Lecture 17 — Scheduling Algorithm Evaluation; UNIX & Windows

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

August 12, 2023

Part I

Algorithm Evaluation

What Do We Choose?

In some cases (real-time?) it's obvious what to choose.

Otherwise, what gives the best overall performance?

This requires a clear definition of best…
    Let's assume we have one.

How to Decide?

We need to evaluate the algorithms we've discussed in terms of how well they achieve that goal.

While it is possible to just have a theoretical discussion about it where we say we think *A* is better than *B* but worse than *C*, that's not a very scientific approach.

What we'd like to do instead is gather some data and make the decision using it.

Deterministic Modelling

Looks like writing test scenarios for a lab project.

Test scenario has inputs/outputs and check correctness/performance.

An example of a simple test case looks like this, where all five processes arrive at time 0 in this order:

Process ID	Burst Time
1	10
2	29
3	3
4	7
5	12

Then run the target scheduling algorithms.

Wait times:

FCFS: 28
SJF: 13
RR(10): 23

Total time of execution is the same… right?

Process Switch Penalty

A more precise version of this evaluation would assign a penalty to the process switch.

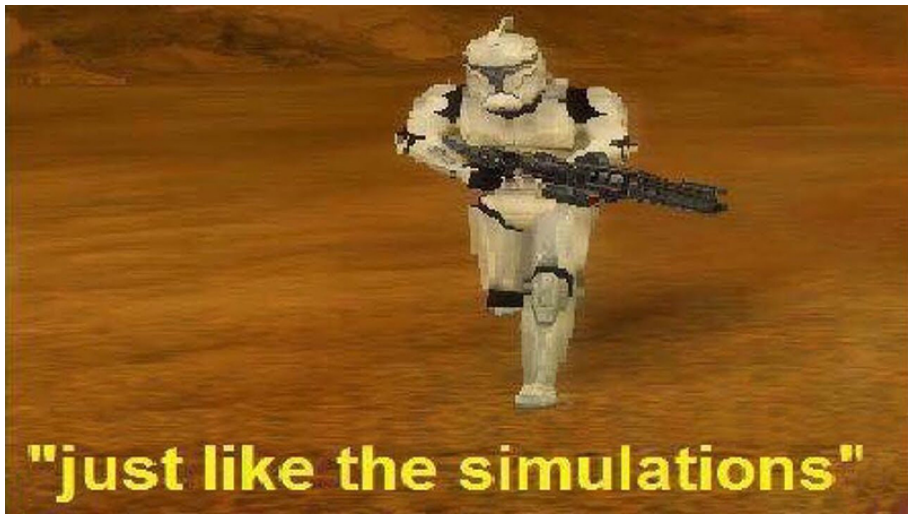
Context Switches

FCFS and SJF have 4 switches; RR has 7.

Maybe context switch time is so small as to be irrelevant?

More Realistic

Test cases that look more realistic will not have everything be statically initialized at the beginning and all tasks waiting.



The nice thing about the deterministic modelling approach is that the tests are reproducible and produce consistent results on every run.

Oh no, Math

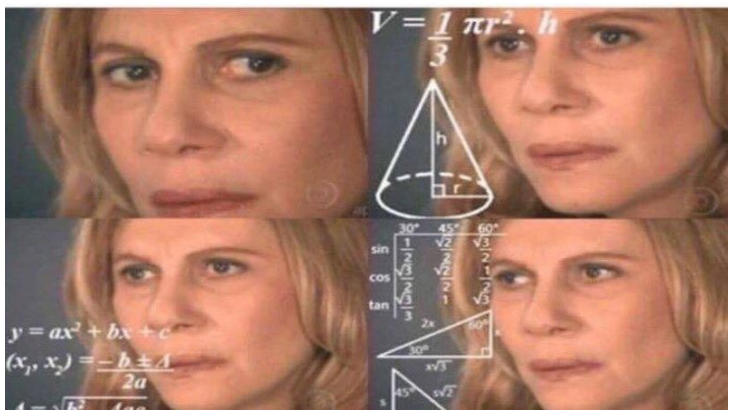
The real behaviour of most systems looks very different from how the deterministic approach models it.

Real systems have a lot of randomness in them…

Queueing Theory

Example: login times are somewhat random, but not totally.

That leads us to queueing theory.



That's too advanced for this course.

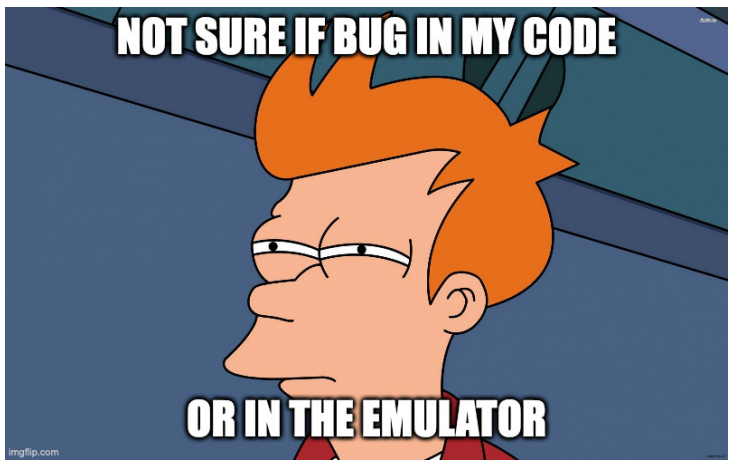
The simulation, if designed appropriately, will account for more things like clock interrupts and context switch times.

It’s possible to do an all-software simulation…

The simulation is only as accurate as you make it and there are a lot of things that might be difficult to simulate accurately.

Embedded Platform?

Your system may come with an emulator that saves you a lot of work!



Simulation Limitation

Simulations are always based on some assumptions.

A general purpose OS also needs to work for many different HW configurations…

Actually, let’s consider some real OS algorithms.

Part II

Windows & UNIX Scheduling

Scheduling in Desktop OSes

In this lecture we will examine how real commercial operating systems schedule their processes and threads.

We will examine UNIX and Windows scheduling.

We will see what approaches are used and what is interesting/novel about them.

Traditional UNIX

The traditional UNIX scheduling is really ancient; as in System V R3 and BSD 4.3.
It was replaced in SVR4 (which had some real-time support).

Multilevel feedback system using Round Robin within each of the queues.

Time slicing is implemented and the default time slice is a (very long) 1 second.

So if a process does not block or complete within 1 s, then it will be preempted.

Priority is based on the process type as well as the execution history.

Processor utilization for a process j is calculated for an interval i as:

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

And the priority is for process j at interval i is calculated by the formula:

$$P_j(i) = B_j + \frac{CPU_j}{2} + N_j$$

where B_j is the base priority of process j and N_j is the "nice" value of process j.

Be Nice

The “nice” value is a UNIX way to allow a user to voluntarily reduce the priority of a process to be “nice” to other users (but honestly, who uses this?).

Actually, the answer to that question is: system administrators.

An admin can “re-nice” a process and make it somewhat nicer than it would otherwise be.

The *CPU* and *N* components of the equation are restricted to prevent a process from migrating outside of its assigned category.

A process is assigned to a given category based on what kind of process it is.

To put it in simple terms, the OS puts its own needs first and tries to make the best use of resources it can.

From highest to lowest priority, the categories are:

1. Swapper (move processes to and from disk)
2. Block I/O device control (e.g., disk)
3. File manipulation
4. Character I/O device control (e.g., keyboard)
5. User processes

Yes, unfortunately, user processes get piled at the bottom of the list.

It should provide for efficient use of I/O devices and tends to penalize processor-bound processes at the expense of I/O bound processes.

CPU-bound processes should be able to carry on executing when an I/O-bound process waits for the I/O operation to complete.

When an I/O operation is finished, we would like to start the next I/O operation so the I/O device is not waiting.

This strategy is reasonably effective for a general-purpose, time-sharing operating system.

Upgrading to SVR4

SVR4: highest priority to real-time, then kernel, then user-mode.

Big differences are:

(1) more priority levels – 160 broken down into three types

(2) preemption points

Preemption points were needed!

FreeBSD

FreedBSD has a scheduling process that's quite similar to that of SVR4.

Interactivity scoring mechanism: which are user-interactive?

Scoring Mechanism

We'll define the maximum interactivity score as m, the runtime of the thread as r and the sleep time of the thread as s.

More sleep time than run time? The interactivity score is calculated as $(\frac{m}{2})(\frac{r}{s})$

For those that have more run time than sleep time: $(\frac{m}{2})(1+\frac{r}{s})$.

Affinity and Load-Balancing

We already covered affinity and load-balancing.

FreeBSD has push and pull mechanisms.

Pull: bit mask to indicate it's idle.

Push: Twice per second tasks to equalize highest and lowest CPUs.

Windows Scheduling

Windows schedules threads using a priority-based, preemptive scheduling algorithm, ensuring that the highest priority thread runs.

The official name for the selection routine is the dispatcher.

A thread runs until it is preempted, blocks, terminates, or its time slice expires.

Windows Scheduling

If a higher priority thread is unblocked, it will preempt a lower priority thread.

Windows has 32 different priority levels, the regular (priority 1 to 15) and real-time classes (16 to 31).

A memory management task runs at priority 0.

The dispatcher maintains a queue for each of the scheduling priorities and goes through them from highest to lowest until it finds something to do.

If there is nothing else currently ready, the System Idle Process will “run”.

Windows Priority Classes

There are six priority classes you can set a process to via Task Manager:

1. Realtime
2. High
3. Above Normal
4. Normal
5. Below Normal
6. Low

A process is usually in the Normal class.

Windows Thread Priorities

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Windows Time Slicing

If a process reaches the end of a time slice, the thread is interrupted.

Unless it is in the real-time category, its priority will be lowered, to a minimum of the base priority of each class.

When a process that was blocked on something (e.g., a wait or I/O), its priority is temporarily boosted (unless it is real-time, in which case it cannot be boosted).

The amount of the boost depends on what the event was.

A process that was waiting for keyboard input gets a bigger boost than one that was waiting for a disk operation, for example.

Windows Priority: UI

The operating system also gives a priority to whatever process is running in the selected foreground window.

This is different from foreground vs. background processes, because the definition of a foreground process was one that was user-interactive.

Here, the distinction is which of the user-interactive processes is currently “on top” in the UI.

Not only does this process get a priority boost, but it also gets longer time slices.

It highlights the different heritages of Windows and UNIX…