Name: Bramha Nimbalkar

Roll no: 7

Division: E

ASSIGNMENT 7

Write a program in C to implement FCFS CPU scheduling with arrival time. (o/p-response time, turnaround time, idle time, CPU utilization time, completion time and CPU Utilization)

```c
#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[], int at[]) {
    int service_time[n];
    service_time[0] = 0;
    wt[0] = 0;

    for (int i = 1; i < n; i++) {
        service_time[i] = service_time[i - 1] + bt[i - 1];
        wt[i] = service_time[i] - at[i];
        if (wt[i] < 0)
            wt[i] = 0;
    }
}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}

void findAvgTime(int processes[], int n, int bt[], int at[]) {
    int wt[n], tat[n];

    findWaitingTime(processes, n, bt, wt, at);
    findTurnAroundTime(processes, n, bt, wt, tat);

    printf("Process  Arrival Time  Burst Time  Waiting Time  Turnaround Time\n");
    int total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, at[i], bt[i], wt[i], tat[i]);
    }

    printf("\nAverage waiting time = %.2f\n", (float)total_wt / (float)n);
    printf("Average turnaround time = %.2f\n", (float)total_tat / (float)n);
}
```

```
int main() {
    int processes[] = {1, 2, 3};
    int n = sizeof(processes) / sizeof(processes[0]);
    int burst_time[] = {7, 4, 1};
    int arrival_time[] = {0, 2, 4};

    findAvgTime(processes, n, burst_time, arrival_time);
    return 0;
}
```

Output:

Write a program in C to implement SJF CPU scheduling (non-preemptive)(o/p-response time, turnaround time , waiting time, average waiting time.)

```
#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[]) {
    wt[0] = 0;

    for (int i = 1; i < n; i++) {
        wt[i] = 0;
        for (int j = 0; j < i; j++)
            wt[i] += bt[j];
    }
}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}

void findAvgTime(int processes[], int n, int bt[]) {
    int wt[n], tat[n];

    findWaitingTime(processes, n, bt, wt);
    findTurnAroundTime(processes, n, bt, wt, tat);

    printf("Process  Burst Time  Waiting Time  Turnaround Time\n");
    int total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
```

```
        printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    }

    printf("\nAverage waiting time = %.2f\n", (float)total_wt / (float)n);
    printf("Average turnaround time = %.2f\n", (float)total_tat / (float)n);
}

int main() {
    int processes[] = {1, 2, 3, 4};
    int n = sizeof(processes) / sizeof(processes[0]);
    int burst_time[] = {6, 8, 7, 3};

    // Sort processes based on burst times in ascending order
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (burst_time[j] > burst_time[j + 1]) {
                int temp = burst_time[j];
                burst_time[j] = burst_time[j + 1];
                burst_time[j + 1] = temp;

                temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }

    findAvgTime(processes, n, burst_time);
    return 0;
}
```

Output:

```
Process  Burst Time  Waiting Time  Turnaround Time
1               3               0               3
2               6               3               9
3               7               9               16
4               8               16              24

Average waiting time = 7.00
Average turnaround time = 13.00
```

Write a program in C to implement SJF CPU scheduling (preemptive)(o/p-response time, turnaround time , waiting time, average waiting time.)

```
#include <stdio.h>

// Structure to represent a process
typedef struct Process {
```

```c
    int pid;        // Process ID
    int burst_time; // Burst time
    int arrival_time; // Arrival time
    int remaining_time; // Remaining burst time
} Process;

int findShortestJob(Process processes[], int n, int time) {
    int shortest = -1;
    int min_remaining_time = __INT_MAX__;

    for (int i = 0; i < n; i++) {
        if (processes[i].arrival_time <= time && processes[i].remaining_time <
min_remaining_time && processes[i].remaining_time > 0) {
            shortest = i;
            min_remaining_time = processes[i].remaining_time;
        }
    }

    return shortest;
}

void preemptiveSJF(Process processes[], int n) {
    int total_wt = 0; // Total waiting time
    int total_tat = 0; // Total turnaround time
    int complete = 0; // Count of completed processes
    int time = 0; // Current time

    printf("Gantt Chart: ");
    while (complete != n) {
        int shortest_job = findShortestJob(processes, n, time);

        if (shortest_job == -1) {
            printf("- idle - ");
            time++;
            continue;
        }

        processes[shortest_job].remaining_time--;

        printf("P%d ", processes[shortest_job].pid);

        if (processes[shortest_job].remaining_time == 0) {
            complete++;
            int finish_time = time + 1;
            int turnaround_time = finish_time - processes[shortest_job].arrival_time;
            int waiting_time = turnaround_time - processes[shortest_job].burst_time;

            total_wt += waiting_time;
            total_tat += turnaround_time;
        }

        time++;
    }
```

```
    printf("\nAverage waiting time = %.2f\n", (float)total_wt / n);
    printf("Average turnaround time = %.2f\n", (float)total_tat / n);
}

int main() {
    Process processes[] = {
        {1, 6, 1, 6},
        {2, 8, 1, 8},
        {3, 7, 2, 7},
        {4, 3, 3, 3}
    };

    int n = sizeof(processes) / sizeof(processes[0]);

    preemptiveSJF(processes, n);

    return 0;
}
```

Output:

Write a program in C to implement Priority CPU scheduling (preemptive)(o/p-response time, turnaround time , waiting time, average waiting time.)

```
#include <stdio.h>
#include <stdbool.h>

// Structure to represent a process
typedef struct Process {
    int pid;            // Process ID
    int burst_time;     // Burst time
    int arrival_time;   // Arrival time
    int priority;       // Priority
    int remaining_time; // Remaining burst time
} Process;

int findHighestPriority(Process processes[], int n, int time) {
    int highest = -1;
    int min_priority = __INT_MAX__;

    for (int i = 0; i < n; i++) {
        if (processes[i].arrival_time <= time && processes[i].priority < min_priority &&
processes[i].remaining_time > 0) {
            highest = i;
            min_priority = processes[i].priority;
```

```
        }
    }

    return highest;
}

void preemptivePriorityScheduling(Process processes[], int n) {
    int total_wt = 0; // Total waiting time
    int total_tat = 0; // Total turnaround time
    int complete = 0; // Count of completed processes
    int time = 0; // Current time

    printf("Gantt Chart: ");
    while (complete != n) {
        int highest_priority = findHighestPriority(processes, n, time);

        if (highest_priority == -1) {
            printf("- idle - ");
            time++;
            continue;
        }

        processes[highest_priority].remaining_time--;

        printf("P%d ", processes[highest_priority].pid);

        if (processes[highest_priority].remaining_time == 0) {
            complete++;
            int finish_time = time + 1;
            int turnaround_time = finish_time - processes[highest_priority].arrival_time;
            int waiting_time = turnaround_time - processes[highest_priority].burst_time;

            total_wt += waiting_time;
            total_tat += turnaround_time;
        }

        time++;
    }

    printf("\nAverage waiting time = %.2f\n", (float)total_wt / n);
    printf("Average turnaround time = %.2f\n", (float)total_tat / n);
}

int main() {
    Process processes[] = {
        {1, 6, 1, 2, 6},
        {2, 8, 1, 1, 8},
        {3, 7, 2, 4, 7},
        {4, 3, 3, 3, 3}
    };

    int n = sizeof(processes) / sizeof(processes[0]);

    preemptivePriorityScheduling(processes, n);
```

```
    return 0;
}
```

Output:

Write a program in C to implement Round Robin CPU scheduling(o/p-response time, turnaround time , waiting time, average waiting time.)

```c
#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum) {
    int remaining_time[n];
    for (int i = 0; i < n; i++)
        remaining_time[i] = bt[i];

    int t = 0;
    while (1) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (remaining_time[i] > 0) {
                done = 0;
                if (remaining_time[i] > quantum) {
                    t += quantum;
                    remaining_time[i] -= quantum;
                } else {
                    t = t + remaining_time[i];
                    wt[i] = t - bt[i];
                    remaining_time[i] = 0;
                }
            }
        }
        if (done == 1)
            break;
    }
}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
        tat[i] = bt[i] + wt[i];
}

void findAvgTime(int processes[], int n, int bt[], int quantum) {
    int wt[n], tat[n];
    findWaitingTime(processes, n, bt, wt, quantum);
```

```
    findTurnAroundTime(processes, n, bt, wt, tat);

    printf("Process  Burst Time  Waiting Time  Turnaround Time\n");
    int total_wt = 0, total_tat = 0;
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
        printf("%d\t\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    }

    printf("\nAverage waiting time = %.2f\n", (float)total_wt / (float)n);
    printf("Average turnaround time = %.2f\n", (float)total_tat / (float)n);
}

int main() {
    int processes[] = {1, 2, 3};
    int n = sizeof(processes) / sizeof(processes[0]);
    int burst_time[] = {7, 4, 1};
    int quantum = 2;

    findAvgTime(processes, n, burst_time, quantum);
    return 0;
}
```

Output:

```
Process  Burst Time  Waiting Time  Turnaround Time
1               7               5               12
2               4               5               9
3               1               4               5

Average waiting time = 4.67
Average turnaround time = 8.67
```