OS Assignment 3

```
#include <stdio.h>
#include <stdlib.h>
struct info {
	int Pid; // Process ID
	int Bt; // Burst Time
	int At; // Arrival Time
	int P; // Priority (for priority scheduling)
	int Ct; // Completion Time
	int Wt; // Waiting Time
	int Tat; // Turnaround Time
	int Rt; // Remaining Time (for preemptive scheduling)
};
struct gantt_chart {
	int pid;
	int start; // Start Time of execution
	int end; // End Time of execution
};
// Quick Sort
int partition(struct info arr[], int low, int high) {
	int pivot = arr[high].At;
	int i = low - 1;
	struct info temp;
	for (int j = low; j < high; j++) {
		if (arr[j].At < pivot) {
			i++;
			temp = arr[i];
			arr[i] = arr[j];
			arr[j] = temp;
		}
	}
	temp = arr[i + 1];
	arr[i + 1] = arr[high];
	arr[high] = temp;
	return i + 1;
}
void quick_sort(struct info arr[], int low, int high) {
	if (low < high) {
		int pi = partition(arr, low, high);
		quick_sort(arr, low, pi - 1);
		quick_sort(arr, pi + 1, high);
	}
}
// FCFS Scheduling
void fcfs(struct info arr[], int n) {
	quick_sort(arr, 0, n - 1); // Sorting by Arrival Time
	int completed = 0;
	float total_waiting_time = 0;
	float total_tat_time = 0;
	for (int i = 0; i < n; i++) {
		if(i == 0){
```

```
            completed = arr[i].At + arr[i].Bt;
        }
        else if(arr[i].At > completed){
            completed += arr[i].At + arr[i].Bt;
        }else{
            completed += arr[i].Bt;
        }
        arr[i].Ct = completed;
        arr[i].Tat = arr[i].Ct - arr[i].At;
        arr[i].Wt = arr[i].Tat - arr[i].Bt;
        if (arr[i].Wt < 0){
            arr[i].Wt = 0;
        }
        total_waiting_time += arr[i].Wt;
        total_tat_time += arr[i].Tat;
    }
    // Display Process Order
    printf("Processes completed in following order:\n");
    printf("PID\tBT\tAT\tCT\tWT\tTAT\n");
    for (int i = 0; i < n; i++){
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", arr[i].Pid, arr[i].Bt, arr[i].At, arr[i].Ct,
arr[i].Wt, arr[i].Tat);
    }
    printf("Average Waiting Time = %.2f\n", total_waiting_time / n);
    printf("Average Turnaround Time = %.2f\n", total_tat_time / n);
}
// SJF Non-Preemptive
void sjf_non_preemptive(struct info arr[], int n) {
    quick_sort(arr, 0, n - 1); // Sorting by Arrival Time
    int is_completed[n];
    for (int i = 0; i < n; i++) {
        is_completed[i] = 0;
    }
    int completed = 0;
    int current_time = 0;
    float total_waiting_time = 0;
    float total_tat_time = 0;
    struct gantt_chart gantt[n];
    int chart_idx = 0;
    while (completed < n) {
        int idx = -1;
        int min_bt = 100;
        for (int i = 0; i < n; i++) {
            if (arr[i].At <= current_time && !is_completed[i]) {
                if (arr[i].Bt < min_bt) {
                    min_bt = arr[i].Bt;
                    idx = i;
                }
                else if (arr[i].Bt == min_bt) {
                    if (arr[i].At < arr[idx].At) {
                        idx = i;
                    }
            }
```

```
                        }
                }
                if (idx != -1) {
                        gantt[chart_idx].pid = arr[idx].Pid;
                        gantt[chart_idx].start = current_time;
                        current_time += arr[idx].Bt;
                        gantt[chart_idx].end = current_time;
                        chart_idx++;
                        arr[idx].Ct = current_time;
                        arr[idx].Tat = arr[idx].Ct - arr[idx].At;
                        arr[idx].Wt = arr[idx].Tat - arr[idx].Bt;
                        total_waiting_time += arr[idx].Wt;
                        total_tat_time += arr[idx].Tat;
                        is_completed[idx] = 1;
                        completed++;
                } else{
                        current_time++;
                }
        }
        // Display Process Order
        printf("PID\tBT\tAT\tCT\tWT\tTAT\n");
        for (int i = 0; i < n; i++){
                printf("P%d\t%d\t%d\t%d\t%d\t%d\n", arr[i].Pid, arr[i].Bt, arr[i].At, arr[i].Ct,
arr[i].Wt, arr[i].Tat);
        }
        printf("Average Waiting Time = %.2f\n", total_waiting_time / n);
        printf("Average Turnaround Time = %.2f\n", total_tat_time / n);
        // Display Gantt Chart
        printf("\nGantt Chart:\n-");
        for(int i = 0; i < chart_idx; i++){
                printf("-----");
        }
        printf("\n|");
        for (int i = 0; i < chart_idx; i++) {
                printf(" P%d |", gantt[i].pid);
        }
        printf("\n-");
        for(int i = 0; i < chart_idx; i++){
                printf("-----");
        }
        printf("\n");
        printf("%d", gantt[0].start);
        for (int i = 0; i < chart_idx; i++) {
                if(gantt[i].end < 10){
                        printf("    %d", gantt[i].end);
                }
                else{
                        printf("   %d", gantt[i].end);
                }
        }
        printf("\n");
}
// SJF Preemptive
void sjf_preemptive(struct info arr[], int n) {
```

```
quick_sort(arr, 0, n - 1); // Sorting by Arrival Time
int is_completed[n];
int completed = 0;
int current_time = 0;
int prev = -1;
float total_waiting_time = 0;
float total_tat_time = 0;
struct gantt_chart gantt[100];
int chart_idx = 0;
for (int i = 0; i < n; i++) {
        arr[i].Rt = arr[i].Bt;
        is_completed[i] = 0;
}
while (completed < n) {
        int idx = -1, min_rt = 100;
        for (int i = 0; i < n; i++) {
                if (arr[i].At <= current_time && !is_completed[i] && arr[i].Rt <
                        min_rt && arr[i].Rt > 0) {
                        min_rt = arr[i].Rt;
                        idx = i;
                }
        }
        if (idx != -1) {
                if (prev != idx) {
                        if (prev != -1){
                                gantt[chart_idx++].end = current_time;
                        }
                        gantt[chart_idx].pid = arr[idx].Pid;
                        gantt[chart_idx].start = current_time;
                        prev = idx;
                }
                arr[idx].Rt--;
                current_time++;
                if (arr[idx].Rt == 0) {
                        arr[idx].Ct = current_time;
                        arr[idx].Tat = arr[idx].Ct - arr[idx].At;
                        arr[idx].Wt = arr[idx].Tat - arr[idx].Bt;
                        total_waiting_time += arr[idx].Wt;
                        total_tat_time += arr[idx].Tat;
                        is_completed[idx] = 1;
                        completed++;
                }
        } else {
        if (prev != -1) {
                gantt[chart_idx++].end = current_time;
                prev = -1;
        }
        current_time++;
        }
}
if (prev != -1) {
        gantt[chart_idx++].end = current_time;
}
// Display Process Order
```

```
    printf("PID\tBT\tAT\tCT\tWT\tTAT\n");
    for (int i = 0; i < n; i++){
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", arr[i].Pid, arr[i].Bt, arr[i].At, arr[i].Ct,
arr[i].Wt, arr[i].Tat);
    }
    printf("Average Waiting Time = %.2f\n", total_waiting_time / n);
    printf("Average Turnaround Time = %.2f\n", total_tat_time / n);
    // Display Gantt Chart
    printf("\nGantt Chart:\n-");
    for(int i = 0; i < chart_idx; i++){
        printf("-----");
    }
    printf("\n|");
    for (int i = 0; i < chart_idx; i++) {
        printf(" P%d |", gantt[i].pid);
    }
    printf("\n-");
    for(int i = 0; i < chart_idx; i++){
        printf("-----");
    }
    printf("\n");
    printf("%d", gantt[0].start);
    for (int i = 0; i < chart_idx; i++) {
        if(gantt[i].end < 10){
            printf("    %d", gantt[i].end);
        }
        else{
            printf("   %d", gantt[i].end);
        }
    }
    printf("\n");
}
// Priority Preemptive
void priority_preemptive(struct info arr[], int n) {
    quick_sort(arr, 0, n - 1); // Sorting by Arrival Time
    int is_completed[n];
    int completed = 0;
    int current_time = 0;
    int prev = -1;
    float total_waiting_time = 0;
    float total_tat_time = 0;
    struct gantt_chart gantt[100];
    int chart_idx = 0;
    for (int i = 0; i < n; i++) {
        arr[i].Rt = arr[i].Bt;
        is_completed[i] = 0;
    }
    while (completed < n) {
        int idx = -1, max_priority = -1;

        for (int i = 0; i < n; i++) {
            if (arr[i].At <= current_time && !is_completed[i] && arr[i].P >
max_priority && arr[i].Rt > 0) {
                max_priority = arr[i].P;
```

```
                    idx = i;
                }
            }
            if (idx != -1) {
                if (prev != idx) {
                    if (prev != -1){
                        gantt[chart_idx++].end = current_time;
                    }
                    gantt[chart_idx].pid = arr[idx].Pid;
                    gantt[chart_idx].start = current_time;
                    prev = idx;
                }
                arr[idx].Rt--;
                current_time++;
                if (arr[idx].Rt == 0) {
                    arr[idx].Ct = current_time;
                    arr[idx].Tat = arr[idx].Ct - arr[idx].At;
                    arr[idx].Wt = arr[idx].Tat - arr[idx].Bt;
                    total_waiting_time += arr[idx].Wt;
                    total_tat_time += arr[idx].Tat;
                    is_completed[idx] = 1;
                    completed++;
                }
            } else {
                if (prev != -1) {
                gantt[chart_idx++].end = current_time;
                prev = -1;
                }
                current_time++;
            }
        }
        if (prev != -1){
            gantt[chart_idx++].end = current_time;
        }
        // Display Process Order
        printf("PID\tBT\tAT\tP\tCT\tWT\tTAT\n");
        for (int i = 0; i < n; i++){
            printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", arr[i].Pid, arr[i].Bt, arr[i].At, arr[i].P, arr[i].Ct, arr[i].Wt, arr[i].Tat);
        }
        printf("Average Waiting Time = %.2f\n", total_waiting_time / n);
        printf("Average Turnaround Time = %.2f\n", total_tat_time / n);
        // Display Gantt Chart
        printf("\nGantt Chart:\n-");
        for(int i = 0; i < chart_idx; i++){
            printf("-----");
        }
        printf("\n|");
        for (int i = 0; i < chart_idx; i++) {
            printf(" P%d |", gantt[i].pid);
        }
        printf("\n-");
        for(int i = 0; i < chart_idx; i++){
            printf("-----");
```

```
    }
    printf("\n");
    printf("%d", gantt[0].start);
    for (int i = 0; i < chart_idx; i++) {
        if(gantt[i].end < 10){
            printf("    %d", gantt[i].end);
        }
        else{
            printf("   %d", gantt[i].end);
        }
    }
    printf("\n");
}
// Priority Non-Preemptive
void priority_non_preemptive(struct info arr[], int n) {
    quick_sort(arr, 0, n - 1); // Sorting by Arrival Time
    int is_completed[n];
    int completed = 0;
    int current_time = 0;
    float total_waiting_time = 0;
    float total_tat_time = 0;
    struct gantt_chart gantt[n];
    int chart_idx = 0;
    for (int i = 0; i < n; i++){
        is_completed[i] = 0;
    }
    while (completed < n) {
        int idx = -1, max_priority = -1;
        for (int i = 0; i < n; i++) {
            if (arr[i].At <= current_time && !is_completed[i] && arr[i].P >
max_priority) {
                max_priority = arr[i].P;
                idx = i;
            }
        }
        if (idx != -1) {
            gantt[chart_idx].pid = arr[idx].Pid;
            gantt[chart_idx].start = current_time;
            current_time += arr[idx].Bt;
            gantt[chart_idx].end = current_time;
            arr[idx].Ct = current_time;
            arr[idx].Tat = arr[idx].Ct - arr[idx].At;
            arr[idx].Wt = arr[idx].Tat - arr[idx].Bt;
            total_waiting_time += arr[idx].Wt;
            total_tat_time += arr[idx].Tat;
            is_completed[idx] = 1;
            completed++;
            chart_idx++;
        } else{
            current_time++;
        }
    }
    // Display Process Order
    printf("PID\tBT\tAT\tP\tCT\tWT\tTAT\n");
```

```
        for (int i = 0; i < n; i++){
                printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", arr[i].Pid, arr[i].Bt, arr[i].At,
arr[i].P, arr[i].Ct, arr[i].Wt, arr[i].Tat);
        }
        printf("Average Waiting Time = %.2f\n", total_waiting_time / n);
        printf("Average Turnaround Time = %.2f\n", total_tat_time / n);
        // Display Gantt Chart
        printf("\nGantt Chart:\n-");
        for(int i = 0; i < chart_idx; i++){
                printf("-----");
        }
        printf("\n|");
        for (int i = 0; i < chart_idx; i++) {
                printf(" P%d |", gantt[i].pid);
        }
        printf("\n-");
        for(int i = 0; i < chart_idx; i++){
                printf("-----");
        }
        printf("\n");
        printf("%d", gantt[0].start);
        for (int i = 0; i < chart_idx; i++) {
                if(gantt[i].end < 10){
                        printf("    %d", gantt[i].end);
                }
                else{
                        printf("   %d", gantt[i].end);
                }
        }
        printf("\n");
}
// Round Robin
void round_robin(struct info arr[], int n, int tq) {
        quick_sort(arr, 0, n - 1); // Sorting by Arrival Time
        int is_completed[n];
        int completed = 0;
        int current_time = 0;
        int prev = -1;
        float total_waiting_time = 0;
        float total_tat_time = 0;
        struct gantt_chart gantt[100];
        int chart_idx = 0;
        int front = 0;
        int rear = 0;
        int ready[100];
        for (int i = 0; i < n; i++) {
                arr[i].Rt = arr[i].Bt;
                is_completed[i] = 0;
                if (arr[i].At == 0){
                        ready[rear++] = i;
                }
        }
        while (completed < n) {
                if (front == rear) {
```

```
            current_time++;
            for (int i = 0; i < n; i++){
                if (arr[i].At == current_time){
                    ready[rear++] = i;
                }
            }
            continue;
        }
        int idx = ready[front++];
        if (arr[idx].Rt > 0) {
            if (prev != -1) {
                gantt[chart_idx++].end = current_time;
            }
            gantt[chart_idx].pid = arr[idx].Pid;
            gantt[chart_idx].start = current_time;
            prev = idx;
            int exec_time = 0;
            if(arr[idx].Rt < tq){
                exec_time = arr[idx].Rt;
            }else{
                exec_time = tq;
            }
            arr[idx].Rt -= exec_time;
            current_time += exec_time;
            for (int t = current_time - exec_time + 1; t <= current_time; t++){
                for (int i = 0; i < n; i++){
                    if (arr[i].At == t){
                        ready[rear++] = i;
                    }
                }
            }
            if (arr[idx].Rt == 0) {
                arr[idx].Ct = current_time;
                arr[idx].Tat = arr[idx].Ct - arr[idx].At;
                arr[idx].Wt = arr[idx].Tat - arr[idx].Bt;
                total_waiting_time += arr[idx].Wt;
                total_tat_time += arr[idx].Tat;
                is_completed[idx] = 1;
                completed++;
            } else {
                ready[rear++] = idx;
            }
        }
    }
    if(prev != -1){
        gantt[chart_idx++].end = current_time;
    }
    // Display Process Order
    printf("PID\tBT\tAT\tCT\tWT\tTAT\n");
    for (int i = 0; i < n; i++){
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", arr[i].Pid, arr[i].Bt, arr[i].At, arr[i].Ct,
arr[i].Wt, arr[i].Tat);
    }
    printf("Average Waiting Time = %.2f\n", total_waiting_time / n);
    printf("Average Turnaround Time = %.2f\n", total_tat_time / n);
    // Display Gantt Chart
    printf("\nGantt Chart:\n-");
    for(int i = 0; i < chart_idx; i++){
        printf("-----");
    }
    printf("\n|");
    for (int i = 0; i < chart_idx; i++) {
        printf(" P%d |", gantt[i].pid);
    }
    printf("\n-");
    for(int i = 0; i < chart_idx; i++){
        printf("-----");
    }
    printf("\n");
    printf("%d", gantt[0].start);
    for (int i = 0; i < chart_idx; i++) {
        if(gantt[i].end < 10){
            printf("    %d", gantt[i].end);
        }
        else{
            printf("   %d", gantt[i].end);
        }
    }
    printf("\n");
}
// Main Function
int main() {
    int count, choice;
    printf("Enter the number of processes : ");
    scanf("%d", &count);
    struct info* array = (struct info*) malloc(count * sizeof(struct info)); // dynamic memory allocation
    // malloc : memory allocation. sizeof(struct) : total size of 1 struct.
    // count * sizeof(struct) : total size needed to be allocated struct info* array: pointer used to point to first element of array
    // Taking Input from User
    printf("Enter %d process info: \n", count);
    for (int i = 0; i < count; i++) {
        printf("Enter the Pid of process %d : ", i + 1);
        scanf("%d", &array[i].Pid);
        printf("Enter the Arrival time of process %d : ", i + 1);
        scanf("%d", &array[i].At);
        printf("Enter the Burst time of process %d : ", i + 1);
        scanf("%d", &array[i].Bt);
        array[i].Rt = array[i].Bt;
        printf("\n");
    }

    printf("\nProgram Menu:\nOptions:\n 1) FCFS Scheduling\n 2) Preemptive SJF\n 3) Non-Preemptive SJF\n 4) Preemptive Priority\n 5) Non-Preemptive Priority\n 6) Round Robin\n 7) Exit\n");
    printf("Enter Your Choice: ");
    scanf("%d", &choice);
    int time_quantum;
    switch (choice) {
        case 1:
            fcfs(array, count);
            break;
        case 2:
            sjf_preemptive(array, count);
            break;
        case 3:
            sjf_non_preemptive(array, count);
            break;
        case 4:
            printf("Specify the priority of the processes - \n");
            for (int i = 0; i < count; i++) {
            printf("Enter the Priority of process %d : ", i + 1);
            scanf("%d", &array[i].P);
            }
            priority_preemptive(array, count);
            break;
        case 5:
            printf("Specify the priority of the processes - \n");
            for (int i = 0; i < count; i++) {
            printf("Enter the Priority of process %d : ", i + 1);
            scanf("%d", &array[i].P);
            }
            priority_non_preemptive(array, count);
            break;
        case 6:
            printf("Enter Time Quantum: ");
            scanf("%d", &time_quantum);
            round_robin(array, count, time_quantum);
            break;
        case 7:
            printf("Exit\n");
            break;
        default:
            printf("Invalid Choice\n");
    }
    return 0;
}
```

OUTPUT:-

FCFS:

```
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ gcc Process_Scheduling.c -o
PS
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ ./PS
Enter the number of processes : 5
Enter 5 process info:
Enter the Pid of process 1 : 1
Enter the Arrival time of process 1 : 0
Enter the Burst time of process 1 : 3
Enter the Pid of process 2 : 2
Enter the Arrival time of process 2 : 2
Enter the Burst time of process 2 : 3
Enter the Pid of process 3 : 3
Enter the Arrival time of process 3 : 3
Enter the Burst time of process 3 : 1
Enter the Pid of process 4 : 4
Enter the Arrival time of process 4 : 5
Enter the Burst time of process 4 : 4
Enter the Pid of process 5 : 5
Enter the Arrival time of process 5 : 8
Enter the Burst time of process 5 : 2
Program Menu:
Options:
 1) FCFS Scheduling
 2) Preemptive SJF
 3) Non-Preemptive SJF
 4) Preemptive Priority
 5) Non-Preemptive Priority
 6) Round Robin
 7) Exit
Enter Your Choice: 1
Processes completed in following order:
PID BT AT CT WT TAT
P1 3 0 3 0 3
P2 3 2 6 1 4
P3 1 3 7 3 4
P4 4 5 11 2 6
P5 2 8 13 3 5
Average Waiting Time = 1.80
Average Turnaround Time = 4.40
Preemptive SJF
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ gcc Process_Scheduling.c -o
PS
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ ./PS
Enter the number of processes : 4
Enter 4 process info:
Enter the Pid of process 1 : 1
Enter the Arrival time of process 1 : 0
Enter the Burst time of process 1 : 8
Enter the Pid of process 2 : 2
Enter the Arrival time of process 2 : 1
Enter the Burst time of process 2 : 4
Enter the Pid of process 3 : 3
Enter the Arrival time of process 3 : 2
Enter the Burst time of process 3 : 9
Enter the Pid of process 4 : 4
Enter the Arrival time of process 4 : 3
Enter the Burst time of process 4 : 5
Program Menu:
Options:
 1) FCFS Scheduling
 2) Preemptive SJF
 3) Non-Preemptive SJF
 4) Preemptive Priority
 5) Non-Preemptive Priority
 6) Round Robin
 7) Exit
Enter Your Choice: 2
PID BT AT CT WT TAT
P1 8 0 17 9 17
P2 4 1 5 0 4
P3 9 2 26 15 24
P4 5 3 10 2 7
Average Waiting Time = 6.50
Average Turnaround Time = 13.00
Gantt Chart:
-------------------------
| P1 | P2 | P4 | P1 | P3 |
-------------------------
0 1 5 10 17 26
Non-Preemptive SJF
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ gcc Process_Scheduling.c -o
PS
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ ./PS
Enter the number of processes : 4
Enter 4 process info:
Enter the Pid of process 1 : 1
Enter the Arrival time of process 1 : 0
Enter the Burst time of process 1 : 8
Enter the Pid of process 2 : 2
Enter the Arrival time of process 2 : 1
Enter the Burst time of process 2 : 4
Enter the Pid of process 3 : 3
Enter the Arrival time of process 3 : 2
Enter the Burst time of process 3 : 9
Enter the Pid of process 4 : 4
Enter the Arrival time of process 4 : 3
Enter the Burst time of process 4 : 5
Program Menu:
Options:
 1) FCFS Scheduling
 2) Preemptive SJF
 3) Non-Preemptive SJF
 4) Preemptive Priority
 5) Non-Preemptive Priority
 6) Round Robin
 7) Exit
Enter Your Choice: 3
PID BT AT CT WT TAT
P1 8 0 8 0 8
P2 4 1 12 7 11
P3 9 2 26 15 24
P4 5 3 17 9 14
Average Waiting Time = 7.75
Average Turnaround Time = 14.25
Gantt Chart:
```

```
---------------------
| P1 | P2 | P4 | P3 |
---------------------
0 8 12 17 26
Preemptive Priority
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ gcc Process_Scheduling.c -o PS
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ ./PS
Enter the number of processes : 5
Enter 5 process info:
Enter the Pid of process 1 : 1
Enter the Arrival time of process 1 : 0
Enter the Burst time of process 1 : 4
Enter the Pid of process 2 : 2
Enter the Arrival time of process 2 : 1
Enter the Burst time of process 2 : 3
Enter the Pid of process 3 : 3
Enter the Arrival time of process 3 : 2
Enter the Burst time of process 3 : 1
Enter the Pid of process 4 : 4
Enter the Arrival time of process 4 : 3
Enter the Burst time of process 4 : 5
Enter the Pid of process 5 : 5
Enter the Arrival time of process 5 : 4
Enter the Burst time of process 5 : 2
Program Menu:
Options:
 1) FCFS Scheduling
 2) Preemptive SJF
 3) Non-Preemptive SJF
 4) Preemptive Priority
 5) Non-Preemptive Priority
 6) Round Robin
 7) Exit
Enter Your Choice: 4
Specify the priority of the processes -
Enter the Priority of process 1 : 2
Enter the Priority of process 2 : 3
Enter the Priority of process 3 : 4
Enter the Priority of process 4 : 5
Enter the Priority of process 5 : 5
PID BT AT P CT WT TAT
P1 4 0 2 15 11 15
P2 3 1 3 12 8 11
P3 1 2 4 3 0 1
P4 5 3 5 8 0 5
P5 2 4 5 10 4 6
Average Waiting Time = 4.60
Average Turnaround Time = 7.60
Gantt Chart:
-----------------------------------
| P1 | P2 | P3 | P4 | P5 | P2 | P1 |
-----------------------------------
0 1 2 3 8 10 12 15
```

Non-Preemptive Priority

```
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ gcc Process_Scheduling.c -o
PS
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ ./PS
Enter the number of processes : 5
Enter 5 process info:
Enter the Pid of process 1 : 1
Enter the Arrival time of process 1 : 0
Enter the Burst time of process 1 : 4
Enter the Pid of process 2 : 2
Enter the Arrival time of process 2 : 1
Enter the Burst time of process 2 : 3
Enter the Pid of process 3 : 3
Enter the Arrival time of process 3 : 2
Enter the Burst time of process 3 : 1
Enter the Pid of process 4 : 4
Enter the Arrival time of process 4 : 3
Enter the Burst time of process 4 : 5
Enter the Pid of process 5 : 5
Enter the Arrival time of process 5 : 4
Enter the Burst time of process 5 : 2
Program Menu:
Options:
 1) FCFS Scheduling
 2) Preemptive SJF
 3) Non-Preemptive SJF
 4) Preemptive Priority
 5) Non-Preemptive Priority
 6) Round Robin
 7) Exit
Enter Your Choice: 5
Specify the priority of the processes -
Enter the Priority of process 1 : 2
Enter the Priority of process 2 : 3
Enter the Priority of process 3 : 4
Enter the Priority of process 4 : 5
Enter the Priority of process 5 : 5
PID BT AT P CT WT TAT
P1 4 0 2 4 0 4
P2 3 1 3 15 11 14
P3 1 2 4 12 9 10
P4 5 3 5 9 1 6
P5 2 4 5 11 5 7
Average Waiting Time = 5.20
Average Turnaround Time = 8.20
Gantt Chart:
-------------------------
| P1 | P4 | P5 | P3 | P2 |
-------------------------
0 4 9 11 12 15
```

Round Robin

```
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ gcc Process_Scheduling.c -o
PS
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$ ./PS
```

```
Enter the number of processes : 6
Enter 6 process info:
Enter the Pid of process 1 : 1
Enter the Arrival time of process 1 : 5
Enter the Burst time of process 1 : 5
Enter the Pid of process 2 : 2
Enter the Arrival time of process 2 : 4
Enter the Burst time of process 2 : 6
Enter the Pid of process 3 : 3
Enter the Arrival time of process 3 : 3
Enter the Burst time of process 3 : 7
Enter the Pid of process 4 : 4
Enter the Arrival time of process 4 : 1
Enter the Burst time of process 4 : 9
Enter the Pid of process 5 : 5
Enter the Arrival time of process 5 : 2
Enter the Burst time of process 5 : 2
Enter the Pid of process 6 : 6
Enter the Arrival time of process 6 : 6
Enter the Burst time of process 6 : 3
Program Menu:
Options:
 1) FCFS Scheduling
 2) Preemptive SJF
 3) Non-Preemptive SJF
 4) Preemptive Priority
 5) Non-Preemptive Priority
 6) Round Robin
 7) Exit
Enter Your Choice: 6
Enter Time Quantum: 3
PID BT AT CT WT TAT
P4 9 1 30 20 29
P5 2 2 6 2 4
P3 7 3 33 23 30
P2 6 4 27 17 23
P1 5 5 32 22 27
P6 3 6 21 12 15
Average Waiting Time = 16.00
Average Turnaround Time = 21.33
Gantt Chart:
-------------------------------------------------------------
| P4 | P5 | P3 | P2 | P4 | P1 | P6 | P3 | P2 | P4 | P1 | P3 |
-------------------------------------------------------------
1 4 6 9 12 15 18 21 24 27 30 32 33
friday@friday-VirtualBox:~/Desktop/OS/Assignment3$
```