

CPU SCHEDULING

Shivaram Venkataraman

CS 537, Spring 2020

ADMINISTRIVIA

- Project Ia is due tomorrow (Jan 29) at 10pm
- Still on the waitlist?
- Project Ib out tomorrow

AGENDA / LEARNING OUTCOMES

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?

RECAP

RECAP: SCHEDULING MECHANISM

Process: Abstraction to virtualize CPU

Role of the OS

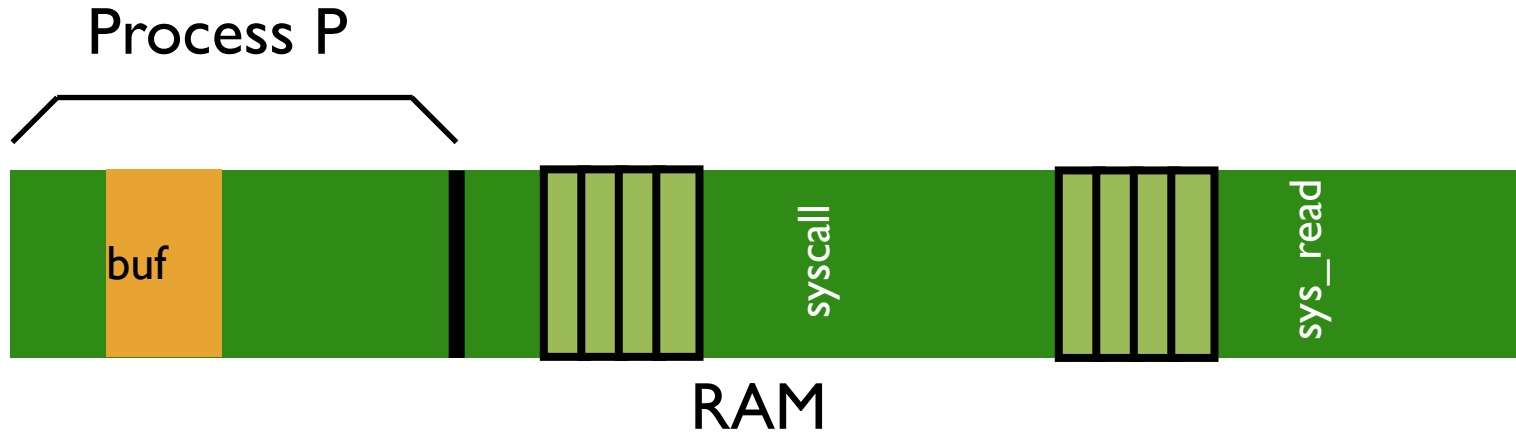
Protection: How can we ensure user process can't harm others?

Sharing: Reschedule processes for fairness, efficiency

RECAP: SYSCALL

Separate user-mode from kernel mode for security

Syscall: call kernel mode functions



DISPATCH MECHANISM

OS runs **dispatch loop**

```
while (1) {  
    run process A for some time-slice  
    stop process A and save its context  
    load context of another process B  
}
```

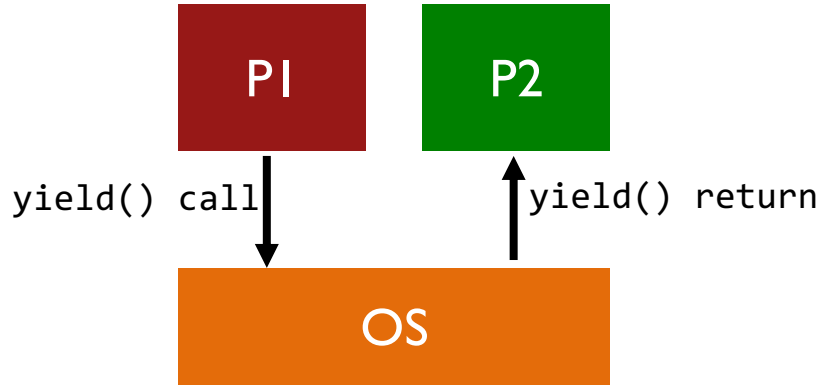
Question 1: How does dispatcher gain control?

Question 2: What must be saved and restored?

HOW DOES DISPATCHER GET CONTROL?

- Option 1: **Cooperative Multi-tasking**: Trust process to relinquish CPU through traps
- Examples: System call or error (illegal instruction or divide by zero) etc.
 - Provide special `yield()` system call

Disadvantages?



TIMER-BASED INTERRUPTS

Option 2: **Timer-based Multi-tasking**

Guarantee OS can obtain control periodically

Enter OS by enabling periodic alarm clock

Hardware generates timer interrupt (CPU or separate chip)

Example: Every 10ms

User must not be able to mask timer interrupt

Operating System

Hardware

Program

Process A

timer interrupt
save regs(A) to k-stack(A)
move to kernel mode
jump to trap handler

Handle the trap

Call switch() routine

save kernel regs(A) to proc-struct(A)

restore kernel regs(B) from proc-struct(B)

switch to k-stack(B)

return-from-trap (into B)

Operating System

Hardware

Program Process A

Handle the trap
Call switch() routine
save kernel regs(A) to proc-struct(A)
restore kernel regs(B) from proc-struct(B)
switch to k-stack(B)
return-from-trap (into B)

timer interrupt
save regs(A) to k-stack(A)
move to kernel mode
jump to trap handler

restore regs(B) from k-stack(B)
move to user mode
jump to B's IP

Process B

SUMMARY

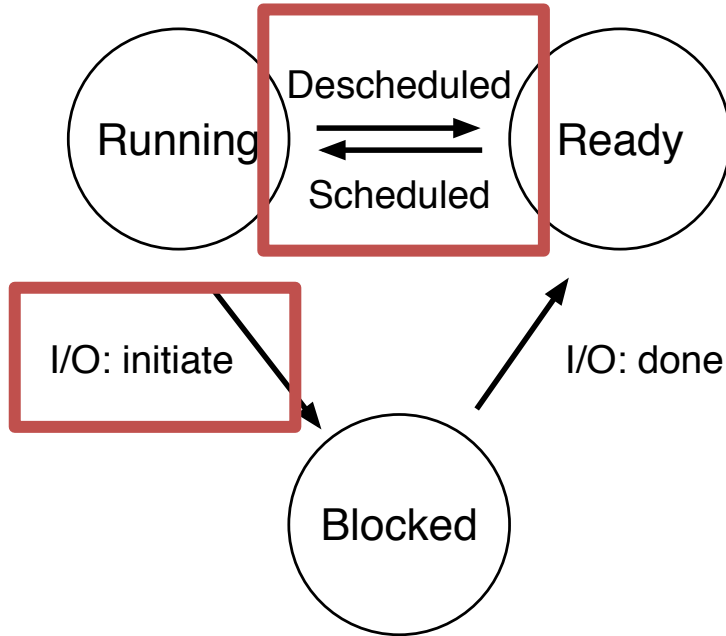
Process: Abstraction to virtualize CPU

Use time-sharing in OS to switch between processes

Key aspects

- Use system calls to run access devices etc. from user mode

- Context-switch using interrupts for multi-tasking



POLICY ?

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

APPROACH

Assumptions

Metric



Scheduling
policy

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 1: TURNAROUND TIME

Turnaround time = *completion_time* - *arrival_time*

Example:

Process A arrives at time $t = 10$, finishes $t = 30$

Process B arrives at time $t = 10$, finishes $t = 50$

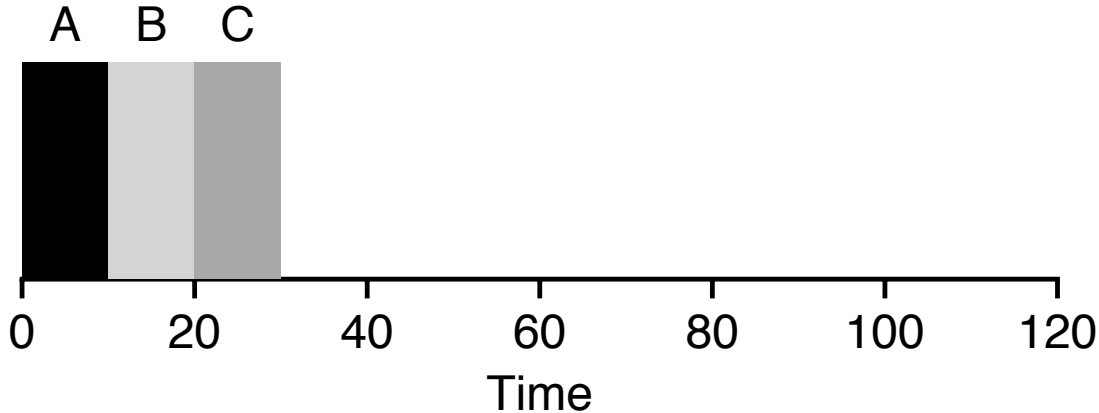
Turnaround time

A = 20, B = 40

Average = 30

FIFO / FCFS

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



Average
Turnaround
Time =

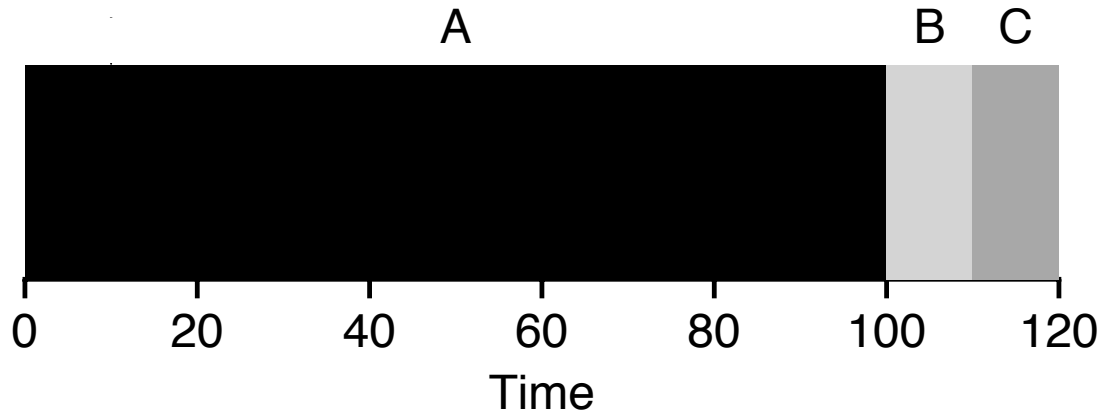
ASSUMPTIONS

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

QUIZ3

<https://tinyurl.com/cs537-sp20-quiz3>

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



Average
Turnaround Time ?

What is one schedule
that could be better?

CHALLENGE

Turnaround time suffers when short jobs must wait for long jobs

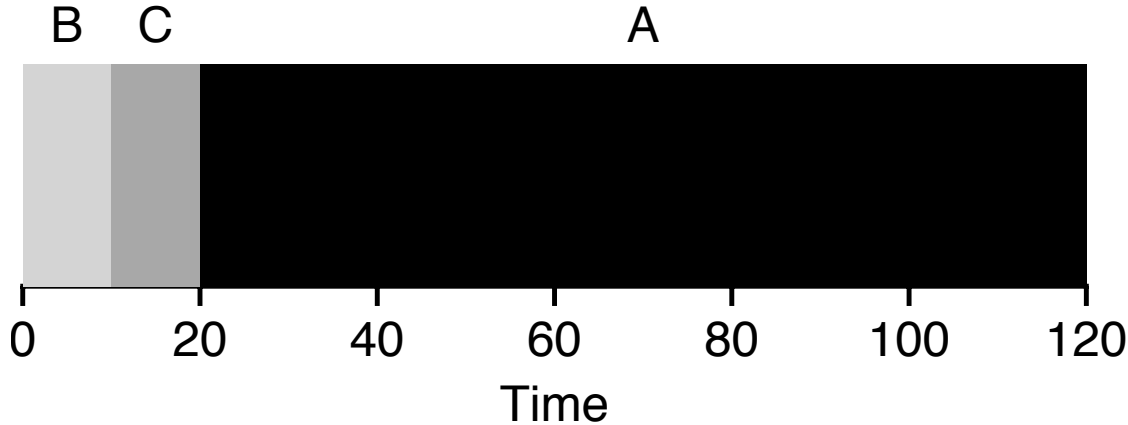
New scheduler:

SJF (Shortest Job First)

Choose job with smallest run time!

SHORTEST JOB FIRST (SJF)

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



Average
Turnaround
Time

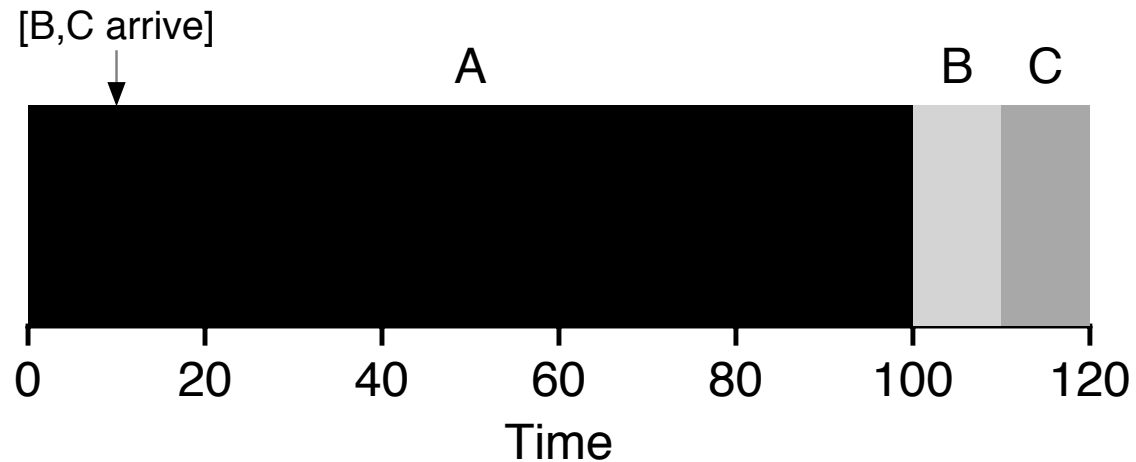
ASSUMPTIONS

- ~~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

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

What will be the schedule with SJF?

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



Average
Turnaround
Time ?

$$(100 + 110 + 120) / 3 = 110s$$

PREEMPTIVE SCHEDULING

Previous schedulers:

FIFO and SJF are non-preemptive

Only schedule new job when previous job voluntarily relinquishes CPU

New scheduler:

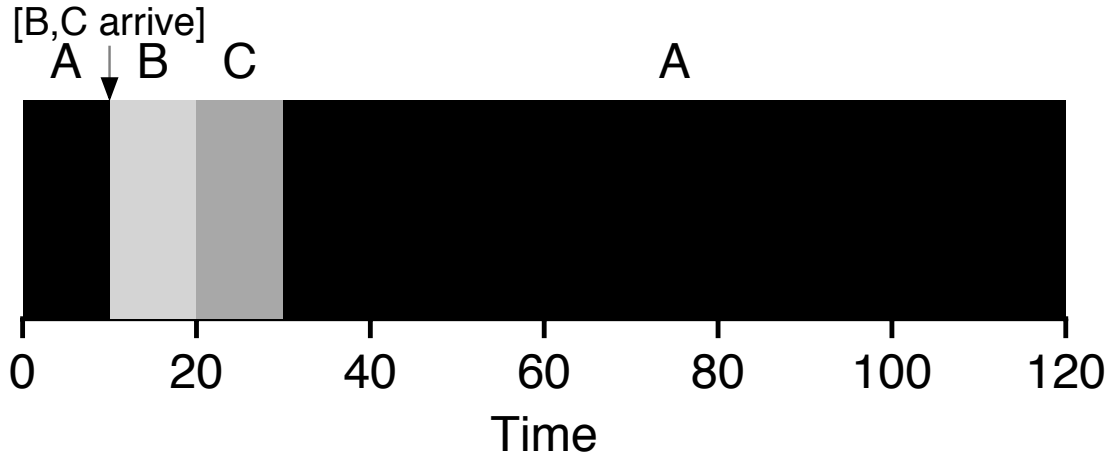
Preemptive: Schedule different job by taking CPU away from running job

STCF (Shortest Time-to-Completion First)

Always run job that will complete the quickest

PREMPTIVE SCTF

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



Average
Turnaround
Time

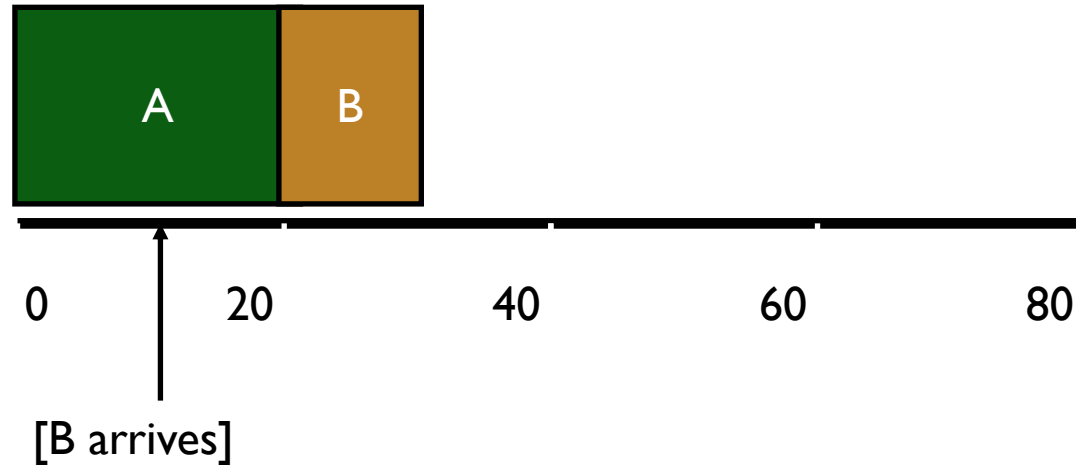
$$(10 + 20 + 120) / 3 = 50s$$

METRIC 2: RESPONSE TIME

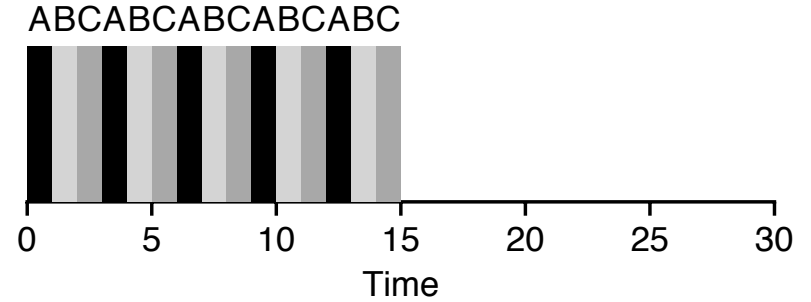
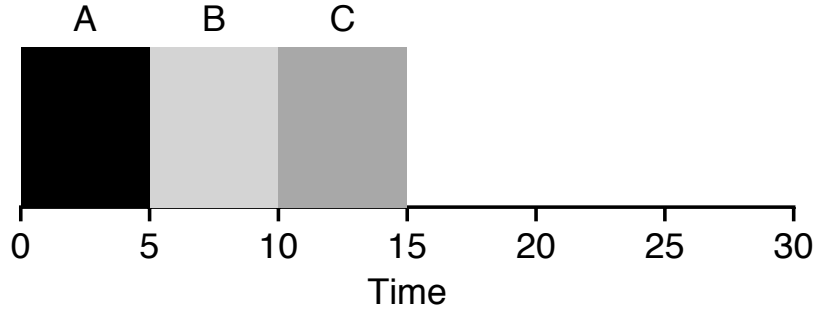
Response time = *first_run_time* - *arrival_time*

B's turnaround: 20s

B's response: 10s



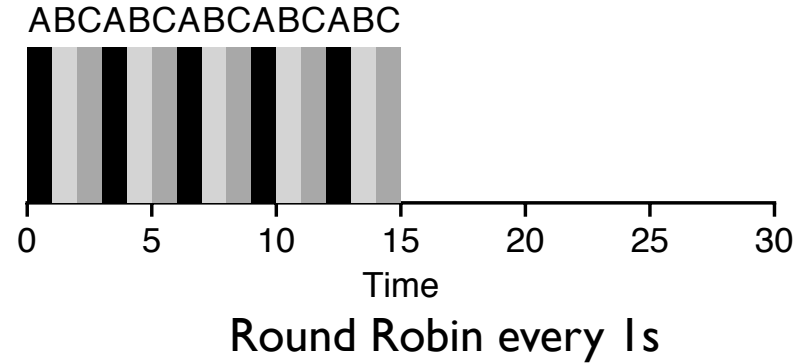
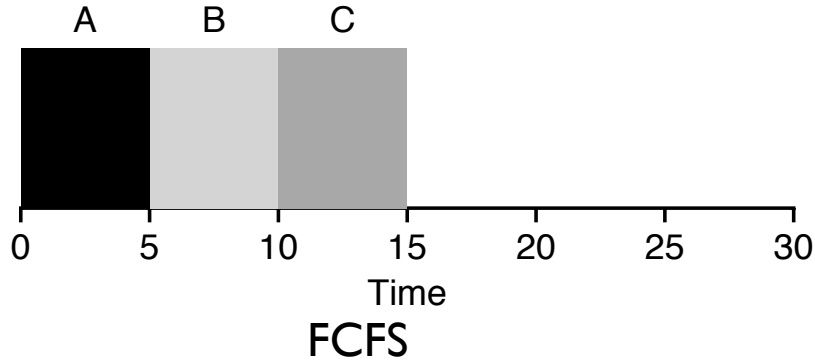
ROUND ROBIN SCHEDULER



Key idea: Switch more often to reduce response time

QUIZ4

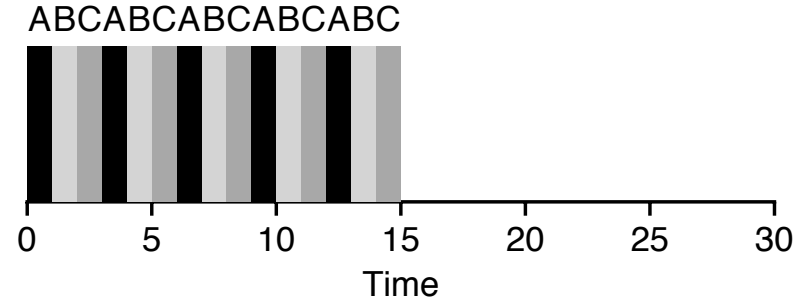
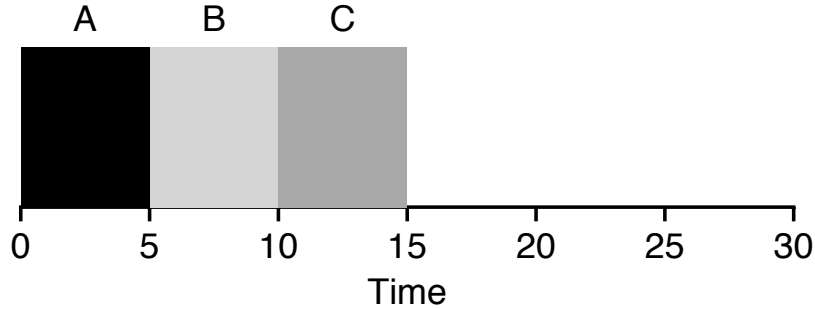
<https://tinyurl.com/cs537-sp20-quiz4>



Average Response Time?

Average Turnaround Time?

QUIZ4: ROUND ROBIN



Average Response Time

$$(0 + 5 + 10)/3 = 5s$$

$$(0 + 1 + 2)/3 = 1s$$

Average Turnaround Time

$$(5 + 10 + 15)/3 = 10s$$

$$(13 + 14 + 15)/3 = 14s$$

TRADE-OFFS

Round robin increases turnaround time, decreases response time

Tuning challenges:

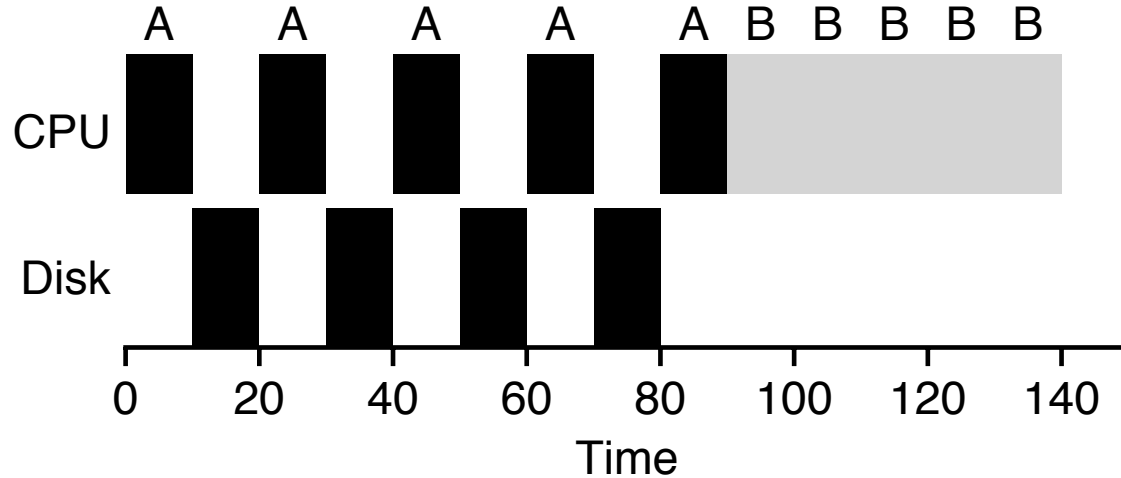
What is a good time slice for round robin?

What is the overhead of context switching?

ASSUMPTIONS

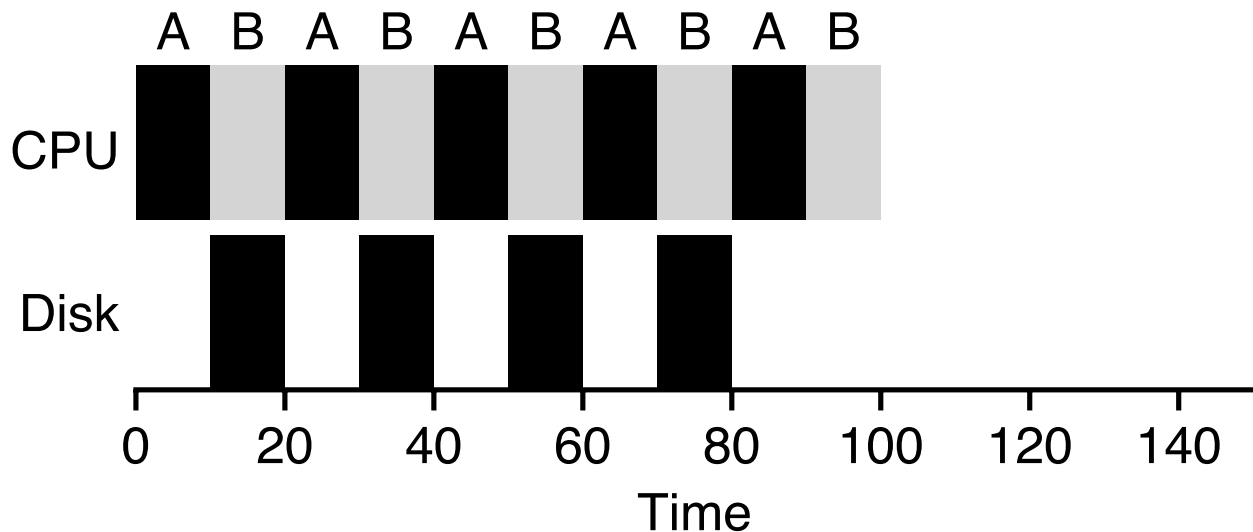
- ~~1. Each job runs for the same amount of time~~
- ~~2. All jobs arrive at the same time~~
3. All jobs only use the CPU (no I/O)
4. Run-time of each job is known

NOT IO AWARE



Job holds on to CPU while blocked on disk!

I/O AWARE SCHEDULING



Each CPU burst is shorter than Job B

With SCTF, Job A preempts Job B

Treat Job A as separate CPU bursts.

When Job A completes I/O, another Job A is ready

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
- “batch” programs care about turnaround time

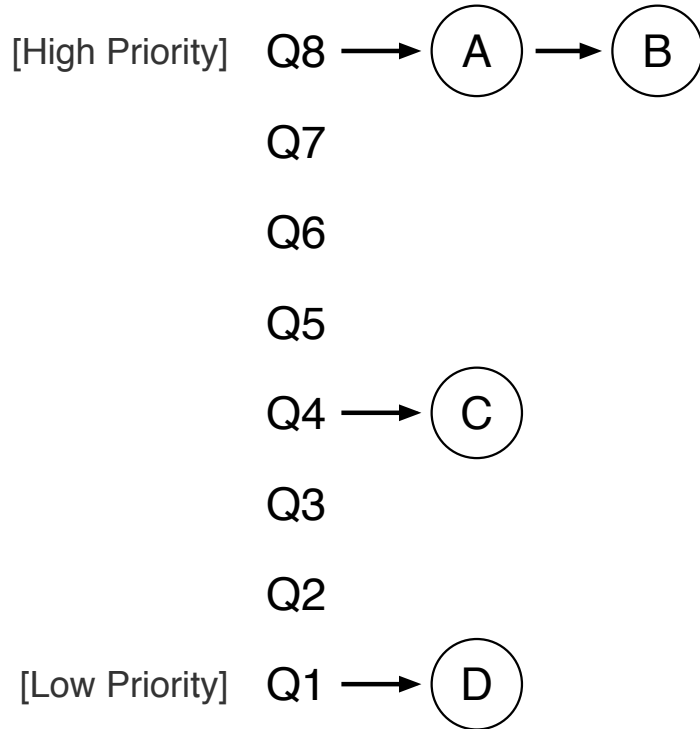
Approach:

Multiple levels of round-robin

Each level has higher priority than lower level

Can preempt them

MLFQ EXAMPLE



“Multi-level” – Each level is a queue!

Rules for MLFQ

Rule 1: If $\text{priority}(A) > \text{Priority}(B)$
A runs

Rule 2: If $\text{priority}(A) == \text{Priority}(B)$,
A & B run in RR

CHALLENGES

How to set priority?

What do we do when a new process arrives?

Does a process stay in one queue or move between queues?

Approach:

Use past behavior of process to predict future!

Guess how CPU burst (job) will behave based on past CPU bursts

MORE MLFQ RULES

Rule 1: If $\text{priority}(A) > \text{Priority}(B)$, A runs

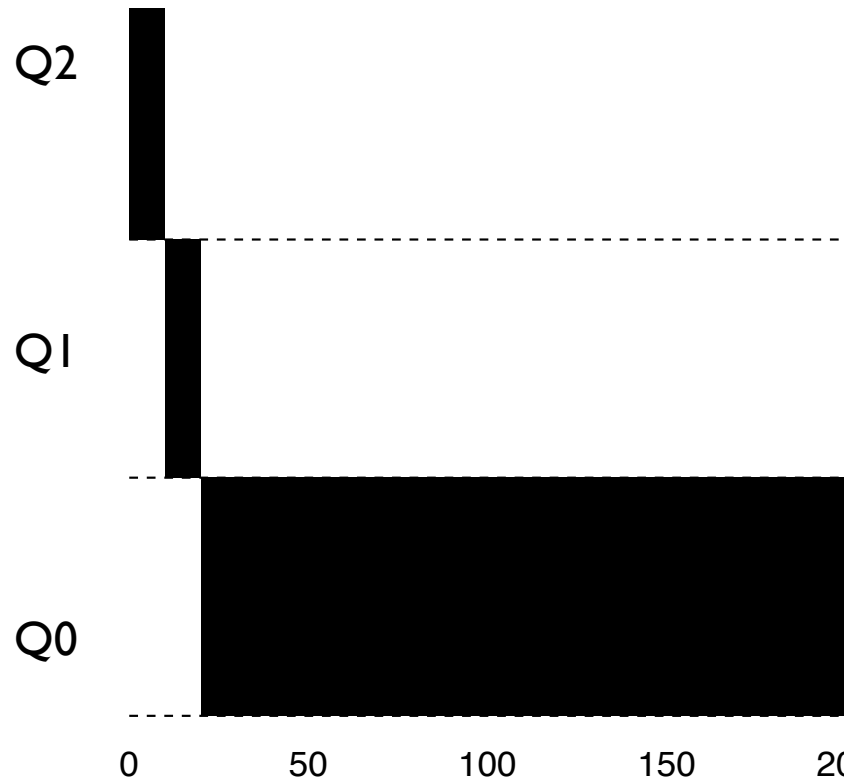
Rule 2: If $\text{priority}(A) == \text{Priority}(B)$, A & B run in RR

Rule 3: Processes start at top priority

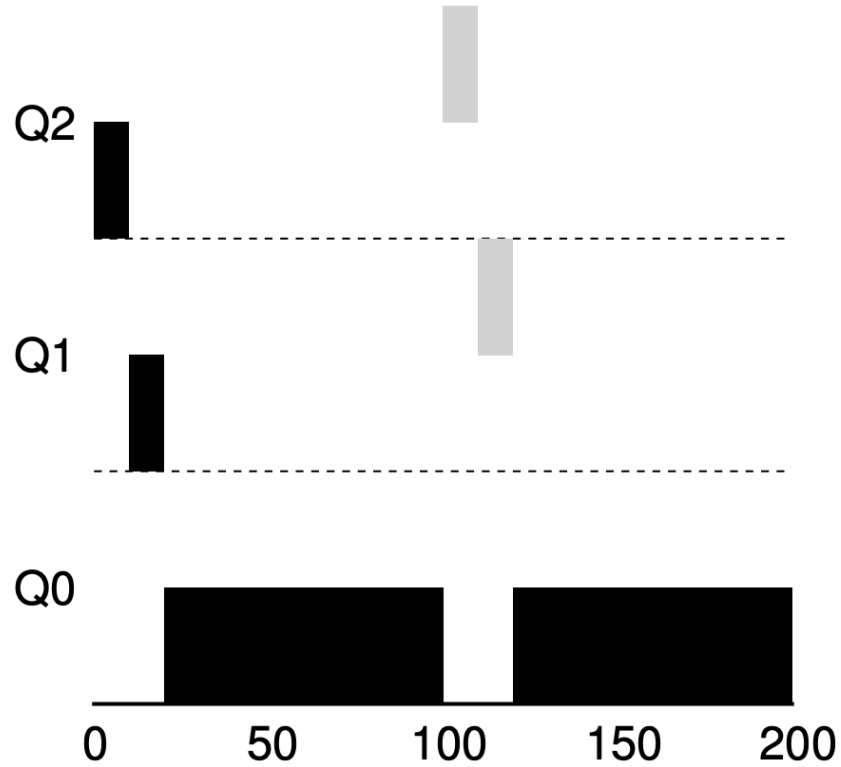
Rule 4: If job uses whole slice, demote process

(longer time slices at lower priorities)

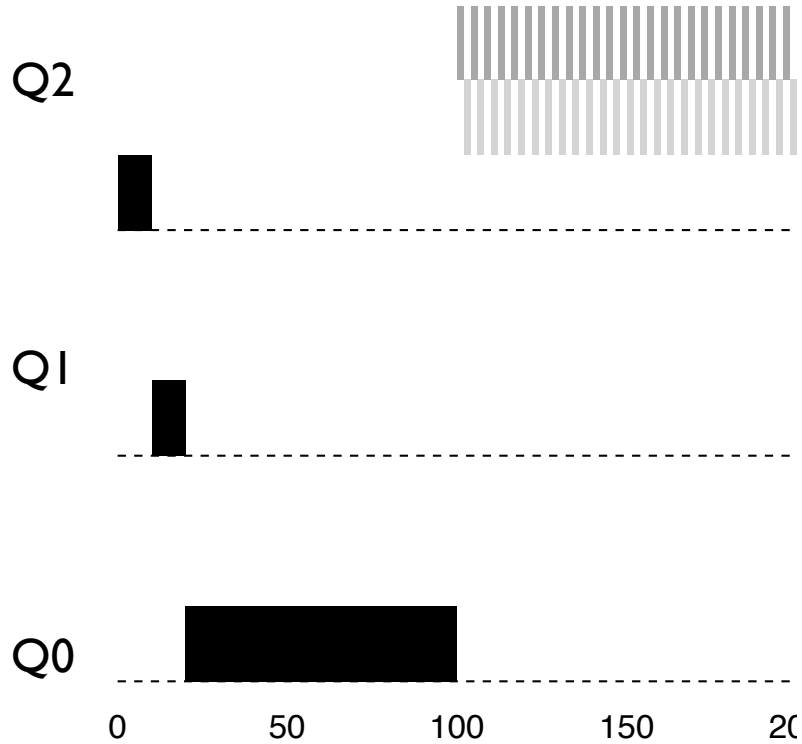
ONE LONG JOB



INTERACTIVE PROCESS JOINS

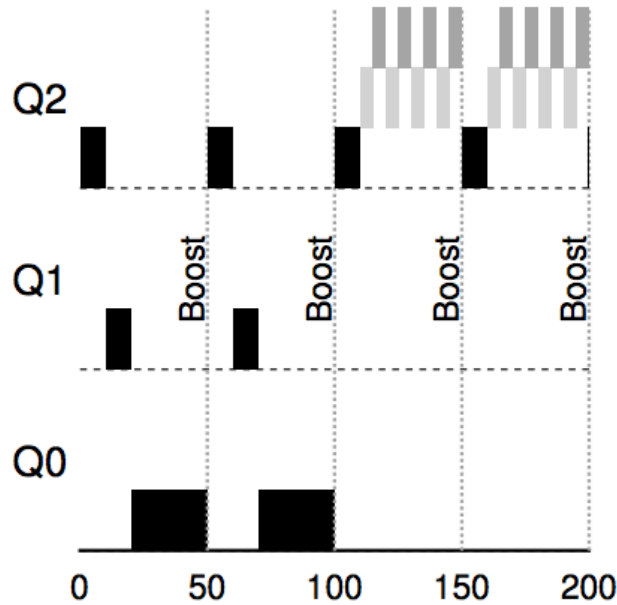
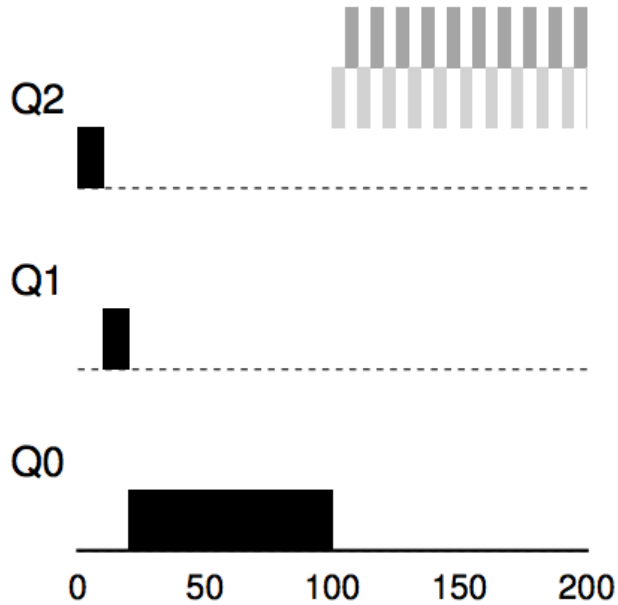


MLFQ PROBLEMS?



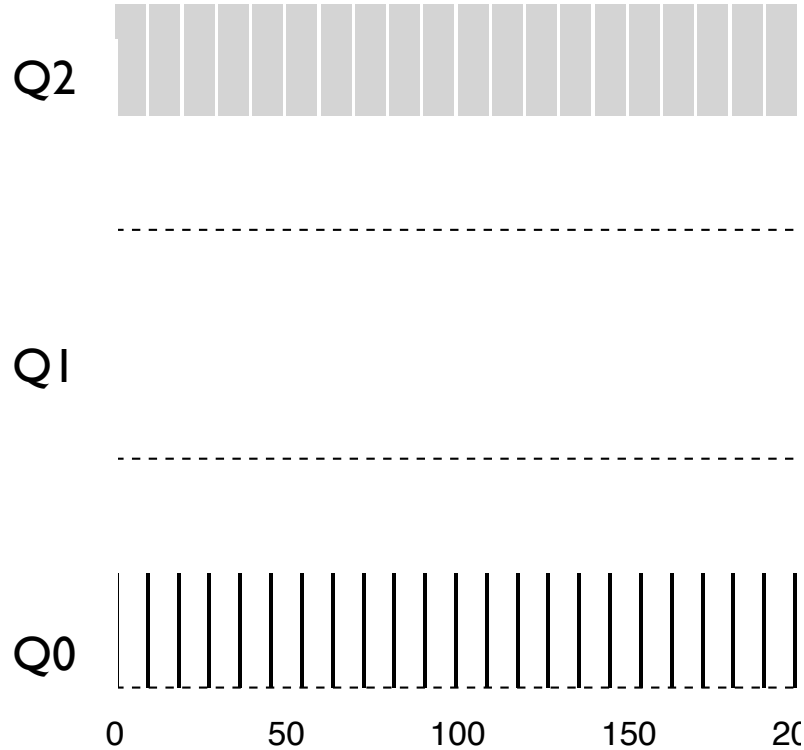
What is the problem with this schedule ?

AVOIDING STARVATION



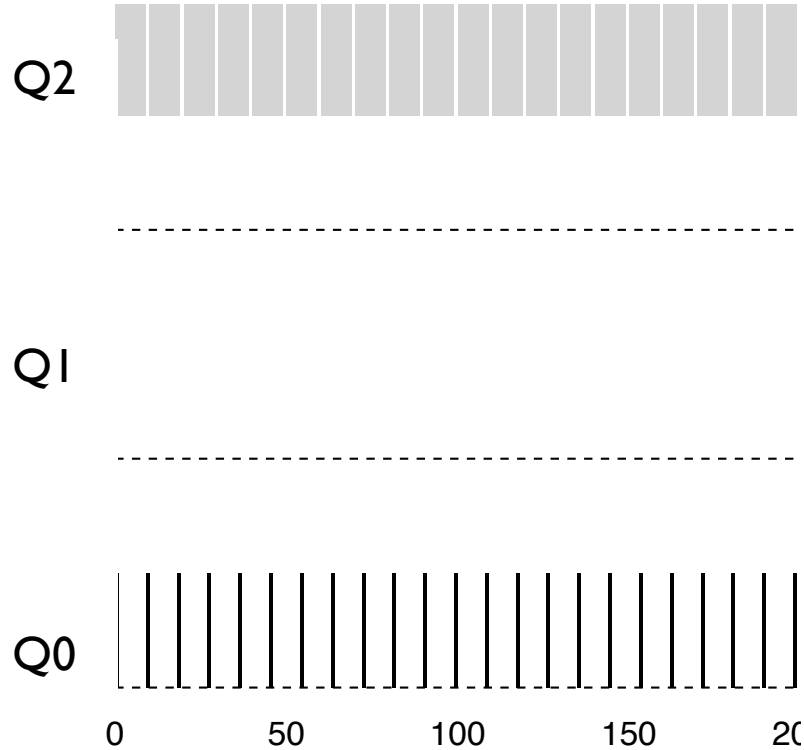
Rule 5: After some time period S , move all the jobs in the system to the topmost queue.

GAMING THE SCHEDULER ?



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

GAMING THE SCHEDULER ?



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

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 Policies

Understand **workload characteristics** like arrival, CPU, I/O

Scope out goals, **metrics** (turnaround time, response time)

Approach

Trade-offs based on goals, metrics (RR vs. SCTF)

Past behavior is good predictor of future behavior?

NEXT STEPS

Project Ia: Due **tomorrow** at 10.00pm

Project Ib: Out tomorrow

Thursday class, discussion

- More advanced scheduling policies

- Summary / review of process, CPU scheduling

- xv6 introduction, walk through

- Go through xv6 context switch / syscall?