Round Robin Process Scheduler — Linux Guide & Report

1. Objective

This report explains the implementation of a simple Round-Robin CPU scheduler in C using a time quantum of 2. The program simulates process execution, prints a Gantt chart, and outputs waiting and turnaround times.

2. Environment Requirements

Linux OS (Ubuntu recommended)

GCC compiler

Text editor (nano, vim, VS Code)

3. File Creation

Create a file named **scheduler.c**:

```
nano scheduler.c
```

4. Code (C Program)

```
#include <stdio.h>

#include <stdlib.h>

#define TIME_QUANTUM 2

#define NUM_PROCESSES 5

typedef struct {

int pid;

int burst;

int remaining;
```

```
int waiting;
int turnaround;
} Process;

int main() {
Process p[NUM_PROCESSES];
int burstTimes[NUM_PROCESSES] = {5, 3, 8, 6, 4};

printf("=== Round Robin (Time Quantum = 2) ===\n\n");

for(int i = 0; i < NUM_PROCESSES; i++) {
p[i].pid = i + 1;
p[i].burst = burstTimes[i];
p[i].remaining = burstTimes[i];
p[i].waiting = 0;
}

int time = 0;
int done;
printf("Gantt Chart:\n");

do {
done = 1;
for(int i = 0; i < NUM_PROCESSES; i++) {
if (p[i].remaining > 0) {
done = 0;
printf("| P%d (%d→", p[i].pid, time);

if (p[i].remaining > TIME_QUANTUM) {
time += TIME_QUANTUM;
```

```
p[i].remaining -= TIME_QUANTUM;
} else {
time += p[i].remaining;
p[i].waiting = time - p[i].burst;
p[i].remaining = 0;
}

printf("%d) ", time);
}
}
} while(!done);

printf("|\n\n");

for(int i = 0; i < NUM_PROCESSES; i++) {
p[i].turnaround = p[i].waiting + p[i].burst;
}

printf("PID\tBurst\tWaiting\tTurnaround\n");
for(int i = 0; i < NUM_PROCESSES; i++) {
printf("P%-2d\t%-5d\t%-7d\t%-10d\n",
p[i].pid, p[i].burst, p[i].waiting, p[i].turnaround);
}

return 0;
}
```

5. Compile the Code

gcc scheduler.c -o scheduler

6. Run the Program

./scheduler

7. Output Explanation

Gantt Chart — Shows execution slices of each process.

Waiting Time — Time a process waits before full execution.

Turnaround Time — Total time from start to finish of the process.

8. Conclusion

The project demonstrates how Round-Robin scheduling works at the OS level. It shows time-slicing, fairness, and queue-based execution.