University of BRISTOL

Computer Systems B

COMS20012

Introduction to Operating Systems and Security

bristol.ac.uk

University of BRISTOL

FCFS, SJF, SRTF, Round Robin

bristol.ac.uk

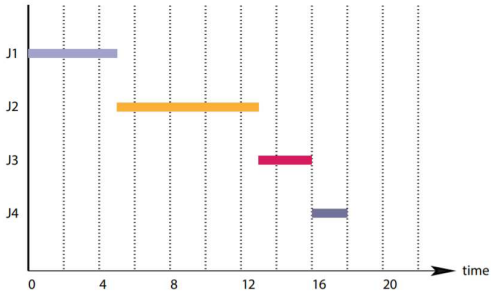
Simple Scheduling

- A set of job Jx to schedule
  - Job arrival time a_x
  - Job running time r_x
- Only one job at a time (multicore later this week)

bristol.ac.uk

First come first served

- Jobs run in order of arrival
- Simple
- No starvation
- If you remember week 5 that is how early computers worked

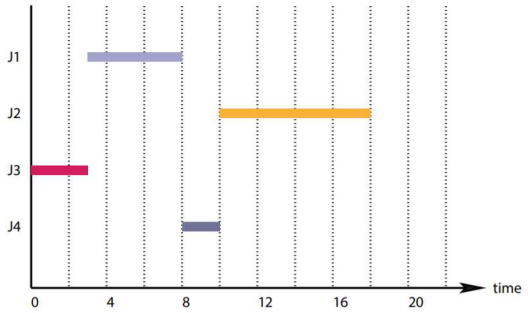


Job	J1	J2	J3	J4
arrival (a_i)	0	0	0	5
run time (r_i)	5	8	3	2

bristol.ac.uk

Shortest Job First

- Jobs run in increasing order of runtime
- Minimize the average **turnaround** time
- **Starvation is possible**

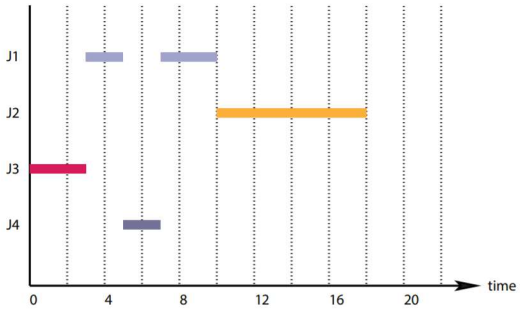


Job	J1	J2	J3	J4
arrival (a_i)	0	0	0	5
run time (r_i)	5	8	3	2

bristol.ac.uk

Shortest Remaining Time First

- Preemptive version of SJF
- Arriving jobs can preempt running jobs
- Select job with the shortest remaining time
- **Starvation is possible**



Job	J1	J2	J3	J4
arrival (a_i)	0	0	0	5
run time (r_i)	5	8	3	2

bristol.ac.uk

University of BRISTOL

Problems?

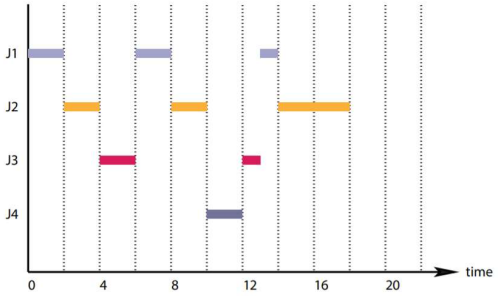
bristol.ac.uk

Problems

- Obviously, **starvation**
- How do you predict running time in practice?
  - Imagine a modern interactive application (e.g., video game)
- No illusion of concurrency
  - Job make progress until they are completed
  - … or pre-empted by a “quicker” job for some algorithms

Round Robin

- Preemptive FCFS
  - Pick jobs as they arrive
  - Execute them for a quantum (a.k.a. time slice)
- OS/161 scheduler
- Switch between thread at regular time interval
  - Implemented via hardware timer
- Provide the illusion of concurrency



Job	J1	J2	J3	J4
arrival (a_i)	0	0	0	5
run time (r_i)	5	8	3	2

bristol.ac.uk

How does it work?

- Week 6 Video 5
- We saw how trap were handled
- We focused on syscall
- Here we want to look at timer interrupt
- … let's see how hardware interrupt are handled

How does it work?

- Week 6 Video 5
- We saw how trap were handled
- We focused on syscall
- Here we want to look at timer interrupt
- … let's see how hardware interrupt are handled

Pause the video and look through the code as needed

bristol.ac.uk

How does it work?

- *kern/arch/mips/locore/trap.c*
- Our trap logic
- Line 188
  - Call to mainbus_interrupt which handles the interrupt logic
- Implemented in *kern/arch/sys161/dev/lamebus_machdep.c*
- Line 298
- In this function we determine the cause of the interrupt

How does it work?

- What happens on hardware timer?
  - Reset the timer (line 318)
  - Call to hardclock (line 320)
- hardclock is implemented in *kern/thread/clock.c*
  - Line 93
- thread_consider_migration
  - Implemented in kern/thread/thread.c (line 881)
  - Migrate to another core (we discuss that more later)
- schedule
  - Implemented in kern/thread/thread.c (line 855)
  - Does nothing
  - Can be used to implement more complex algorithm (future video this week)
- thread_yield

thread_yield

- Implemented in kern/thread/thread.c (line 840)
- Calls thread_switch (line 500)
  - Implement the logic to pick the next thread to execute!

thread_yield

- Implemented in kern/thread/thread.c (line 840)
- Calls thread_switch (line 500)
  - Implement the logic to pick the next thread to execute!
- **We dig into this in the next video!**

bristol.ac.uk

University of BRISTOL

Thank you

bristol.ac.uk