Computer systems architectures  
CM12002

Russell Bradford

2023/24

### 1. Scheduling

We now look at the question of how to choose which process to run next out of all those in the Ready state

This is an *extremely* difficult problem and still has not been solved to everybody’s satisfaction

So we only have a quick overview

### 2. Scheduling

A list of scheduling algorithms, from Wikipedia:

Borrowed-Virtual-Time Scheduling (BVT), Completely Fair Scheduler (CFS), Critical Path Method of Scheduling, Deadline-monotonic scheduling (DMS), Deficit round robin (DRR), Dominant Sequence Clustering (DSC), Earliest deadline first scheduling (EDF), Elastic Round Robin, Fair-share scheduling, First In, First Out (FIFO), also known as First Come First Served (FCFS), Gang scheduling, Genetic Anticipatory, Highest response ratio next (HRRN), Interval scheduling, Last In, First Out (LIFO), Job Shop Scheduling, Least-connection scheduling, Least slack time scheduling (LST), List scheduling, Lottery Scheduling, Multilevel queue, Multilevel Feedback Queue, Never queue scheduling, scheduler, Proportional Share Scheduling, Rate-monotonic scheduling (RMS), Round-robin scheduling (RR), Shortest expected delay scheduling, Shortest job next (SJN), Shortest remaining time (SRT), Staircase Deadline scheduler (SD), “Take” Scheduling, Two-level scheduling, Weighted fair queuing (WFQ), Weighted least-connection scheduling, Weighted round robin (WRR), Group Ratio Round-Robin:

### 3. Scheduling

And they are just the ones people can be bothered to write about on Wikipedia

### 4. Scheduling

Think of the problems

* Try to give each process its fair share of CPU time
* and no starvation of any process
* Try to make interactive processes respond in human timescales
* Try to give as much computation time as possible to compute-heavy processes
* Ensuring critical real-time processes are dealt with before it is too late

### 5. Scheduling

* Try to service peripherals in a timely way
* Understanding the various requirements of hardware: mice and printers are slow; networks and disks are medium; memory is fast
* Try to distribute work amongst multiple devices; e.g, CPUs, disks and networks
* Try to make best use of the hardware and use it efficiently
* Try to make behaviour predictable: we don’t want wildly erratic behaviour
* Try to degrade gracefully under heavy load
* And so on

### 6. Scheduling

And do it all quickly!

### 7. Scheduling

Here we shall concentrate on CPU scheduling, but remember the CPU is just one resource of many

A related problem is *I/O scheduling*, managing requests and responses to other devices, such as disks, to make best use of them

I/O scheduling is important, but we shall not talk about it here

But we will note in passing that the various schedulers for the various resources may not agree on what should be done next!

### 8. Scheduling

All this needs to be quantified somehow so we can use the numbers to make choices

Example measurements include:

* CPU cycles used
* Memory used
* Disk used
* Network used
* Etc.

### 9. Scheduling

And we can quantify results

* Throughput; more or fewer jobs finished in a given time
* Turnaround; response time: interactive response is snappy or sluggish
* Real-time; we *must* deal with this data now else the car will crash (deadlines)
* Money; we’ve been given money to get this data ready in the next hour
* Etc.

### 10. Scheduling

All this information was originally collected to figure out how much money to charge the user

These days most people are not so worried about charging as we all have our own computers. We are more concerned about making the best use of our computer

Though it’s still important: cloud services (e.g., Amazon, Google, Microsoft) sell time on their machines

They charge based on disk storage, data input and output and compute (CPU) used

There’s nothing new in Computer Science: just recurring fashions!

### 11. Scheduling

#### Algorithms

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

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

### 12. 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 jobs on large supercomputers

### 13. 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

### 14. 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

### 15. 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
* But still not really good for either interactive or real-time; may have to wait a long time for a slice of time

### 16. Scheduling

#### Algorithms

**Round Robin**

More suited to systems where all the processes are fairly similar; e.g., dedicated appliances like network routers that have to decide how share network capacity fairly

### 17. 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

### 18. 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

### 19. 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

### 20. 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: and it might move between the two over time

### 21. 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!