Lecture 6 — Uniprocessor Scheduling

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

March 3, 2023

Scheduling is very complex with multiple threads and multiple processors.

To keep things simple, we'll start with one processor.

The basic premise: decide when a process gets to execute.

Four types of scheduling:

1. **Long-Term Scheduling**
2. **Medium-Term Scheduling**
3. **Short Term Scheduling**
4. **I/O Scheduling** (for later)

Long-Term Scheduling

The long-term scheduler determines which programs run at all.

How many jobs do we plan to allow concurrently?

Controls the transition from “new” to the “ready state”.

Long-Term Scheduling

Does not happen much on desktop systems.

The user is responsible for deciding what programs to open.

Sometimes there are per-user limits: e.g, max 100 processes.

Long term example: server-based games (Diablo III) denying a request for a new game due to server load.

Mobile OSes like Android can be more aggressive.

Medium-Term Scheduling

More interesting than long-term; related to swapping.

A swapped out process cannot run in the immediate future.
- But will before too long...

When swapped in, the short term scheduler decides.

Short-Term Scheduling

Sometimes called the dispatcher.

The medium and long term scheduler are all about someday and sometime.

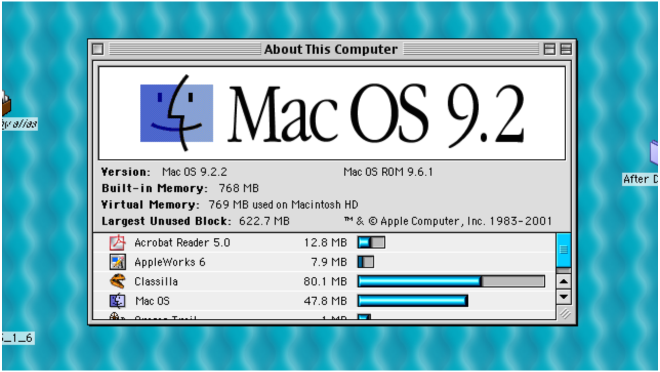
The short term scheduler is about "what are we going to do *right now*".

The short term scheduler is going to run a lot, so it is very important.

It will often run after certain things occur.

Co-operative multitasking, short term scheduling will only take place if:

- The currently executing process yields the CPU; or
- The currently executing process terminates (voluntarily or with an error).



This is not how most operating systems work these days.

If the process does yield or terminate, then the short term scheduler will run.

What we will discuss from here on out is pre-emptive multitasking.

The operating system, and not the running process, is responsible for deciding when it's time to switch processes.

Short-Term Scheduling

Some pre-emptive systems still have the concept of yield, and that is still an occasion to run the scheduler.

Similarly, in pre-emptive systems, processes still terminate.

Short-Term Scheduling: When?

The dispatcher will certainly run when a process becomes blocked.

Examples: I/O operations (network?), block on mutex, page fault...

Short-Term Scheduling: When?

Another time to make a scheduling decision is after handling an interrupt.

After the interrupt is handled, we can return to execution exactly where we left off, or we can go somewhere else.

The original process is suspended already, so why not leave it in that state?

Short-Term Scheduling: When?

System calls like `fork` and even signalling on a semaphore may also provide good opportunities to switch from one process to another.

By invoking the operating system (system call), the caller is suspended.

So it is necessary to decide what process executes next.

Short-Term Scheduling: When?

We acknowledged this in the discussion of `fork` by saying it was not known if the parent or child would execute next.

Semaphores: we do not even know which of the processes waiting on the semaphore will be the one to receive the signal.

Even then, which of the signalling process and waiting process will resume?

Short-Term Scheduling: When?

Finally, there is also time slicing.

If time slices are defined as *t* units, every *t* time units, there will be an interrupt generated by the clock.

The interrupt handler runs the short term scheduler to choose a process to run, so that different processes run (seemingly-)concurrently.

Process Behaviour

Processes tend to alternate periods of computing with input/output requests.

These tend to alternate; the CPU does a lot of work, called a CPU Burst,

Then some I/O; the period where it is waiting is called the I/O Burst.

After the I/O is completed, the CPU can go at it again.

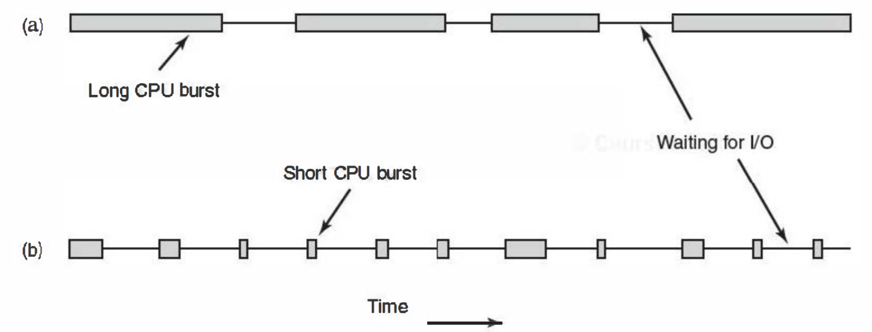
Process Behaviour

How much of each a process does allows for classification of the process.

Processes that spend most of their time computing are called CPU-Bound.

The alternative is a process that mostly waits for I/O: I/O-Bound

CPU- and I/O-Bound Processes



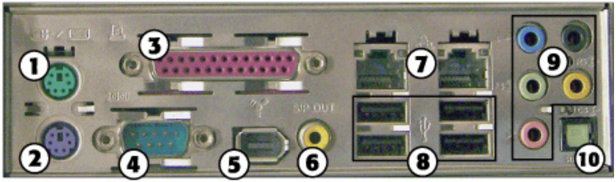
CPU- and I/O-Bound Processes

An I/O-Bound program will tend to have short CPU bursts, of course.

CPUs have gotten faster at a rate much higher than the rate of I/O speedup.

A new CPU comes out every few months and they get faster.

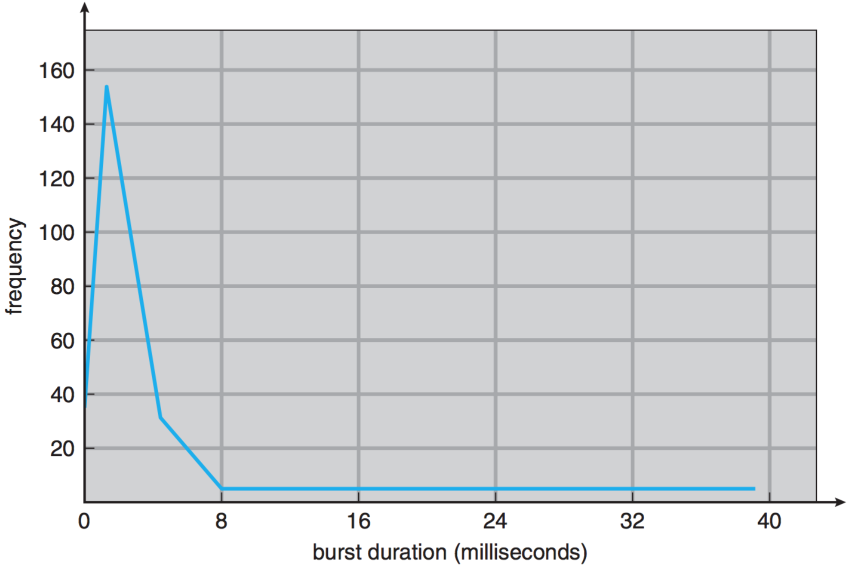
I/O standards like Serial-ATA and USB and such change very slowly, over the course of years.



1. PS/2 mouse port
2. PS/2 keyboard port
3. Parallel port
4. Serial port
5. IEEE 1394a port
6. SPDIF coaxial digital audio port
7. Ethernet ports
8. USB ports
9. 1/8-inch mini-jack audio ports
10. SPDIF optical digital audio port

So it makes sense that over time programs tend towards being I/O-Bound.

CPU Burst Histogram



Bursts

But what does this have to do with scheduling?

The long term scheduler may attempt to keep a balance between CPU- and I/O-Bound tasks to get the best resource utilization.

This requires that the long term scheduler have some idea about which processes are which.

Bursts

Another is that if the disk is slow…

When the disk has nothing to do, the short term scheduler should immediately schedule a process that is likely to issue a disk request.

The goals of scheduling depend significantly on the objectives of the system.

First we have to decide what we want.

Scheduling Criteria

If the system is supposed to respond to events within a certain period of time (real time system), this matters a lot to scheduling.

Perhaps the goal is for the CPU to be used maximally (as in a supercomputer).

Or maybe the most important thing is for users to feel like the system answers them quickly when they issue a command.

Scheduling Criteria

As usual when making a decision, we could just decide randomly.

We need evaluation criteria.

Scheduling Criteria

We will examine and define the following scheduling criteria:

1. Turnaround time.
2. Response time.
3. Deadlines.
4. Predictability.
5. Throughput.
6. Processor utilization.
7. Fairness.
8. Enforcing priorities.
9. Balancing resources.

Scheduling Algorithm Goals

The priorities of these different goals depend on the kind of system it is.

All Systems:

- Fairness
- Priorities
- Balancing Resources

Scheduling Algorithm Goals

The priorities of these different goals depend on the kind of system it is.

Batch Systems:

- Throughput
- Turnaround time
- CPU Utilization

Scheduling Algorithm Goals

The priorities of these different goals depend on the kind of system it is.

Interactive Systems:

- Response time.
- Predictability.

Each process's priority is typically an integer.

In UNIX, a lower number is higher priority; Windows is the opposite.

With a priority value assigned, it can be used to make decisions.

Imagine P_1 wants resource R_1 and P_2 wants that same resource.

If the priority of $P_1 > P_2$, choose to assign the resource to P_1.

Priorities

The OS or the program author may be responsible for assigning a priority.

These priorities may change over time with various criteria.

System administrators can change priorities, usually.

Priorities

Yes, this does mean that some processes are treated better than others.

  Sometimes solely as the result of inheritance.

User Priority Input

Users may have a say in priority, in at least a limited way.

In Windows, for example, as a user it may be possible to set a task priority.

Giving this to users was probably a bad idea, because users often do it wrong.

User Priority Input

If you have a long, CPU-Bound task, the right thing to do is to give it a low priority and not a high one.

You might expect that the high priority will get the task done faster...

It kills the performance of the system and makes users unhappy...
   Even though that's what they asked for!

Choosing the Next Process

In some systems, the highest priority non-blocked process will always run.

This is a great way of making sure that higher priority processes have right-of-way, but a terrible way of ensuring fairness and preventing starvation.

So, scheduling is not as easy as just finding the highest priority thing to do…