Scheduling

Some Assumptions:

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 completion.
4. All jobs only use the CPU. (They perform no I/O)
5. The runtime of each job is unknown. Most unrealistic assumption ae

Scheduling Metrics:

A metric is just something that we use to measure something.

Turnaround Time: A performance metric formulated as

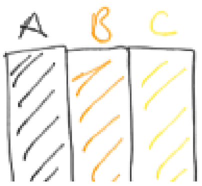
$$T_{turnaround} = T_{completion} - T_{arrival}.$$

Another metric is fairness.

Performance and fairness are often at odds in scheduling.

First in First out (FIFO)

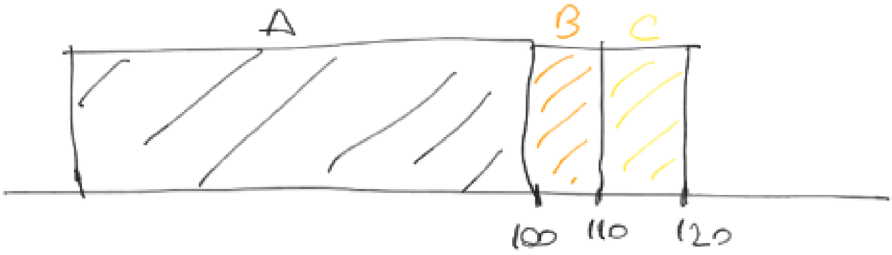
As its name suggest, who arrives first takes the CPU.



A arrives just before B
B arrives just before C.

$$\text{Average Turnaround Time} = \frac{(10-0)+(20-0)+(30-0)}{3} = 20$$

~~Assumption 1~~

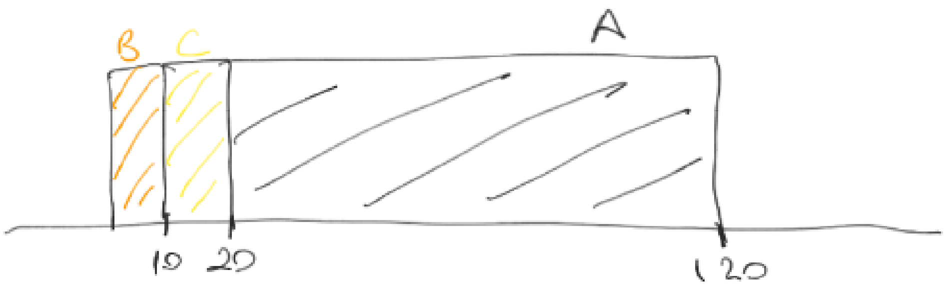


$$\frac{(100-0)+(110-0)+(120-0)}{3} = 110$$

This problem is generally referred as <u>convoy effect</u>, where a number of relatively-short potential costumers of a resource get queued behind a heavyweight resource consumer.

Shortest Job First: (SJF)

It runs the shortest job first, then the next shortest, and so on...

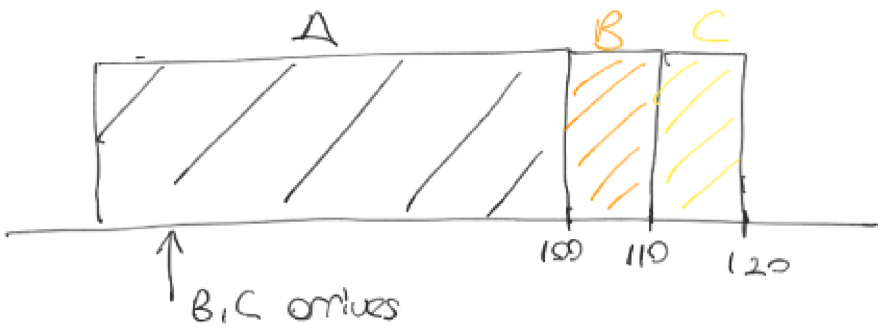


$(10-0)+(20-0)+(120-0)$

$$\frac{(10-0)+(20-0)+(120-0)}{3} = 50$$

Under 2.~5. assumptions SJF is the optimal scheduling algorithm.

Assumption 2

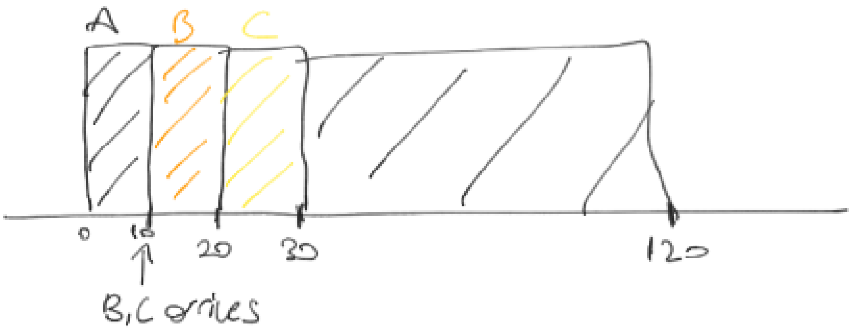


$$\frac{(100-0)+(110-10)+(120-10)}{3} = 103.33$$

Shortest Time-to Completion First (STCF)

Assumption 3

Any time a new job enters the system, the STCF scheduler determines which of the remaining jobs has the least time left, and schedules that one.



$$\frac{(120-0)+(20-10)+(30-10)}{3} = 50$$

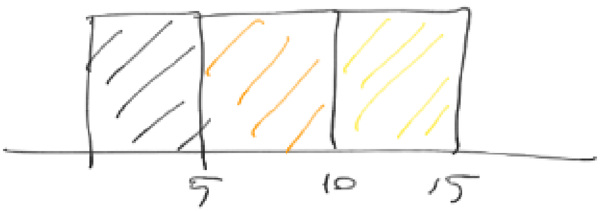
New Metric : Response Time :

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

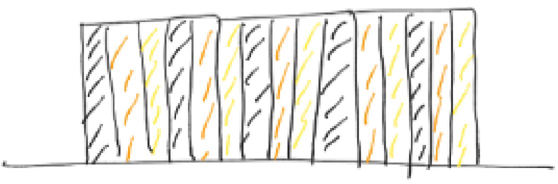
STCF is good for turnaround time but bad for response time.

Round - Robin (RR)

Instead of running jobs to completion, RR runs a job for a time slice and then switches to next job on the queue.



$$\underbrace{\frac{0+(5-0)+(10-0)}{3} = 5}_{\text{response T.}}$$



$$\underbrace{\frac{0+1+2}{3} = 1}_{\text{response T.}}$$

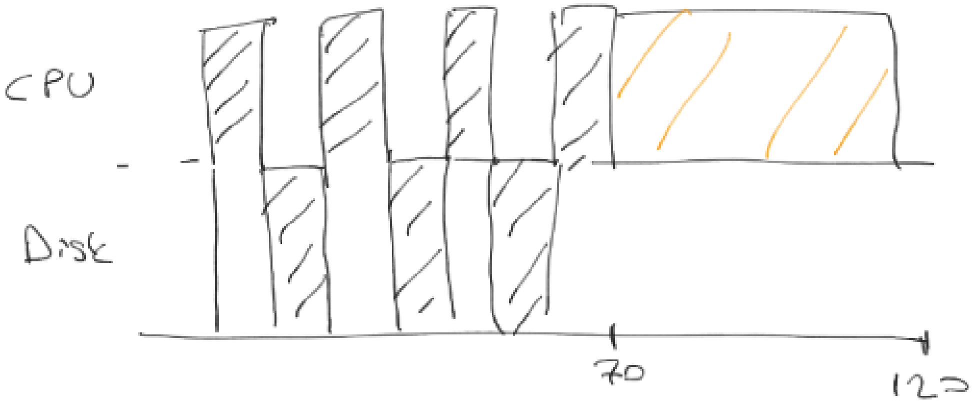
Making the time slice too short is problemetic, cost of context switching will dominate overall performance.

Round-Robin is good for response time, but bad for turnaround time.

Incorporating I/O

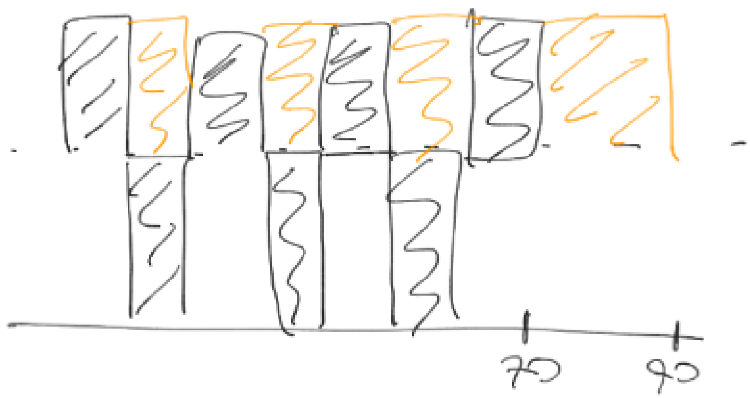
Overlap enables higher utilization: When performing disk I/O starting the operation and then switching to other work is a good idea.

~~Assumption 4~~



$$\frac{70+120}{2} = 105$$

BAD IDEA



$$\frac{90+70}{2} = 80$$

GOOD IDEA.

Multi-level Feedback Queue (MLFQ):

Tip: Learn from past to predict future. Such approaches work when jobs have phases of behaviour and are thus predictable.

MLFQ has a number of distinct queues, each assigned a different priority level.

Rather than giving a fixed priority to each job, MLFQ varies the priority of a job based on its observed behaviour.

It doesn't know whether a job will be a short job or a long running job; it first assumes it might be a short job, thus giving the job high priority. In this manner MLFQ approximates SJF.

Starvation: If there are too many interactive jobs in the system, they will combine to consume all CPU time and thus long running jobs will never receive any CPU. (They starve)

Game the Scheduler: "Gaming the scheduler" generally refers to the idea of doing something sneaky to trick the scheduler into giving you more than your fare share of the resources (i.e by running for 99% of a order of the resources. (i.e. by running [illegible] in a time slice before relinquishing CPU)

A program may change its behaviour over time.

The Priority Boost ↑. So

1) If Priority(A) > Priority(B), A runs
2) If Priority(A) = Priority(B), A & B in RR
3) When a job enters the system, it is placed at the highes priority.
4) Once a job uses up its time allotment at a given level, its priority is reduced.
5) After some time period S, move all the jobs in the system to the topmost queue.

↑ higher priority : shorter time slices. (interactive works)

↓ lower priority : longer time slices

Default values is like

- 60 queues
- lowest priority : 300 ms, highest priority : 20 ms

MLFQ can deliver excellent overall performance for short running interactive jobs, and is fair and makes progress for long running CPU intensive workloads.

Proportional Share

Tickets are used to represent the share of a resource that a process should receive.

Use of Randomness:

- Random is lightweight, requiring little state to track alternatives.
- Random can be quite fast.

A → 75 tickets (0 ~ 75)

B → 25 tickets (76 ~ 99)

63	85	70	39	76	17	29
A	B	A	A	B	A	A

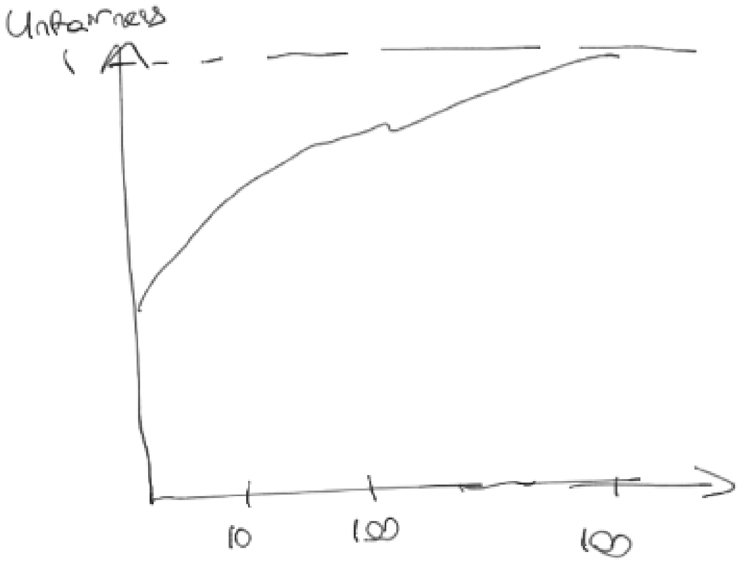
Ticket Mechanisms

Ticket currency allows a user with a set of tickets to allocate tickets among their own jobs in whatever currency they would like, then automatically converts currency to global value.

With ticket transfer a process can temporarily hand of its tickets to another process.

Ticket inflation can be applied in an environment where a group of processes trust one another.

* Very easy to implementation.



$$\text{Unfairness Metric} = \frac{T_{\text{first job completion}}}{T_{\text{second job completion}}}$$

"How to assign tickets?" is a hard question.

Strick Scheduling

A deterministic fair-share scheduler.

Each job in the system has a stride, which is inverse in proportion to the number of tickets it has.

10000

Tickets	Stride
A → 100	A → 100
B → 50	B → 200
C → 250	C → 40

A	B	C	Who Runs
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
			⋮

- Lottery scheduling has one nice property that stride scheduling does not: no global state.
- Imagine a new job enters the system, what should its value be? 0 → monopolizes the CPU.

Last modified: Sep 17, 2019