Lecture 20 — Real-Time Scheduling

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

March 10, 2024

Real-Time Scheduling

Real-Time Scheduling

Real-Time scheduling is just scheduling for real-time systems.

But what is a real-time system?

Supposed to respond to events within some real (wall-clock) time.

There are deadlines, and there are consequences for missing deadlines.

Fast is not as important as predictable.

The term task is used to refer to something that needs doing.

Yes, the scheduler operates on threads rather than specific things to do.

Let’s say each task corresponds to a thread trying to some work.

Task Types

Not quite… Hard, Firm, Soft.

Hard and Soft Real-Time

Hard real-time: it has a deadline that must be met to prevent an error, prevent some damage to the system, or for the answer to make sense.

If a task is attempting to calculate the position of an incoming missile, a late answer is no good.

A soft real-time task has a deadline that is not, strictly speaking, mandatory; missing the deadline degrades the quality of the response, but it is not useless.

Firm

Firm is about severity of consequences.

A firm deadline is one in which the response is useless if it arrives a little too late.

A hard deadline is one in which the system itself fails if it doesn’t meet the deadline.

Desktop OSes and Real-Time

Most of the operating systems you are familiar with (standard Desktop/Server Linux, Mac OS, Windows) are not very suitable to real time systems.

They make few guarantees, if any, about service.

When there are consequences for missing deadlines, this kind of thing matters.

Userspace?

Remember Java's "stop the world" garbage collector scenario.

Why is the OS not suitable for real-time operations?

Arbitrary Timelines

How long does it take to run a system call?

If you measure it, is that the average scenario? Worst-case?

Interrupts? Concurrency-control mechanisms?

It’s Different

Much of the difference of real-time systems is in scheduling.

Hard Real-Time Failure

If a task is hard real-time, there are two scenarios in which it might not complete before its deadline.

The first is that it is scheduled too late; like an assignment that will take two hours to complete being started one hour before the deadline.

Hard Real-Time Failure

If that is the case, the system will likely reject the request to start the task, or perhaps never schedule the task to run at all.

Why waste computation time on a task that will not finish in time?

… Could it have been completed if scheduled better?

Hard Real-Time Failure

The second scenario is that at the time of starting, completion was possible.

For whatever reason (e.g., other tasks with higher priority have occurred) it is no longer possible to meet the deadline.

Hard Real-Time Failure

In that case, execution of the task may be terminated partway through so that no additional effort is wasted on a task that cannot be completed.

… Could this task have been completed if scheduled better?

Properties of Real-Time Systems

Real-time systems are considered to be unique in five key areas:

1. Determinism
2. Responsiveness
3. User (administrator) control
4. Reliability
5. Fail-Soft operation

Determinism

Determinism: operations are predictable.

Perfect determinism probably won’t happen; just need guarantees.

Nondeterminism?

Nondeterminism isn’t necessarily bad: e.g., caching.

The presence of caching makes the worst case no worse and the best case much better – this sounds like free performance to me!

Can you think of something else?

Responsiveness

Responsiveness includes not only the time to execute the interrupt handler…

Also the time it takes to start the interrupt handler and what happens if the interrupt is itself interrupted by another higher-priority interrupt.

Administrator Control

Administrator control could be much less or much more!

Typical OS is somewhere in between.

Fail-soft: do our best even if it's not possible to succeed at all tasks.

Scheduling is Central

Scheduling Algorithm Selections

Choosing the wrong algorithm guarantees failure.

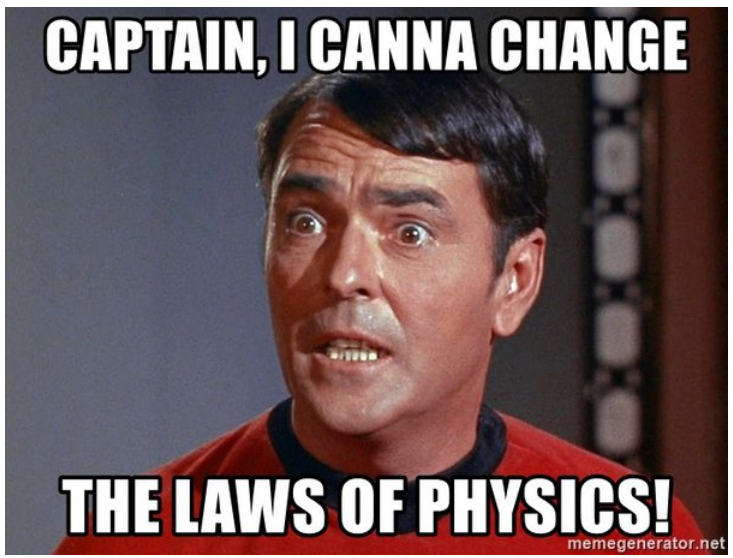
Scheduling algorithms we've discussed so far are inadequate.

Fairness and minimizing average response time are irrelevant.

Stay On Target!

The goal is to make sure hard real-time tasks complete.
And as many as possible of the soft real-time tasks.

Right algorithm? Success is possible but not guaranteed.



Preemption is the Priority

Non-preemptive algorithms won't work here.

Immediate preemption makes sense.

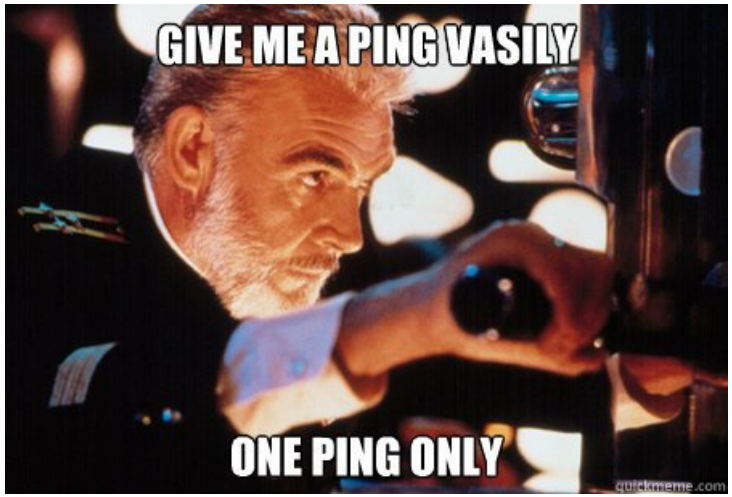
Why, when we can make timeslices super small?

What Kind of Tasks?

Every task can be categorized as one of the four kinds:

1. Fixed-Instance
2. Periodic
3. Aperiodic
4. Sporadic

Fixed Instance



This kind of task happens only a fixed number of times (usually once).

Periodic Tasks

If a task repeats at regular intervals, it's periodic.

Periodic Tasks

Periodic tasks have two relevant attributes:

: τ_k, the period;

c_k, the worst-case computation time

We can calculate the utilization of periodic tasks:

$$U = \sum_{k=1}^{n} \frac{c_k}{\tau_k}$$

Overloaded

If $U > 1$, it means the system is overloaded: there are too many periodic tasks.

We cannot guarantee that the system can execute all tasks and meet the deadlines.

Otherwise, we can devise a schedule.

Only Periodic?

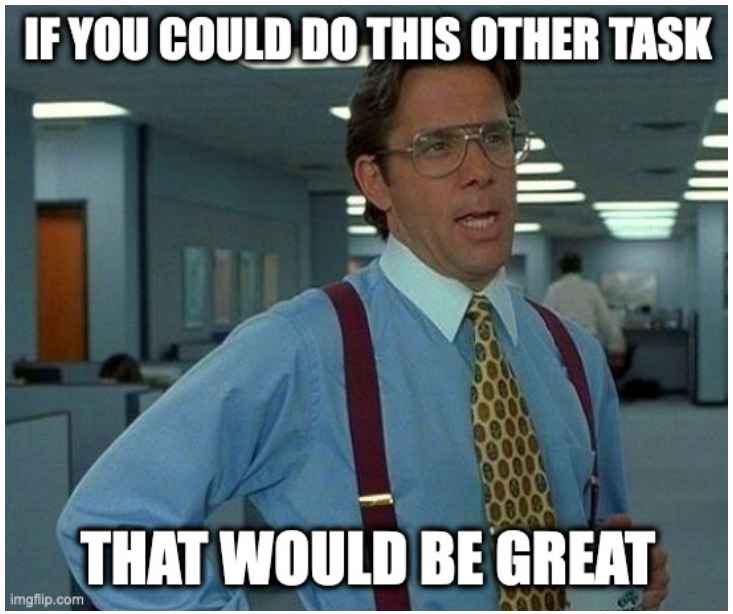
If the only tasks in the system are periodic ones, then we can create a fixed schedule for them.

Example: classes at UW.

But life is rarely so nice.

Aperiodic Tasks

Aperiodic tasks occur irregularly with no minimum interval.



If we expect an average arrival rate of 3 requests per second, there is still a 1.2% chance that eight or more requests appear within 1 second

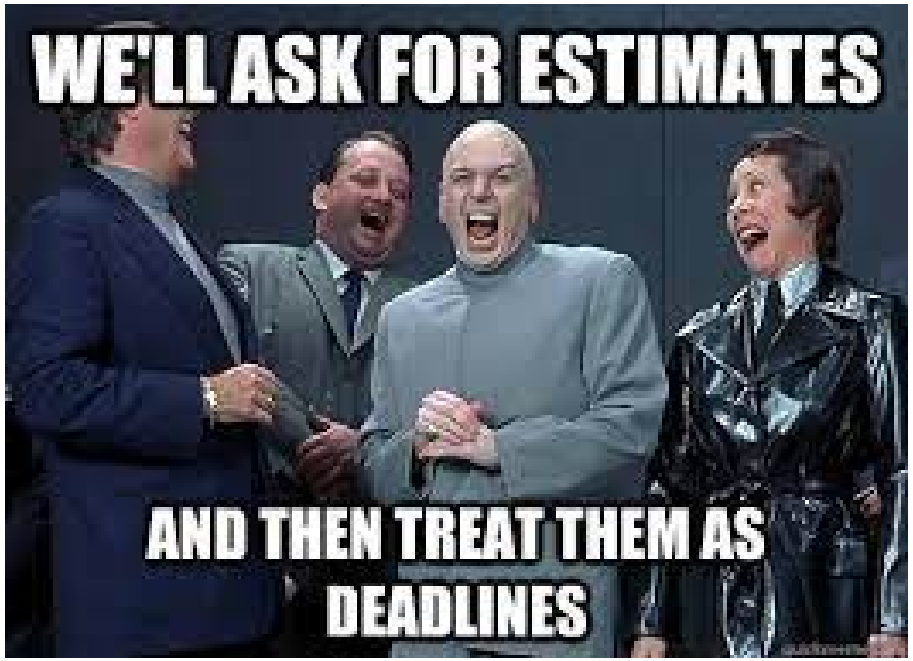
Sporadic Task

Sporadic tasks are aperiodic, but they require meeting deadlines.

We need a guarantee that there is a minimum time τ_k between occurrences of the event.

They can overload the system?

Where do task execution times come from?



Worst Case Scenario

We always assume the worst case scenario!

There's two ways to do it: source code analysis and empirical testing.

Err on the side of caution… More money? Worse battery life?

Airline Example

Caused by lack of capacity…

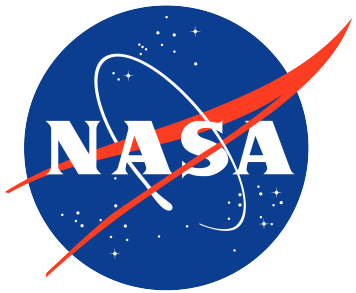
Code Analysis

Code analysis is exactly what it sounds like.

Try to figure out how long the execution takes on the longest path.

Overestimate; ignores pipelining and compiler optimizations…

Estimation is Hard



NASA/JPL Guidelines:

1. No recursion and no `goto`
2. Loops must have a fixed bound
3. No dynamic memory allocation after init

Empirical Observation

If you already have a version n of the system and want to build version $n + 1$: extrapolate?

Otherwise, simulate.

Simulation must be representative or misleading.

Confidence Intervals

It's possible to estimate using known mathematical techniques the worst case runtime with the confidence interval.

We might say that the maximum time is t with 99% confidence, based on the standard deviation.

Ready to Schedule Some?

Now that we understand real-time systems and classification…

Let’s actually schedule some!