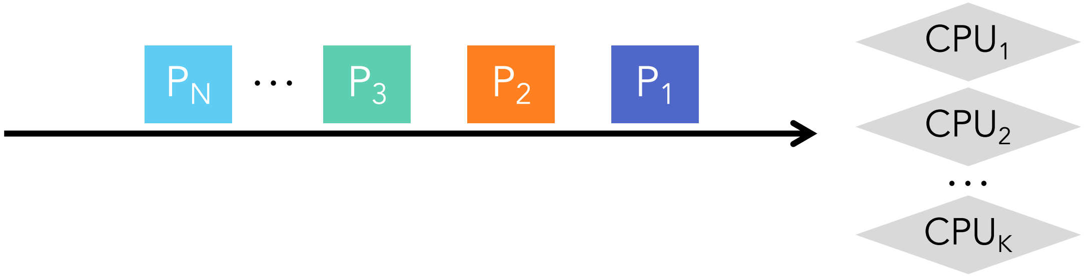
Winter 2025

EECS 482: Introduction to Operating Systems

Lecture 17: CPU Scheduling

Prof. Ryan Huang

CPU scheduling



The scheduling problem:

- Have N jobs ready to run
- Have $\mathrm{K} \geq 1$ CPUs

Choose which job to run on which CPU, for how long?

- We'll refer to schedulable entities as jobs – could be processes, threads, people, etc.

How to choose

- Project 2 and 3: FIFO
- Many other policies possible

Scheduling criteria

Why do we care?

- How do we measure the effectiveness of a scheduling algorithm?

Two broad goals

- Performance
- Fairness

Performance

Throughput – # of processes that complete per unit time

- $\# \; jobs/time$ (higher is better)

Turnaround time – time for each process to complete

- $T_{finish} - T_{start}$ (lower is better)

Response time – time from request to *first* response

- $T_{response} - T_{request}$ (lower is better)
- e.g., key press to echo (not launch to exit)

Above criteria are affected by secondary criteria

- *Waiting time* – $Avg(T_{wait})$ time each process waits in the ready queue
- *CPU utilization* – $\%CPU$ fraction of time CPU doing productive work

Fairness

Second goal of CPU scheduling: share CPU among threads in “fair” manner

What does “fair” mean?

Fairness may conflict with performance

Starvation = extremely unfair

First-come, first-served (FCFS)

Run jobs in order that they arrive

- E.g., Say P_1 needs 24 sec, while P_2 and P_3 need 3.
- Say P_2, P_3 arrived immediately after P_1, get:

Throughput: 3 jobs / 30 sec = 0.1 jobs/sec

Turnaround Time: P_1 : 24, P_2 : 27, P_3 : 30

- Average TT: (24 + 27 + 30) / 3 = 27

Can we do better?

FCFS continued

Suppose we scheduled P_2, P_3, then P_1

- Would get:

Throughput: 3 jobs / 30 sec = 0.1 jobs/sec

Turnaround Time: P_1 : 30, P_2 : 3, P_3 : 6

- Average TT: (30 + 3 + 6) / 3 = 13 – much less than 27

Lesson: scheduling algorithm can reduce TT

Can a scheduling algorithm improve throughput?

- Yes, if jobs require both computation and I/O

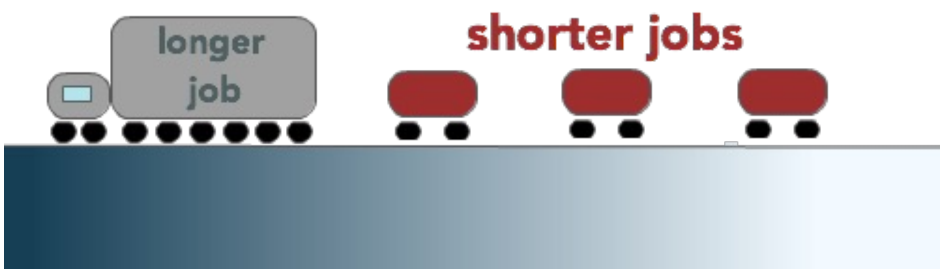
FCFS limitations

FCFS algorithm is non-preemptive in nature

- Once CPU time has been allocated to a process, other processes can get CPU time only after the current process has finished or gets blocked.

Short jobs can be stuck behind long ones

- This property of FCFS scheduling is called *Convoy Effect*



Shortest job first (SJF)

Shortest Job First (SJF)

- Choose the job with the smallest expected CPU burst
  - Person with smallest # of items in shopping cart checks out first

Example

- Three jobs, P_1 needs 8 sec, P_2 needs 4 sec, P_3 needs 2 sec

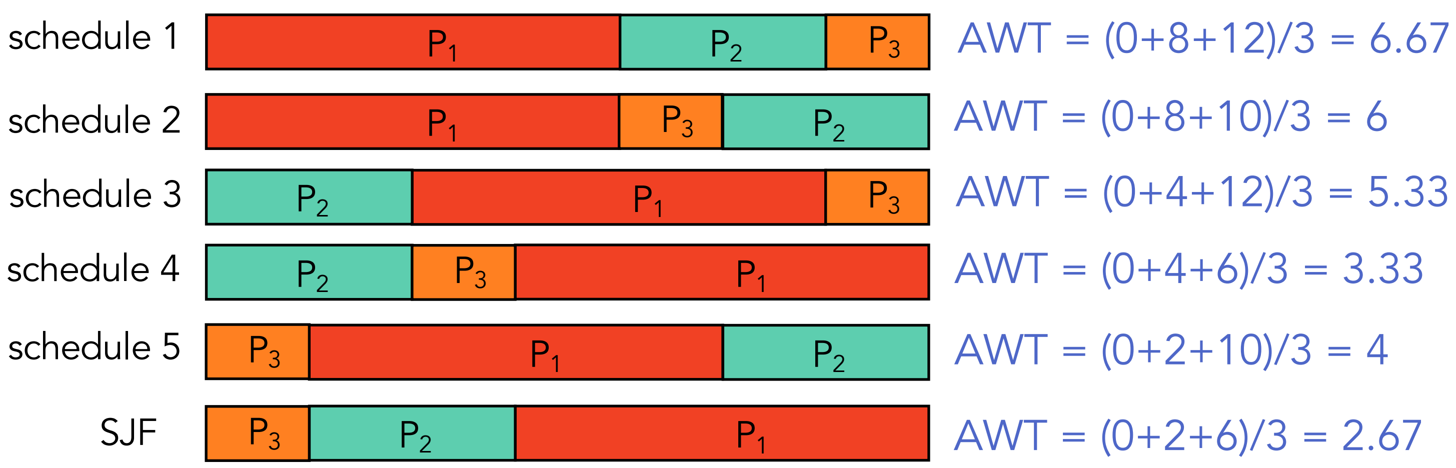
Average Waiting Time: (0 + 2 + 6) / 3 = 2.67

SJF is optimal

SJF has *provably* optimal (minimum) *average waiting time (AWT)*

Previous example: P_1 8 sec, P_2 4 sec, P_3 2 sec

- How many possible schedules?



SJF limitations

Can potentially lead to unfairness or starvation

Impossible to know size of CPU burst ahead of time

- Like choosing person in line without looking inside cart

How can you make a reasonable guess?

- Estimate CPU burst length based on past
- E.g., exponentially weighted average

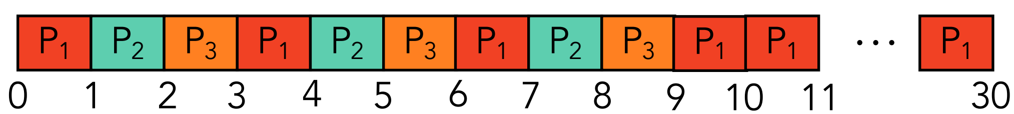
Round robin (RR)

FIFO + periodic preemption

- Each job is given a time slice called a quantum
- Preempt job after duration of quantum
- When preempted, move to back of FIFO queue

Example:

- Three jobs, P_1 needs 24 sec, while P_2 and P_3 need 3 sec
- Quantum is 1 sec



- Average TT: (30 + 8 + 9) / 3 = 15.67
  - Compared to FCFS: 27

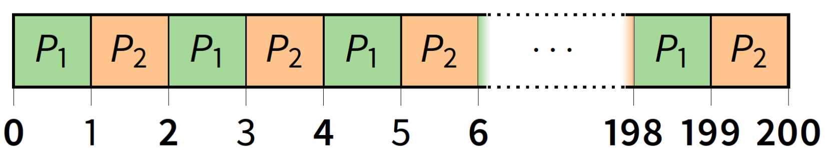
Round robin vs. FCFS

Round robin achieved lower average turnaround time than FCFS in last example

- Is it always the case?

Another example:

- Two jobs, both arrive at time 0, both take 100 sec



- What would the average turnaround time be with RR?
  - Turnaround Time: P_1=199, P_2=200
  - Average TT: (199 + 200) / 2 = 199.5
- How does it compare to FCFS?
  - Turnaround Time: P_1=100, P_2=200
  - Average TT: (100+200) / 2 = 150

Round robin trade-offs

Advantages:

- Fair allocation of CPU across jobs
- Low average waiting time when job lengths vary
- Good for responsiveness if small number of jobs

Disadvantages

- Longer average turnaround time when job lengths are uniform
- Context switches are frequent and need to be very fast

Round robin time quantum

How to pick quantum?

- What if time slice is too long?
- What if time slice is too short?
- Want much larger than context switch cost
- Majority of CPU bursts should be less than quantum
- But not so large system reverts to FCFS

Typical values: 1–100 msec

Priority

Priority scheduling

- Associate a numeric priority with each process
  - E.g., smaller number means higher priority (Unix/BSD)
- Give CPU to the process with highest priority

Problem?

- Starvation: low-priority jobs can wait indefinitely

Solution?

- "Aging"
  - Increase priority as a function of waiting time
  - Decrease priority as a function of CPU consumption

Scheduling summary

Many different policies: FCFS, round robin, SJF, priority, deadline, proportional share, etc.

- OS schedulers use different policies in different settings
  - Interactive, CPU-bound, batch, etc.
- Still actively researched today!

Combining scheduling algorithms to optimize for multiple objectives

- Have multiple queues
- Use a different algorithm for each queue
- Move processes among queues
- Example: Multiple-level feedback queues (MLFQ)