OS-2 (CS3523) Theory Assignment 1

CO22BTECH11006
Om Dave

1 Amdahl's Law

Speed up gain is given by,

$$S = \frac{1}{(1-P) + \frac{P}{N}}$$

Where:

- S is the theoretical speedup of the execution of the whole task,
- P is the proportion of the execution time that the improvement affects (parallelizable part),
- N is the number of processing units (cores).

1.1 40 percent parallel

1. **With eight processing cores** (N = 8, P = 0.40):

$$S = \frac{1}{(1-0.40) + \frac{0.40}{8}} = 1.54$$

2. **With sixteen processing cores** (N = 16, P = 0.40):

$$S = \frac{1}{(1 - 0.40) + \frac{0.40}{16}} = 1.6$$

1.2 67 percent parallel

1. **With two processing cores** (N = 2, P = 0.67):

$$S = \frac{1}{(1 - 0.67) + \frac{0.67}{2}} = 1.50$$

2. **With four processing cores** (N = 4, P = 0.67):

$$S = \frac{1}{(1 - 0.67) + \frac{0.67}{4}} = 2.01$$

1.3 90 percent parallel

1. **With four processing cores** (N = 4, P = 0.90):

$$S = \frac{1}{(1 - 0.90) + \frac{0.90}{4}} = 3.08$$

2. **With eight processing cores** (N = 8, P = 0.90):

$$S = \frac{1}{(1 - 0.90) + \frac{0.90}{8}} = 4.71$$

2 Multi-threading thread allocation

2.1 How many threads will you create to perform the input and output? Explain.

- Create one thread for both input and output operations since they are sequential tasks. Parallelizing these operations is unlikely to significantly improve performance, and managing multiple threads for I/O could introduce unnecessary overhead.

2.2 How many threads will you create for the CPU-intensive portion of the application? Explain

- Create four threads for the CPU-intensive portion to make the most of the four available processors. This approach ensures efficient parallel execution and maximizes CPU utilization during the CPU-bound phase of the application. Using fewer than four threads would underutilize the processors, while using more than four threads may introduce unnecessary context switching and overhead, potentially reducing overall performance due to increased contention for CPU resources.

3 Non-preemptive Scheduling

Process	Burst time	Priority
P1	5	4
P2	3	1
P3	1	2
P4	7	2
P5	4	3

Table 1: Processes with burst time and priority

3.1 FCFS (First come First served)

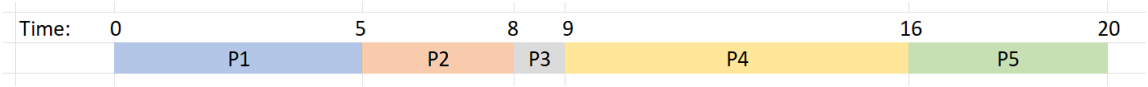

Figure 1: FCFS Scheduling

- Turnaround time: Completion time - arrival time
  - Turnaround time for P1 : 5 - 0 = 5
  - Turnaround time for P2 : 8 - 0 = 8
  - Turnaround time for P3 : 9 - 0 = 9
  - Turnaround time for P4 : 16 - 0 = 16
  - Turnaround time for P5 : 20 - 0 = 20

- Waiting time: Turnaround time - burst time
  - Waiting time for P1 : 5 - 5 = 0
  - Waiting time for P2 : 8 - 3 = 5
  - Waiting time for P3 : 9 - 1 = 8
  - Waiting time for P4 : 16 - 7 = 9
  - Waiting time for P5 : 20 - 4 = 16

  Avg waiting time = (0+5+8+9+16)/5 = 7.6 ms

3.2 SJF (Shortest job first)

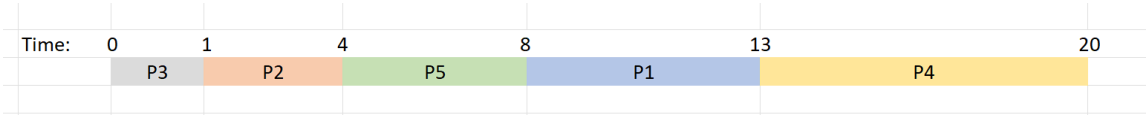

Figure 2: SJF Scheduling

- Turnaround time: Completion time - arrival time
  - Turnaround time for P1 : 13 - 0 = 13
  - Turnaround time for P2 : 4 - 0 = 4
  - Turnaround time for P3 : 1 - 0 = 1
  - Turnaround time for P4 : 20 - 0 = 20
  - Turnaround time for P5 : 8 - 0 = 8
- Waiting time: Turnaround time - burst time
  - Waiting time for P1 : 13 - 5 = 8
  - Waiting time for P2 : 4 - 3 = 1

- Waiting time for P3 : 1 - 1 = 0
- Waiting time for P4 : 20 - 7 = 13
- Waiting time for P5 : 8 - 4 = 4

Avg waiting time = (8+1+0+13+4)/5 = 5.2 ms

3.3 Non-preeemptive Priority

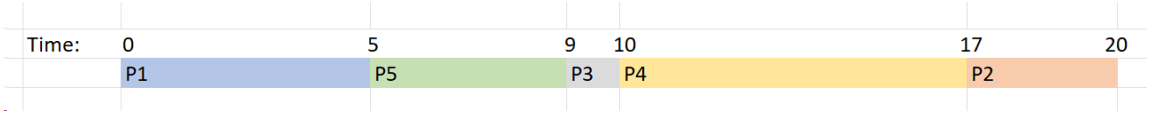

Figure 3: Non-preemptive priority

- Turnaround time: Completion time - arrival time
  - Turnaround time for P1 : 5 - 0 = 5
  - Turnaround time for P2 : 20 - 0 = 20
  - Turnaround time for P3 : 10 - 0 = 10
  - Turnaround time for P4 : 17 - 0 = 17
  - Turnaround time for P5 : 9 - 0 = 9
- Waiting time: Turnaround time - burst time
  - Waiting time for P1 : 5 - 5 = 0
  - Waiting time for P2 : 20 - 3 = 17
  - Waiting time for P3 : 10 - 1 = 9
  - Waiting time for P4 : 17 - 7 = 10
  - Waiting time for P5 : 9 - 4 = 5

Avg waiting time = (0+17+9+10+5)/5 = 8.2 ms

3.4 Round Robin

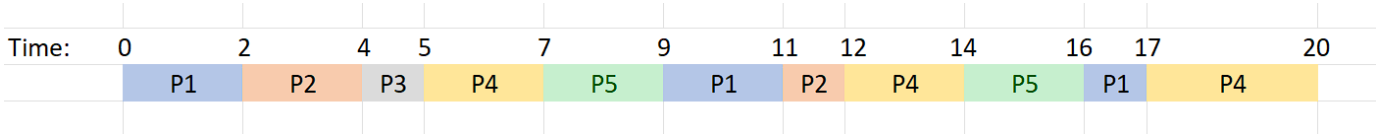

Figure 4: Round Robin (Quantum: 2)

- Turnaround time: Completion time - arrival time
  - Turnaround time for P1 : 17 - 0 = 17
  - Turnaround time for P2 : 12 - 0 = 12
  - Turnaround time for P3 : 5 - 0 = 5
  - Turnaround time for P4 : 20 - 0 = 20
  - Turnaround time for P5 : 16 - 0 = 16
- Waiting time: Turnaround time - burst time
  - Waiting time for P1 : 17 - 5 = 12
  - Waiting time for P2 : 12 - 3 = 9
  - Waiting time for P3 : 5 - 1 = 4
  - Waiting time for P4 : 20 - 7 = 13
  - Waiting time for P5 : 16 - 4 = 12

  Avg waiting time = (12+9+4+13+12)/5 = 10 ms

3.5 Best Waiting time

The average waiting times for the above methods are as follows:

- FCFS: 7.6 ms
- SJF: 5.2 ms
- Priority: 8.2 ms
- Round Robin: 10 ms

SJF (Shortest Job first) results in minimum average waiting time.

4 Tasks with periods

- Period of process 1: p1 = 50ms
- Period of process 2: p2 = 75ms
- Burst time of process 1: t1 = 25ms
- Burst time of process 2: t2 = 30ms

4.1 RMS (Rate monotonic scheduling)

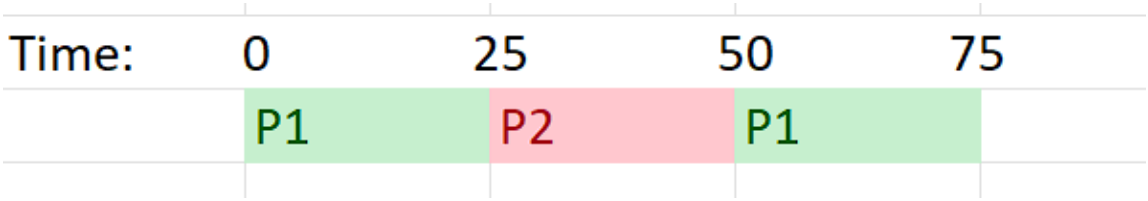

Figure 5: Gantt Chart for RMS: P2 executes only for 25ms in its first period which is less than its burst time t2=30ms

Rate-Monotonic Scheduling is a priority algorithm that assigns higher priorities to tasks with shorter periods. Since, p1 is less than p2, process 1 will have higher priority. P1 starts first and finishes its work at the 25th time unit. Then, P2 begins to work until the 50th time unit. At that time, P1 is ready to work again, so it interrupts P2 and starts its task, completing it by the 75th time unit. At this point, P2 can resume, but it still has 5 time units of work left from its first cycle. This means it doesn't finish its task on time. Hence, these 2 processes can't be schedules using rate monotonic scheduling.

4.2 EDF (Earliest deadline first scheduling)

It is a dynamic priority algorithm so that the priority of a task can change during runtime. The highest priority job is the one with the earliest deadline, If two tasks have the same absolute deadlines, chose one of them randomly.

Deadline of P1 is 50 and that of P2 is 75. So, scheduling using EDF, P1 will be scheduled first due to earlier deadline. So, P1 will start at time 0 and finish at time 25. New deadline of P1 is 100 and that of P2 is 75. So, P2 will start executing and continue till 55. New deadline of P2 is 150 and that of P1 is 100. So, P1 will start at time

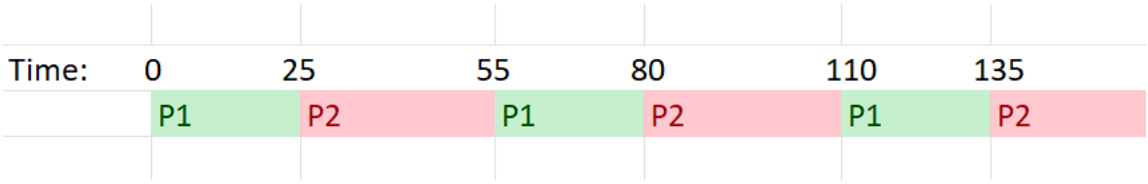

Figure 6: EDF: Earliest deadline first scheduling

55 and end at 80. New deadline of P1 as well as P2 is 150. This will keep on repeating.

5 Dynamically changing priorities

α : Priority change rate for a waiting process
β : Priority change rate for a running process

a. When $\beta > \alpha > 0$:

This condition implies that the priority of a process increases faster when it is running (β) compared to when it is waiting (α), with both rates of change being positive. This scenario favors a process that is running, as its priority accelerates more rapidly than those waiting. So no other process will preempt the running process and it will run to its completion. Hence, this algorithm would perform like FCFS - First come first served, as the process which would run first, would have its priority increasing untill it finishes, only then next process is run.

b. When $\alpha < \beta < 0$:

In this scenario, the priority of a process decreases over time, whether it is running or waiting, but decreases at a slower rate (β) when running than when waiting (α). Let all processes in the ready queue begin

with an initial priority of 0, denoted as $P_0 = 0$. Their priority declines consistently at rate a (where $\alpha < 0$). The last process to join the queue is running because it has the highest priority at the start. It won't be stopped by the processes that were already waiting because it loses priority slower ($a < b < 0$) than they do. However, if a new process comes into the queue, it will take over because it starts with the highest priority of zero (Other process whose initial priority was 0, would have lower priority by now as their priority is decreasing). Hence, this algorithm would perform like LCFS - last come first served scheduling.