Scheduling

Levels of scheduling 4.2.2

Scheduling algorithms 5.3

Scheduling of processes

Which processes should be picked to run?

Different solutions for different needs.

Batch systems

Keep the machine going

Time-sharing systems

Keep the users going

Real-time systems (including multimedia, virtual reality etc)

Always deal with the important things first

Levels of scheduling

Batch systems

Very long-term scheduler

can this user afford it?

administrative decisions - students can't enter jobs between 10pm and 6am

Long-term scheduler

may enforce administrative decisions

which jobs (currently spooled) should be accepted into the system

need to know about resource requirements

- How many CPU seconds? (need a way of encouraging users to try to be accurate in their estimation)
- How many files, tapes, pages of output?

invoked when jobs leave the system

Medium-term scheduler

if things get out of balance

suspend this process and swap it out

Short-term scheduler

which of the runnable jobs should go next

Time-sharing systems

No long-term and medium-term scheduler

unless there is a batch submission mode

users perform this task

Real-time systems

Priorities are really important.

Real-time processes must be handled within certain times.

Other processes will suffer delays rather than the real-time processes.

Medium-term scheduling can be important to maintain adequate response for the real-time processes.

What do most processes spend most of their time doing?

Bursts of CPU and IO activity.

Scheduling algorithms 5.3

FCFS - again

Simple - no time wasted to determine which process should run next

Round-robin scheduling is FCFS with time slices

What is wrong with treating every process equally?

One way to tune this is to change length of the time slice.

What effect does a long time slice have?

What effect does a short time slice have?

What is the 80% rule?

Should we have different length time slices for different processes?

What behaviour do we get if we give CPU intensive processes long time slices?

What behaviour do we get if we give CPU intensive processes short time slices?

SJF - shortest job first

actually shortest next CPU burst

Optimal - minimum average wait time.

Unfortunately how do we know?

We may have this information at the long-term scheduling phase of a batch system.

We could use the previous CPU bursts of this process to predict its coming behaviour.

Another form of shortest job first

How much longer does each job need to complete?

Get processes out of the system as quickly as possible.

Under this mechanism, processes which hog the CPU lose their positions to those which don't.

A form of **priority**.

In this case the longer the CPU burst the worse the priority.

Other systems preserve priorities

Set before a process runs

When a new process arrives it is placed in the position in the ready queue corresponding to its priority.

It is possible to get starvation

Priorities can vary over the life of the process

The system makes changes according to the process' behaviour

CPU usage, IO usage, memory requirements

Aging

Priorities sometimes are used to allocate percentages of process time

e.g. a process of a worse priority might be scheduled after five processes of a better priority.

This prevents starvation, but better priority processes will still run more often.

A common way to do this is with multiple priority queues

Some systems keep processes on the same queue

others move processes from queue to queue

Some are absolute - worse priority queues only run a process if no better queues have any waiting

Some have different selection strategies

Some allocate different time slices

Processes can be moved from queue to queue because of their behaviour

CPU intensive processes are commonly put on worse priority queues

What behaviour does this encourage?

Processes which haven't run for a long time can be moved to better priority queues

Multiple processors

We presume all processes can run on all processors

(not always true)

Maintain a shared queue

Let each processor select the next process from the queue

Or let one processor determine which process goes to which processor