Assignment 2 - Process Scheduling, Memory Management

SYSC 4001

Claire Villanueva 101220553

Divya Vithiyatharan 101196047

November 13, 2023

Part I

a) Kernel Structure

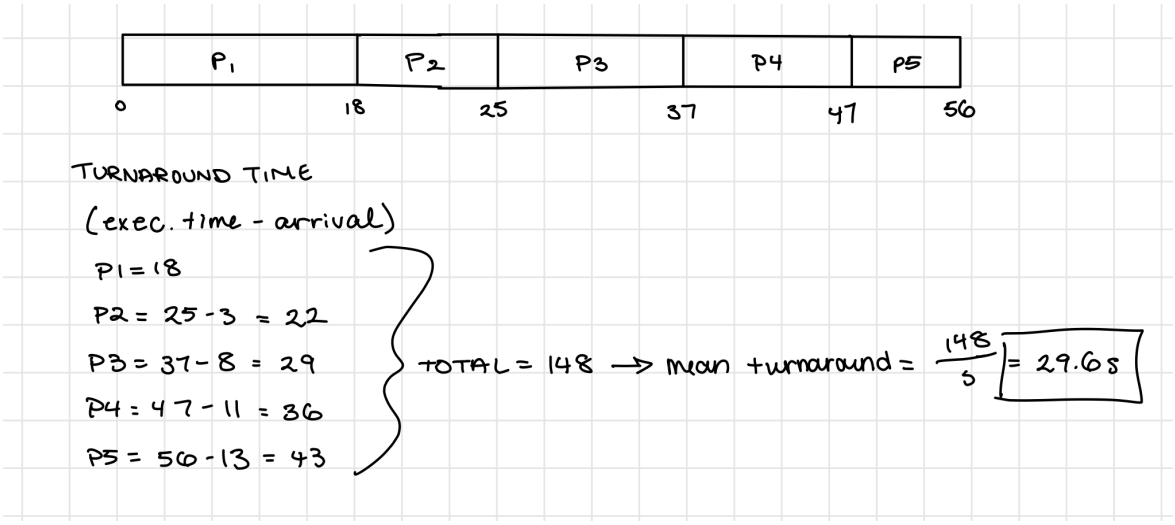
i) Higher Priority Process Arrives - In the event of a higher-priority process arriving in a preemptive scheduler, the OS Kernel interrupts the running process, moving it from the Running state to the Ready state. The CPU scheduler assesses priorities, allocating the CPU to the higher-priority process if present. If not, it selects the next available process for execution.

ii) Time Quantum Expires - In a Round Robin scheduler (with priorities), if the algorithm timer expires, indicating that the process has used up its time quantum, the OS Kernel interrupts the running process, causing it to abandon CPU use and transitioning from the Running state to the Ready state. The CPU scheduler then prioritizes processes, allocating the CPU to a higher-priority process if available.

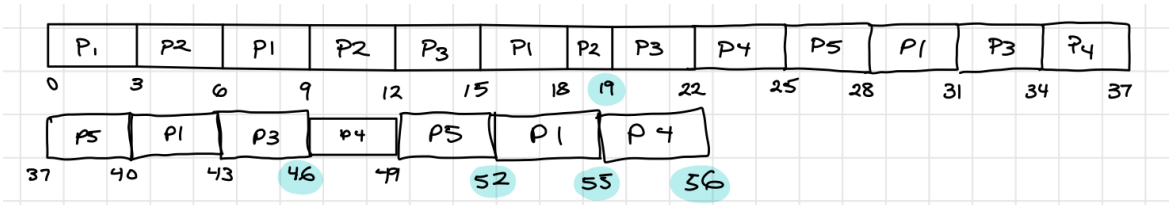
iii) Waits for External Event: If a process must wait for an external event (eg. reading/writing a file or an user interaction) it voluntarily releases the CPU. This could be a non-preemptive scheduler, where processes abandon CPU use voluntarily while waiting for external events.

b) Mean Turnaround Time

i) FCFS



ii) RR with 3s Quantum



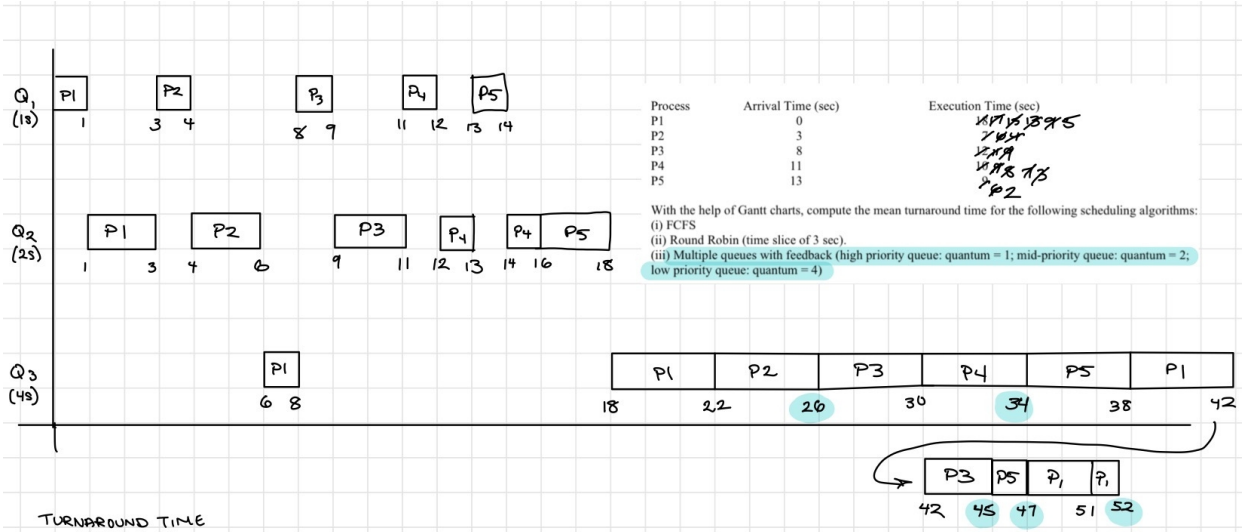
Process	Arrival Time (sec)	Execution Time (sec)
P1	0	
P2	3	
P3	8	
P4	11	
P5	13	

TURNAROUND TIME
(completion time - arrival)

P1 = 55
P2 = 19 - 3 = 16
P3 = 46 - 8 = 38
P4 = 56 - 11 = 45
P5 = 52 - 13 = 39

mean turnaround = $\frac{55+16+38+45+39}{5}$ = 38.6s

iii) Multiple Queues with Feedback



Process	Arrival Time (sec)	Execution Time (sec)
P1	0	
P2	3	
P3	8	
P4	11	
P5	13	

With the help of Gantt charts, compute the mean turnaround time for the following scheduling algorithms:
(i) FCFS
(ii) Round Robin (time slice of 3 sec).
(iii) Multiple queues with feedback (high priority queue: quantum = 1; mid-priority queue: quantum = 2; low priority queue: quantum = 4)

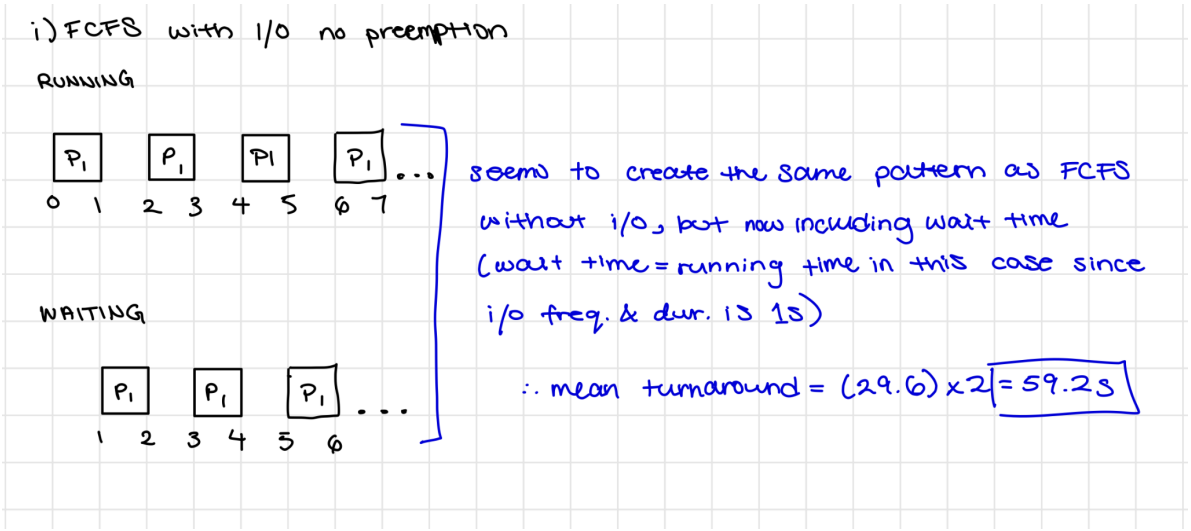
TURNAROUND TIME
(completion time - arrival)

P1 = 52
P2 = 26 - 3 = 23
P3 = 45 - 8 = 37
P4 = 34 - 11 = 23
P5 = 47 - 13 = 34

mean turnaround = $\frac{52+23+37+23+34}{5}$ = 33.8s

c) **With I/O**

i) **FCFS**



After analyzing the other algorithms (RR with 3 second quantum and multiple queues with feedback), all three algorithms will produce the same mean turnaround time since each process has an I/O frequency and duration of 1 second. As a result of the 1 second I/O frequency and duration, this will cause each process in each algorithm to go into I/O after one second, and another process will take over the CPU. Thus, every scheduler behaves the same and results in a mean turnaround of 59.2 seconds.

d) **Algorithms if they are in Favour of Short/Long Processes**

i) FCFS - This scheduler executes processes in the order in which they arrived in the ready queue; the process that arrives first in the ready queue, will be executed first. This scheduler does not favor or discriminate against short processes as FCFS simply executes the first process to arrive, no matter the execution time.

ii) RR - This scheduler provides a sense of fairness as each process is assigned a fixed time quantum, where a process will be preempted if it tries to execute longer than the assigned quantum. This algorithm allows shorter processes to be executed more frequently and also guarantees longer processes to have a chance to execute.

iii) Multi-Level - This scheduler allows processes to move between queues with the intention of separating processes according to the characteristics of their burst time. If a process has a short burst time, it may be favored and moved to a higher priority queue. Conversely, if a process utilizes too much CPU time or has a long burst time, it might be moved to a lower priority queue. Additionally, a process waiting for an extended period in a lower priority queue may be promoted to a higher priority queue. The concept of aging ensures that processes waiting in

lower priority queues are given an opportunity to move to higher priority queues over time, preventing potential resource starvation.

e) Memory Management

102K	205K	43K	180K	70K	125K	91K	150K

i) First Fit : Allocate the first hole that is big enough

102K	205K	43K	180K	70K	125K	91K	150K
Job 4 (90K)	Job 1 (122K)		Job 2 (105K)				

Waiting Queue: Job 3

Job 1 :

1. Job 1 (122K) > Hole 102K
2. Job 1 (122K) < Hole 205K : fill

Job 2 :

1. Job 2 (105K) > Hole 102K
2. Hole 205K : filled
3. Job 2 (105K) > Hole 43K
4. Job 2 (105K) < Hole 180K : fill

Job 3 :

1. Job 3 (203K) > Hole 102K
2. Hole 205K : filled
3. Job 3 (203K) > Hole 43K
4. Hole 180K : filled
5. Job 3 (203K) > Hole 70K
6. Job 3 (203K) > Hole 125K
7. Job 3 (203K) > Hole 91K
8. Job 3 (203K) > Hole 150K

Job 3 is placed into the Waiting State until there is an available hole that fits Job 3 (ie. until Hole 2 (205K) if empty).

Job 4 :

1. Job 4 (90K) < Hole 102K : fill

ii) Best Fit : Allocate the smallest hole that is big enough; must search entire list, unless ordered by size

102K	205K	43K	180K	70K	125K	91K	150K

	Job 3 (203K)				Job 1 (122K)	Job 4 (90K)	Job 2 (105K)

Job 1 (122K) - Searched through all the memory partitions for a partition that fits, the smallest amount of wasted memory is 3K if placed in 125K.
Job 2 (105K) - Searched through all the memory partitions for a partition that fits, the smallest amount of wasted memory is 20K if placed in 125K, but that is filled so the next partition available is 150K with 45K wasted.
Job 3 (203K) - Searched through all the memory partitions for a partition that fits, the only partition that fits is 205K.
Job 4 (90K) - Searched through all the memory partitions for a partition that fits, the smallest amount of wasted memory is 1K if placed in 91K.

iii) Worst Fit : Allocate the largest hole; must also search entire list

f)

102K	205K	43K	180K	70K	125K	91K	150K
	Job 1 (122K)		Job 2 (105K)				Job 4 (90K)

Waiting Queue: Job 3

Job 1 (122K) - Searched through all the memory partitions for the largest partition that fits, which is 205K.
Job 2 (105K) - Searched through all the memory partitions for the largest partition that fits, but since the 205K is already filled, place it in the next largest partition which is 180K.
Job 3 (203K) - Searched through all the memory partitions for the largest partition that fits, which is only 205K; Job 3 is placed into the Waiting State until there is an available hole that fits Job 3 (ie. 205K is empty).
Job 4 (90K) - Searched through all the memory partitions for the largest partition that fits, but since the 205K and 180K are already filled, place it in the next largest partition which is 150K.

Part II:

The instructions to run the simulation program are provided in the README.md inside the project folder, sysc4001_A2.

iii) Simulation Execution Report

In our simulation analysis, we simulated 10 different scenarios using our FCFS, Round Robin, and External Priorities algorithms, and analyzed their resulting throughput, average turnaround time, and wait time. We simulated scenarios in regards to mostly I/O bound processes, mostly CPU processes, and processes with similar I/O and CPU bursts.

For our CPU Bound processes experiment, we inputted processes with a significantly large CPU time, and we observed that FCFS demonstrates better performance out of the three algorithms as its resulting throughput was found to be 0.00005 processes per ms and its average turnaround time was 49600 ms. Whereas the priority and round robin (with a throughput of 0.00048 jobs/ms) scheduler had an average turnaround time of 51600ms and 59000ms, respectively. This makes sense since FCFS minimizes context switching (saving the state of a process and loading the state of another) as FCFS does not preempt processes, which is beneficial for CPU-bound scenarios.

For our I/O Bound processes experiment, we inputted processes with frequent I/O to test for FCFS and external priority schedulers. We found that FCFS had an average turnaround time of 34000 ms; whereas, external priorities had an average turnaround time of 30000 ms. Each scheduler had a similar value for throughput and average wait time, however because external priorities resulted in an average turnaround time of 30000 ms, this proves that external priorities had a better performance for I/O bound processes. Ideally, round robin would have the best performance due to its time slicing and fair algorithm.

For our processes with similar CPU Bursts, we also tested for priority handling where some processes shared the same high priority. In this scenario, we saw that FCFS exhibited a throughput of 0.000093 jobs per ms, an average turnaround time of 24200 ms, and an average wait time of 1800 ms. We also saw that the External Priorities algorithm resulted in a throughput of 0.000093 jobs per ms, an average turnaround time of 27800 ms, and an average wait time of 1800 ms. Alternatively, Round Robin resulted in a throughput of 0.000086 jobs per ms, average turnaround time of 31600 ms, and an average wait time of 1801 ms. This proves that FCFS and External Priorities both perform well with similar CPU bursts, and shared high priorities.

In conclusion, we have observed that Round Robin tends to exhibit high throughput, but also higher wait times in certain cases with long CPU processes. We also saw that FCFS performs well in CPU bound processes with a high throughput and low turnaround time. Finally, External

Priorities was shown to effectively manage wait times and succeed in high throughput in priority driven scenarios.

iv) Memory Management Report

The implemented system utilizes the first-fit algorithm to allocate memory for processes in the ready state. If a process is unable to cannot be allocated memory because there are no holes that will fit the process, the memory is put into the waiting state. When a process enters the ready state, the system searches for the first available block of memory that can accommodate the process's size. Then any process in the waiting state will be moved into the ready state put in the first hole the process fits in.

Despite the successful implementation of the first-fit algorithm, there is a bug in calculating the total amount of memory used. This incorrect calculation unfortunately affects the accurate determination of the total amount of free memory available. Due to the inaccurate reporting of memory usage and free memory, it may lead to inefficient resource utilization and hinder the identification of memory-related issues. But, based on the output in my csv files, the processes are being allocated into the correct memory partition (the first available one that fits). Plus, the processes are being completely removed once the process enters the Running state. Thus, I am pretty confident my program conceptually is correct.