Scheduling

Algorithms

We now look a just a few of the simplest scheduling algorithms

Scheduling

Algorithms

We now look a just a few of the simplest scheduling algorithms

Exercise. Have a look at textbooks for gruesome detail on the relative performances of these algorithms

Scheduling

Algorithms

Run until completion

Scheduling

Algorithms

Run until completion

First in, first out (FIFO); non-preemptive batch, as on pre-OS machines

Scheduling

Algorithms

Run until completion

First in, first out (FIFO); non-preemptive batch, as on pre-OS machines

- Good for large amounts of computation

Scheduling

Algorithms

Run until completion

First in, first out (FIFO); non-preemptive batch, as on pre-OS machines

- Good for large amounts of computation
- No overheads of multitasking

Scheduling

Algorithms

Run until completion

First in, first out (FIFO); non-preemptive batch, as on pre-OS machines

- Good for large amounts of computation
- No overheads of multitasking
- Poor interaction with other hardware; can't process while printing (recall spooling)

Scheduling

Algorithms

Run until completion

First in, first out (FIFO); non-preemptive batch, as on pre-OS machines

- Good for large amounts of computation
- No overheads of multitasking
- Poor interaction with other hardware; can't process while printing (recall spooling)
- No interactivity

Scheduling

Algorithms

Run until completion

First in, first out (FIFO); non-preemptive batch, as on pre-OS machines

- Good for large amounts of computation
- No overheads of multitasking
- Poor interaction with other hardware; can't process while printing (recall spooling)
- No interactivity

Clearly not suitable for modern machines?

Scheduling

Algorithms

Run until completion

First in, first out (FIFO); non-preemptive batch, as on pre-OS machines

- Good for large amounts of computation
- No overheads of multitasking
- Poor interaction with other hardware; can't process while printing (recall spooling)
- No interactivity

Clearly not suitable for modern machines?

Actually still the basis for large supercomputers

Scheduling

Algorithms

Shortest Job First

Scheduling

Algorithms

Shortest Job First

Shortest-time-to-completion runs next; non-preemptive

Scheduling

Algorithms

Shortest Job First

Shortest-time-to-completion runs next; non-preemptive

- No multitasking

Scheduling

Algorithms

Shortest Job First

Shortest-time-to-completion runs next; non-preemptive

- No multitasking
- Good throughput

Scheduling

Algorithms

Shortest Job First

Shortest-time-to-completion runs next; non-preemptive

- No multitasking
- Good throughput
- Similar behaviour to FIFO on average

Scheduling

Algorithms

Shortest Job First

Shortest-time-to-completion runs next; non-preemptive

- No multitasking
- Good throughput
- Similar behaviour to FIFO on average
- Long jobs suffer and might get starved

Scheduling

Algorithms

Shortest Job First

Shortest-time-to-completion runs next; non-preemptive

- No multitasking
- Good throughput
- Similar behaviour to FIFO on average
- Long jobs suffer and might get starved
- Difficult to estimate time-to-completion, so reliant on the job description for this information

Scheduling

Algorithms

Run until completion plus cooperative multitasking

Scheduling

Algorithms

Run until completion plus cooperative multitasking

Non-preemptive

Scheduling

Algorithms

Run until completion plus cooperative multitasking

Non-preemptive

- Weak multitasking

Scheduling

Algorithms

Run until completion plus cooperative multitasking

Non-preemptive

- Weak multitasking
- Uses round-robin or similar to choose another task on relinquish

Scheduling

Algorithms

Run until completion plus cooperative multitasking

Non-preemptive

- Weak multitasking
- Uses round-robin or similar to choose another task on relinquish
- Poor interactivity

Scheduling

Algorithms

Run until completion plus cooperative multitasking

Non-preemptive

- Weak multitasking
- Uses round-robin or similar to choose another task on relinquish
- Poor interactivity
- Easy for a process to starve other processes

Scheduling

Algorithms

Run until completion plus cooperative multitasking

Non-preemptive

- Weak multitasking
- Uses round-robin or similar to choose another task on relinquish
- Poor interactivity
- Easy for a process to starve other processes
- Hard to write “good citizen” programs

Scheduling

Algorithms

Run until completion plus cooperative multitasking

Non-preemptive

- Weak multitasking
- Uses round-robin or similar to choose another task on relinquish
- Poor interactivity
- Easy for a process to starve other processes
- Hard to write “good citizen” programs

Scheduling

Algorithms

Run until completion plus cooperative multitasking

Non-preemptive

- Weak multitasking
- Uses round-robin or similar to choose another task on relinquish
- Poor interactivity
- Easy for a process to starve other processes
- Hard to write “good citizen” programs

Was used on millions of personal computers for a long time

Scheduling

Algorithms

Preemptive Round Robin

Scheduling

Algorithms

Preemptive Round Robin

Give each process, in turn, a fixed time slice

Scheduling

Algorithms

Preemptive Round Robin

Give each process, in turn, a fixed time slice

- Multitasking

Scheduling

Algorithms

Preemptive Round Robin

Give each process, in turn, a fixed time slice

- Multitasking
- Gives interactive processes the same time as compute processes

Scheduling

Algorithms

Preemptive Round Robin

Give each process, in turn, a fixed time slice

- Multitasking
- Gives interactive processes the same time as compute processes
- No starvation

Scheduling

Algorithms

Preemptive Round Robin

Give each process, in turn, a fixed time slice

- Multitasking
- Gives interactive processes the same time as compute processes
- No starvation
- Better interactivity than cooperative systems

Scheduling

Algorithms

Preemptive Round Robin

Give each process, in turn, a fixed time slice

- Multitasking
- Gives interactive processes the same time as compute processes
- No starvation
- Better interactivity than cooperative systems
- Not good for either interactive or real-time; may have to wait a long time for a slice of time

Scheduling

Algorithms

Round Robin

More suited to systems where all the processes are of equal (or nearly equal) importance; e.g., dedicated appliances like network routers that have to decide how share network capacity fairly

Shortest Remaining Time

Scheduling

Algorithms

Shortest Remaining Time

Time slice, pick next process by the estimate of the shortest time remaining; preemptive

Scheduling

Algorithms

Shortest Remaining Time

Time slice, pick next process by the estimate of the shortest time remaining; preemptive

- Good for short jobs

Scheduling

Algorithms

Shortest Remaining Time

Time slice, pick next process by the estimate of the shortest time remaining; preemptive

- Good for short jobs
- Good throughput

Scheduling

Algorithms

Shortest Remaining Time

Time slice, pick next process by the estimate of the shortest time remaining; preemptive

- Good for short jobs
- Good throughput
- Long jobs still can be starved

Scheduling

Algorithms

Shortest Remaining Time

Time slice, pick next process by the estimate of the shortest time remaining; preemptive

- Good for short jobs
- Good throughput
- Long jobs still can be starved
- Still hard to make estimates of times

Scheduling

Algorithms

Least Completed Next

Scheduling

Algorithms

Least Completed Next

The process that has consumed the least amount of CPU time next

Scheduling

Algorithms

Least Completed Next

The process that has consumed the least amount of CPU time next

- All processes make equal process in terms of CPU time

Scheduling

Algorithms

Least Completed Next

The process that has consumed the least amount of CPU time next

- All processes make equal process in terms of CPU time
- Interactive processes get good attention as they use relatively little CPU

Scheduling

Algorithms

Least Completed Next

The process that has consumed the least amount of CPU time next

- All processes make equal process in terms of CPU time
- Interactive processes get good attention as they use relatively little CPU
- Long jobs can be starved by lots of small jobs

Scheduling

Algorithms

These algorithms have good points, but they also have bad points: so “obviously” we just need to tweak them a bit

Scheduling

Algorithms

These algorithms have good points, but they also have bad points: so “obviously” we just need to tweak them a bit

But beware of patching and tweaking without having a good overview of what’s happening

Scheduling

Algorithms

These algorithms have good points, but they also have bad points: so “obviously” we just need to tweak them a bit

But beware of patching and tweaking without having a good overview of what’s happening

Many a system has ended up with a scheduler that’s large, slow and impossible to understand

Scheduling

Algorithms

These algorithms have good points, but they also have bad points: so “obviously” we just need to tweak them a bit

But beware of patching and tweaking without having a good overview of what’s happening

Many a system has ended up with a scheduler that’s large, slow and impossible to understand

And impossible to fix when you stumble across the next deficiency

Scheduling

Algorithms

At the very least we need to take interactivity, priority, and more into account

Scheduling

Algorithms

At the very least we need to take interactivity, priority, and more into account

How do we know if a process is interactive or compute intensive?

Scheduling

Algorithms

At the very least we need to take interactivity, priority, and more into account

How do we know if a process is interactive or compute intensive?

Watch how much I/O is happening and how long we are waiting for it: high I/O per compute is interactive, low is compute intensive

Scheduling

Algorithms

At the very least we need to take interactivity, priority, and more into account

How do we know if a process is interactive or compute intensive?

Watch how much I/O is happening and how long we are waiting for it: high I/O per compute is interactive, low is compute intensive

A process can be a mix of both, of course: it might move between the two over time

Scheduling

Algorithms

Similarly, priorities can be

Scheduling

Algorithms

Similarly, priorities can be

- Static. Unchanging through the life of the process. Very simple, but unresponsive to change (e.g., a process that alternates interactivity with urgent computation)

Scheduling

Algorithms

Similarly, priorities can be

- Static. Unchanging through the life of the process. Very simple, but unresponsive to change (e.g., a process that alternates interactivity with urgent computation)
- Dynamic. Priority responds to changes in the load. Harder to get right, more expensive to compute.

Scheduling

Algorithms

Similarly, priorities can be

- Static. Unchanging through the life of the process. Very simple, but unresponsive to change (e.g., a process that alternates interactivity with urgent computation)
- Dynamic. Priority responds to changes in the load. Harder to get right, more expensive to compute.
- Purchased. Pay more, get higher priority!

Scheduling

Algorithms

Highest Response Ratio Next

Scheduling

Algorithms

Highest Response Ratio Next

A variant of SRT, where we take the time a process has been waiting since its last time slice into account

Highest Response Ratio Next

A variant of SRT, where we take the time a process has been waiting since its last time slice into account

$$\text{Dynamic priority} = \frac{\text{time so far in system}}{\text{cpu used so far}}$$

Scheduling

Algorithms

Highest Response Ratio Next

A variant of SRT, where we take the time a process has been waiting since its last time slice into account

$$\text{Dynamic priority} = \frac{\text{time so far in system}}{\text{cpu used so far}}$$

- A process executes repeated time slices until its priority drops below that of another process

Scheduling

Algorithms

Highest Response Ratio Next

A variant of SRT, where we take the time a process has been waiting since its last time slice into account

$$\text{Dynamic priority} = \frac{\text{time so far in system}}{\text{cpu used so far}}$$

- A process executes repeated time slices until its priority drops below that of another process
- Tries to avoid starvation

Scheduling

Algorithms

Highest Response Ratio Next

A variant of SRT, where we take the time a process has been waiting since its last time slice into account

$$\text{Dynamic priority} = \frac{\text{time so far in system}}{\text{cpu used so far}}$$

- A process executes repeated time slices until its priority drops below that of another process
- Tries to avoid starvation
- Long jobs will eventually get a slice

Scheduling

Algorithms

Highest Response Ratio Next

- New jobs get immediate attention as CPU time is near 0

Scheduling

Algorithms

Highest Response Ratio Next

- New jobs get immediate attention as CPU time is near 0
- But now critical shorter jobs might not finish in time as they could get scheduled after a long-waiting job

Scheduling

Algorithms

Highest Response Ratio Next

- New jobs get immediate attention as CPU time is near 0
- But now critical shorter jobs might not finish in time as they could get scheduled after a long-waiting job
- This needs frequent re-evaluation of priorities to get good behaviour, which implies small timeslices, and so lots of scheduling overhead