Scheduling

How to give every process its time?

By Matthew Tancreti
For COM S 352
Iowa State University

Scheduling

The *workload* of a system consists of many processes with often different and conflicting requirements and user expectations

Some processes just want a lot of CPU time,
while others require a quick response to I/O,
and users expect applications to be responsive

What **scheduling policies** will best meet user expectations?

"Strategy is the art of making use of time and space. I am less concerned about the latter than the former. Space we can recover, lost time never." – Napoleon Bonaparte

Napoleon Bonaparte [source]

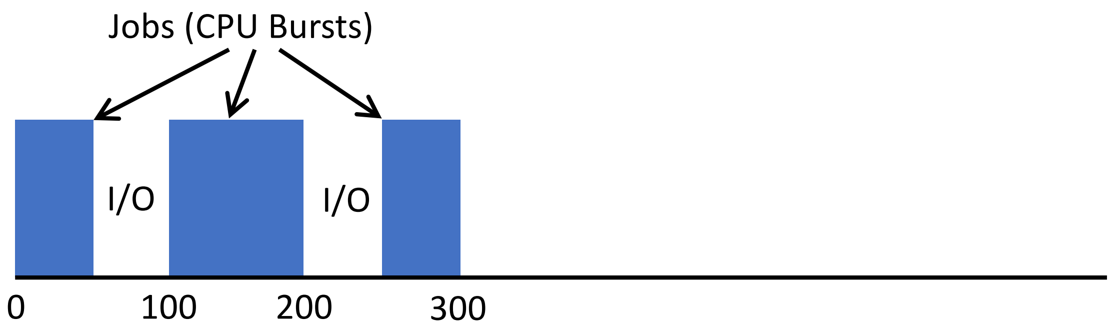
Jobs

Real processes alternate between needing CPU time and waiting for I/O

We call time when process needs CPU time a *job* (or *CPU burst*)

The time when process needs to wait is a *I/O burst*

Example: An OS that has only a single process looks like this



Jobs With Time-Sharing

A scheduler may not let a process complete its job in one run on the CPU

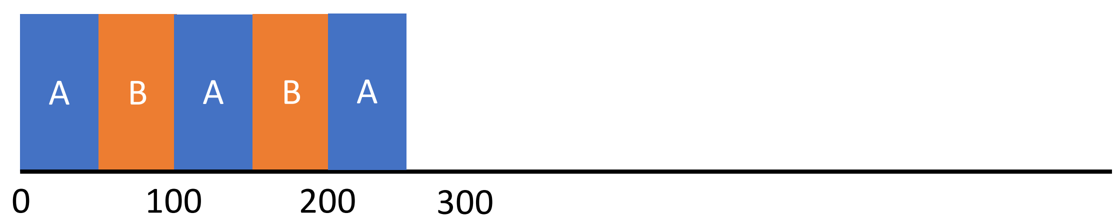
On time-sharing systems, the scheduler may *preempt* a process, swapping it with a different process before it has completed its job

Example:

Process A has a CPU burst of 150ms

Process B has a CPU burst of 100ms

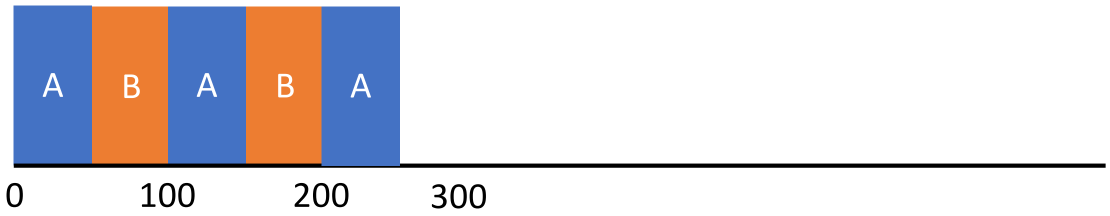
What is one way they could be scheduled?



Metrics

$T_{arrival}$ - time when job fist enters ready state
$T_{completion}$ - time when job finishes
$T_{firstrun}$ - time when job starts its first run on the CPU

Example: Suppose jobs A and B both arrive at time 0 and execute as shown



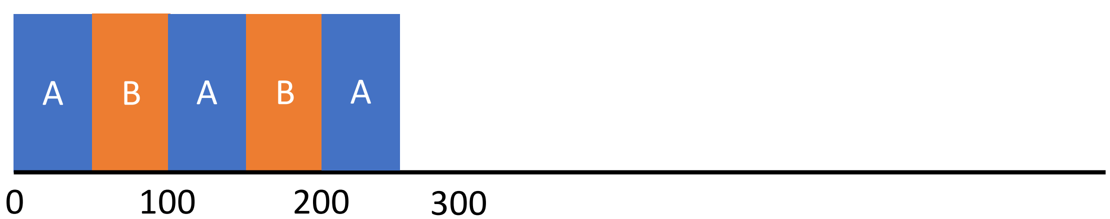
A: $T_{firstrun\_A} = 0$, $T_{completion\_A} = 250$
B: $T_{firstrun\_B} = 50$, $T_{completion\_B} = 200$

Metric: Turnaround Time

The time to complete a job

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

Example: Suppose jobs A and B both arrive at time 0 and execute as shown



A: $T_{turnaround\_A} = 250 - 0 = 250$

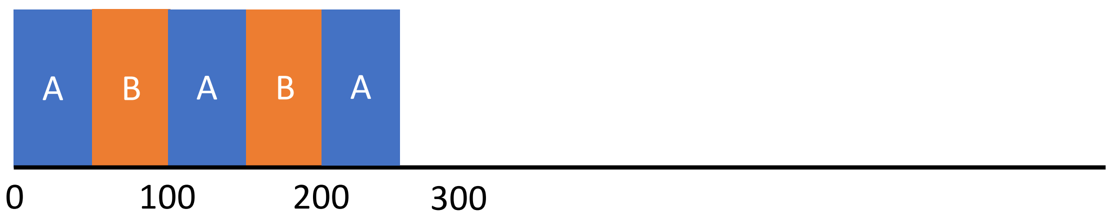
B: $T_{turnaround\_B} = 200 - 0 = 200$

Metric: Response Time

The time to first execution on CPU

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

Example: Suppose jobs A and B both arrive at time 0 and execute as shown



A: $T_{response\_A} = 0 - 0 = 0$

B: $T_{response\_B} = 50 - 0 = 50$

Why Different Metrics?

Turnaround time tells time to complete jobs, good for *CPU bound* processes, where getting enough CPU runtime is the main concern

Response time tells how long to respond to I/O, good for *I/O bound* (*interactive*) processes, which have short CPU bursts and frequent I/O

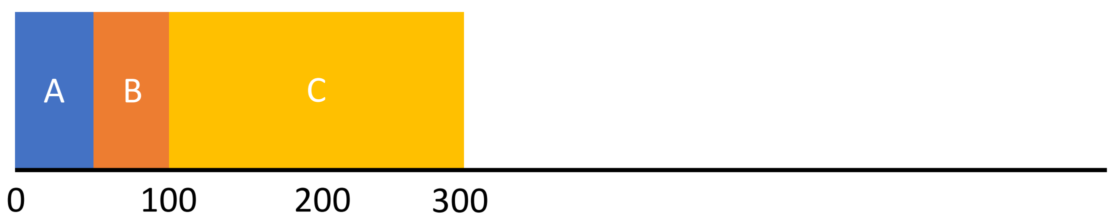
There are other metrics, later we will look at *fairness*

Policy: FIFO

Implementation: FIFO queue
Preemption: None
Advantage: Easy to implement

Example: Jobs arrive in the order A, B and C

	$T_{arrival}$	Runtime
A	0	50
B	0	50
C	0	200



$T_{average\_turnaround}$ = (50 + 100 + 300) / 3 = 150
$T_{average\_response}$ = (0 + 50 + 100) / 3 = 50

Problem with FIFO

Example: Jobs arrive in the order A, B and C

	$T_{arrival}$	Runtime
A	0	200
B	0	50
C	0	50

$T_{average\_turnaround}$ = (200 + 250 + 300) / 3 = 250

$T_{average\_response}$ = (0 + 200 + 250) / 3 = 150

Performance depends on arrival order, large upfront CPU burst can hurt turnaround and response time

Policy: Shortest Job First (SJF)

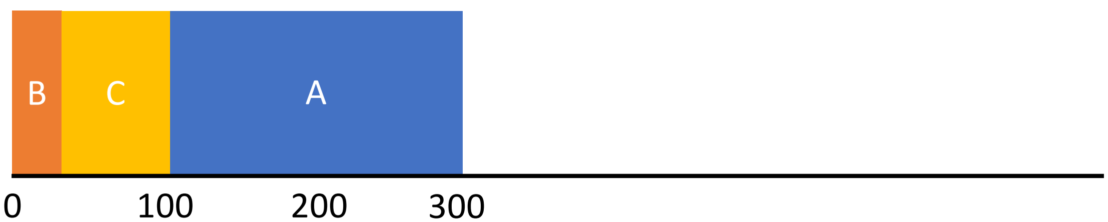
Implementation: Priority queue sorted by job length (shortest first)

Preemption: None

Advantage: Optimal average turnaround and response time when all jobs arrive at same time

Example: Jobs arrive in the order A, B and C

	$T_{arrival}$	Runtime
A	0	200
B	0	25
C	0	75



$T_{average\_turnaround} = (25 + 100 + 300) / 3 = 142$

$T_{average\_response} = (0 + 25 + 100) / 3 = 42$

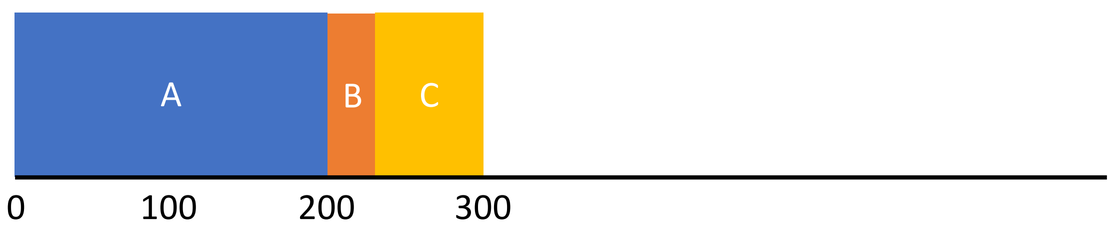
Problem with SJF

What happens if short jobs arrive after starting a long job?

Back to high turnaround and response times

Example:

	$T_{arrival}$	Runtime
A	0	200
B	5	25
C	10	75



$T_{average\_turnaround}$ = ((200 - 0) + (225 - 5) + (300 - 75)) / 3 = 215

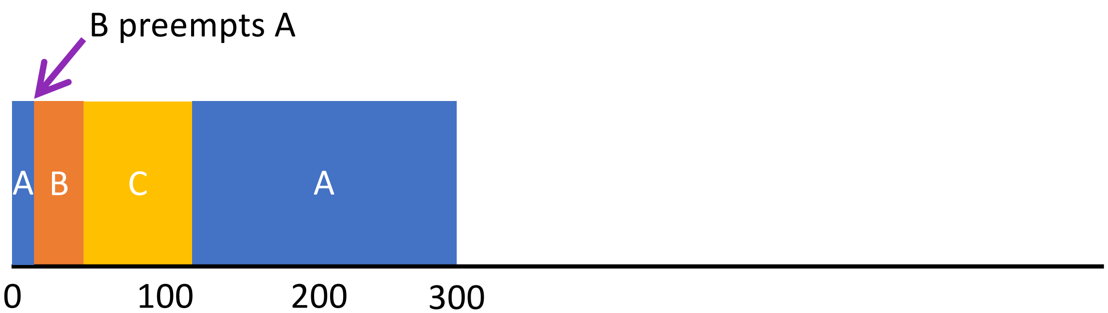
$T_{average\_response}$ = ((0 - 0) + (200 - 5) + (300 - 10)) / 3 = 162

Policy: Shortest Time-to-Completion First (STCF)

Implementation: Priority queue sorted by time to completion (shortest first)
Preemption: If new job arrives with a shorter time to completion, it preempts
Advantage: Short jobs don't need to wait for a long job to complete

Example: Jobs arrive in the order A, B and C

	$T_{arrival}$	Runtime
A	0	200
B	5	25
C	10	75



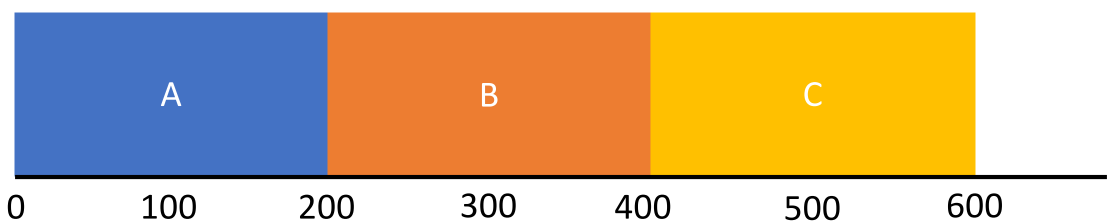
$T_{average\_turnaround}$ = ((300-0) + (30-5) + (105-10)) / 3 = 140
$T_{average\_response}$ = (0 + 0 + (30-10)) / 3 = 6.67

Problem with STCF

Preemption gives STCF better response time in some cases, but there are still cases where response time can be poor

Example: Jobs arrive in the order A, B and C

	$T_{arrival}$	Runtime
A	0	200
B	0	200
C	0	200



$T_{average\_turnaround}$ = (200 + 400 + 600) / 3 = 400

$T_{average\_response}$ = (0 + 200 + 400) / 3 = 200

Policy: Round Robin (RR)

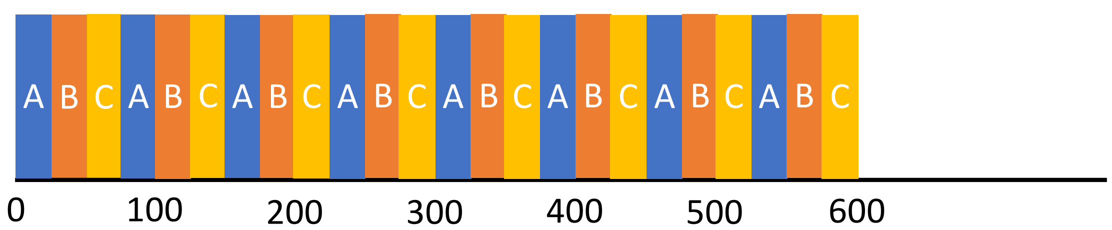
Implementation: FIFO queue

Preemption: Job on CPU gets *time-slice*, preempt when time expired

Advantage: Low response time

Example: Jobs arrive in the order A, B and C; time-slice is 25

	$T_{arrival}$	Runtime
A	0	200
B	0	200
C	0	200



$T_{average\_turnaround}$ = (550 + 575 + 600) / 3 = 575

$T_{average\_response}$ = (0 + 25 + 50) / 3 = 25

Pro and Con of RR

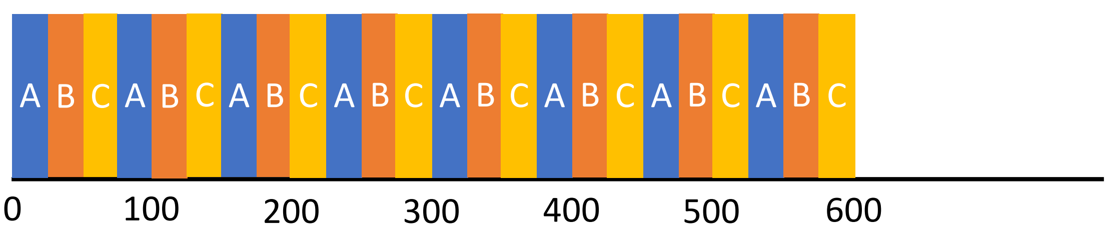
Pro: RR has great response time with gaurenteed upper bound

worst_case_response_time = time_slice x (num_jobs – 1)

Con: Bad average turnaround time, frequent context switches reduce CPU efficiency

Example: Jobs arrive in the order A, B and C; time-slice is 25

	$T_{arrival}$	Runtime
A	0	200
B	0	200
C	0	200



$T_{average\_turnaround}$ = (550 + 575 + 600) / 3 = 575

$T_{worst\_case\_response}$ = 25 x (3 – 1) = 50

Another Problem with SJF and STCF: Oracle

Scheduler doesn’t know how long it will take for a job to complete

SJF and STCF require an *oracle*, the ability to see into the future!

How can we get the benefits of STCF in a real system?