Imperial College London

Operating Systems

Scheduling

Course 211
Spring Term 2018-2019

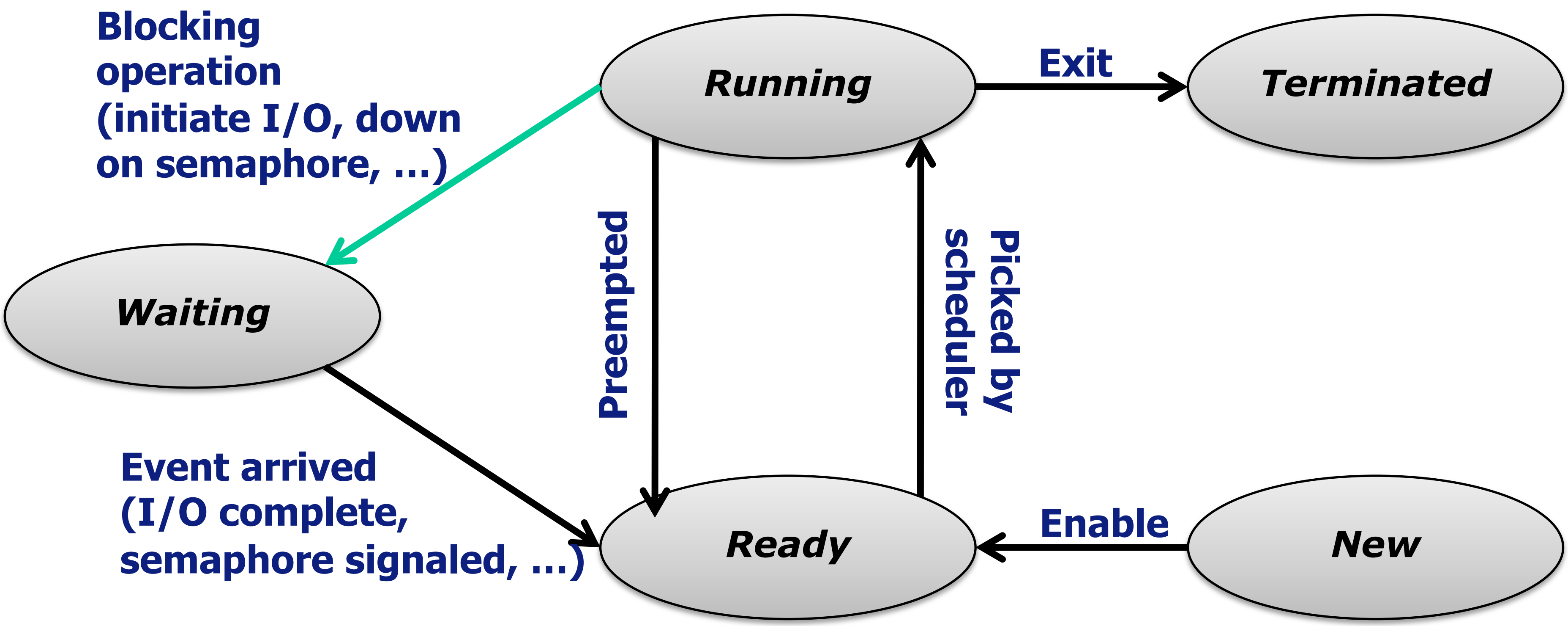
http://www.imperial.ac.uk/computing/current-students/courses/211/calendar/

(Slides courtesy of Cristian Cadar)

Peter Pietzuch

prp@doc.ic.ac.uk
http://www.doc.ic.ac.uk/~prp

Process States



- **New**: the process is being created
- **Ready**: runnable & waiting for processor
- **Running**: executing on a processor
- **Waiting/Blocked**: waiting for an event
- **Terminated**: process is being deleted

If multiple processes are ready, which one should be run?

Goals of Scheduling Algorithms

Ensure fairness

- Comparable processes should get comparable services

Avoid indefinite postponement

- No process should starve

Enforce policy

- E.g. priorities

Maximize resource utilisation

- CPU, I/O devices

Minimize overhead

- From context switches, scheduling decisions

Goals of Scheduling Algorithms

Batch systems:

- Throughput: Jobs per unit of time
- Turnaround time: Time between job submission & completion

Interactive systems:

- Response time crucial: Time between request issued and first response

Real-time systems:

- Meeting deadlines:
  - Soft deadlines: e.g. leads to degraded video quality
  - Hard deadline: e.g. leads to plane crash

Preemptive vs. Non-Preemptive Scheduling

Non-preemptive

- Let process run until it blocks or voluntarily releases CPU

Preemptive

- Let process run for a maximum amount of fixed time
- Requires clock interrupt

CPU-Bound vs. I/O-Bound Processes

CPU-bound processes

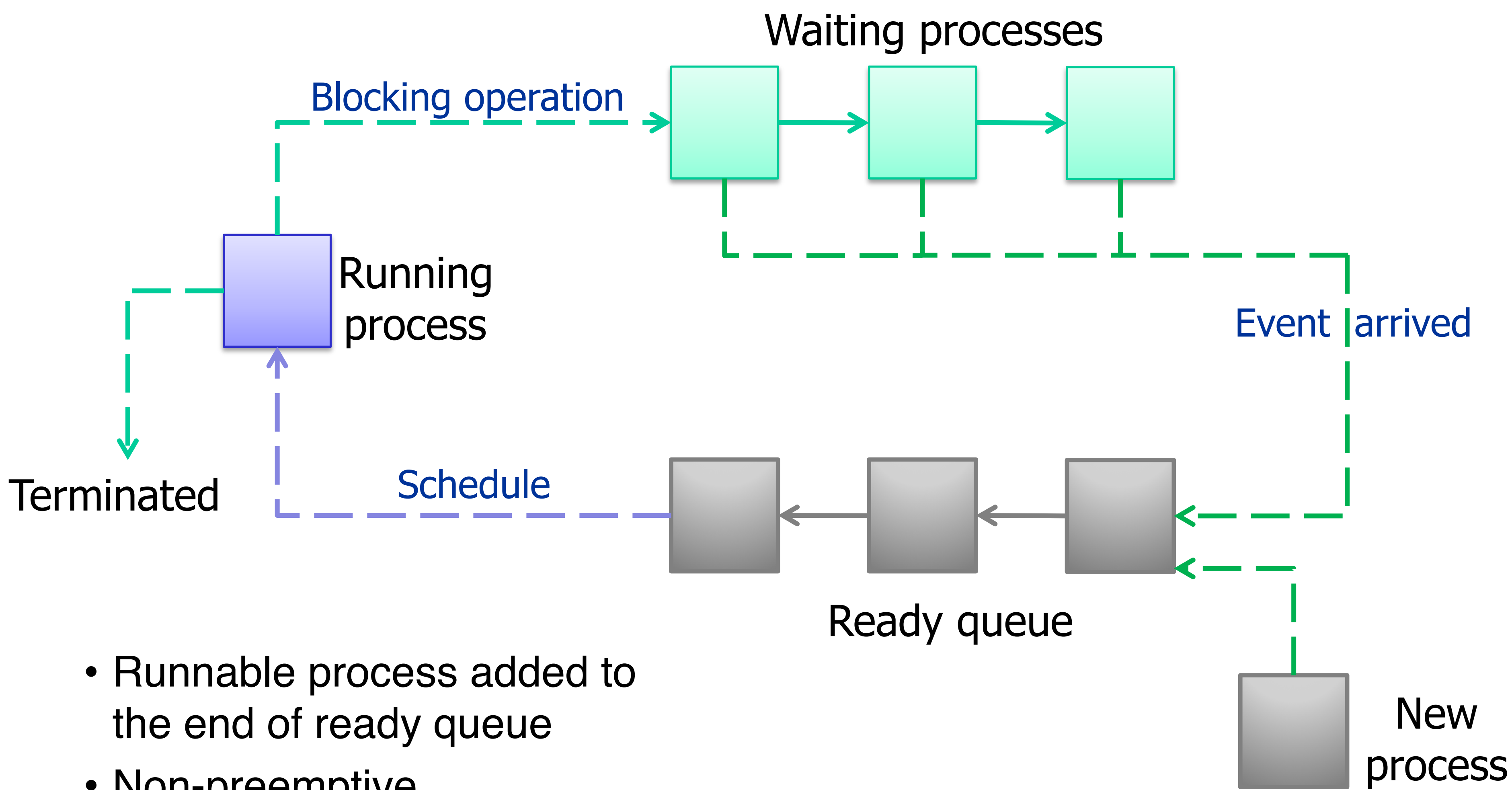
– Spend most of their time using the CPU

I/O-bound processes

– Spend most of their time waiting for I/O

– Tend to only use CPU briefly before issuing I/O request

First-Come First-Served (FCFS) (Non-Preemptive)



- Runnable process added to the end of ready queue
- Non-preemptive

FCFS Advantages

- No indefinite postponement
  - All processes are eventually scheduled
- Really easy to implement

FCFS Disadvantages

What happens if a long job is followed by many short jobs?

- E.g. 1h, 1s, 1s, 1s, with jobs 2-4 submitted just after job 1

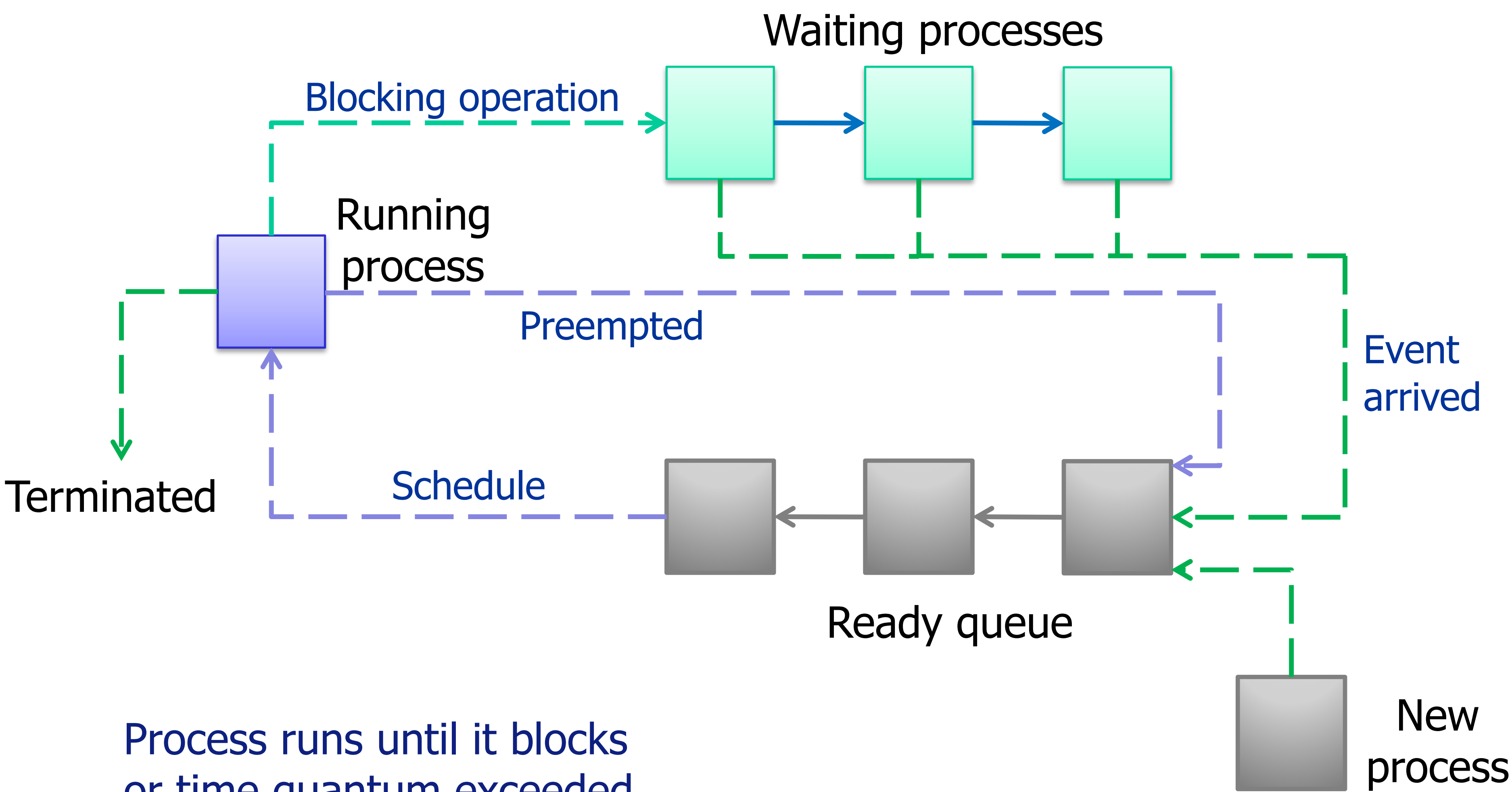
3600s	*1s*	*1s*	*1s*

- Throughput?
- Average turnaround time?

1s	*1s*	*1s*	*3600s*

- Throughput?
- Average turnaround time?

Round-Robin Scheduling (RR)



Process runs until it blocks or time quantum exceeded

Round-Robin

Fairness

- Ready jobs get equal share of the CPU

Response time

- Good for small number of jobs

Average turnaround time:

- Low when run-times differ
- Poor for similar run-times

RR Quantum (Time Slice)

RR Overhead:

- 4 ms quantum, 1 ms context switch time: 20% of time → overhead
- 1 s quantum, 1ms context switch time: only 0.1% of time → overhead

Large quantum:

- Smaller overhead
- Worse response time
  - Quantum = ∞ → FCFS

Small quantum:

- Larger overhead
- Better response time

RR Quantum

Choosing a quantum value:

- Should be much larger than context switch cost
- But provide decent response time

Typical values: 10 ms – 200 ms

Some example values for standard processes (values vary depending on process type and behaviour, priority, …):

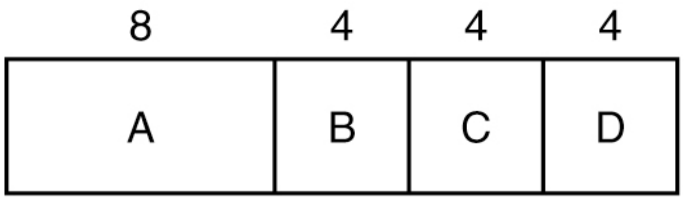
- Linux: 100 ms
- Windows client: 20 ms
- Windows server: 180 ms

Shortest Job First (SJF)

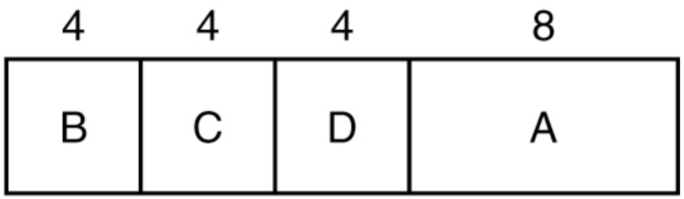
Non-preemptive scheduling with run-times known in advance

Pick the shortest job first

- Process with shortest CPU burst



FCFS turnaround time:



SJF turnaround time:

- Provably optimal when all jobs are available simultaneously

Shortest Remaining Time (SRT)

Preemptive version of shortest job first

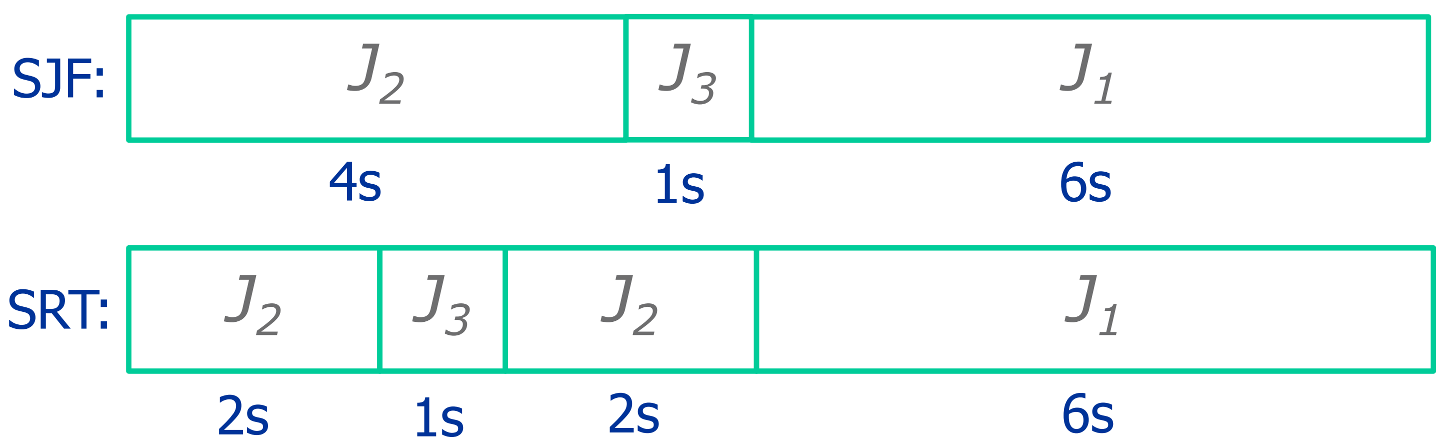
– Again, runtimes have to be known in advance

Choose process whose remaining time is shortest

– When new process arrives with execution time less than the remaining time for the running process, run it

Allows new short jobs to get good service

Example: 3 jobs: J_1 = 6s, J_2 = 4s, J_3 = 1s arrives after 2s



Shortest Remaining Time (SRT)

What if a running process is almost complete and a shorter job arrives?

Knowing Run-times in Advance

Run-times are usually not available in advance

Compute CPU burst estimates based on heuristics?

- E.g. based on previous history
- Not always applicable

User-supplied estimates?

- Need to counteract cheating to get higher priority
- E.g. terminate or penalise processes after they exceed their estimated run-time

Fair-Share Scheduling

Users are assigned some fraction of the CPU

- Scheduler takes into account who owns a process before scheduling it

E.g. two users each with 50% CPU share

- User 1 has 4 processes: A, B, C, D
- User 2 has 2 processes: E, F

What does a fair-share RR scheduler do?

Priority Scheduling

Jobs are run based on their priority

- Always run the job with the highest priority

Priorities can be **externally defined** (e.g. by user) or based on some **process-specific metrics** (e.g. their expected CPU burst)

Priorities can be **static** (i.e. they do not change) or **dynamic** (they may change during execution)

Example: Consider 3 processes arriving at essentially the same time with externally defined static priorities A = 4, B = 7, C = 1, where a higher value means higher priority.

- Processes are run to completion in the order B, A, C

General-Purpose Scheduling

Favour short and I/O-bound jobs

- Get good resource utilisation
- And short response times

Quickly determine the nature of job and adapt to changes

- Processes have periods when they are I/O-bound and periods when they are CPU-bound

Multilevel Feedback Queues

A form of priority scheduling

- Shortest remaining time also a form of priority scheduling

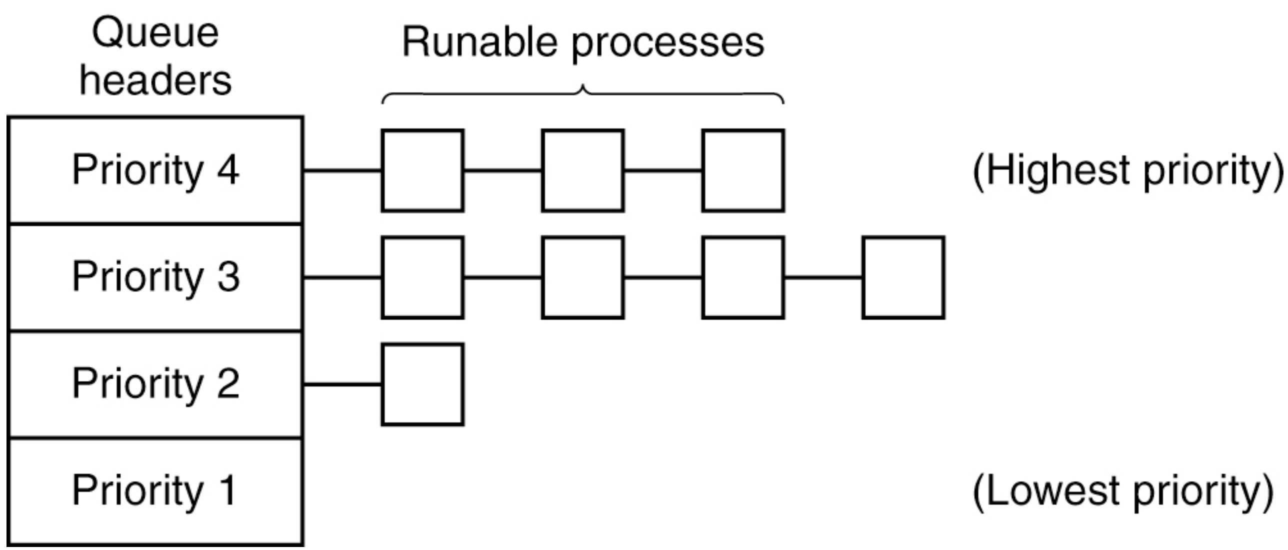
Implemented by many OSs:

- Windows Vista, Windows 7
- Mac OS X
- Linux 2.6 - 2.6.23
- Pintos!

Multilevel Feedback Queues

One queue for each priority level

- Run job on highest non-empty priority queue
- Each queue can use different scheduling algorithm
  - Usually round-robin



Multilevel Feedback Queues

Need to determine current nature of job

- I/O-bound? CPU-bound?

Need to worry about starvation of lower-priority jobs

Feedback mechanism:

- Job priorities recomputed periodically, e.g. based on how much CPU they have recently used
  - Exponentially-weighted moving average
- **Aging**: increase job's priority as it waits

Multilevel Feedback Queues

Not very flexible

- Applications basically have no control
- Priorities make no guarantees
  - What does priority 15 mean?

Does not react quickly to changes

- Often needs warm-up period
  - Running system for a while to get better results
- Problem for real-time systems, multimedia apps

Cheating is a concern

- Add meaningless I/O to boost priority?

Cannot donate priority

Lottery Scheduling [Waldspurger and Weihl 1994]

Jobs receive lottery tickets for various resources

- E.g. CPU time

At each scheduling decision, one ticket is chosen at random and the job holding that ticket wins

Example: 100 lottery tickets for CPU time,
P1 has 20 tickets

- Chance of P1 running during the next CPU quantum: 20%
- In the long run, P1 gets 20% of the CPU time

Lottery Scheduling

- Number of lottery tickets meaningful
  - Job holding p% of tickets, gets p% of resource
  - Unlike priorities
- Highly responsive:
  - New job given p% of tickets has the p% chance to get the resource at the **next** scheduling decision
- No starvation
- Jobs can exchange tickets
  - Allows for priority donation
  - Allows cooperating jobs to achieve certain goals
- Adding/removing jobs affect remaining jobs proportionally
- Unpredictable response time
  - What if interactive process is unlucky for a few lotteries?

Summary

- Scheduling algorithms often need to balance conflicting goals
  - E.g. ensure fairness, enforce policy, maximise resource utilisation

- Different scheduling algorithms appropriate in different contexts
  - E.g. batch systems vs interactive systems vs real-time systems

- Well-studied scheduling algorithms include:
  - First-Come First-Served FCFS, Round Robin, Shortest Job First (SJF), Shortest Remaining Time (SRT), Multilevel Feedback Queues and Lottery Scheduling

Tutorial Questions

State which of the following are true and which false. Justify your answers.

1. Interactive systems generally use non-preemptive processor scheduling.

2. Turnaround times are more predictable in preemptive than in non-preemptive systems.

3. One weakness of priority scheduling is that the system will faithfully honor the priorities, but the priorities themselves may not be meaningful.

Tutorial Questions

1. Interactive systems generally use non-preemptive processor scheduling.

2. Turnaround times are more predictable in preemptive than in non-preemptive systems.

3. One weakness of priority scheduling is that, while a system may faithfully honour the priorities, the priorities themselves may not be meaningful.