Scheduling: Introduction

Previous chapter: How to switch b/w processes → context switch mechanism
Current chapter: When to switch b/w processes → scheduling policies.

Early times (1940–1950s): A queue of jobs (on punch cards, tapes) would arrive at the machine room.
- computers ran one job at a time.
- the operator decided the job order.
- CPU idle during I/O

Even though the term CPU scheduling wasn't used yet, the act of choosing which job runs next is fundamentally the same decision modern OS schedulers make.
- only now, it's automated, faster, and more complex.

What are the factors to keep in mind, while developing a scheduling policy.

- Assumptions on the work load (like, duration, arrival time, etc.)
- What metrics to optimize? (maximize utilization) minimize response time etc.)

What are the assumptions about the processes?

1. Each job runs for the same amount of time.
2. All jobs arrive at the same time.
3. Once started, each job runs to its completion.
4. All jobs only use the CPU (i.e., no I/O)
5. The run-time of each job is known

The above assumptions are unrealistic and we will get rid of each of them one by one.

Which are the metrics that we are interested in optimizing?

* Average turnaround time
* Average response time

The turnaround time of a job is defined as the time at which the job completes minus the time at which the job arrived in the system.

$$\text{i.e., } T_{turnaround} = T_{completion} - T_{arrival}$$

By Assumption 2, we have $T_{arrival} = 0$

$\Rightarrow$ under these assumptions, $T_{turnaround} = T_{completion}$

Scheduling policies

(1) First In, First Out (FIFO)

eg:- This is a typical serving policy in the grocery stores.

Imagine three jobs, say A, B, and C arrive in the system around the same time. (ie., $T_{arrival} = 0$), and each job runs for 10 seconds.
What would be the average turnaround time for these jobs in FIFO?

$T_{comp}(A) = 10$

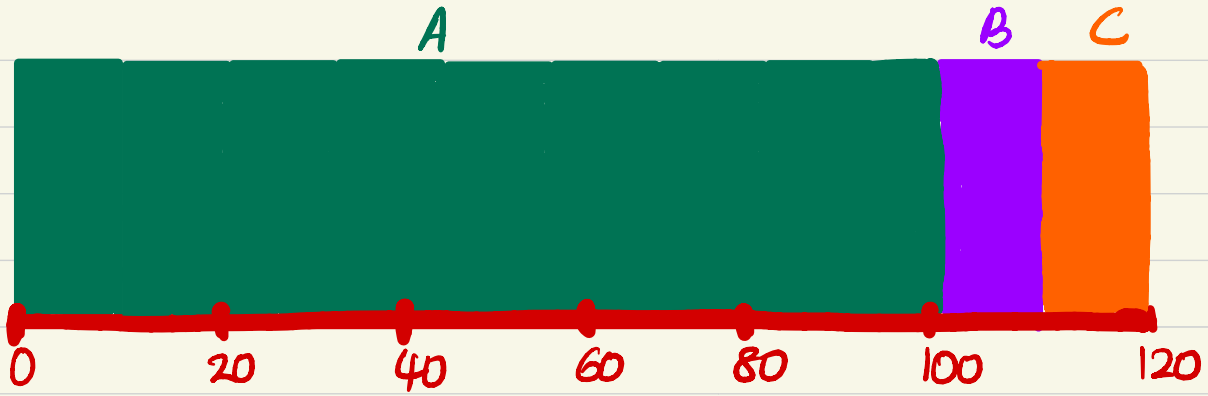
$T_{comp}(B) = 20$

$T_{comp}(C) = 30$

$\Rightarrow$ average turnaround time $= \frac{10+20+30}{3} = \underline{\underline{20}}$

Relax assumption 1 and use the same FIFO strategy.
↳ i.e., each job need not have the same running time.

eg:- 3 jobs A, B, and C where A runs for 100 seconds while B and C runs for 10 each.



$T_{comp}(A) = 100$, $T_{comp}(B) = 110$, $T_{comp}(C) = 120$

⇒ average turnaround = 110, which is pretty bad!

The above problem is generally referred to as convoy effect, where a number of relatively short potential consumers of a resource get queued behind a heavyweight resource consumer.

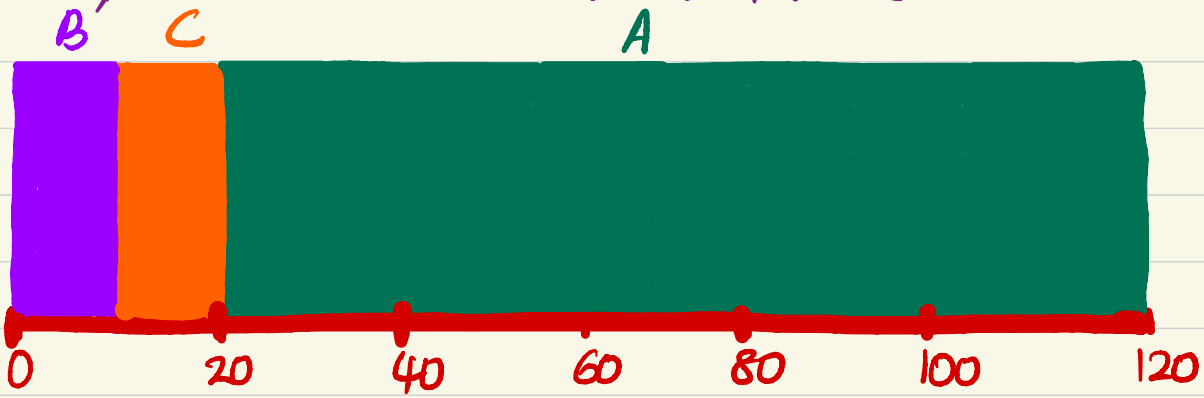
How to overcome this problem?

Grocery stores adapts the idea of "ten-items-or-less" to deal convoy effect.

Our next scheduling strategy is inspired by this idea.

(2) Shortest Job First (SJF)

Recall the previous example, 3 jobs A, B, and C where A runs for 100 seconds, while B and C runs for 10 each. SJF works as follows.



$T_{comp}(B) = 10$ $T_{comp}(B) = 20$

$T_{comp}(A) = 120$

$\Rightarrow$ average turnaround

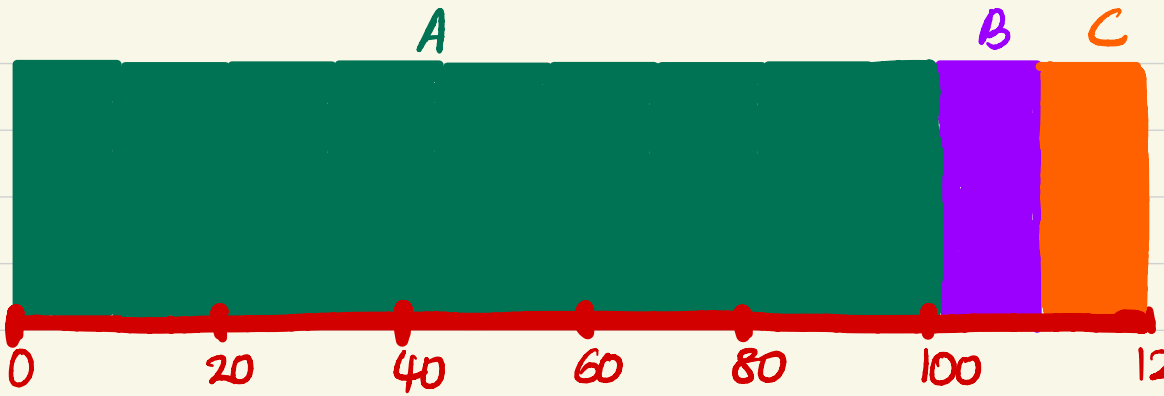
$= \frac{10+20+120}{3} = 50,$

Which is a good improvement!

Now, what if we need to relax Assumption 2?

i.e., every job need not start at the same time as well!

Let us modify the previous example. This time let A arrives at $t=0$ and needs to run for 100 seconds, where as B and C arrive at $t=10$ and each need to run for 10 seconds. Then SJF works as follows:



Turnaround (A) = 100 − 0 = 100

Turnaround (B) = 110 − 10 = 100

Turnaround (C) = 120 − 10 = 110

$\Rightarrow$ average turnaround $= \frac{310}{3} = 103.3$ seconds,

which is also pretty bad!

So what to do now? $\longrightarrow$ Relax Assumption 3

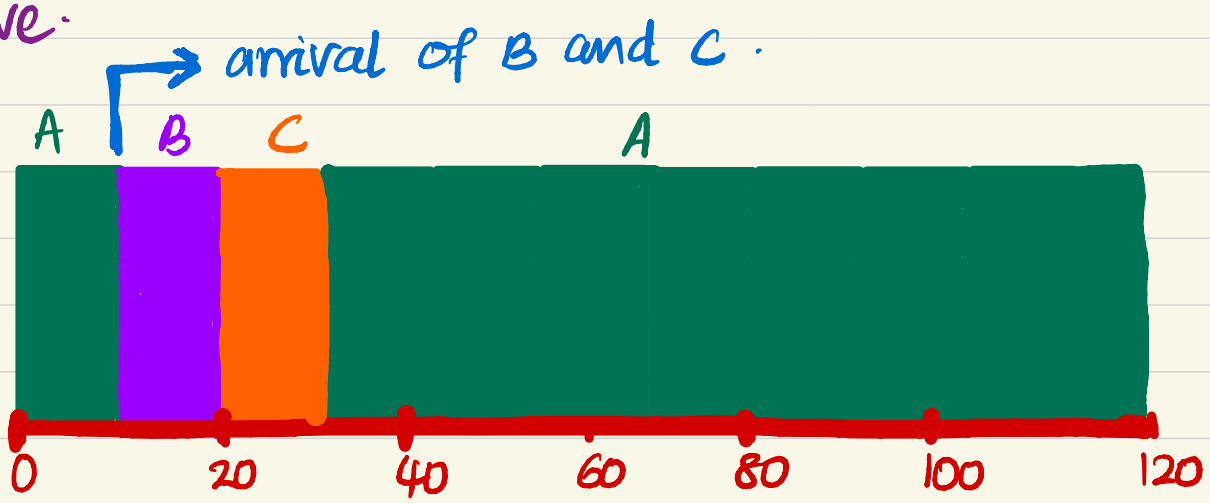
i.e., jobs need not run to completion.

In particular, using timer interrupt and context switching mechanisms, add preemption to SJF. The new version is called

(3) Preemptive Shortest Job First (PSJF)
or Shortest Time to Completion First (STCF).

i.e., When a new job enters the system:
- determine which of the remaining jobs (including the new job) has the least time left, and schedules that one.

now in the above example, the scheduler can stop A when B and C arrive.



Turnaround (A) = 120 − 0 = 120

Turnaround (B) = 20 − 10 = 10

Turnaround (C) = 30 − 10 = 20

⇒ average turnaround

$= \frac{120+10+20}{3} = \frac{150}{3} = 50$ seconds,

which is better than SJF.

* STCF is a great policy for turnaround time (under the necessary assumptions)

A new metric : response time

In time sharing systems, users would sit at a terminal and demand interactive performance from the system as well.

Note that STCF and related previous policies are not particularly good for response time

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

(i.e., time from when the job arrives in a system to the first time it is scheduled)

Even in STCF, if three jobs arrive at the same time, the third job has to wait for the previous jobs to run in their entirety before being scheduled just once.

$\Rightarrow$ bad response time and interactivity.

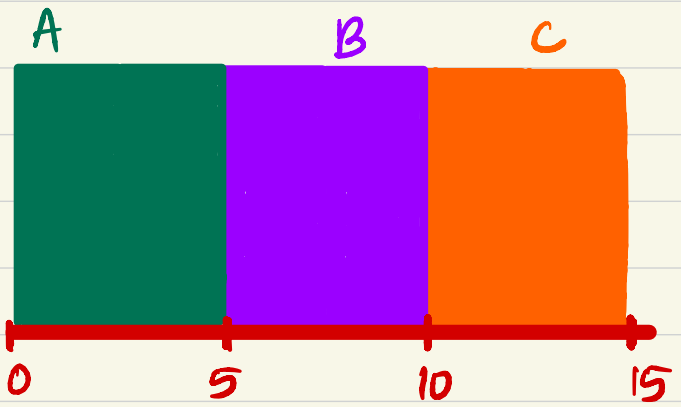
Question: How can we build a scheduler that is sensitive to response time?

(4) Round Robin (RR) (also known as time slicing scheduling)

* Run a job for a time slice and then switch to the next job in the run queue until the jobs are finished.
* time slice is sometimes called a scheduling quantum.
* It repeatedly does so until the jobs are finished.
* the length of the time slice must be a multiple of the timer-interrupt period.

For example, suppose the jobs A, B, and C arrive at the same time in the system, and each of them wish to run for 5 seconds.

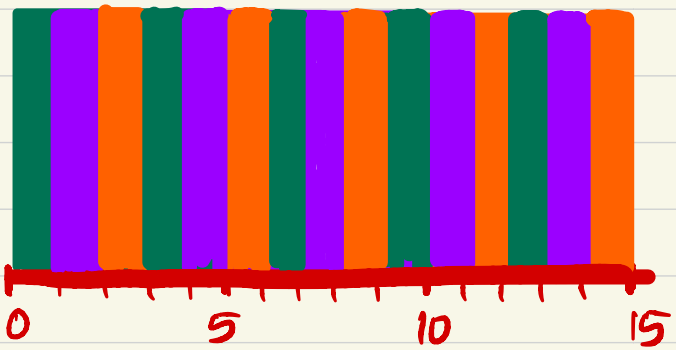
What does an STF scheduler do?



⇒ response time (A) = 0
response time (B) = 5
response time (C) = 10

⇒ average response time = $\frac{0+5+10}{3}$ = 5 seconds

In contrast to above RR with a time slice of 1 second would cycle through jobs quickly.



response time (A) = 0
response time (B) = 1
response time (C) = 2
⇒ average response time = 1, which is great!

What is the biggest challenge in RR?

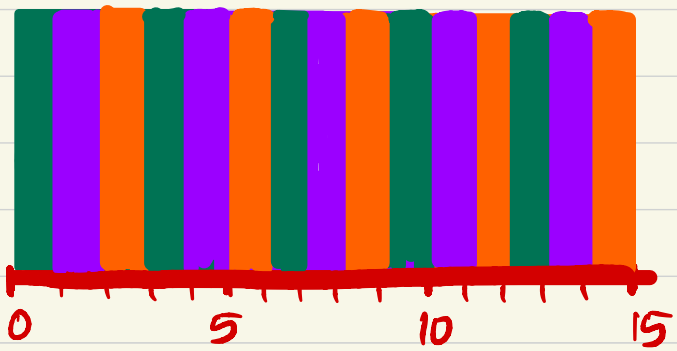
The length of the time slice is critical!

Shorter time slice ⇒ better response time
but the cost of context switching will dominate overall performance

Longer time slice ⇒ amortize the cost of switching
but worse response time.

Deciding on the length of the time slice presents a trade-off to a system designer. In fact, RR with a reasonable time slice, is an excellent scheduler if response time is our only metric.

But, what about the metric turnaround time



$T_{arrival}(A) = T_{arrival}(B) = T_{arrival}(C) = 0$

⇒ $Turnaround(A) = T_{comp}(A) = 13$
$Turnaround(B) = T_{comp}(B) = 14$
$Turnaround(C) = T_{comp}(C) = 15$

⇒ average turnaround = 14, which is pretty awful!

RR is fair, but performs poorly on metrics such as turnaround time.

To summarize, under the current assumptions (4 and 5)

STCF → optimizes turnaround time, but bad response time

RR → optimizes response time, but bad turnaround time.

Relaxing Assumption 4

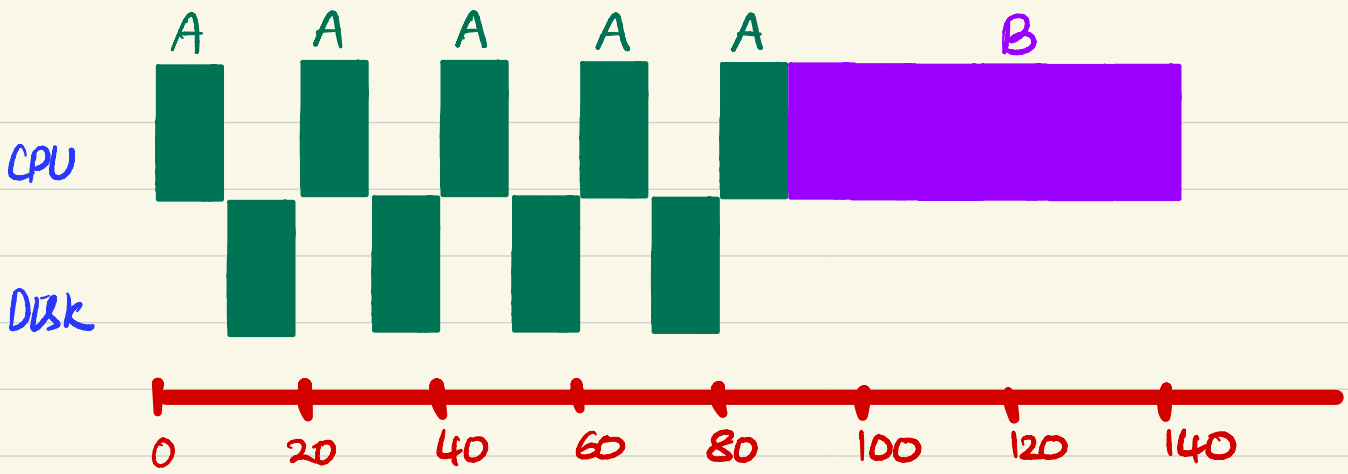
↳ i.e., jobs can also perform I/O.

Incorporating I/O

eg:- Let A and B be two processes such that A and B need 50 ms of CPU time each.

- A runs for 10 ms and then issues an I/O request
- I/O each takes 10 ms
- B simply uses the CPU for 50 ms and performs no I/O
- the scheduler runs A first, then B after.

We intend to build STCF scheduler.



A better approach is as follows:

- treat each 10 ms sub-job of A as an independent job.
- Thus, at the beginning the choice is b/w 10 ms A or a 50 ms B.
- STCF clearly choose 10 ms A

when A initiates I/o request, job A is blocked waiting for I/o completion

- Thus scheduler schedules B.

When the I/o completes, an interrupt is raised, the OS moves A from blocked to ready and again make a choice.