Carleton University
Department of Systems and Computer Engineering
SYSC 4001 Operating Systems Fall 2025

Assignment 2 – Process Scheduling, Memory Management

Timur Grigoryev 101276841(Student 1) [1a) i) , 1a) iii), 1b) , 1c)2.]
Rounak Mukherjee 101116888 (Student 2) [1a) ii) , 1a) iv) , 1c)1.]

Both -> Part II

Rounak Mukherjee

Part I – Concepts

a) ii) Round Robin (time slice of 4 ms).

Ready Queue Management:

- Processes are added to the ready queue when they arrive
- After executing for 4 ms (or completing), the process goes to the back of the queue
- The next process in the queue gets the CPU

Detailed Timeline

Time	Event	Ready Queue (before execution)	Process Executed	Remaining Time
0	P1 arrives	[P1:12]	P1	12->8
4	P1 quantum expires	[P1:8]	P1	8->7
5	P2 arrives	[P1:7, P2:8]	P1	7->6
6	Switch to P2	[P2:8,P1:6]	P2	8->6
8	P3 arrives	[P2:6, P1:6, P3:3]	P2	6->4
10	P2 quantum expires	[P1:6, P3:3, P2:4]	P1	6->2
14	P1 quantum expires	[P3:3, P2:4, P1:2]	P3	3->0 ✓
15	P4 arrives	[P2:4, P1:2, P4:6]	-	-
17	P3 completes	[P2:4, P1:2, P4:6]	P2	4->0 ✓
20	P5 arrives	[P1:2, P4:6, P5:5]	-	-
21	P2 completes	[P1:2, P4:6, P5:5]	P1	2->0 ✓

23	P1 completes	[P4:6, P5:5]	P4	6->2
27	P4 quantum expires	[P5:5, P4:2]	P5	5->1
31	P5 quantum expires	[P4:2, P5:1]	P4	2->0 ✓
33	P4 completes	[P5:1]	P5	1->0 ✓
34	All complete	[]	-	-

Timeline:

Time 0-4: P1 executes (arrives at 0, ready queue: [P1])

- P1: 12 → 8 ms remaining

Time 4-5: P1 continues (only process, 1 ms more)

- At time 5: P2 arrives
- P1: 8 → 7 ms remaining

Time 5-9: Context switch to P2 (ready queue: [P2, P1])

- P2 executes for 4 ms (quantum)
- At time 8: P3 arrives
- P2: 8 → 4 ms remaining

Time 9-13: P1 executes (ready queue: [P1, P3, P2])

- P1 executes for 4 ms (quantum)
- P1: 7 → 3 ms remaining

Time 13-16: P3 executes (ready queue: [P3, P2, P1])

- P3 executes for 3 ms and **completes**
- At time 15: P4 arrives
- P3: 3 → 0 ms ✓

Time 16-20: P2 executes (ready queue: [P2, P1, P4])

- P2 executes for 4 ms and **completes**
- At time 20: P5 arrives
- P2: 4 → 0 ms ✓

Time 20-23: P1 executes (ready queue: [P1, P4, P5])

- P1 executes for 3 ms and **completes**
- P1: 3 → 0 ms ✓

Time 23-27: P4 executes (ready queue: [P4, P5])

- P4 executes for 4 ms (quantum)
- P4: 6 → 2 ms remaining

Time 27-31: P5 executes (ready queue: [P5, P4])

- P5 executes for 4 ms (quantum)
- P5: 5 → 1 ms remaining

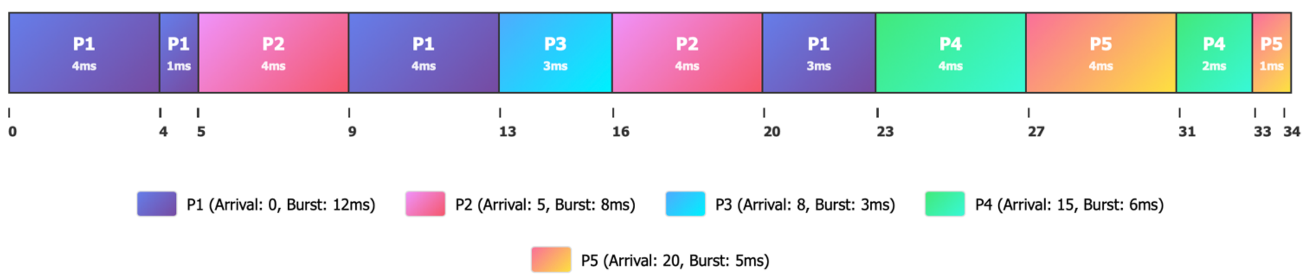
Time 31-33: P4 executes (ready queue: [P4, P5])

- P4 executes for 2 ms and **completes**
- P4: 2 → 0 ms ✓

Time 33-34: P5 executes (ready queue: [P5])

- P5 executes for 1 ms and **completes**
- P5: 1 → 0 ms ✓

1. Gantt Chart



2. Completion Time for Each Process

Process	Arrival Time	Execution Time	Completion Time
P1	0	12	23 ms
P2	5	8	20 ms
P3	8	3	16 ms
P4	15	6	33 ms
P5	20	5	34 ms

3. Turnaround Time for Each Process

Turnaround Time = Completion Time – Arrival Time

Process	Completion Time	Arrival Time	Turnaround Time
P1	23	0	23 - 0 = 23 ms

P2	20	5	20 – 5 = 15 ms
P3	16	8	16 – 8 = 8 ms
P4	33	15	33 – 15 = 18 ms
P5	34	20	34 - 20 = 14 ms

4. Mean Turnround Time

Mean Turnaround Time = Sum of all Turnaround Times / Number of Processes

Mean Turnaround Time = (23 + 15 + 8 + 18 + 14) / 5

Mean Turnaround Time = 78 / 5

Mean Turnaround Time = 15.6 ms

Summary Table

Process	Arrival	Execution	Completion	Turnaround	Waiting Time
P1	0	12	23	23	11
P2	5	8	20	15	7
P3	8	3	16	8	5
P4	15	6	33	18	12
P5	20	5	34	14	9
Average	-	-	-	15.6	8.8

(Waiting Time = Turnaround Time – Exection Time)

iv) Multiple queues with feedback (high-priority queue: quantum = 2; mid-priority queue: quantum = 3; low-priority queue: FIFO)

Algorithm Parameters

- **High-Priority Queue (Q1)**: Time Quantum = 2 ms, Pre-emptive
- **Mid-Priority Queue (Q2)**: Time Quantum = 3 ms, Pre-emptive
- **Low-Priority Queue (Q3)**: FIFO (First-Come-First-Served), Non-pre-emptive

MLFQ Rules:

1. All processes enter at the **highest priority queue (Q1)**
2. If a process uses its entire quantum in Q1, it is **demoted to Q2**
3. If a process uses its entire quantum in Q2, it is **demoted to Q3**
4. If a process completes before its quantum expires, it terminates

5. **Priority:** Q1 > Q2 > Q3 (always execute from highest queue first)
6. When a higher-priority process arrives, it pre-empts lower-priority processes

<u>Execution Steps</u>:

Initial State:

Process	Arrival Time	Execution Time	Current Queue	Remaining Time
P1	0	12	Q1	12
P2	5	8	-	8
P3	8	3	-	3
P4	15	6	-	6
P5	20	5	-	5

Timeline:

Time 0-2: P1 enters Q1 and executes

- P1 uses full quantum (2 ms) → **demoted to Q2**
- P1: 12 → 10 ms remaining
- Queues: Q1[], Q2[P1:10], Q3[]

Time 2-4: P1 executes in Q2 (no other processes)

- P1 uses 2 ms of its 3 ms quantum
- P1: 10 → 8 ms remaining
- At time 4: Still in Q2 (hasn't used full quantum yet)

Time 4-5: P1 continues in Q2

- P1 uses 1 more ms (total 3 ms quantum used) → **demoted to Q3**
- P1: 8 → 7 ms remaining
- At time 5: P2 arrives
- Queues: Q1[P2:8], Q2[], Q3[P1:7]

Time 5-7: P2 enters Q1 (pre-empts P1 in Q3)

- P2 uses full quantum (2 ms) → **demoted to Q2**
- P2: 8 → 6 ms remaining
- Queues: Q1[], Q2[P2:6], Q3[P1:7]

Time 7-8: P2 executes in Q2

- P2 uses 1 ms of its 3 ms quantum
- P2: 6 → 5 ms remaining
- At time 8: P3 arrives

Time 8-10: P3 enters Q1 (pre-empts P2 in Q2)

- P3 uses full quantum (2 ms) → **demoted to Q2**
- P3: 3 → 1 ms remaining
- Queues: Q1[], Q2[P2:5, P3:1], Q3[P1:7]

Time 10-12: P2 continues in Q2 (2 more ms to complete quantum)

- P2 uses 2 more ms (total 3 ms quantum used) → **demoted to Q3**
- P2: 5 → 3 ms remaining
- Queues: Q1[], Q2[P3:1], Q3[P1:7, P2:3]

Time 12-13: P3 executes in Q2

- P3 uses 1 ms and **completes** (before quantum expires)
- P3: 1 → 0 ms ✓
- Queues: Q1[], Q2[], Q3[P1:7, P2:3]

Time 13-15: P1 executes in Q3 (FIFO, non-pre-emptive)

- P1 uses 2 ms
- P1: 7 → 5 ms remaining
- At time 15: P4 arrives

Time 15-17: P4 enters Q1 (pre-empts P1 in Q3)

- P4 uses full quantum (2 ms) → **demoted to Q2**
- P4: 6 → 4 ms remaining
- Queues: Q1[], Q2[P4:4], Q3[P1:5, P2:3]

Time 17-20: P4 executes in Q2

- P4 uses full quantum (3 ms) → **demoted to Q3**
- P4: 4 → 1 ms remaining
- At time 20: P5 arrives
- Queues: Q1[P5:5], Q2[], Q3[P1:5, P2:3, P4:1]

Time 20-22: P5 enters Q1 (pre-empts processes in Q3)

- P5 uses full quantum (2 ms) → **demoted to Q2**
- P5: 5 → 3 ms remaining
- Queues: Q1[], Q2[P5:3], Q3[P1:5, P2:3, P4:1]

Time 22-25: P5 executes in Q2

- P5 uses full quantum (3 ms) → **demoted to Q3**

- P5: 3 → 0 ms **completes** exactly at quantum
- Queues: Q1[], Q2[], Q3[P1:5, P2:3, P4:1]

Time 25-30: P1 executes in Q3 (FIFO, first in queue)

- P1 runs to completion (non-pre-emptive in Q3)
- P1: 5 → 0 ms ✓
- Queues: Q1[], Q2[], Q3[P2:3, P4:1]

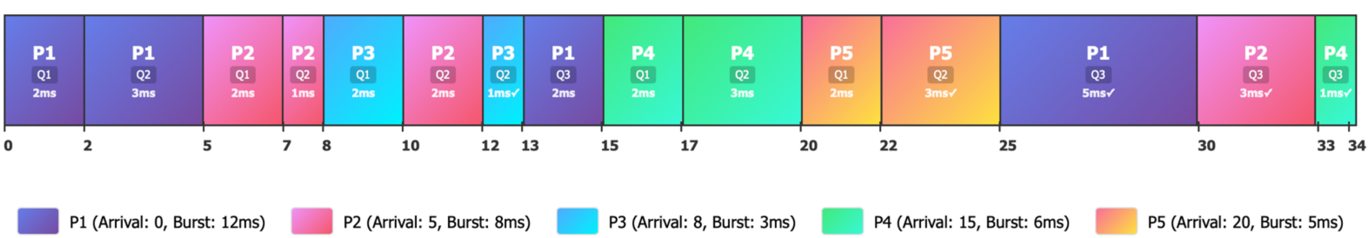
Time 30-33: P2 executes in Q3 (FIFO)

- P2 runs to completion
- P2: 3 → 0 ms ✓
- Queues: Q1[], Q2[], Q3[P4:1]

Time 33-34: P4 executes in Q3 (FIFO)

- P4 runs to completion
- P4: 1 → 0 ms ✓
- Queues: Q1[], Q2[], Q3[]

1. Gantt Chart



Q1 (High Priority) Quantum = 2ms Pre-emptive ; Q2 (Mid Priority) Quantum = 3ms Pre-emptive ; Q3(Low Priority) FIFO Non-pre-emptive

2. Completion Time for Each Process

Process	Arrival Time	Execution Time	Completion Time
P1	0	12	30 ms
P2	5	8	33 ms
P3	8	3	13 ms
P4	15	6	34 ms
P5	20	5	25 ms

3. Turnaround Time for Each Process

Turnaround Time = Completion Time – Arrival Time

Process	Completion Time	Arrival Time	Turnaround Time
P1	30	0	30 – 0 = 30 ms
P2	33	5	33 – 5 = 28 ms
P3	13	8	13 – 8 = 5 ms
P4	34	15	34 – 15 = 19 ms
P5	25	20	25 – 20 = 5 ms

4. Mean Turnaround Time

Mean Turnaround Time = Sum of all Turnaround Times / Number of Processes

Mean Turnaround Time = (30 + 28 + 5 + 19 + 5) / 5

Mean Turnaround Time = 87 / 5

Mean Turnaround Time = 17.4 ms

Summary Table

Process	Arrival	Execution	Queue Path	Completion	Turnaround	Waiting
P1	0	12	Q1->Q2->Q3	30	30	18
P2	5	8	Q1->Q2->Q3	33	28	20
P3	8	3	Q1->Q2	13	5	2
P4	15	6	Q1->Q2->Q3	34	19	13
P5	20	5	Q1->Q2	25	5	0
Average	-	-	-	27	17.4	10.6

- P3 and P5 had the best turnaround times (5 ms each) because they completed in higher-priority queues
- P1 had the longest turnaround time (30 ms) as it was demoted to Q3 and had to wait for all higher-priority processes
- P2 also suffered (28 ms) by being in Q3 and waiting for P1 to complete first

- The feedback mechanism penalizes long-running processes by demoting them to lower queues
- Short processes benefit from starting in the high-priority queue and completing quickly

c) Memory Management Problem – Multiple Partition Allocation

Initial Setup

Free List (in order):

Position	Hole Size	Status
1	85 KB	Free
2	340 KB	Free
3	28 KB	Free
4	195 KB	Free
5	55 KB	Free
6	160 KB	Free
7	75 KB	Free
8	280 KB	Free

Jobs to Allocate:

Job	Arrival Time	Memory Requirement
J1	t_1	140 KB
J2	t_2	82 KB
J3	t_3	275 KB
J4	t_4	65 KB
J5	t_5	190

i) First Fit Algorithm (1)

Strategy: Allocate the job to the first hole that is large enough.

Allocation Process:

J1 (140 KB):

- Scan from position 1: 85 KB (too small)
- Position 2: 340 KB ≥ 140 KB ✓ Allocate here
- Remaining in position 2: 340 - 140 = 200 KB

J2 (82 KB):

- Scan from position 1: 85 KB ≥ 82 KB ✓ Allocate here
- Remaining in position 1: 85 - 82 = 3 KB

J3 (275 KB):

- Position 1: 3 KB (too small)
- Position 2: 200 KB (too small)
- Position 3: 28 KB (too small)
- Position 4: 195 KB (too small)
- Position 5: 55 KB (too small)
- Position 6: 160 KB (too small)
- Position 7: 75 KB (too small)
- Position 8: 280 KB ≥ 275 KB ✓ Allocate here
- Remaining in position 8: 280 - 275 = 5 KB

J4 (65 KB):

- Position 1: 3 KB (too small)
- Position 2: 200 KB ≥ 65 KB ✓ Allocate here
- Remaining in position 2: 200 - 65 = 135 KB

J5 (190 KB):

- Position 1: 3 KB (too small)
- Position 2: 135 KB (too small)
- Position 3: 28 KB (too small)
- Position 4: 195 KB ≥ 190 KB ✓ Allocate here
- Remaining in position 4: 195 - 190 = 5 KB

First Fit Results:

Allocation Table:

Job	Allocated Partition	Partition Size	Job Size	Internal Fragmentation
J1	Position 2	340 KB	140 KB	0 KB (split)
J2	Position 1	85 KB	82 KB	0 KB (split)
J3	Position 8	280 KB	275 KB	0 KB (split)
J4	Position 2 (remaining)	200 KB	65 KB	0 KB (split)
J5	Position 4	195 KB	190 KB	0 KB (split)

Remaining Free Memory:

Position	Size
1	3 KB
2	135 KB
3	28 KB
4	5 KB
5	55 KB
6	160 KB
7	75 KB
8	5 KB

Total Free Memory Remaining: 3 + 135 + 28 + 5 + 55 + 160 + 75 + 5 = **466 KB**

Total Internal Fragmentation: 0 KB (assuming partitions are split exactly)

Total External Fragmentation: 466 KB (free memory that cannot be used because it's fragmented into small pieces)

ii) Best Fit Algorithm (1)

Strategy: Allocate the job to the smallest hole that is large enough.

Allocation Process:

J1 (140 KB):

- Available holes ≥ 140 KB: 340 KB, 195 KB, 160 KB, 280 KB
- Smallest: 160 KB (position 6) ✓ Allocate here
- Remaining: 160 - 140 = 20 KB

J2 (82 KB):

- Available holes ≥ 82 KB: 85 KB, 340 KB, 195 KB, 280 KB
- Smallest: 85 KB (position 1) ✓ Allocate here
- Remaining: 85 - 82 = 3 KB

J3 (275 KB):

- Available holes ≥ 275 KB: 340 KB, 280 KB
- Smallest: 280 KB (position 8) ✓ Allocate here
- Remaining: 280 - 275 = 5 KB

J4 (65 KB):

- Available holes ≥ 65 KB: 340 KB, 195 KB, 75 KB
- Smallest: 75 KB (position 7) ✓ Allocate here
- Remaining: 75 - 65 = 10 KB

J5 (190 KB):

- Available holes ≥ 190 KB: 340 KB, 195 KB
- Smallest: 195 KB (position 4) ✓ Allocate here
- Remaining: 195 - 190 = 5 KB

Best Fit Results:

Allocation Table:

Job	Allocated Partition	Partition Size	Job Size	Internal Fragmentation
J1	Position 6	160 KB	140 KB	0 KB (split)
J2	Position 1	85 KB	82 KB	0 KB (split)
J3	Position 8	280 KB	275 KB	0 KB (split)
J4	Position 7	75 KB	65 KB	0 KB (split)
J5	Position 4	195 KB	190 KB	0 KB (split)

Remaining Free Memory:

Position	Size
1	3 KB
2	340 KB
3	28 KB
4	5 KB
5	55 KB
6	20 KB
7	10 KB
8	5 KB

Total Free Memory Remaining**:** 3 + 340 + 28 + 5 + 55 + 20 + 10 + 5 = **466 KB**

Total Internal Fragmentation**:** 0 KB (assuming partitions are split exactly)

Total External Fragmentation**:** 466 KB

iii) Worst Fit Algorithm (1)

Strategy: Allocate the job to the largest hole available.

Allocation Process:

J1 (140 KB):

- Largest hole: 340 KB (position 2) ✓ Allocate here
- Remaining: 340 - 140 = 200 KB

J2 (82 KB):

- Largest hole: 280 KB (position 8) ✓ Allocate here
- Remaining: 280 - 82 = 198 KB

J3 (275 KB):

- Largest hole: 200 KB (position 2 remaining) - too small
- Next largest: 198 KB (position 8 remaining) - too small
- Next: 195 KB (position 4) - too small
- Cannot allocate J3

Since J3 cannot be allocated, let's continue with remaining jobs:

J4 (65 KB):

- Largest hole: 200 KB (position 2 remaining) ✓ Allocate here
- Remaining: 200 - 65 = 135 KB

J5 (190 KB):

- Largest hole: 198 KB (position 8 remaining) ✓ Allocate here
- Remaining: 198 - 190 = 8 KB

Worst Fit Results:

Allocation Table:

Job	Allocated Partition	Partition Size	Job Size	Status
J1	Position 2	340 KB	140 KB	Allocated
J2	Position 8	280 KB	82 KB	Allocated

J3	-	-	275 KB	NOT ALLOCATED
J4	Position 2 (remaining)	200 KB	65 KB	Allocated
J5	Position 8 (remaining)	198 KB	190 KB	Allocated

Remaining Free Memory:

Position	Size
1	85 KB
2	135 KB
3	28 KB
4	195 KB
5	55 KB
6	160 KB
7	75 KB
8	8 KB

Total Free Memory Remaining: 85 + 135 + 28 + 195 + 55 + 160 + 75 + 8 = **741 KB**

Total Internal Fragmentation: 0 KB

Total External Fragmentation: 741 KB (including J3's requirement of 275 KB that couldn't be satisfied)

Note: 3 (275 KB) could not be allocated because no single hole was large enough, even though total free memory (741 KB) exceeds the requirement.

Summary Comparison

Algorithm	Jobs Allocated	Total Free Memory	External Fragmentation	Largest Free Block
First Fit	5/5 (100%)	466 KB	466 KB	160 KB
Best Fit	5/5 (100%)	466 KB	466 KB	340 KB
Worst Fit	4/5 (80%)	741 KB	741 KB	195 KB

Observations:

1. First Fit and Best Fit successfully allocated all jobs
2. Worst Fit failed to allocate J3 because it fragmented large holes early
3. Best Fit leaves the largest single free block (340 KB), making it best for future allocations
4. Worst Fit performed worst, leaving more total free memory but unable to satisfy all requests

Round Robin (time slice of 4 ms) algorithm

RR shares the CPU *fairly* among ready processes by giving each one a fixed time slice (quantum) in turn. If a process doesn't finish within its slice, it is pre-empted and placed at the end of the ready queue. This repeats until all processes finish.
Assumptions:
• Single CPU, zero context-switch overhead.
• Ready queue is FIFO.
• New arrivals are enqueued at the tail of the ready queue as soon as they arrive.
• If the CPU becomes idle and no process is ready, the clock jumps to the next arrival time.
• Ties are broken FCFS by arrival time.
We should RR algorithm because –
- Good response time and fairness for time-sharing systems.
- The smaller the quantum, the better the response time but the higher the context-switch overhead; the larger the quantum, the more it behaves like FCFS.
Algorithm (pseudocode):

```
q = 4  // ms
time = 0
Ready = FIFO queue

while there exist unfinished processes:
   enqueue to Ready every process with arrival_time ≤ time
   if Ready empty:
      time = next arrival time
      continue
   p = Ready.pop_front()
   run p for t = min(q, remaining_time[p])
   time += t
   decrease remaining_time[p] by t
   enqueue to Ready every process that arrived during this run
   if remaining_time[p] > 0:
      Ready.push_back(p)   // preempt and requeue
```

```
else:
   record p's completion time
```

Part II – Concurrent Processes in Unix

Explanation of the "fork" System Call

➔ The fork() system call is a fundamental Unix/Linux system call used to create a new process. It creates a child process that is an exact copy of the parent process (the process that calls fork()).

Working of the "fork" System Call:

When a process calls fork(), the operating system:

1. Creates a new process (child) that is a duplicate of the calling process (parent)
2. Copies the parent's memory space to the child, including Code (program instructions), Data (variables and their current values), stack, heap
3. Assigns a unique PID to the child process
4. Returns different values to distinguish parent from child:

    - In the parent process: fork() returns the PID of the child (a positive integer)
    - In the child process: fork() returns 0
    - On failure: fork() returns -1 (no child created)

Return Values

```c
pid_t pid = fork();

if (pid < 0) {
    // Error: fork failed
    perror("Fork failed");
}
else if (pid == 0) {
    // Child process code
    printf("I am the child process\n");
}
else {
    // Parent process code
    printf("I am the parent, my child's PID is %d\n", pid);
}
```

Timur Grigoryev

Process	Arrival Time (ms)	Execution Time (ms)
P1	0	12
P2	5	8
P3	8	3
P4	15	6
P5	20	5

1a

i) FCFS

P1	P2	P3	P4	P5

0 12 20 23 29 34

CT

P1 = 12ms
P2 = 20ms
P3 = 23ms
P4 = 29ms
P5 = 34ms

Turnaround time (C-A)

P1 = 12-0 = 12ms
P2 = 20-5 = 15ms
P3 = 23-8 = 15ms
P4 = 29-15 = 14ms
P5 = 34-20 = 14ms

average turnaround time = $\frac{(12+15+15+14+14)}{5}$

= 14ms

ii)

P1	P1	P2	P3	P4	P1	P2	P4	P5	P1	P5

0 4 8 12 15 19 23 27 29 33 37 38

CT

P1 = 37ms
P2 = 27ms
P3 = 15ms
P4 = 29ms
P5 = 38ms

Turnaround time

P1 = 37 - 0 = 37ms
P2 = 27 - 5 = 22ms
P3 = 15 - 8 = 7ms
P4 = 29 - 5 = 14ms
P5 = 38 - 20 = 18ms

mean
Avg turnaround = $\frac{37+22+7+14+18}{5}$

= 19.6

iii)

P1	P3	P1	P4	P5	P2

0 8 11 15 21 26 34

CT

P1 = 15ms
P2 = 34ms
P3 = 11ms
P4 = 21ms
P5 = 26ms

turnaround time

P1 = 15-0 = 15ms
P2 = 34-5 = 29m
P3 = 11-8 = 3ms
P4 = 21-15 = 4ms
P5 = 26-20 = 6ms

mean = $\frac{15+29+3+4+6}{5}$

= 11.78ms

iv)

| P1 | P1 | P2 | P2 | P3 | P2 | P3 | P1 | P4 | P4 | P5 | P5 | P1 | P2 | P4 |

0 2 5 7 8 10 12 13 15 17 20 22 25 30 33 34

CT

P1 = 30ms
P2 = 33ms
P3 = 13ms
P4 = 34ms
P5 = 25ms

Turnaround time

P1 = 30 - 0 = 30ms
P2 = 33 - 5 = 28ms
P3 = 13 - 8 = 5ms
P4 = 34 - 15 = 19ms
P5 = 25 - 20 = 5ms

$$\text{mean} = \frac{30+28+5+19+5}{5}$$

$= 17.3\text{ms}$

b)

| P1 | I/O | P1 | I/O | P2 | P1 | P2 | P3 | P1 | P2 | P3 | P4 | P1 | P2 | P5 | P4 | P1 | P5 | P4 | P5 |

0 2 2.5 4.5 5 7 9 11 13 15 17 18 20 22 24 26 28 30 32 34 35

CT

P1 = 30ms
P2 = 24ms
P3 = 18ms
P4 = 34ms
P5 = 35ms

turnaround time

P1 = 30 - 0 = 30ms
P2 = 24 - 9 = 15ms
P3 = 18 - 8 = 10ms
P4 = 34 - 15 = 19ms
P5 = 35 - 20 = 15ms

$$\text{mean} = \frac{30+15+10+19+15}{5}$$

$= 18.57\text{ms}$

ii) RR

| P1 | I/O | P1 | I/O | P2 | P1 | P2 | P3 | P1 | P2 | P3 | P_4 | P_1 | P2 | P_5 | P_4 | P_1 | P_5 | P_4 | P5 |

0 2 2.5 4.5 5 7 9 11 13 15 17 18 20 22 24 26 28 30 32 34 35

CT

P_1 = 30 ms
P_2 = 24 ms
P_3 = 18 ms
P_4 = 34 ms
P_5 = 35 ms

Turnaround time

P_1 = 30 - 0 = 30ms
P_2 = 24 - 5 = 19ms
P_3 = 18 - 8 = 10ms
P_4 = 34 - 15 = 19ms
P_5 = 35 - 20 = 15ms

$$\text{mean} = \frac{30+19+10+19+15}{5}$$

$= 18.56\text{ms}$

iii) SJF

| P1 | I/O | P1 | I/O | P1 | P2 | P1 | P3 | P1 | P3 | P1 | P2 | P1 | P4 | P1 | P4 | P2 | P4 | P5 |

0 2 2.5 4.5 5 7 7.5 8 10 10.5 11.5 12.5 13 15 15.5 17.5 19 19.5 21.5 22

| P4 | P5 | P2 | P5 | P2 | P5 | P2 | I/O | P2 | I/O | P2 |

22 24 25.5 26 28 28.5 29.5 31 31.5 33.5 34 36

Turnaround time

P_1 = 17.5ms
P_2 = 36ms
P_3 = 11.5ms
P_4 = 24ms
P_5 = 29.5ms

$P_1 = 17.5 - 0 = 17.5$ms
$P_2 = 36 - 5 = 31$ms
$P_3 = 11.5 - 8 = 3.5$ms
$P_4 = 24 - 15 = 9$ms
$P_5 = 29.5 - 20 = 9.5$ms

$$\text{mean} = \frac{17.5+31+3.5+9+9.5}{5}$$

$= 14.2$ms

iv)

| P1 | I/O | P1 | I/O | P2 | P1 | P2 | P3 | P1 | P2 | P3 | P4 | P1 | P2 | P5 | P4 | P1 | P5 | P4 | P5 |

0 2 2.5 4.5 5 7 9 11 13 15 17 18 20 22 24 26 28 30 32 34 35

CT

P_1 = 30ms
P_2 = 24ms
P_3 = 18ms
P_4 = 34ms
P_5 = 35ms

Turnaround time

$P_1 = 30 - 0 = 30$ms
$P_2 = 24 - 5 = 19$ms
$P_3 = 18 - 8 = 10$ms
$P_4 = 34 - 15 = 19$ms
$P_5 = 35 - 20 = 15$ms

$$\text{mean} = \frac{30+19+10+19+15}{5}$$

mean = 18.6ms

Part 1c2)

Following the application of the First Fit, Best Fit, and Worst Fit allocation techniques to the specified memory configuration, the following findings were noted:

First Fit generated a modest external fragmentation of 466 KB scattered across numerous tiny holes, however it was effective in allocating all five jobs (100% success). Because it stops at the first appropriate hole, it is quick, but over time, it tends to fragment memory close to the beginning.

Additionally, Best Fit left the largest single contiguous block (340 KB), which increases the likelihood of future allocations, while allocating all jobs with the same total free RAM (466 KB). Although it could take longer to find the smallest hole, it is more space-efficient and ideal for systems that routinely assign minor duties.

worst fit only allocated four of the five jobs (80 % success) and left 741 KB of fragmented free memory. It performed the worst overall because it broke large holes too early, making it unable to satisfy later large requests.

Best Fit provides the best balance for systems with frequent small allocations because it minimizes wasted space. First Fit is preferable for mixed workloads where allocation speed is more important than perfect space efficiency. Worst Fit should be avoided due to its high fragmentation and low allocation efficiency.

part1)

FCFS is a non-pre-emptive scheduling algorithm where the process that arrives first gets the CPU first. Processes are executed in order of their arrival time; the CPU stays busy until the current process completes. If a process arrives while another is running, it waits in the ready queue. The process with the smallest arrival time is selected first if there's a tie it's broken by order of appearance.

part2)

The exec() system call in Unix replaces the current running process with a completely new program. When a process calls exec, the operating system loads the specified executable into the process's memory space overwriting its previous code, data, stack, and heap and then starts executing the new program from its entry point, usually the main() function. Importantly, the process keeps the same PID and open file descriptors, but everything else is replaced by the new program. This is why exec() is often used together with fork(): first, fork() creates a new child process, then the child calls exec() to start running a different program. Once exec() succeeds, the old program is gone and exec() never returns to the original code. If it fails if the program file doesn't exist, it returns -1 and the process continues executing the old code. The exec() transforms an existing process into a new one without creating another process.

Timur Grigoryev - Implementation and Documentation
Rounak Mukherjee - Implementation and Testing

Links to GitHub repos:

1. https://github.com/Ronyisreal/SYSC4001_A2_P2 (Part II)

2. https://github.com/Ronyisreal/SYSC4001_A2_P3/tree/main (Part III)