CS 6013: Systems 1
Operating Systems + Computer Architecture

Lecture 10: Scheduling (Part I)

Book: Chapters 6 and 7

Review

- We discussed fork(), exec(), wait().
- We talked about LDE and the challenges therein.
  - First challenge: restricting access to sensitive state of the CPU. **Solution**: Privilege Rings.
  - Second challenge: preventing denial of service without sacrificing efficiency. **Solution**: Context switch and scheduling.
- Separate mechanism (details of context switch) from policy (when to context switch)
- Today: wrap-up context switch, dive into process scheduling (policy).

Recall: 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
}
```

Context-switch

Cooperative vs Preemptive

- Cooperative multitasking relies on processes to yield(), bad idea

- **Use preemptive multitasking!**
  - OS does context switching periodically.
  - Use a timer-based interrupt.
  - Ensure processes all get slices of CPU time.

Operating System	**Hardware**	**Program**
		Process A **…**
	Syscall or timer interrupt Hw switches to kstack Raises to kernel mode Save regs(A) to kstack(A) Jump to trap handler	
Handle the trap Call **switch()** routine Save kstack(A) to PCB(A) Restore regs(B) from PCB(B) Switch to kstack(B) Return-from-trap (into B)		
	Restore regs(B) from kstack(B) Move to user mode Jump to B’s IP	
		Process B **…**

CPU Virtualization: Two Components

Dispatcher (thus far)

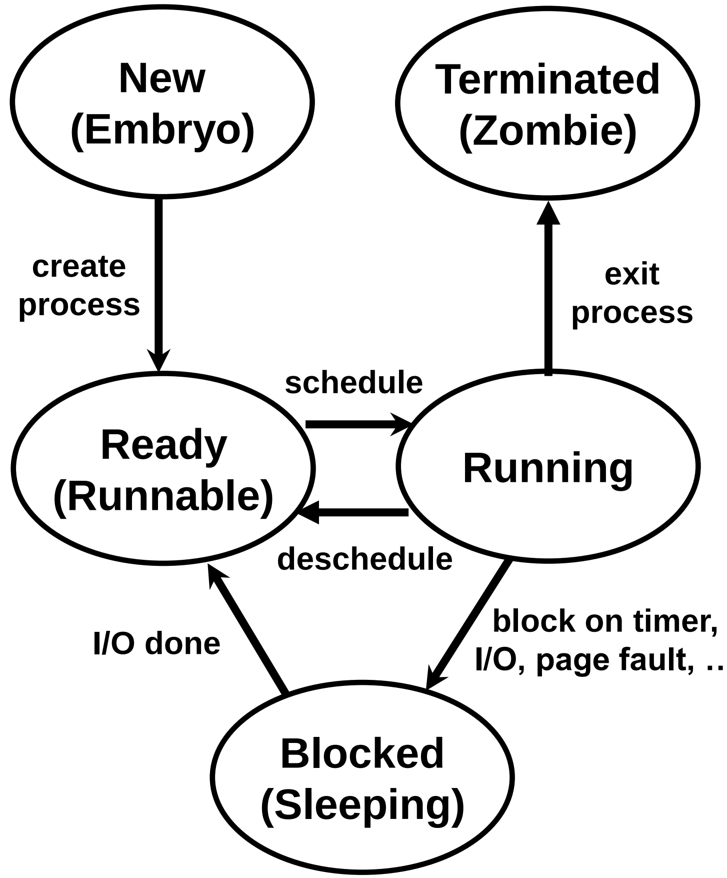
- Low-level mechanism
- Performs context-switch
    - Switch from user mode to kernel mode
    - Save execution state (registers) of old process in PCB
    - Insert PCB in ready queue
    - Load state of next process from PCB to registers
    - Switch from kernel to user mode
    - Jump to instruction in new user process

- Scheduler (now)
    - Policy to determine which process gets CPU when

Review: Process States

How to transition?
("mechanism")

When to transition?
("policy")



Vocabulary

Workload: set of **job** descriptions (arrival time, run time)

- Job: View as current CPU burst of a process
- Process alternates between CPU and I/O
- Moves between ready and blocked queues

Scheduler: logic that decides which ready job to run

Metric: measurement of scheduling quality

Scheduling Performance Metrics

Minimize turnaround time

- Do not want to wait long for job to complete
- Completion_time – arrival_time

Minimize response time

- Schedule interactive jobs promptly so users see output quickly
- Initial_schedule_time – arrival_time

Minimize waiting time

- Do not want to spend much time in Ready queue

Maximize throughput

- Want many jobs to complete per unit of time

Maximize resource utilization

- Keep expensive devices busy

Minimize overhead

- Reduce number of context switches

Maximize fairness

- All jobs get same amount of CPU over some time interval

Workload 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

Scheduling Basics

Workloads:
arrival_time
run_time

Schedulers:
FIFO
SJF
STCF
RR

Metrics:
turnaround_time
response_time

Example: Workload, Scheduler, Metric

JOB	arrival_time (s)	run_time (s)
A	~0	10
B	~0	10
C	~0	10

FIFO: First In, First Out

- also called FCFS (first come first served)
- run jobs in *arrival_time* order

What is our turnaround? *completion_time* - *arrival_time*

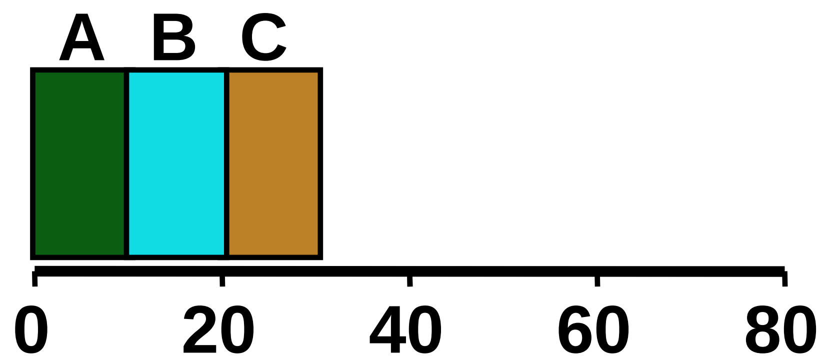
FIFO: Event Trace

JOB	arrival_time (s)	run_time (s)
A	~0	10
B	~0	10
C	~0	10

Time	
0	A arrives
0	B arrives
0	C arrives
0	run A
10	complete A
10	run B
20	complete B
20	run C
30	complete C

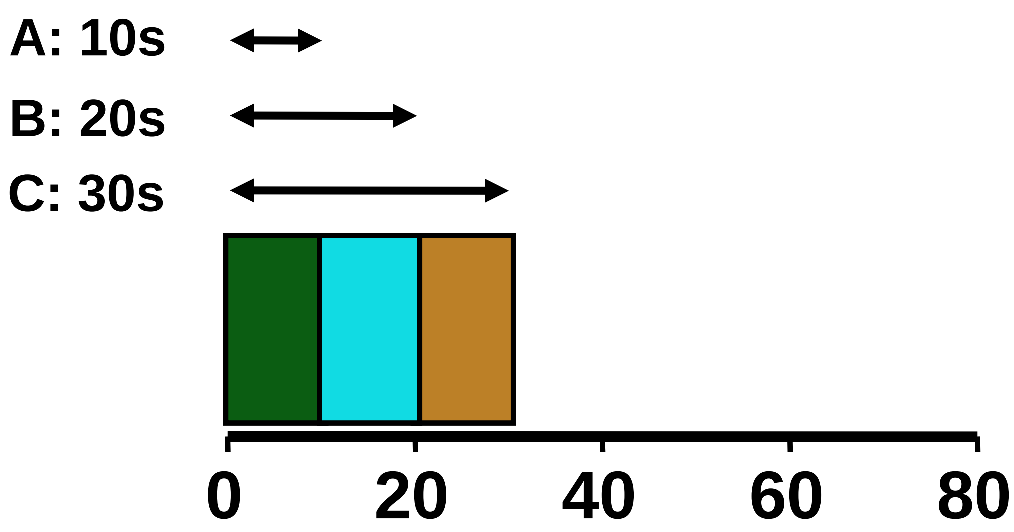
FIFO (Identical Jobs)

JOB	arrival_time (s)	run_time (s)
A	~0	10
B	~0	10
C	~0	10



Gantt chart: **Illustrates how jobs are scheduled over time on a CPU**
(this is a general type of chart used to illustrate how things are laid out over some time period, used across engineering/management/operations/finance)

FIFO (Identical Jobs)



What is the average turnaround time?

turnaround_time = completion_time - arrival_time

(10 + 20 + 30) / 3 = 20s

Scheduling Basics

Workloads:
- arrival_time
- run_time

Schedulers:
- FIFO
- SJF
- STCF
- RR

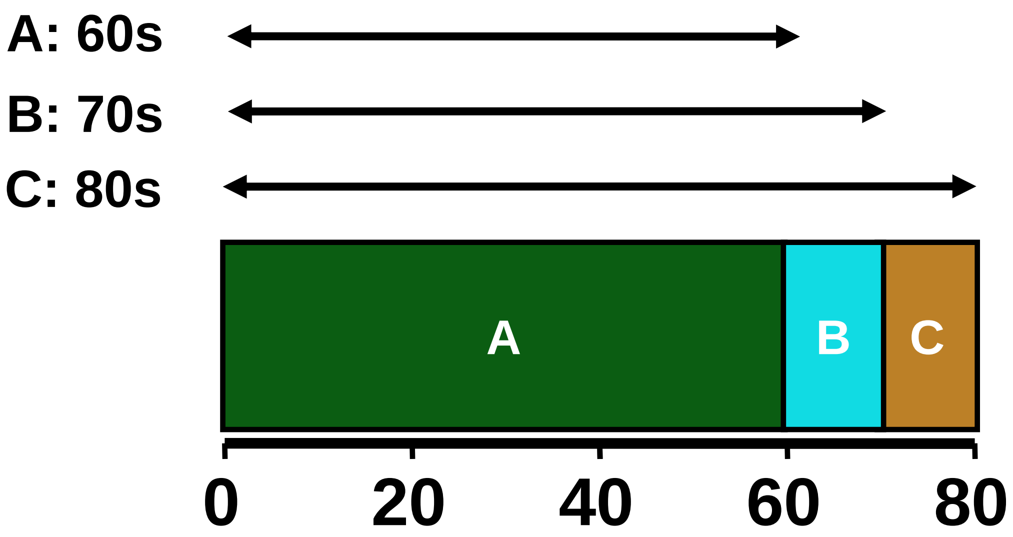
Metrics:
- turnaround_time
- response_time

Workload 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. The run-time of each job is known

Counterexample: Big First Job

JOB	arrival_time (s)	run_time (s)
A	~0	**60**
B	~0	10
C	~0	10



Average turnaround time: 70 s

Convoy Effect

Passing the Tractor

Problem with Previous Scheduler:

FIFO: Turnaround time suffers when short jobs wait behind long jobs

New scheduler:

SJF (Shortest Job First)

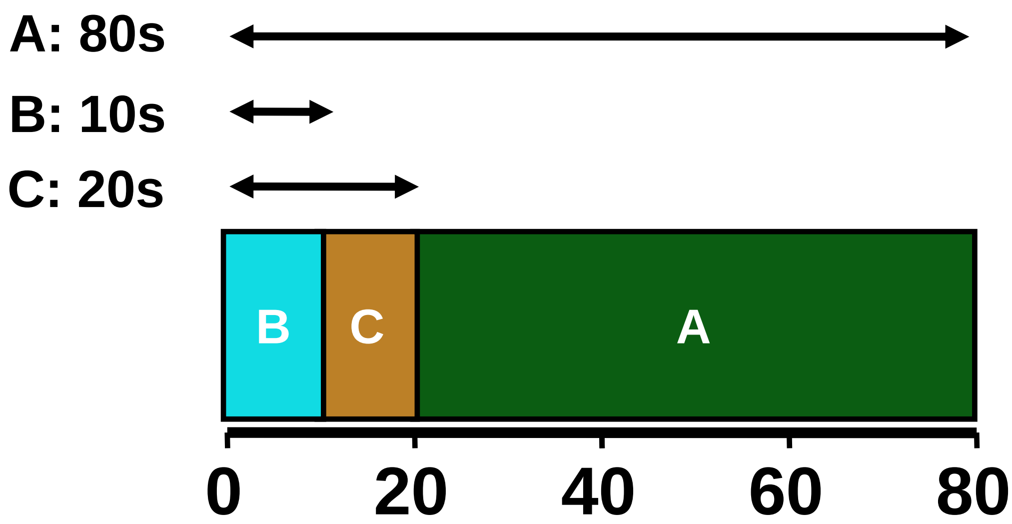
Choose job with smallest *run_time*

Shortest Job First

JOB	arrival_time (s)	run_time (s)
A	~0	**60**
B	~0	10
C	~0	10

What is the average turnaround time with SJF?

SJF Turnaround Time



What is the average turnaround time with SJF?

(80 + 10 + 20) / 3 = ~36.7s

Average turnaround with FIFO: 70s

SJF optimal in minimizing turnaround time (when no preemption)

Shorter job before longer job improves turnaround time of short more than it harms turnaround time of long

Scheduling Basics

Workloads:
- arrival_time
- run_time

Schedulers:
- FIFO
- SJF
- STCF
- RR

Metrics:
- turnaround_time
- response_time

Workload Assumptions

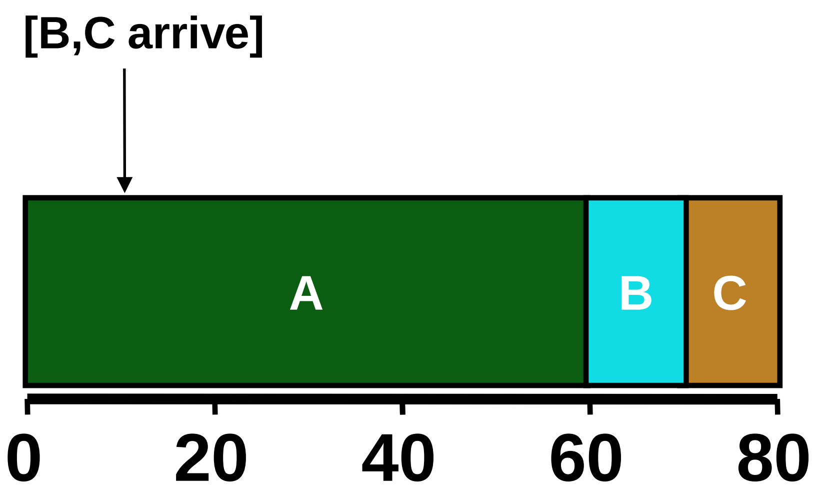
1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. All jobs only use the CPU (no I/O)
4. The run-time of each job is known

Shortest Job First (Arrival Time)

JOB	arrival_time (s)	run_time (s)
A	~0	60
B	**~10**	10
C	**~10**	10

What is the average turnaround time with SJF?

Stuck Behind a Tractor Again



JOB	arrival_time (s)	run_time (s)
A	~0	60
B	**~10**	10
C	**~10**	10

What is the average turnaround time?

(60 + (70 – 10) + (80 – 10)) / 3 = 63.3s

Preemptive Scheduling

Previous schedulers:

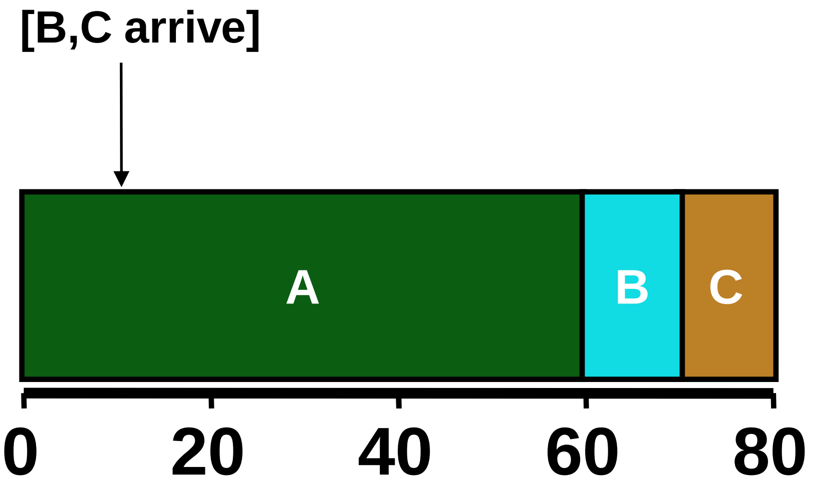
- FIFO and SJF are **non-preemptive**
- Only schedule new job when previous job voluntarily relinquishes CPU (performs I/O or exits)

New scheduler:

- **Preemptive**: Potentially schedule different job at any point by taking CPU away from running job
- **STCF (Shortest Time-to-Completion First)**
- Always run job that will complete the quickest

Non-Preemptive: SJF

JOB	arrival_time (s)	run_time (s)
A	~0	60
B	**~10**	10
C	**~10**	10

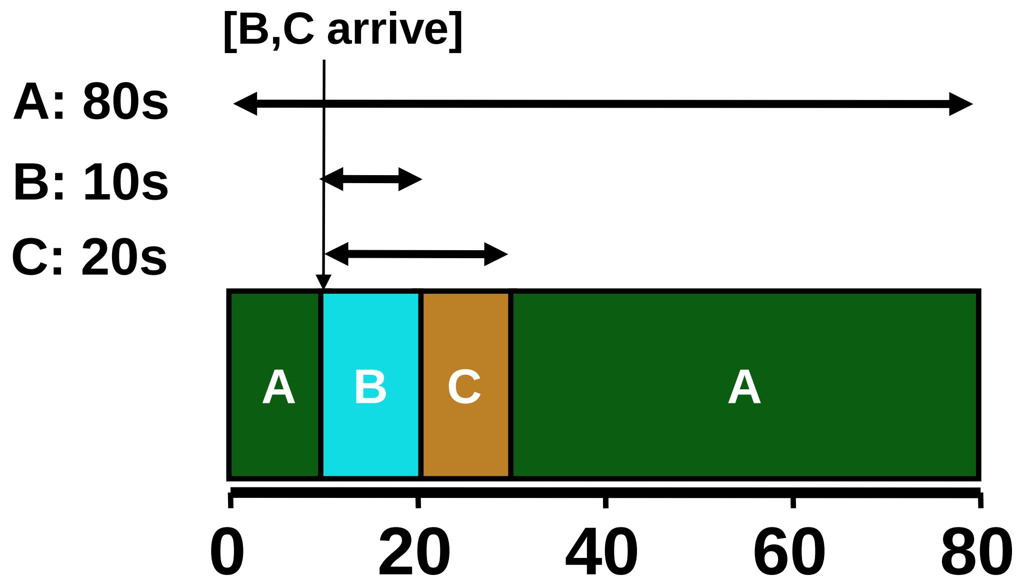


Average turnaround time:

(60 + (70 – 10) + (80 – 10)) / 3 = 63.3s

Preemptive: STCF

JOB	arrival_time (s)	run_time (s)
A	~0	60
B	**~10**	10
C	**~10**	10



Average turnaround time with STCF? 36.7

Average turnaround time with SJF: 63.3s

Scheduling Basics

Workloads:
- arrival_time
- run_time

Schedulers:
- FIFO
- SJF
- STCF
- RR

Metrics:
- turnaround_time
- response_time

Response Time

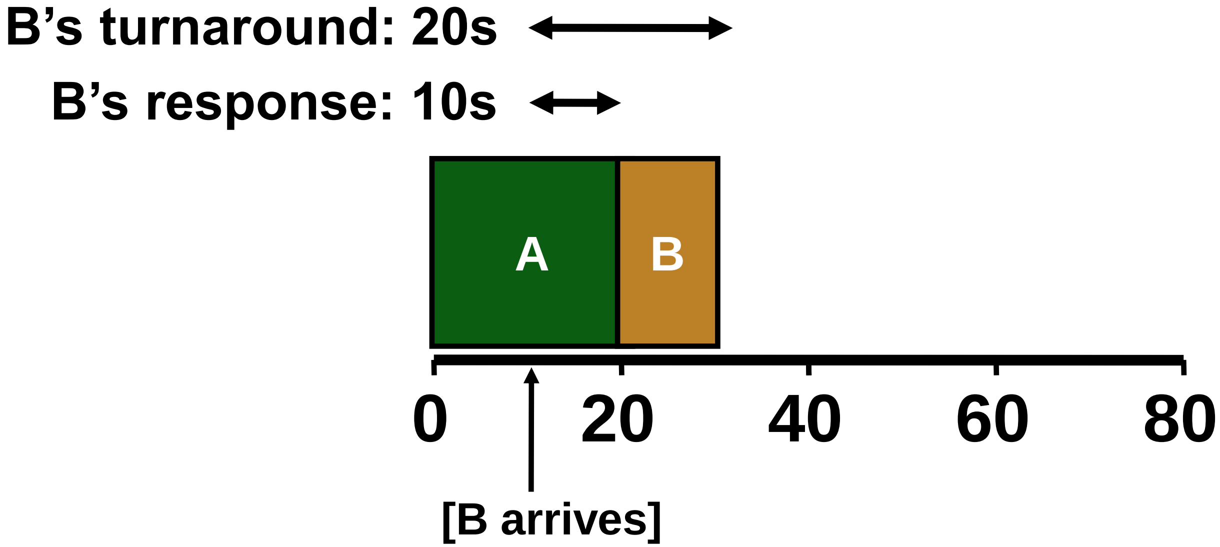
SCTF okay for batch systems, but for time sharing systems when a job completes is less important

Sometimes care about when job starts instead of when it finishes

New metric:

response_time = first_run_time - arrival_time

Response vs. Turnaround



Round-Robin Scheduler

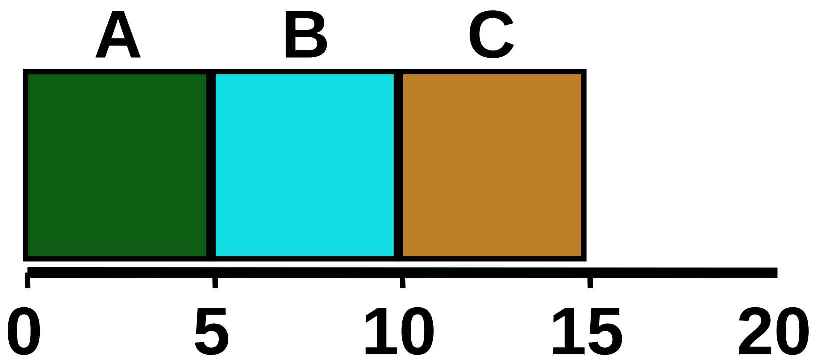
Previous schedulers:

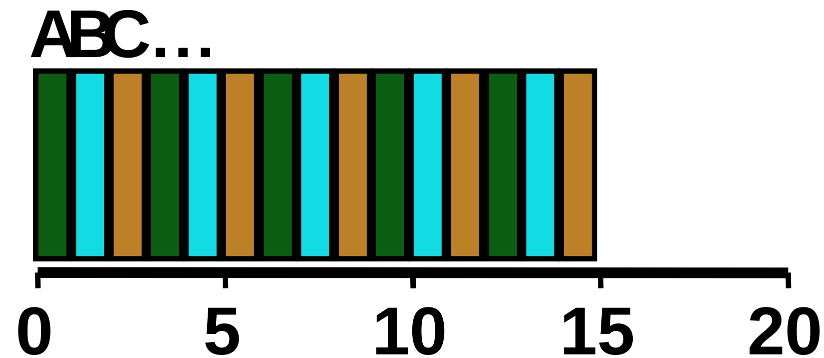
- FIFO, SJF, and STCF can have poor response time

New scheduler: RR (Round Robin)

- Alternate ready processes
- Switch after fixed-length **time-slice** (or **quantum**)

FIFO vs Round-Robin





Avg Response Time?
(0+5+10)/3 = 5

Avg Response Time?
(0+1+2)/3 = 1

In what way is RR worse?

Avg turn-around time with equal job lengths is horrible

Other reasons why RR could be better?

If run-times unknown, short jobs get chance to run, finish fast

Fair

Summary

Understand goals (metrics) and workload, then design scheduler around that

General purpose schedulers need to support processes with different goals