From last time

Explain why the time slice in pre-emptive process scheduling algorithms is normally significantly longer than the time needed for a context switch (2 marks)

Why is a schedule giving lowest average turnaround time the same as that giving lowest average waiting time? (1 mark)

Given a set of jobs with known processing time, all available to run, explain why repeatedly running the shortest job next gives the lowest average turnaround time. (3 marks)

What is a CPU burst and an I/O burst? What is a CPU-bound and an I/O bound process? Why is it a good strategy in process scheduling to give higher priority to I/O bound processes? (4 marks)

The University of Manchester

COMP25111: Operating Systems

Lecture 7: Process Scheduling 2

John Gurd

School of Computer Science, University of Manchester

Autumn 2015

Overview & Learning Outcomes

Shortest First

Priorities – static & dynamic

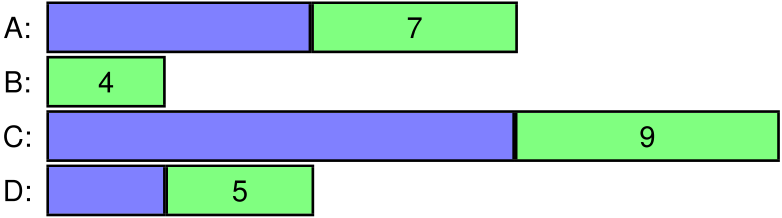
Multiple Queues

Shortest-Job-First (SJF)

last lecture – starting the shortest job first gave a better result

From a set of ready processes, start process with smallest CPU burst
– minimises average turnaround & waiting time

e.g. processes A, B, C, D arrive together,
CPU-burst time: A=7, B=4, C=9, D=5

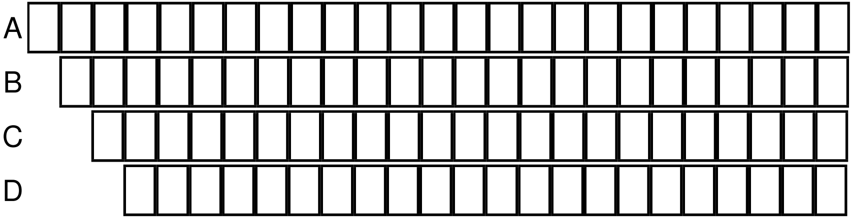


Shortest Remaining Time First/Next (SRTF)

Preemptive (not time-sliced) version of SJF:

For each newly-ready process:
if CPU-burst $<$ time to complete running process,
then context-switch & run the new process

Question: as previous example, but staggered arrival:



Shorter average waiting time than (non-preemptive) SJF

Problems with Shortest-First

Starvation: A process may be overlooked repeatedly

How to predict CPU-burst length

Priority Scheduling

So far, implicitly assumed that all processes are equally important

Use **Process Priority** to e.g.
– run first/last
– give longer/shorter time slice
...

e.g. SJF: “highest priority” = shortest CPU-burst

Many variants of priority scheduling
Major problem **starvation** – ensure low-priority processes eventually run

Separate **Policy** & **Mechanism**

Static vs Dynamic Priorities

Static (externally defined) priorities: predetermined for each process

Dynamic (internally defined) priorities: assigned by the system to achieve certain goals

May use both externally and internally defined priorities

Multiple Queues

How to map **priority** $\rightarrow$ scheduling decisions?

e.g. higher priority $\rightarrow$ longer time slice?

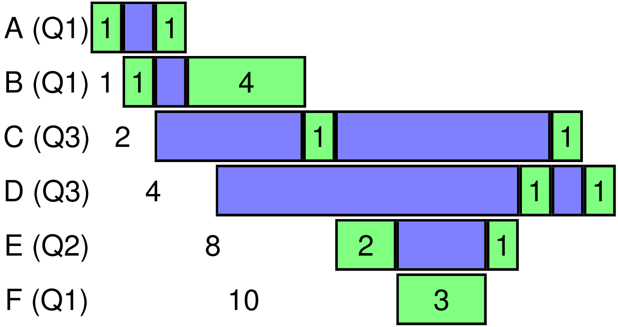
more convenient: ready state has **multiple queues**, each with its own priority

Options:
– Higher-priority queues must be empty for low-priority processes to run at all
– Higher priority queues get more time than lower priority queues
– Processes may move between queues (**dynamically adjusted**)

Example

3 queues: Q1 $>$ Q2 $>$ Q3

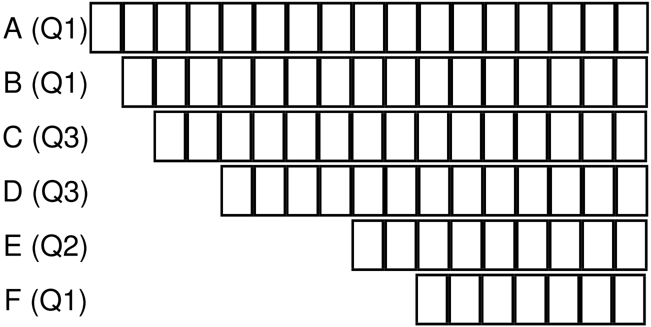
1st scenario:
– Q1 must be empty for Q2 processes to run; Q1 & Q2 empty for Q3 processes
– Run processes round-robin for 1 quantum each



Example ctd.

2nd scenario: (repeatedly)
– 3 quanta for Q1, then 2 quanta for Q2, then 1 quantum for Q3
– each queue applies round-robin (time-slice = 1 quantum)

Question: A-F as before:



Multilevel Feedback Queues

3 queues: Q1 $>$ Q2 $>$ Q3

– Q1 must be empty for Q2 processes to run; Q1 & Q2 empty for Q3 processes
– a process in Q1 gets 1 quanta, in Q2 gets 2 quanta, in Q3 gets 4 quanta
– every process is initially assigned to Q1.
– A process using all its time slice will go down one queue
– A process not using all its time slice, will go up one queue.
(i.e. higher priority for I/O-bound, lower priority for CPU-bound)

Another example

(similar to Linux process scheduling prior to version 2.5)

3 queues: Q1 $>$ Q2 $>$ Q3
Q1=FIFO, Q2=Round-Robin, Q3=priority based (1 to 40)
process in Q1 (Q2) always higher priority than one in Q2 (Q3)

Q3: each process given a quantum, q, (or credits)
q = (quantum_left / 2) + priority

reduce quantum of running process by 1 per clock tick
– if quantum=0, remove process from CPU

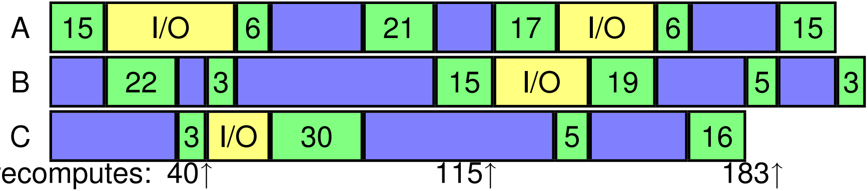
select ready process with the highest quantum;
if all ready processes have 0 quantum,
– recompute quantum of all processes in the system

if a process with a higher quantum than the one running
becomes ready, then it is set running

example ctd.

Three processes A, B, C (in the 3rd queue)
initial priority & quantum A=23, B=22, C=21

A: 15 CPU, 28 I/O, 44 CPU, 24 I/O, 21 CPU ...
B: 40 CPU, 22 I/O, 27 CPU ...
C: 3 CPU, 9 I/O, 51 CPU ...



Question: what triggers each recompute? what new quanta?

Disadvantages

– does not scale well

– boosting I/O-bound processes is not optimal

For linux fans: source code & books available

Not a topic to cover with a couple of slides

More process/thread scheduling

Many more algorithms
UNIX & Windows have priority/multiqueue algorithms

Real-time systems may require other algorithms
(need to meet deadline) e.g.
– Earliest Deadline First (start process with first deadline)
– Least Slack First (start process with smallest:
deadline – completion_time)

Process scheduling settled – many open problems with multiprocessors, hyperthreading

Evaluation: analytic (deterministic), queuing theory, simulation

Summary of key points

Shortest First

Priorities – static & dynamic

Multiple Queues

Multilevel feedback queue scheduling
– most generic (configurable for different policies)
– most complex

Next: process synchronisation

Your Questions

For next time

Explain briefly how starvation may occur in process scheduling. (2 marks)

In round-robin scheduling, new processes are typically placed at the end of the ready-state queue rather than at the beginning. Suggest a good reason for this. (2 marks)

A scheduler uses a time-slice of 4.5msec, and a context switch takes 0.5msec. What percentage of CPU time is spent on executing process instructions: (a) if processes use the whole time-slice? (b) if processes only need 0.5msec CPU-bursts? In general, how would you improve the percentage of CPU time spent on executing process instructions? (3 marks)

Exam Questions

i) An SRTF scheduler also uses time-slices (of length 1). Show what is run in each time-slice if these processes are present at the start: A needing 3 time-units of CPU, B needing 6, C needing 7, & D needing 8. What is the average turnaround time? (5 marks)

ii) The scheduler is modified so, after a process has had a time-slice, instead of just using the length of the remaining CPU-burst of a process, it uses this **minus** the time since the process last received a time-slice. If two or more processes are tied, the one with the shorter remaining CPU-burst is chosen. Show what is run in each time quantum for the same set of processes. What is the average turnaround time? (5 marks)

iii) What scheduling defect is the modification in (ii) attempting to remedy, and how effective is it? (2 marks)

Glossary

Shortest Remaining Time First/Next (SRTF)
Starvation
Priority
Static Priority
Dynamic Priority
Scheduling Policy
Scheduling Mechanism
Multiple Scheduling Queues
Dynamically Adjusted Scheduling Queues
Mulitlevel Feedback Queue Scheduling

Reading

OSC: sections 5.3.2-5.3.6, 5.8 (skim through 5.5-5.7)

OSCJ: sections 5.3.2-5.3.6, 5.9 (skim through 5.5-5.8)

older OSCJ: sections 6.3.2-6.3.6, 6.7.3, 6.10 (skim through 6.5-6.9)

MOS3: sections 2.4.2-2.4.3 up to Shortest Process Next, 2.7 (skim through 2.4.4)

MOS2: sections 2.5.2-2.5.3 up to page 146, 2.7 (skim through 2.5.4)