CPU Virtualization: Scheduling

CS 537: Introduction to Operating Systems

Louis Oliphant

University of Wisconsin - Madison

Fall 2024

Administrivia

- Project 1 Due Sep 13th, 11:59pm
- Project 2 Released, Sep 13th
  - Work with xv6 Operating System
  - Create a new system call
  - Create user-level program that uses the new system call
- Will also release **optional** Docker Container for development of xv6 projects
- Can't make an exam time? Fill out this form to request taking during alternate exam time.

Agenda

- Scheduling
  - How does the OS decide what process to run?
  - What are some of the metrics to optimize for?
- Policies
  - How to handle interactive and batch processes?
  - What to do when OS doesn't have complete information?

Review: CPU Virtualization

- A **process** is an OS abstraction for managing a running program. The **process list** contains **PCB** entries for each process and the PCB contains OS managed information (e.g. **process state**, **context**, open files, etc.).
- The OS manages processes using **Limited Direct Execution**:
  - timer **interrupts** to regain control and enforce sharing of the CPU
  - Privilege levels (user-mode and kernel-mode) with **trap** and **return-from-trap** instructions
  - **System calls** to provide services to a process while maintaining security of resources

Review: CPU Virtualization (cont.)

- API for programs to work with processes:
  - `fork()` for duplicating a process
  - `exec()` for replacing a process memory image
    - Can replace stdin, stdout, or stderr before calling exec so process's stream gets redirected.
  - `wait()` for a parent process to wait on any of its children to finish

Vocabulary

Workload Set of **jobs** (arrival time, run time)

Job Current Execution of a process
- Alternates between CPU and I/O
- Moves between ready and blocked queues

Scheduler Decides which ready job to run

Metric Measurement of scheduling quality

Scheduling Approach

Assumptions

1. Each job runs for the same amount of time.
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

Metric

Turnaround Time

Metric 1: Turnaround Time

$$T_{turnaround} = T_{completion} - T_{arrival}$$

Example:

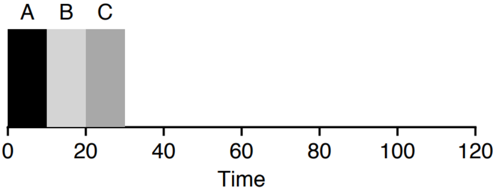
Process A arrives at time t = 10, finishes t = 30
Process B arrives at time t = 10, finishes t = 50

Turnaround Time:

A = 20, B = 40
Average = 30

Policy 1: FIFO / FCFS

Job	arrival(s)	run time (s)	turnaround (s)
A	~0	10	
B	~0	10	
C	~0	10	



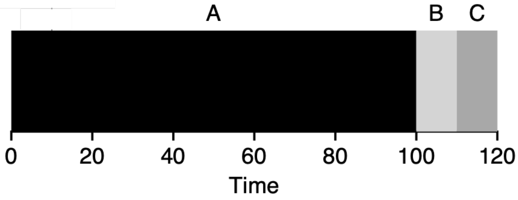
Average
Turnaround
Time =

Assumptions

1. Each job runs for the same amount of time.
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

Short Jobs Suffer

Job	Arrival(s)	run time (s)
A	~0	100
B	~0	10
C	~0	10



what is a possible schedule that could be better?

Challenge

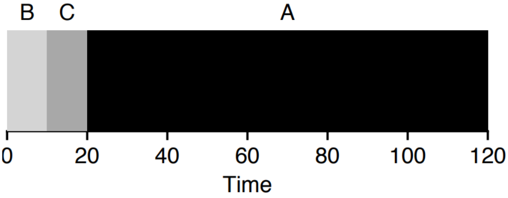
Turnaround time suffers when short jobs must wait for long jobs

New scheduler:

- SJF (Shortest Job First)
- Choose job with smallest runtime

Policy 2: SJF

Job	Arrival(s)	run time (s)	Turnaround (s)
A	~0	100	
B	~0	10	
C	~0	10	



Average Turnaround Time

Assumptions

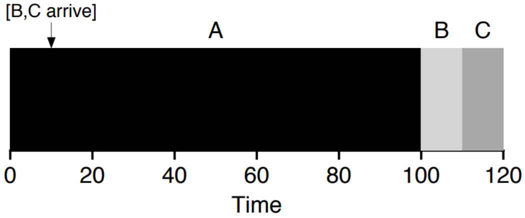
1. ~~Each job runs for the same amount of time.~~
2. All jobs arrive at the same time
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

What will be the Schedule with SJF?

Job	Arrival(s)	run time (s)	Turnaround (s)
A	~0	100	
B	10	10	
C	10	10	

SJF Average Turnaround Time

Job	Arrival(s)	run time (s)	Turnaround (s)
A	~0	100	
B	10	10	
C	10	10	



Average Turnaround Time?

Policy 3: STCF (Preemptive Scheduling)

Previous Schedulers:

FIFO and SJF are non-preemptive

Only schedule new job when previous job voluntarily relinquishes CPU

New Scheduler:

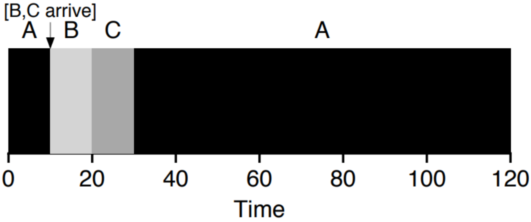
Preemptive: Schedule different job by taking CPU away from running job

STCF (Shortest Time-to-Completion First)

Always run job that will complete the quickest

Preemptive STCF

Job	Arrival(s)	run time (s)	Turnaround (s)
A	~0	100	
B	10	10	
C	10	10	



Average Turnaround Time?

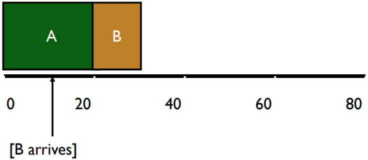
Metric 2: Response Time

$$T_{response} = T_{firstrun} - T_{arrival}$$

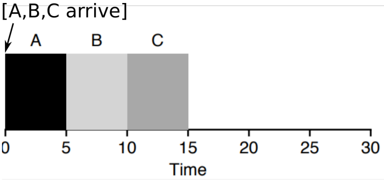
Example:

Process B arrives at time t = 10, starts t = 20, finishes t = 30

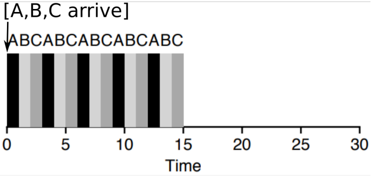
B's turnaround = 20s
B's response = 10s



Policy 4: Round Robin



FCFS



Round robin every 1s

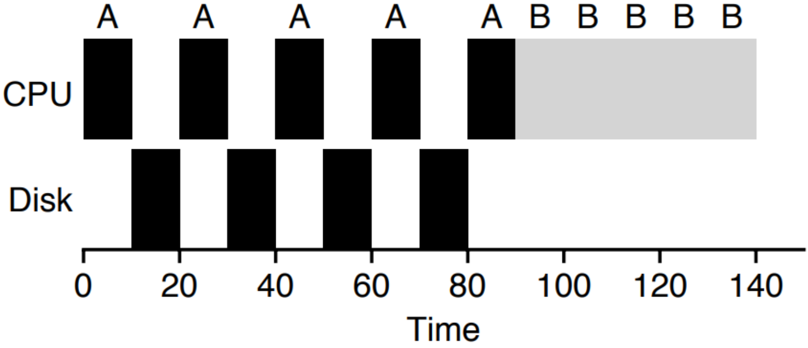
- Key Idea: Switch more often to reduce response time.
- Challenges:
    - Tuning: What is a good time slice?
    - What is the overhead of a context switch?

Quiz 2: https://tinyurl.com/cs537-fa24-q2

Assumptions

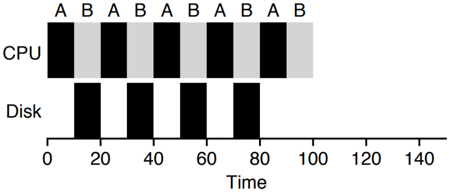
1. ~~Each job runs for the same amount of time.~~
2. ~~All jobs arrive at the same time~~
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

Not I/O Aware



Job Holds Onto CPU!

I/O Aware Scheduling



- Treat Job A as several separate jobs (A_1, A_2, A_3, etc.), one for each CPU burst.
- When Job A_n completes I/O, another Job A_{n+1} is ready
- Each CPU burst is shorter than job B
- With STCF, Job A preempts job B

Assumptions

1. ~~Each job runs for the same amount of time.~~
2. ~~All jobs arrive at the same time~~
3. ~~All jobs only use the CPU (no I/O)~~
4. Run-time of each job is known

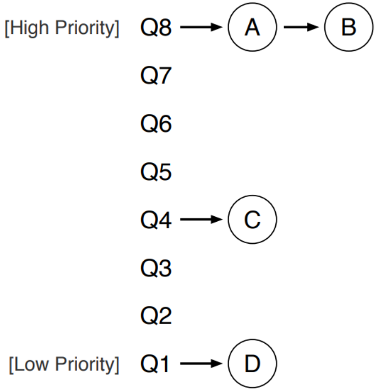
Multi-Level Feedback Queue (MLFQ)

- General Purpose Scheduler
- Must support two job types with distinct goals:
  - "interactive" programs care about response time (RR optimal)
  - "batch" programs care about turnaround time (STCF optimal)

How can a scheduler both minimize response time for interactive jobs and minimize turnaround for batch jobs **without knowing *a priori*** the job length?

- Approach: (Won Turing Award)
  - Multiple levels of round-robin
  - Each level has higher priority than lower level
  - Can preempt them

MLFQ Example



RULES:
Rule 1: If Priority(A) $>$ Priority(B) then A runs

Rule 2: If Priority(A) $==$ Priority(B) then A&B run in RR

Challenges

- How to set the starting priority of a job?
- How jobs move between queues?

Approach:

- Use past behavior to predict future behavior!

More MLFQ Rules

Rule 1: If Priority(A) $>$ Priority(B) then A runs

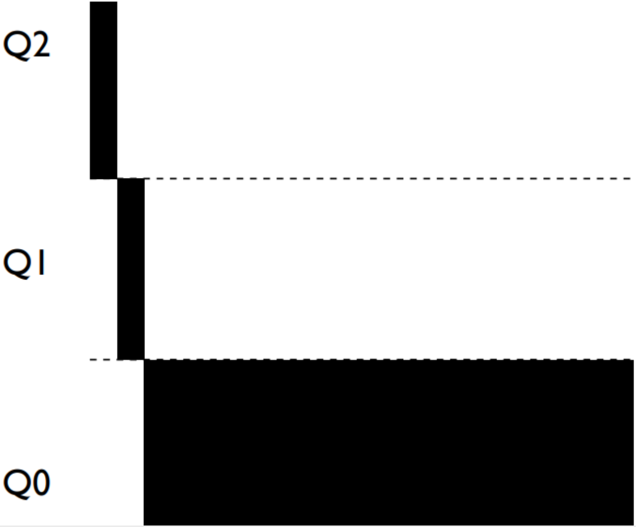
Rule 2: If Priority(A) $==$ Priority(B) then A&B run in RR

Rule 3: Jobs start at top priority

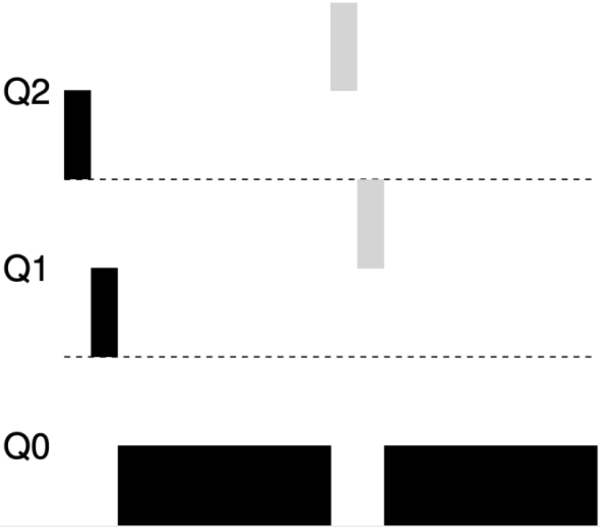
Rule 4: If job uses whole time slice, demote process

(longer time slices at lower priorities)

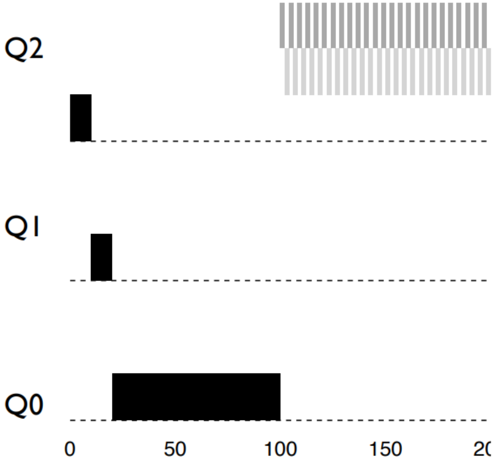
One Long Job



Short Job Joins

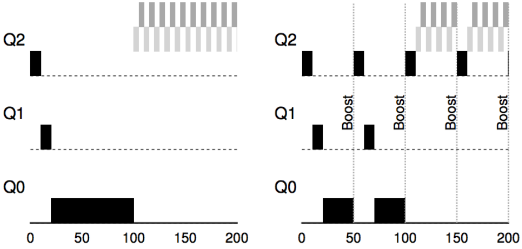


Interactive Jobs Mixed with Batch Jobs



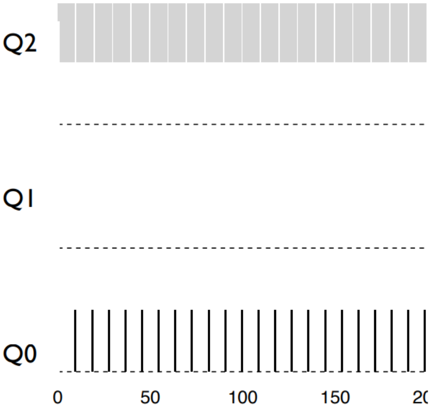
What is the problem?

Avoiding Starvation: Boosting



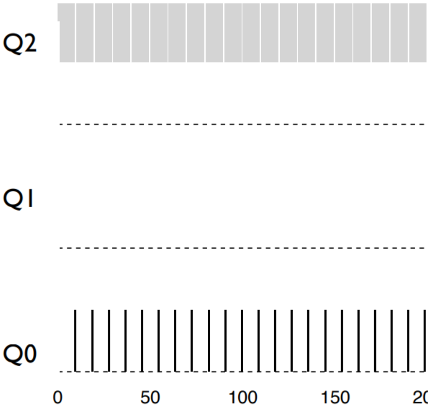
Rule 5: After some time period S, move all jobs to the topmost queue.

Gaming The Scheduler



Job could trick scheduler by doing I/O just before time-slice ends

Gaming The Scheduler



Job could trick scheduler by doing I/O just before time-slice ends

Rule 4*: Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced.

Summary

- Scheduling
  - How does OS decide?
    Understand workload characteristics (e.g. job types, arrival times)
  - What metrics to optimize?
    Select metrics that match goals (e.g. turnaround, response)
- Policies
  - How handle interactive vs. batch jobs?
    understand trade-offs based on goals, metrics (RR vs. STCF)
  - What to do with incomplete information?
    use past to predict future

Other CPU Scheduling Concepts

- Proportional Share Scheduler
  - Try to guarantee each job obtain percentage of CPU time
  - Lottery Scheduling (probabilistic fair-share)
  - Stride Scheduling (deterministic fair-share)
  - Completely Fair Scheduler (Linux's scheduler)
- Multiprocessor Scheduling
  - Multiprocessor Architecture (e.g. caching – coherence, affinity)
  - Synchronization
  - Single Queue Multiprocessor Scheduling (SQMS)
  - Multi-Queue Multiprocessor Scheduling (MQMS)
  - O(1), BFS, Completely Fair Scheduler