CS 111: Operating System Principles

Lecture 7

Basic Scheduling

1.0.0

Jon Eyolfson

April 13, 2021

There are Preemptible and Non-preemptible Resources

A preemptible resource can be taken away and used for something else

e.g. a CPU

The resource is shared through scheduling

A non-preemptible resource can not be taken away without acknowledgment

e.g. disk space

The resource is shared through allocations and deallocations

Note: Parallel and distributed systems may allow you to allocate a CPU

A Dispatcher and Scheduler Work Together

A dispatcher is a low-level mechanism

Responsible for context switching

A scheduler is a high-level policy

Responsible for deciding which processes to run

The Scheduler Runs Whenever a Process Changes State

First let’s consider non-preemptable processes

- Once the process starts, it runs until completion

In this case, the scheduler will only make a decision when the process terminates

Preemptive allows the operating system to run the scheduler at will

- Check `uname -v`, your kernel should tell you it’s preemptable

Metrics

Minimize waiting time and response time

Don’t have a process waiting too long (or too long to start)

Maximize CPU utilization

Don’t have the CPU idle

Maximize throughput

Complete as many processes as possible

Fairness

Try to give each process the same percentage of the CPU

First Come First Served (FCFS)

The most basic form of scheduling

The first process that arrives gets the CPU

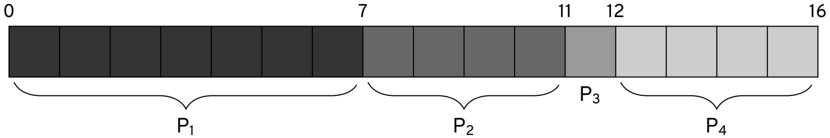
Processes are stored in a FIFO queue in arrival order

A Gantt Chart Illustrates the Schedule

Consider the following processes:

Process	Arrival Time	Burst Time
P_1	0	7
P_2	0	4
P_3	0	1
P_4	0	4

Assume, $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4$. For FCFS, our schedule is:



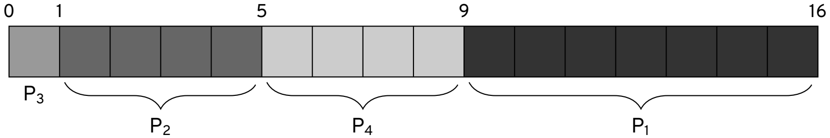
What is the average waiting time?

What Happens to Our Waiting Time with a Different Arrival Order

Consider the same processes:

Process	Arrival Time	Burst Time
P_1	0	7
P_2	0	4
P_3	0	1
P_4	0	4

Assume, $P_3 \rightarrow P_2 \rightarrow P_4 \rightarrow P_1$. For FCFS, our schedule is:



What is the average waiting time now?

Shortest Job First (SJF)

A slight tweak to FCFS, we always schedule the job with the shortest burst time first

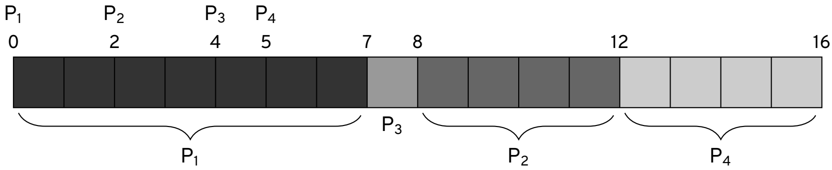
We’re still assuming no preemption

SFJ Minimizes the Average Wait Time over FCFS

Consider the same processes with different arrival times:

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

For SJF, our schedule is (arrival on top):



Average waiting time: $\frac{0+6+3+7}{4} = 4$

SFJ is Not Practical

It is provably optimal at minimizing average wait time (if no preemption)

You will not know the burst times of each process

- You could use the past to predict future executions

You may starve long jobs (they may never execute)

Shortest Remaining Time First (SRTF)

Changing SJF to run with preemptions requires another tweak

We'll assume that our minimum execution time is one unit

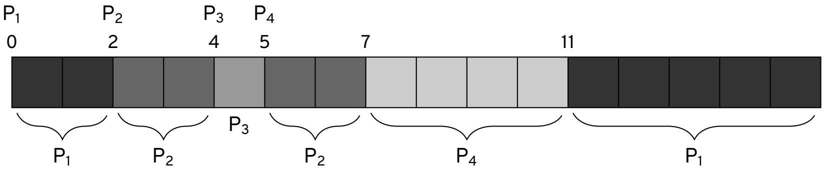
Similar to SJF, this optimizes the average waiting time

SRTF Reduces the Average Wait Time Compared to SFJ

Consider the same processes and arrival times as SFJ:

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

For SRTF, our schedule is (arrival on top):



Average waiting time: $\frac{9+1+0+2}{4} = 3$

Round-Robin (RR)

So far we haven’t handled fairness (it’s a trade off with other metrics)

The operating system divides execution into time slices (or quanta)

- An individual time slice is called a quantum

Maintain a FIFO queue of processes similar to FCFS

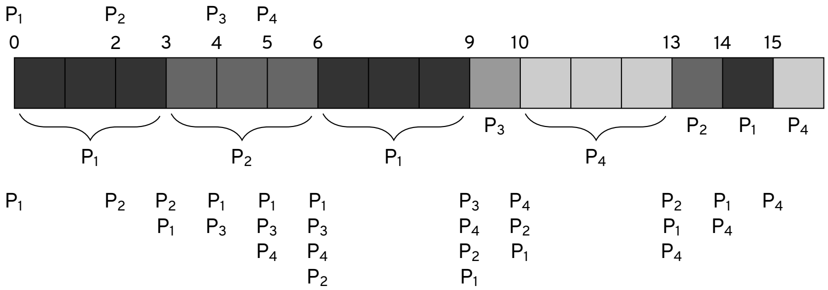
- Preempt if still running at end of quantum and re-add to queue

What are practical considerations for determining quantum length?

RR with a Quantum Length of 3 Units

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

For RR, our schedule is (arrival on top, queue on bottom):



Metrics for RR (3 Unit Quantum Length)

Number of context switches: 7

Average waiting time: $\frac{8+8+5+7}{4} = 7$

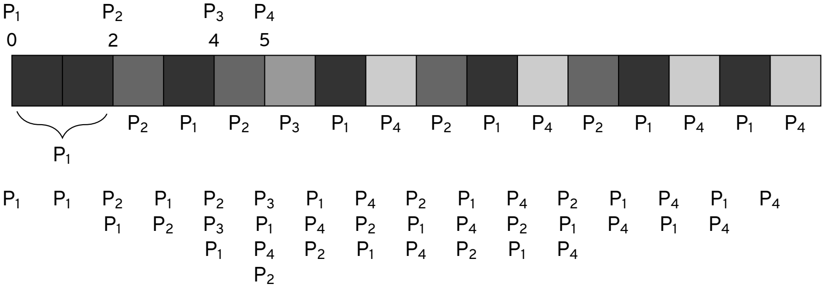
Average response time: $\frac{0+1+5+5}{4} = 2.75$

Note: on ties (a new process arrives while one is preempted), favor the new one

RR with a Quantum Length of 1 Units

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

For RR, our schedule is (arrival on top, queue on bottom):



Metrics for RR (1 Unit Quantum Length)

Number of context switches: 14

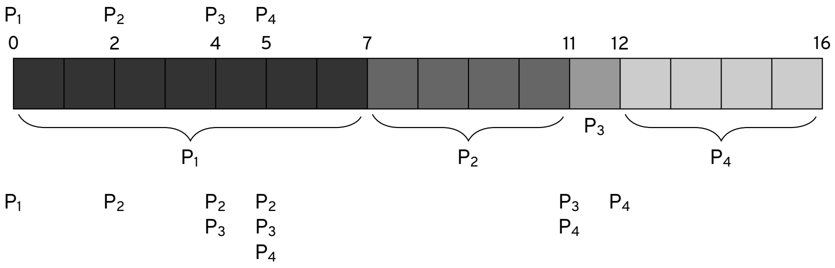
Average waiting time: $\frac{8+6+1+7}{4} = 5.5$

Average response time: $\frac{0+0+1+2}{4} = 0.75$

RR with a Quantum Length of 10 Units

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

For RR, our schedule is (arrival on top, queue on bottom):



Metrics for RR (10 Unit Quantum Length)

Number of context switches: 3

Average waiting time: $\frac{0+5+7+7}{4} = 4.75$

Average response time: $\frac{0+5+7+7}{4} = 4.75$

Note: in this case it's the same as FCFS without preemptions

RR Performance Depends on Quantum Length and Job Length

RR has low response good interactivity

- Fair allocation of CPU
- Low average waiting time (when job lengths vary)

The performance depends on the quantum length

- Too high and it becomes FCFS
- Too low and there's too many context switches (overhead)

RR has poor average waiting time when jobs have similar lengths

Scheduling Involves Trade-Offs

We looked at few different algorithms:

- First Come First Served (FCFS) is the most basic scheduling algorithm
- Shortest Job First (SJF) is a tweak that reduces waiting time
- Shortest Remaining Time First (SRTF) uses SJF ideas with preemptions
- SRTF optimizes lowest waiting time (or turnaround time)
- Round-robin (RR) optimizes fairness and response time