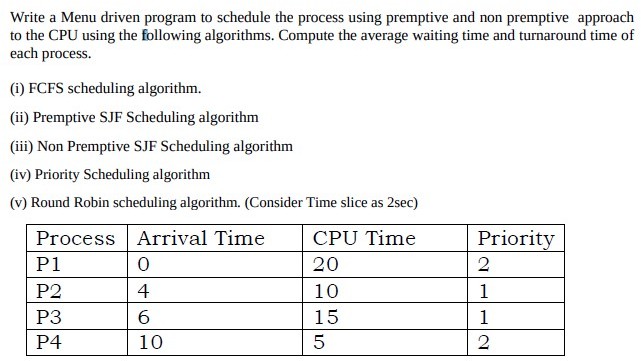
**Operating Systems Lab DA2**

Name:Manas Laud

Reg no:22BKT0060

Q1:



**Code:**

#include <stdio.h> #include <stdlib.h> #include <limits.h>

#define MAX\_PROCESSES 100

struct Process { int id;

int arrival\_time; int burst\_time;

int priority;

int remaining\_time; int completion\_time; int waiting\_time;

int turnaround\_time;

};

void calculate\_times(struct Process processes[], int n) { for (int i = 0; i < n; i++) {

processes[i].turnaround\_time = processes[i].completion\_time - processes[i].arrival\_time;

processes[i].waiting\_time = processes[i].turnaround\_time - processes[i].burst\_time;

}

}

void print\_results(struct Process processes[], int n) { float avg\_waiting\_time = 0, avg\_turnaround\_time = 0;

printf("\nProcess\tArrival Time\tBurst Time\tPriority\tCompletion Time\tWaiting Time\ tTurnaround Time\n");

for (int i = 0; i < n; i++) { printf("P%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",

processes[i].id, processes[i].arrival\_time, processes[i].burst\_time, processes[i].priority, processes[i].completion\_time, processes[i].waiting\_time, processes[i].turnaround\_time);

avg\_waiting\_time += processes[i].waiting\_time; avg\_turnaround\_time += processes[i].turnaround\_time;

}

avg\_waiting\_time /= n; avg\_turnaround\_time /= n;

printf("\nAverage Waiting Time: %.2f\n", avg\_waiting\_time); printf("Aierage Turnaround Time: %.2f\n", avg\_turnaround\_time);

}

void fcfs(struct Process processes[], int n) { int current\_time = 0;

for (int i = 0; i < n; i++) {

if (current\_time < processes[i].arrival\_time) current\_time = processes[i].arrival\_time;

processes[i].completion\_time = current\_time + processes[i].burst\_time; current\_time = processes[i].completion\_time;

}

calculate\_times(processes, n); print\_results(processes, n);

}

void sjf\_preemptive(struct Process processes[], int n) { struct Process temp[MAX\_PROCESSES];

int current\_time = 0, completed = 0;

for (int i = 0; i < n; i++) { temp[i] = processes[i];

temp[i].remaining\_time = processes[i].burst\_time;

}

while (completed != n) {

int shortest = -1, min\_burst = INT\_MAX;

for (int i = 0; i < n; i++) {

if (temp[i].arrival\_time <= current\_time && temp[i].remaining\_time < min\_burst && temp[i].remaining\_time > 0) {

shortest = i;

min\_burst = temp[i].remaining\_time;

}

}

if (shortest == -1) { current\_time++;

} else {

temp[shortest].remaining\_time--; current\_time++;

if (temp[shortest].remaining\_time == 0) { completed++;

temp[shortest].completion\_time = current\_time;

}

}

}

calculate\_times(temp, n); print\_results(temp, n);

}

void sjf\_non\_preemptive(struct Process processes[], int n) { struct Process temp[MAX\_PROCESSES];

int current\_time = 0, completed = 0;

for (int i = 0; i < n; i++) temp[i] = processes[i];

while (completed != n) {

int shortest = -1, min\_burst = INT\_MAX; for (int i = 0; i < n; i++) {

if (temp[i].arrival\_time <= current\_time && temp[i].burst\_time < min\_burst && temp[i].remaining\_time > 0) {

shortest = i;

min\_burst = temp[i].burst\_time;

}

}

if (shortest == -1) { current\_time++;

} else {

temp[shortest].completion\_time = current\_time + temp[shortest].burst\_time; current\_time = temp[shortest].completion\_time; temp[shortest].remaining\_time = 0;

completed++;

}

}

calculate\_times(temp, n); print\_results(temp, n);

}

void priority\_scheduling(struct Process processes[], int n) { struct Process temp[MAX\_PROCESSES];

int current\_time = 0, completed = 0;

for (int i = 0; i < n; i++) temp[i] = processes[i];

while (completed != n) {

int highest\_priority = -1, min\_priority = INT\_MAX; for (int i = 0; i < n; i++) {

if (temp[i].arrival\_time <= current\_time && temp[i].priority < min\_priority && temp[i].remaining\_time > 0) {

highest\_priority = i; min\_priority = temp[i].priority;

}

}

if (highest\_priority == -1) { current\_time++;

} else {

temp[highest\_priority].completion\_time = current\_time + temp[highest\_priority].burst\_time;

current\_time = temp[highest\_priority].completion\_time; temp[highest\_priority].remaining\_time = 0; completed++;

}

}

calculate\_times(temp, n); print\_results(temp, n);

}

void round\_robin(struct Process processes[], int n, int time\_slice) { struct Process temp[MAX\_PROCESSES];

int current\_time = 0, completed = 0;

int queue[MAX\_PROCESSES], front = 0, rear = 0;

for (int i = 0; i < n; i++) { temp[i] = processes[i];

temp[i].remaining\_time = processes[i].burst\_time;

}

while (completed != n) {

for (int i = 0; i < n; i++) {

if (temp[i].arrival\_time == current\_time) { queue[rear] = i;

rear = (rear + 1) % MAX\_PROCESSES;

}

}

if (front == rear) { current\_time++; continue;

}

int index = queue[front];

front = (front + 1) % MAX\_PROCESSES;

if (temp[index].remaining\_time <= time\_slice) { current\_time += temp[index].remaining\_time; temp[index].completion\_time = current\_time; temp[index].remaining\_time = 0; completed++;

} else {

current\_time += time\_slice; temp[index].remaining\_time -= time\_slice; queue[rear] = index;

rear = (rear + 1) % MAX\_PROCESSES;

}

}

calculate\_times(temp, n); print\_results(temp, n);

}

int main() {

struct Process processes[] = {

{1, 0, 20, 2, 20},

{2, 4, 10, 1, 10},

{3, 6, 15, 1, 15},

{4, 10, 5, 2, 5}

};

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

int choice; do {

printf("\nCPU Scheduling Algorithms:\n"); printf("1. FCFS Scheduling\n");

printf("2. Preemptive SJF Scheduling\n"); printf("3. Non-Preemptive SJF Scheduling\n"); printf("4. Priority Scheduling\n");

printf("5. Round Robin Schedling\n"); printf("6. Exit\n");

printf("Enter your choice (processes-6): "); scanf("%d", &choice);

switch (choice){ case 1:

printf("\nFCFS Scheduling:\n"); fcfs(processes, n);

break; case 2:

printf("\nPreemptive SJF Scheduling:\n"); sjf\_preemptive(processes, n);

break; case 3:

printf("\nNon-Preemptive SJF Scheduling:\n"); sjf\_non\_preemptive(processes, n);

break; case 4:

printf("\nPriority Scheduling:\n"); priority\_scheduling(processes, n); break;

case 5:

printf("\nRound Robin Scheduling (Time Slice = 2):\n"); round\_robin(processes, n, 2);

break; case 6:

printf("Exiting program…\n"); break;

default:

printf("Invalid choice. Please try again.\n");

}

}while (choice != 6); return 0;

}

Output:

