【OS】Day12

【Ch7】Scheduling

7.1 Workload Assumptions

We will make the following assumptions about the processes, sometimes called jobs, that are running in the system:

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(i.e. they perform no I/O)
5. The run-time of each job is known

7.2 Scheduling Metrics

We also need a scheduling metric to compare different scheduling policies.

A metric is just something that we use to measure something, and there are a number of different metrics that make sense in scheduling.

For now, let us also simplify by having a single metric: turnaround 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.

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

Turnaround time is a performance metric, which will be our primary focus this chapter.

Another metric of interest is fairness.

Some systems optimise performance on the cost of fairness by reducing the runtime of some processes.

7.3 First In, First Out(FIFO)

The most basic algorithm we can implement is known as First In, First Out(FIFO) scheduling.

Let's do a quick example. Imagine three jobs, A, B, and C, arrive in the system at roughly the same time($T_{arrival}$ = 0). Because FIFO has to put some job first, let's assume A arrives a hair earlier than B, which arrives a hair before C. Assume also that each job takes roughly ten seconds to complete.

What will be the average turnaround time be for these jobs?

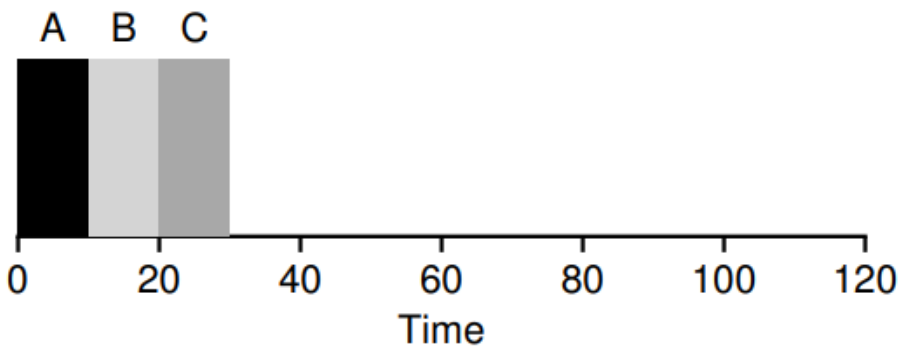

Figure 7.1: **FIFO Simple Example**

From the figure above, we can see that A finishes at 10, B at 20, and C at 30.

Thus, the average turnaround time for three jobs is simply $\frac{10+20+30}{3} = 20$

Now let's relax one of our assumptions. In particular, let's relax assumption 1, and thus no longer assume that each job runs for the same amount of time.

How does FIFO perform now? What kind of workload could you construct to make FIFO perform poorly?

If A runs for 100 seconds while B and C runs for 10 seconds each

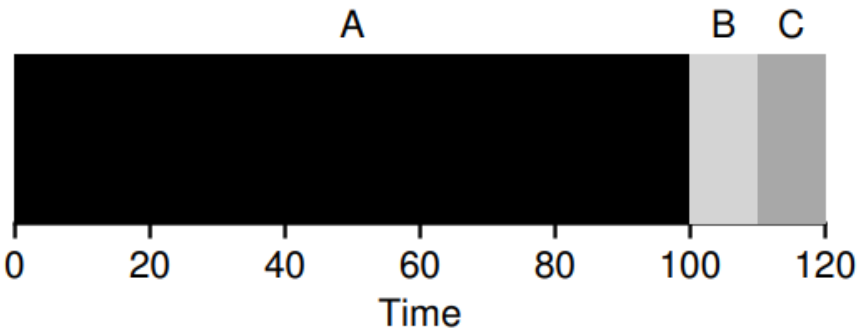

Figure 7.2: **Why FIFO Is Not That Great**

Thus, the average turnaround time for the system is high: a painful 110 seconds($\frac{100+110+120}{3} = 110$).

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

7.4 Shortest Job First(SJF)

A new scheduling discipline known as Shortest Job First(SJF) is introduced to solve the convoy effect issue.

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

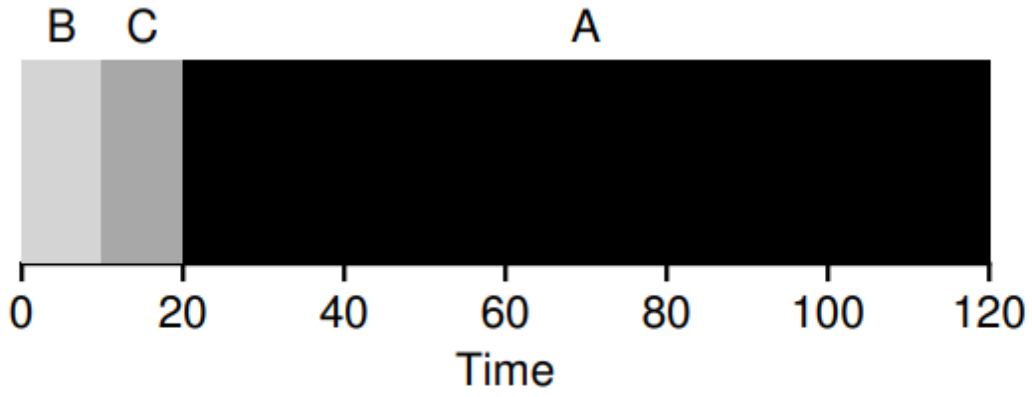

Figure 7.3: **SJF Simple Example**

Implementing the SJF, the average turnaround time for A, B, and C is now 50($\frac{10+20+120}{3} = 50$)

Now, let's relax assumption 2, and now assume that jobs can arrive at any time instead of all at once. *What problems does this lead to?*

This time, assume A arrives at t = 0 and needs to run for 100 seconds, whereas B and C arrive at t = 10 and each need to run for 10 seconds. With the pure SJF, we will get the following graph:

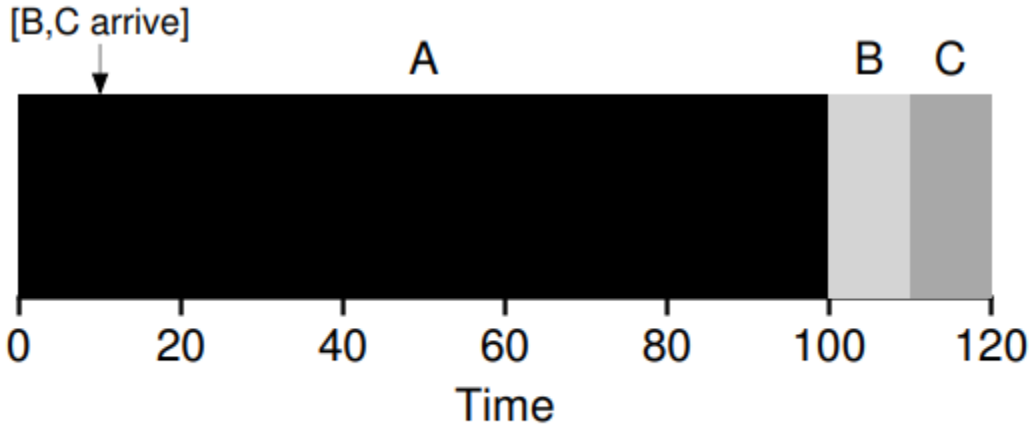

Figure 7.4: **SJF With Late Arrivals From B and C**

Even though B and C arrived shortly after A, they still are forced to wait until A has completed, and thus suffer the same convoy problem. Average turnaround time for three jobs is 103.33 seconds.($\frac{100+(110-10)+(120-10)}{3} = 103.33$).

7.5 Shortest Time-to-Completion First(STCF)

To address this concern, we need to relax assumption 3(that jobs must run to completion).

The scheduler can now do something else when B and C arrive: it can preempt job A and decide to run another job, perhaps continuing A later. SJF by our definition is a non-preemptive scheduler, and thus suffers from the problems described above.

Fortunately, there is a scheduler which does exactly that: add preemption to SJF, known as the Shortest Time-to-Completiong First(STCF).

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

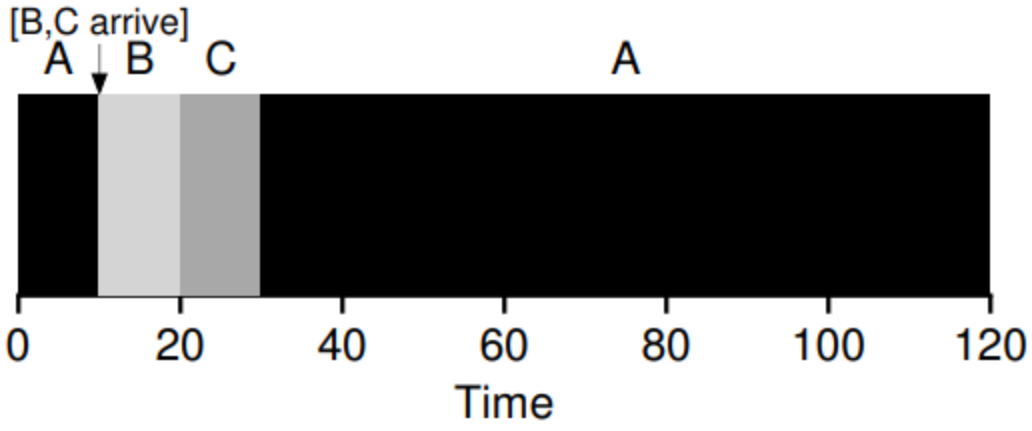

Figure 7.5: **STCF Simple Example**

In this case, the result is much-improved: 50 seconds($\frac{(120-0) + (20-10) + (30-10)}{3} = 50$).