Concerning Interruptions

Stephen B. Jenkins
Institute for Aerospace Research
National Research Council of Canada

Introduction

There's a simple, harmless sounding, four-syllable question that causes much wailing and gnashing of teeth among programmers: "Got a minute?" I always cringe if I hear those words while I'm coding, because by the time I've looked up and answered "No!", the damage is often already done.

A couple of decades ago, near the beginning of my programming career, I used to think that I was unusually sensitive to disturbances, and thus that it was up to me to overcome this "intolerance of interruptions". In the years since, I've come to realize that the effects of being interrupted are a problem for the vast majority of programmers. But I don't expect you to take my word for it. To properly set the stage for the rest of the discussion, I'd like to quote a few of the things that other, far more qualified people have written about interruptions.

First, on the topic of the frequency of interruptions in the workplace:

> *My informal observations in the US indicate that personnel in shared work spaces experience some kind of interruption every 10 minutes.*[1]

> *We found that…people spend on the average eleven and a half minutes in continuous work on a project or theme before they switch to another.*[2]

> *On average the subjects were being interrupted just over 4 times every hour.*[3]

> *Participants in our study reported an average of 50 task shifts over the week.*[4]

Now, you may be saying to yourself: "So what? Everyone gets interrupted. What makes programmers so special?"

> *Programmers […] have to build up a complex mental model of the programming problem, the state of the different variables, and so on. That's why most people do not make good programmers, but it is also why even the best programmer gets into trouble when he or she is interrupted. Bang! Down falls the carefully constructed mental house of cards.*[5]

This destruction of "complex mental models" has some rather serious repercussions. On the subject of the effect of interruptions on a programmer's productivity:

> *DeMarco reports that the recovery time after a phone call is at least 15 minutes. Even though we could not measure recovery time exactly, we believe his estimate to be valid. If more than 10 interrupts occur during a day, the time between interrupts becomes too short to accomplish product development work.*[6]

> *I maintain that programming cannot be done in less than three-hour windows. It takes three hours to spin up to speed, gather your concentration, shift into "right brain mode", and really focus on a problem.*[7]

> *What drives me crazy is that ever since my first job I've realized that as a developer, I usually average about two or three hours a day of productive coding.*[8]

> *The mere prospect of being interrupted is enough to prevent hackers from working on hard problems.*[9]

The toll on a programmer's mental state due to the cumulative effect of all of these interruptions is rather disturbing:

> *…an average worker's functioning IQ falls ten points when distracted by ringing telephones and incoming emails. This drop in IQ is more than double the four point drop seen following studies on the impact of smoking marijuana. Similarly, research on sleep deprivation*

Published (abridged) in *IEEE Computer*, Vol. 39, No. 11, November 2006.

> *suggests that an IQ drop of ten points is equal to missing an entire night of sleep.*[10]
>
> *Many programmers appear to be continually frustrated in attempts to work. They are plagued by noise and interruption, and pessimistic that the situation will ever be improved. The data recorded about actual interruptions supports the view that the so-called "work-day" is made up largely of frustration time.*[11]

The fact that the last quote is from a study published more than 20 years ago unfortunately validates the pessimistic outlook of those interviewed by DeMarco and Lister. What was true in 1985 is still true today: the mass of programmers lead lives of quiet frustration (with apologies to Thoreau).

I hope this short introduction has been sufficient to convince you of the seriousness of the problem. As programmers, many of us – especially those who have worked on low-level code or real-time systems – are well aware of the problem of time lost due to context switches in our operating systems, and how to minimize its impact. We need to become even more aware of, and concerned about, the high cost of cerebral context switches.

Severity of an interruption

I've identified four separate factors that significantly affect the severity, in terms of time lost, of an interruption: its "time", the person causing it, its subject, and the type of work it disrupts. I should point out that the following comments are not the result of any in-depth, multi-person, time and motion study I've performed (Dammit Jim, I'm a programmer, not an efficiency expert!). Rather, they are merely personal observations. Even assuming the most conservative rate of interruption, though, I've probably had to deal with more than 50,000 interruptions during my 20+ years as a professional software developer. That has to count for something.

Time of interruption

By "time of interruption" I mean both its length and its timing relative to other events. While many might think that the length of an interruption is one of its most defining characteristics, my own experience is that a "mental core dump" happens within the first few seconds. After that, the length of time is rather insignificant. The shortest types of interruptions however, someone else's phone ringing or some noise in the hallway, are really only momentary distractions. For some people, they seem to cause little or no problems, while for others they are a source of considerable disruption. It would appear that this is a personal matter, not unlike being a light or heavy sleeper, and beyond conscious control. The timing of an interruption though, influences its severity to a much greater degree. As seen in the introduction, interruptions that occur within 15 or 20 minutes of each other can cause your productivity to fall to near zero.

Person causing the interruption

Depending on your work environment, the person causing the interruption can have a variable effect on its severity. By using some of the techniques in the next section, it is usually possible to reduce the problems caused by intrusions from subordinates and even peers, but it may not be as easy to keep your supervisor (or anyone higher up the food chain) from interrupting you.

Subject of the interruption

The subject of the interruption can significantly affect its severity. While a simple question, such as "Going for coffee?" may not disturb you very much, someone dropping in for a chat about the weekend can cause real damage. I personally find that technical questions are far worse though. Especially those that require some degree of mental effort such as "Do you remember that little app you wrote last year…" The worst types of questions are those that require a complete mental context switch: "I need help debugging my [*insert programming language you haven't used regularly in over 10 years*] code and the client is waiting down the hall; he's really annoyed!" (Lest you think this a rather contrived example, I'd like to point out that this exact scenario happened to me recently.)

The type of work being interrupted

I believe that the type of work being interrupted is the single most critical factor influencing the severity of the damage done. While it seems that writing documentation or doing background research are able to withstand interruptions fairly well, designing algorithms and generating code are far more fragile endeavours. In my personal experience, the worst time to be interrupted is while refactoring code. If I'm in the middle of a large cut-and-paste session, moving and consolidating multiple blocks of code, even a trivial interruption can completely destroy my train of thought. This means not only losing the time to get that train back on the rails, but can also mean the loss of a significant portion of the work done in the minutes before the interruption occurred.

Dealing with interruptions

In dealing with interruptions, I recommend a two-pronged approach: reduce their number and reduce their severity. To that end, I've provided a few suggestions of things you might try; they have worked for me in the past and will work for you as well, hopefully. If the corporate culture where you work is unnecessarily interrupt driven, you'll be doing everyone a favour by bringing about change. (Although they may not thank you for it. Not only are there many people who dislike and resist change, but there are some who actually enjoy the uncertainty and lack of structure inherent in an environment filled with interruptions.)

Reducing their number

Obviously, the best way to deal with an interruption is to avoid it altogether. Here are several things that you can do to try and reduce the number of interruptions to which you are subjected.

1) Disseminate information.
   Generate your own FAQ list, and get others to use it. I created an HTML page titled "README if SBJ is Away" and placed it on an internal web server. Due to my use of the other six interrupt reduction techniques in this list, people tend to go to that page even when I'm not away.

2) Educate your coworkers, managers, admin assistants, etc.
   Be sure to let people know not only how harmful interruptions are, but also when the best and worst times for you to be interrupted are.

3) Communicate using email.
   Rather than using the telephone or talking face-to-face, encourage people to contact you via email and respond to them that way too. There are several benefits to be had. First, it allows you to choose the best time to read and respond. Email also tends to be more succinct, and less prone to vague, rambling interactions. Lastly, the extra work required in writing an email will filter out many unimportant interruptions; people will opt to RTFM ("read the fine manual") or STFW ("search the fine web") instead of bothering you.

4) Isolate yourself physically.
   If you have a door, close it. If you're not allowed to close it due to departmental policies, use a signal of some kind – a sign, a hat over the doorknob, etc. – to let people know that they're not welcome. If you don't have a door because you are in a cubicle, there are still options. Can you move the partitions around? Can you change your orientation, relative to the opening of your cube, to cut down on the number of passers by that decide to drop in for a chat?

5) Isolate yourself electronically.
   Get Caller-ID and voice mail, and use them. Resist the urge to pick up the telephone merely because it's ringing. Disable email notifiers. Wear headphones – even if you don't listen to music! Force people to work hard if they want to interrupt you.

6) Deviate from normal work hours.
   Modifying your hours will allow you to do the things that are the most sensitive to interruptions when others are not around. I suspect this is one reason why so many programmers prefer to work "odd" hours. While the stereotype is to come in late – "at the crack of noon" – and leave very late, I personally prefer to arrive at the office very early: typically 6:00 AM. Yes, that's a *six*.

7) Remonstrate against interruptions.
   At a minimum, practice forming a nasty scowl, and use it whenever anyone comes into your workspace to interrupt you. Cultivate a reputation as a curmudgeon. If being nasty isn't in keeping with your character, try a pained, exasperated look accompanied by a heavy sigh. Or, try rolling your eyes towards the heavens while muttering "How long, O Lord?" If you are interrupted to deal with something that is documented in your FAQ, be sure to let the person know, in some suitably unsubtle manner, that they have bothered you needlessly.

Reducing their severity

For those interruptions that are unavoidable, here are some suggestions that may help you to reduce their severity.

1) Consolidate interruptions.
   Combine interruptions and deal with several at once. Try to foresee other disruptions from the same person and deal with them right away. Respond to email or phone messages right after another interruption has already done its damage. If possible, go to lunch/coffee early or late, depending on the timing of other intrusions on your concentration.

2) Anticipate interruptions.
   If you're on a roll, and you know that you are coming up to a forced interruption (a meeting, lunch, etc.), try to break at a logical point at least five minutes before you need to leave. Use the

time to make a few quick notes, right in the code if nowhere else, of what you were intending to do next; it will be much easier to get started again when you return. This is especially true if you can deliberately leave a simple or enjoyable task with which to begin your next programming session.

3) Compensate by changing the way you work.
If possible, try modifying your development style to a more agile approach. Using a rapid design-write-test cycle can reduce the complexity of the mental model required, minimizing the damage caused by individual interruptions.

4) Procrastinate until a better time.
Once you've been disturbed, if you expect further interruptions in the near future, don't try to write code. You'll only frustrate yourself. Leave the thought intensive stuff for another time. Consider doing something other than programming. Write some documentation. Read that journal/textbook/trade magazine you've been meaning to get to.

5) Capitulate to the situation.
Relax and take a deep breath. Go with the flow. Don't beat yourself up about losing your concentration and needing 15 minutes to get it back – it's normal. If you're not convinced that it's normal, read the papers and articles referred to in the introduction. Getting stressed-out about the loss of productivity will only make things worse.

If all else fails

As a last resort, if you are often interrupted and are unable to reduce the frequency or severity to a level that is acceptable, you may want to consider looking for work elsewhere. While that may sound rather drastic, the stress caused by high levels of frustration can lead to significant health problems. Life is too short to spend 40 or more hours a week doing something that leaves you tense and upset. Even if you decide not to take a job elsewhere, merely going through the process and knowing that you have a way out (a current resume, marketable skills, etc.) may help to lower your anxiety level.

The worst-case scenario: deliberately planned, chronic interruptions

So far, I've only discussed unplanned interruptions, but what about planned interruptions? Obviously the majority of the coping strategies of the previous section cannot help in this case. After all that has been written about the harm caused, you may be wondering "Who on earth would deliberately plan to have programmers chronically interrupted?" Unfortunately, this is effectively what happens when a developer is required to work on two or more projects simultaneously.

> *I feel like when I have two programming projects on my plate at once, the task switch time is something like 6 hours. In an 8-hour day, that means multitasking reduces my productivity to 2 hours per day. […] In fact, the real lesson from all this is that you should* ***never let people work on more than one thing at once.***[12] [Original emphasis]

I realize that sometimes, operational requirements dictate that multiple projects absolutely have to be done concurrently. Just keep in mind the dangers of juggling several things that are costly, dissimilar, and fragile. (A good mental picture might be to imagine your self wearing a harlequin suit, and simultaneously tossing a Tiffany lamp, a Swarovski vase, and a widescreen laptop into the air.) The simplest way to dramatically improve the productivity of programmers that must undertake multiple projects is to allow them to work on those projects serially (at least one full day on each), rather than in parallel.

In defense of interruptions

Despite everything that's been said so far, interruptions are not always undesirable. Being interrupted by a fire alarm is a bad thing when they're just testing the system; it's a good thing when there's a fire. Also, what is time lost to you may be a net gain for your organization. From a systemic point-of-view, the interruptor may be helped more than the interruptee is harmed.

If the times of frequent interruptions happen only occasionally, they can be turned into a benefit by prompting you to "shift gears" and do something other than creating code. I actually wrote the majority of this article during a week when I knew that I'd have a constant stream of interruptions. I work at a research laboratory and because the person responsible for the day-to-day software needs of the experiment was away on vacation, I was expected to take over most of his duties. Somewhat ironically, the knowledge that I would be continuously interrupted, and thus not very productive as a programmer, encouraged me to use the time to write about the problem of interruptions.

Concluding remarks

Now that you've read almost all the way through this article, I have a confession to make. I didn't really write this for you (programmers) to read – that would be "preaching to the choir". I wrote it for your subordinates, coworkers, and supervisors: please share it with them; it might just make your life a little better. In fact, if I want to be 100% honest, I wrote this for *my* subordinates, coworkers, and supervisors to read. It just shows you the lengths to which I'm willing to go in order to reduce the number of interruptions I get.

References

1. C. Jones, *How Office Space Affects Programming Productivity*, IEEE Computer, Vol. 28, No. 1, Jan. 1995, pp. 76-77.

2. V.M. González and G. Mark, *"Constant, constant, multi-tasking craziness": Managing Multiple Working Spheres*, CHI'04 – Conference on Human factors in computing systems, ACM Press, April 2004, pp.113-120.

3. B. O'Connail and D. Frohlich, *Timespace in the workplace: dealing with interruptions*, CHI '95 – Conference on Human Factors in Computing Systems, ACM Press, May 1995, pp. 262-263.

4. M. Czerwinski, E. Horvitz, and E. Wilhite, *A Diary Study of Task Switching and Interruptions*, Proceedings of CHI 2004, ACM Conference on Human Factors in Computing Systems, Apr. 2004

5. J. Nielsen, *Are developers people?*, May, 2001; http://www-106.ibm.com /developerworks/library/it-nielsen4/?dwzone=ibm

6. R. van Solingen, E. Berghout, and F. van Latum, *Interrupts: Just a Minute Never Is*, IEEE Software, Vol. 15, No. 5, Sept. 1998, pp. 97-103.

7. O. Eichhorn, *The Tyranny of Email*, Mar. 2003; http://www.w-uh.com /articles/030308-tyranny_of_email.html

8. J. Spolsky, *Fire And Motion*, Jan. 2002; http://www.joelonsoftware.com /articles/fog0000000339.html

9. P. Graham, *Great Hackers*, July 2004; http://www.paulgraham.com/gh.html

10. Hewlett-Packard Press Release, *Abuse of technology can reduce UK workers' intelligence*, Apr. 2005; http://h40059.www4.hp.com/ featurestories/pdf/Info-Mania_PressRelease.pdf

11. T. DeMarco and T. Lister, *Programmer performance and the effects of the workplace*, Proceedings of the 8th International Conference on Software Engineering, Aug. 1985, pp.268-272.

12. J. Spolsky, *Human Task Switches Considered Harmful*, Feb. 2001; http://www.joelonsoftware.com/ articles/fog0000000022.html

Bio

Stephen B. Jenkins is the senior programmer/analyst at the Aerodynamics Laboratory of the Institute for Aerospace Research, National Research Council of Canada. For the past decade, he has specialized in developing web-based software tools using open source software. Feel free to send email; just don't expect him to respond immediately.

For more information, go to `www.erudil.com`