Active modes of function	Task executed in modes X, Y, Z	Task active/inactive in each mode	Mode list

Table 1: Relationship between Timing Constraints and Timing Requirements.

The remainder of this section provides background to DO178B. Section 2 of this paper will provide a brief description of the current method that is used by Rolls-Royce. Section 3 will present the YUTC approach to the design of a kernel, with section 4 giving timing analysis. Section 5 discusses an approach to certification. Section 6 presents the conclusions of the paper.

1.1 Background

Understandably, the regulatory authorities (the Federal Aviation Authority, the Civil Aviation Authority and the Joint Airworthiness Authority) impose many constraints on system designers of class A systems. Within DO178B [1], class A software is defined as:
"*software whose anomalous behaviour, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.*"
Catastrophic failure consists of failure conditions which could prevent continued safe flight and landing.

As stated earlier, class A systems have traditionally been implemented by cyclic executives. One of the motivations for this is that non-determinism is minimised – the regulatory authorities are able to verify via the scheduling table that appropriate timing requirements will be met. DO178B is supportive of this approach, stating that (section 5.2.2 in [1]):
"*The software design process should avoid introducing complexity because as the complexity of software increase, it becomes more difficult to verify the design and to show that the safety objectives of the software are satisfied.*"

To introduce fixed priority scheduling into class A systems we must ensure that the approach:

- is conservative, and hence understandable;
- is verifiable so that it can be shown the system requirements (including timing) are met;
- allows the gathering of evidence for the safety case. Four properties should be examined for the safety case: functionality (are the requirements met?); resource (does the system have sufficient resources?); timing (does the system meet its timing constraints?); and failure behaviour (how does the system behave in the event of failures?).

2 The Existing Method

This section examines how current systems that are controlled by a cyclic executives are produced. This will provide a benchmark to compare the fixed priority scheduling against. There are four particular issues to be discussed:

- the type of system to be scheduled;
- implementation of the cyclic executive scheduler;
- fault identification and recovery for the existing method;
- identification of the opportunities for improving the existing method.

2.1 The Types of System to be Controlled

The type of system to be controlled is a Full Authority Digital Engine Controller (FADEC), where:

- software is used to control the operation of the hardware to achieve a safe and an efficient response.
- critical functionality is policed by a device that provides fail safe protection. For example, hardware devices may be provided to protect against loss of control of fuel flow.

The data within the system is a mixture of continuous and discrete types that are updated either periodically or sporadically. The processing within the system is asynchronous in nature, however both hard and soft functionality exists. Table 1 explains the relationship between the system that is to be controlled and the requirements that affect the scheduler.

The key requirements that the system imposes on the kernel (scheduler) are:

- the provision of task offsets shall be enforced by the kernel;
- the dispatch of certain tasks must be performed in a particular order, i.e. precedence constraint;
- precedence constrained tasks may have an end-to-end deadline;
- some tasks have jitter constraints (particularly those dealing with input and output devices) requiring the use of an offset to ensure the task is executed when all preceding tasks have completed execution.

2.2 How the Currently Used Cyclic Executive Scheduler is Implemented

The cyclic executive currently used by Rolls-Royce typically consists of a minor cycle with a period of 25ms and a major cycle with a period of 200ms. Various forms of secondary scheduling (round robin, cyclic executive) are used to achieve the other iteration rates that may be required.

The scheduler is synthesised automatically on the first build of the software and then modified manually for subsequent builds. The reason for this approach is that the effect of modifications on the control flow can be determined more easily during regression testing.

Memory protection is not used between tasks or natural software partitions. Instead, a pull data model (i.e. only the tasks that owns the data item can write but all tasks may read the data) combined with good programming practice is used to prevent data corruption.

The Spark Ada subset [5] is used for the purpose of the implementation.

2.3 Fault Detection and Recovery

The fault tolerant properties of the current system can be separated into three distinct parts: redundancy, identification, and recovery. Redundancy is provided by the system consisting of two identical lanes that are connected by a communications channel which is used for data validation. There are two forms of fault identification in the system:

1. *Timing Watchdogs* – a timing watchdog is provided to identify whether cycle overruns occur and then raise a fault flag. The mechanism is discussed in more detail in section 3.3.

2. *Health Levels* – each lane maintains a health level value which is calculated by comparing a number of actual data values with expected data values (obtained by calculating data values using other sources or from the other lane). If the health level falls below an acceptable level then fault recovery is performed.

Fault recovery is, typically, achieved using the *fail stop* method where if the lane controls the engine then a lane change is performed, and the lane is reset.

2.4 Opportunities for Improvement

The opportunities for improvement with the current approach are believed to be:

- *Use of Analysis Techniques*
  Analysis rather than testing should be used to verify the timing requirements are met. Harter [HARTER87] states that analysis is preferable for verification rather than test since it is more efficient and more likely to detect problems. Analysis also provides a quicker and cheaper method of determining the effect of changes in the system requirements. The current approach provides little evidence of whether the system will meet the timing requirements until the first build has been produced. By the time of the first system build, significant commitments have been made to the hardware and software architectures are costly to change. A verification strategy is required that will allow both static and dynamic testing of the system to ensure that the scheduling requirements are met.

- *Implementation Requirements*
  An implementation that better reflects the requirements of the system. The cyclic executive scheduler has provision for a limited selection of iteration rates and no provision for sporadics which results in an inefficient (in terms of processor utilisation) implementation. However, this will require a more stringent requirements specification.

- *Maintenance*
  The maintenance of a heavily loaded (in terms of processor utilisation) cyclic executive scheduler is notoriously difficult resulting in frequent and time consuming reworks being necessary.

- *Better Management of Task Offsets*
  The cyclic executive scheduler does not provide an elegant mechanism for implementing tasks that have offsets. The approach taken with the cyclic executive scheduler is normally to place sufficient functionality between the tasks that have an offset.

3 Kernel Architecture

This section discusses the architectural choices made during the implemention of the fixed priority kernel. Issues and trade-offs related to the implementation are discussed.

There are three principle areas in which the fixed priority scheduler may require a different kernel architecture to the cyclic executive: scheduling policy, handling of sporadics, and handling of timing overruns. These are now discussed.

3.1 Scheduling Policy

Traditional fixed priority scheduling policy is implemented using a preemptive flow of control where the highest priority task is always executed. Often, fixed priority scheduling is accompanied by a mechanism for protecting shared data, e.g. priority ceiling protocol [6]. However, this is not required within the context of the system under consideration due to the data pull model, together with the timing characteristics of the system.

Fixed priority scheduling can lead to non-determinism – it is not clear where in their execution tasks will be preempted by the release of higher priority tasks, leading to an greatly increased number of possible execution scenarios. Also, data flows and updates can be interrupted – a task may be preempted when a data calculation is only partly finished. Both issues could be solved under a pre-emptive scheme, however, both application and kernel software would become more complex.

Alternatively, a non-preemptive flow of control can be used. The choice of non-preemptive is supported by DO178B which requires an implementation whereby the effects of interrupts can easily determined.

For non-preemptive fixed priority scheduling two principle variants can be identified:

1. *tick-driven*
   The tick driven approach uses a clock tick which interrupts task execution to release tasks. To achieve a non- preemptive flow of control when the task execution is interrupted by the clock tick, the task is immediately resumed when the scheduler has updated the run queue. This occurs even if a higher priority task has been released. The problem with this approach is that tasks which are not released at a rate which is a multiple of the clock tick will suffer from release jitter. If the release jitter becomes too great then task iteration rates or the release mechanism may have to be altered so that task deadlines can be met.

2. *co-operative*
   The co-operative scheduler uses a non-preemptive model with task releases dependent on a real-time clock rather than an interrupt source. When a task completes execution, the clock is read, the run queue is updated if a task should be released and the highest priority task is dispatched.

The exact variant of non-preemptive fixed priority scheduling chosen has three major impacts on the system:

1. *Hardware Architecture*
   The infrastructure of the system will be affected by the scheduling policy. The tick driven scheduler requires a hardware clock tick that periodically generates an interrupt. The co-operative scheduler requires a real-time clock that allows the decision of whether to release a task to be made. Therefore, the tick driven approach provides the more reusable approach since it can use the same hardware as the cyclic executive scheduler.

2. *Kernel Overheads*
   Whichever scheduling policy makes the most attempts to update the run queue will have the worst kernel overheads (assuming that the worst case search and release of runnable tasks takes an equivalent time for both schedulers) since an overhead is incurred independent of whether a task is released. A simple test can be derived for determining whether tasks may be ready for release in an attempt to reduce the overhead, i.e.

   - *Co-operative Scheduler Test*: store a value that is the time at which any task will next become runnable so that at the end of each task execution this value can be simply checked against the real-time clock.
   - *Tick Driven Test*: store a value which is the number of clock ticks before any task will become runnable so that when a clock tick arrives this value can be simply checked and recalculated.

3. *Responsiveness*
   The responsiveness (i.e. ability to meet short deadlines) is related to the maximum release jitter and the length of time tasks are blocked by the executions of lower priority tasks (and the length of time consumed by execution of the scheduling policy, which we will assume is the same for tick driven and co-operative). In general, the co-operative scheduler will be more responsive since the release jitter of the tick driven approach can be as large as the blocking time (due to lower priority tasks having to complete their execution), plus the clock tick rate (for tasks with iteration rates that are not multiples of the clock rate). However, the release jitter of the co-operative approach is simply the blocking time.

Hybrid Policy

An alternative approach that has been developed during discussion with software and hardware engineers is the hybrid of the tick driven and co-operative scheduler approach.

The hybrid approach releases the majority of tasks based on a clock tick with a few short period tasks released co-operatively. The benefit of this approach is that it allows a compromise between kernel overheads and task responsiveness. The reason is that between clock ticks, only those tasks that require a quick response need to be checked for release. Therefore, the kernel overheads each time a task finishes execution are reduced when compared with the co-operative approach. Also, the release jitter on the short period tasks benefits from the co-operative approach.

We note that jitter could also be reduced by a more frequent clock tick. However, this would increase the resultant overheads, implying a detrimental affect on the ability to schedule the task set. Also, the clock tick is fixed within the system once hardware platforms have been determined.

3.2 Sporadic Tasks

The scheduling policy will be dependent on the deadline of the sporadic task since the arrival of a sporadic cannot be determined, unlike a periodic, and therefore the worst case must be assumed. Without the use of sporadics the functionality must be modelled as a periodic (typically with an iteration rate less than the minimum inter-arrival time) to achieve the necessary deadline. This will lead to an increased worst case processor utilisation and a task set that is more difficult to schedule.

For certification reasons related to the determinism of control flow, the sporadic tasks shall not be implemented using the usual interrupt scheme. Instead, an event flag shall be raised when the sporadic should be released with the actual task being released when the scheduler next updates the run queue. The event flags would be set by the appropriate hardware devices and memory mapped so that the scheduler may read the value.

To protect against hardware failures causing the event flags to become stuck, a minimum inter-arrival time would be specified to prevent the sporadic being triggered at too fast a rate. The kernel will be responsible for policing that a sporadic is not triggered too fast, without this protection other tasks may miss their deadlines. In some cases which are chosen carefully, a maximum inter-arrival time may also be specified so that the task is called irrespective of the event flag condition. Obviously, the application of the maximum inter-arrival time must be carefully selected and controlled, i.e. an additional check may be required before any functionality is performed.

3.3 Timing Overruns

There are two principle approaches for providing protection against timing overruns which are the tick driven approach, and the task countdown timer approach.

The tick driven timing watchdog approach is where at a regular rate (normally triggered as a multiple of the clock tick rate) a check is performed to ensure that the software is not looping. This check would be based on ensuring that a sufficient number of tasks have been executed between checks (e.g. the sum of the worst case execution time of the tasks that have been executed is greater than the period of the timing watchdog). The response time of the tick driven timing watchdog to detectable faults is twice the period of the timing watchdog.

The task countdown timer approach is where each time a task execution commences a countdown timer is started. The duration of the countdown timer is greater than the Worst Case Execution Time (WCET) of any task in the task set. When the task execution is complete the countdown timer is restarted. If the countdown timer reaches zero then fault recovery is performed. Therefore, the response time to a fault is equal to the duration of the countdown timer.

3.4 Summary

From a technical perspective, the co-operative scheduling technique with countdown timer has the advantage that better responsiveness should be obtained and the lack of interrupts in the system makes

safety analysis simpler. However, cost and risk are determining factors so the tick driven scheduling approach with periodic timing watchdog affords greater reuse of the existing architecture. Therefore, to provide a flexible system with maximum reuse the hybrid approach with a tick driven watchdog is assumed

4 Timing Analysis

This section will show how existing analysis for fully preemptive scheduling can be adapted for the hybrid scheduling approach. A major part of this work is the trade-off of ability to understand the analysis and the pessimism that the analysis introduces. The reason for this approach is that the analysis technique which is used should be intuitively correct to the engineers responsible for the system and the certification authorities rather than an approach which requires complex proofs. The timing analysis is derived from the work of Burns [7] with a starting point of:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \qquad (1)$$

where for task τ_i, R_i represents the worst case response time, C_i the worst case execution time, and B_i the longest time τ_i can be blocked by a lower priority task. The function $hp(i)$ returns the set of tasks with higher priorities than τ_i. Hence, the summation term represents the time τ_i can be interfered with by higher priority tasks. We assume that sporadics can be treated as periodics with period equal to their worst-case inter-arrival time.

It is assumed that, for all τ_i, we have $C_i \leq D_i \leq T_i$.

The principal difference between the preemptive scheduling model and the non-preemptive scheduling model is the blocking time related to lower priority tasks. In a fully preemptive system the blocking time may be zero. However, in a non-preemptive system the blocking time is equal to the longest duration execution that can occur of lower priority tasks between task dispatches.

$$B_i = \underset{k \in lp(i)}{max} (C_k) \qquad (2)$$

where $lp(i)$ represents the set of tasks of lower priority than τ_i.

To solve equation 1 a recurrence relation is formed [7]:

$$R_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j \qquad (3)$$

The recurrence can be initiated with $R_i^0 = C_i$, and terminates if either $R_i^{n+1} > D_i$ or $R_i^{n+1} = R_i^n$.

This relation has been adapted by Audsley to account for the extra interference that may be caused by release jitter which may allow extra instances of higher priority tasks to be executed [AUDSLEY93]:

$$R_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n + J_j}{T_j} \right\rceil C_j \qquad (4)$$

For a periodic task the jitter, J_j, is given by:

$$J_j = T_{clk} - GCD(T_{clk}, T_j) \qquad (5)$$

where T_{clk} is the period of the clock and GCD is the greatest common denominator. For a sporadic task the jitter is given by:

$$J_j = T_{clk} \qquad (6)$$

Now, when the analysis converges (i.e. $R_i^{n+1} = R_i^n$), the worst case response time is calculated as:

$$R_i = R_i^{n+1} + J_i \qquad (7)$$

Now we turn to kernel overheads, namely context switches and implications of the scheduling scheme. The effect of context switches may be modelled using the knowledge that at most two context switches may occur per task. The analysis accommodates this by modifying the WCET of the tasks, i.e.

$$C_i := C_i + C_{s1} + C_{s2} \qquad (8)$$

where C_{s1} is the time taken to perform a context switch into the task, C_{s2} is the time taken to perform a context switch from the task.

The secondary category of kernel overhead to be analysed is the clock model which has two facets for the hybrid scheduler. Those related to the periodic and co-operative updating of the run queue. The co-operative clock model is modelled by modifying the context switch from the task, C_{s2}, to incorporate the time taken to update the run queue, i.e.

$$C_{s2} := C_{s2} + C_{coop} \qquad (9)$$

where C_{coop} can be founded with a WCET analyser.

The periodic clock for tick driven scheduling can be modelled in two ways [7]:

1. *Single Task*
   Here a single task, τ_{clock}, is used to model the updating of the run queue. Task τ_{clock} has the highest priority and the following timing constraints:

Task	T (μs)	C (μs)	D (μs)
A	6250	250	6250
J	11000	1000	11000
B	25000	4000	25000
C	50000	2000	50000
D	100000	1000	100000
E	200000	1000	200000
F	1000000	3000	1000000

Table 2 : Basic Task Set.

ID	P	B (μs)	J (μs)	R(μs)	Met?
clock	1	4000	0	6000	Yes
A	2	4000	0	6250	Yes
J	3	4000	6000	15500	No
B	4	3000	0	15750	Yes
C	5	3000	0	18750	Yes
D	6	3000	0	22000	Yes
E	7	3000	0	23000	Yes
F	8	0	0	23000	Yes

Table 3: Analysis for 6.25ms Clock and Single Task Model of Overheads.

$$T_{clock} = T_{clk}$$

$$C_{clock} = MC_{fixed} + MC_{first} + (N - M)C_{sub}$$

where N represents the number of actual tasks in the system, T_{min} is the shortest period of all tasks, C_{first} is the cost in transferring the first task from the delay queue to the run queue, C_{sub} is the cost in transferring the subsequent tasks from the delay queue to the run queue, and

$$M = \left\lceil \frac{T_{min}}{T_{clk}} \right\rceil$$

2. *Multiple Task*
   Here, an extra task for each actual task models the updating of the run queue. Each additional task has the same iteration rate as the actual task, and a WCET obtained via analysis.

Clearly there is a trade-off between understandability and pessimism of the two approaches. The multiple task approach offers greater accuracy with respect to response times, but is more complex.

ID	P	B (μs)	J (μs)	R(μs)	Met?
clockA	1	0	0	500	Yes
clockJ	2	0	6000	6750	Yes
clockB	3	0	0	1000	Yes
clockC	4	0	0	1250	Yes
clockD	5	0	0	1500	Yes
clockE	6	0	0	1750	Yes
clockF	7	0	0	2000	Yes
A	8	4000	0	7000	No
J	9	4000	6000	14250	No
B	10	3000	0	12250	Yes
C	11	3000	0	15000	Yes
D	12	3000	0	16000	Yes
E	13	3000	0	18250	Yes
F	14	0	0	18250	Yes

Table 4 : Analysis for 6.25ms Clock and Multiple Task Model of Overheads.

4.1 Example of the Analysis

This example shows the analysis given in the previous section, illustrating the effect of release jitter and implementation strategy on the ability to schedule a task set. Table 2 describes the task set that is to be scheduled. Note that the computation times given for each task include basic context switch overheads, as defined by equation (8).

Initially, we analyse the system assuming conventional tick driven fixed priority scheduling, using the single task clock model of overheads. Now, the task set of Table 2 is extended to include a high priority task τ_{clock} which includes all overheads due to the periodic clock. According to the single task model, τ_{clock} has the characteristics of:

- $T_{clock} = D_{clock} = 6250\mu s$
- $C_{clock} = 2000\mu s$

The computation time of the task is calculated assuming:

- $C_{fixed} + C_{first} = 500\mu s$
- $C_{sub} = 250\mu s$

The results of the analysis are given in Table 3. We note that τ_J will miss its deadline.

Now, we analyse the system assuming conventional tick driven fixed priority scheduling, using the multiple task clock model of overheads. Now, the task set of Table 2 is extended into Table 4 including, for each task, a corresponding task modelling its clock overheads (e.g. τ_A has associated task τ_{clockA}).

ID	P	C (μs)	B (μs)	R (μs)	Met?
A	1	750	4500	5250	Yes
J	2	1500	4500	7500	Yes
B	3	4500	3500	11000	Yes
C	4	2500	3500	15750	Yes
D	5	1500	3500	17250	Yes
E	6	1500	3500	18750	Yes
F	7	3500	0	18750	Yes

Table 5: Analysis for a Co-operative Scheduler.

The computation time of τ_{clockA} is given by: $C_{fixed} + C_{first} = 500\mu s$. All other extra tasks can only be released during the same context switch that releases τ_{clockA} (since they have periods that are multiples of T_{clockA}). Hence, they all have computation times equal to $C_{sub} = 250\mu s$. The results of the analysis are given in Table 4. We note that both τ_A and τ_J will miss their deadlines.

Now we consider scheduling the task set using the pure co-operative approach. Initially, we must amend the computation times of the tasks (as given in Table2) to account for the cost of updating the run queue. This is given by equation (9). We assume that $C_{coop} = 500\mu s$. Note that there will be no jitter, due to the co-operative scheduling and that all blocking times are also increased by C_{coop}. Table 5 shows the results of the analysis, with all tasks meeting their deadlines.

Now we consider the hybrid scheduling approach. To eliminate jitter, we assume that τ_A and τ_J are scheduled co-operatively, and the remainder of the tasks are tick-driven. The clock has period of 25ms, sufficient to schedule tasks with no jitter. A single task, τ_{clock}, with the highest priority is used to account for all tick-driven overheads. It has parameters:

- $T_{clock} = D_{clock} = 25000\mu s$
- $C_{clock} = 1500\mu s$

The computation times of τ_A and τ_J are increased (over their values in Table 2) to include C_{coop}. Note that their blocking times do not change. However, since τ_{clock} is the highest priority task, it interferes with τ_A and τ_J. The results of the analysis are given in Table 6 – all tasks are schedulable.

Out of the tick-driven approaches, task τ_J is only schedulable using the hybrid approach. In particular, the worst case response time of τ_J is considerably better using the hybrid approach. Experimentation has been performed to examine the effect of changing the clock tick rate and/or raising the iteration rate of τ_J. The purpose of the experimentation is to determine whether a purely tick driven scheduling approach may result is a schedulable task set. To date, a schedulable solution has not been found with the tick driven approach.

ID	P	C (μs)	B (μs)	R (μs)	Met?
clock	1	1500	0	1500	Yes
A	2	750	4000	6250	Yes
J	3	1500	4000	8500	Yes
B	4	4000	3000	13750	Yes
C	5	2000	3000	15750	Yes
D	6	1000	3000	16750	Yes
E	7	1000	3000	17750	Yes
F	8	3000	0	17750	Yes

Table 6: Analysis of Hybrid 25ms Scheduler with Single Clock Model of Overheads.

Summary

The examples of the analysis show that the hybrid approach provides better responsiveness over the tick driven approach for systems where flexibility (i.e. ability to have a wide range of iteration rates) is required. Similarly, the response time of sporadics with this scheme would be better since they would suffer no release jitter only blocking. Further work is required to generalise and optimise the hybrid scheduling approach.

5 Certification

The purpose of this section is to discuss how the YUTC may satisfy the certification authorities that the fixed priority scheduler provides a scheduler with at least the same integrity as the cyclic executive. The table in Table 7 summarises a safety analysis that has been performed using a technique described by Burns [2]. The principle behind the technique is that the four principle properties (functionality, resource, timing, safety) are examined and recorded for whether the property is met, the nature of the evidence and the assumptions that are made.

Table 7 shows that both static analysis and dynamic testing has been performed to show the correct operation of the system, and to bound memory and processing resource usage. This assumes fault-free conditions. Failure analysis has been performed that shows detectable timing faults are recovered assuming the timing watchdogs do not also fail. For the systems in question, hazards caused by dual random failures are normally accepted as being sufficiently remote, assuming there is no possible common cause and individual failures have a sufficiently low probability.

Property	Nature of Property	Nature of Evidence	Assumptions
Functionality	Dispatcher Correctness (tasks are scheduled at the correct rate and the correct order) Operational Correctness (system invariants are not affected)	Tests using the actual hardware and software showed that the dispatcher met the requirements, The scheduler is produced using statically stored variables – invariants should not be affected.	The kernel infrastructure operates in fault-free conditions. The scheduler is produced to an appropriate standard.
Timing	In all cases the performance of the system is deterministic and schedulable.	Schedulability analysis has been performed that shows the system is schedulable (section 4).	The kernel infrastructure operates in fault-free conditions.
Resource	Memory usage is deterministic and within the allowable limits.	The scheduler is produced using statically stored variables – resource usage proportional to the (static) number of tasks.	The kernel infrastructure operates in fault-free conditions.
Failure Behaviour	In the event of a timing overrun within the system, the fault will be identified within the appropriate time.	The tick driven timing watchdog is the same as that used for the cyclic executive scheduler which has previously been tested. Appropriate hazard analysis of the timing watchdog also exists.	The timing watchdog does not also fail.

Table 7 : Summary of the Kernel Safety Analysis.

6 Conclusions

This paper has shown how a fixed priority kernel can be produced with the minimum rework of an existing system that will meet the certification requirement of class A systems providing a cost effective and technically better scheduling method. Hazard analysis has been used to show that no additional hazards are introduced than exist with the cyclic executive scheduler. A safety analysis technique has been used to gather evidence that can be used as part of the certification case.

A number of kernel design approaches have been examined which have trade-offs between responsiveness and system reuse. The approach chosen will depend on the commercial and technical constraints of the system that is being developed. However, an interesting compromise of tick driven and co-operative scheduling approaches has been identified that provides benefits over the purely tick driven and co-operative scheduling approaches. Further work is required to generalise and optimise the hybrid scheduling approach. The YUTC expect that 1996/7 will see the first "real" practical application of the techniques described in this paper to control an aircraft engine.

References

[1] RTCA Inc., "Software considerations in airborne systems and equipment certification," DO-178B/ED-12B, December 1992.

[2] A. Burns and J. A. McDermid, "Real-time safety-critical systems: Analysis and synthesis," *Software Engineering Journal*, pp. 267–281, November 1994.

[3] C. D. Locke, "Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives," *Real-Time Systems*, vol. 4, no. 1, pp. 37–53, March 1992.

[4] L. Sha and J. B. Goodenough, "Real-time scheduling theory and Ada," *IEEE Computer*, April 1990.

[5] B. A. Carre, *SPARK: The SPADE Ada Kernel (v3.1)*. Program Validation Ltd., 1992.

[6] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronisation," *IEEE Trans. on Comp.*, vol. 39, no. 9, pp. 1175–1185, Sept. 1990.

[7] A. Burns, "Preemptive priority based scheduling: An appropriate engineering approach," in *Advances in Real-Time Systems* (S. Son, ed.), 1993.