**Musa Haji  
230701391  
Ex No. 6a:  
 First Come First Serve (FCFS) Scheduling**

Aim:

To implement the First-come First-serve (FCFS) scheduling technique.

Algorithm:

Get the number of processes from the user.

Read the process name and burst time for each process.

Calculate the total process time (sum of all burst times).

Calculate the total waiting time and total turnaround time for each process:

Waiting time for the first process is 0.

For subsequent processes, the waiting time is the sum of the burst times of all the previous processes.

Turnaround time is the sum of waiting time and burst time for each process.

Display the process name & burst time for each process.

Display the total waiting time, average waiting time, and turnaround time.

Program Code:

#include <stdio.h>

int main() {

int n, i;

float total\_wt = 0, total\_tat = 0;

// Get the number of processes

printf("Enter the number of processes: ");

scanf("%d", &n);

int burst\_time[n], waiting\_time[n], turnaround\_time[n];

// Get the burst time for each process

printf("Enter the burst time of the processes: ");

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

scanf("%d", &burst\_time[i]);

}

// Calculate waiting time for each process

waiting\_time[0] = 0;

for (i = 1; i < n; i++) {

waiting\_time[i] = burst\_time[i - 1] + waiting\_time[i - 1];

}

// Calculate turnaround time for each process

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

turnaround\_time[i] = burst\_time[i] + waiting\_time[i];

}

// Display the process details

printf("\nProcess Burst Time Waiting Time Turn Around Time\n");

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

printf("%d\t\t%d\t\t%d\t\t%d\n", i, burst\_time[i], waiting\_time[i], turnaround\_time[i]);

}

// Calculate total waiting time and total turnaround time

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

total\_wt += waiting\_time[i];

total\_tat += turnaround\_time[i];

}

// Calculate average waiting time and average turnaround time

float avg\_wt = total\_wt / n;

float avg\_tat = total\_tat / n;

// Display average waiting time and average turnaround time

printf("\nAverage waiting time is: %.2f", avg\_wt);

printf("\nAverage Turnaround Time is: %.2f\n", avg\_tat);

return 0;

}

Sample Output:

Enter the number of processes:

3

Enter the burst time of the processes:

24 3 3

Process Burst Time Waiting Time Turn Around Time

0 24 0 24

1 3 24 27

2 3 27 30

Average waiting time is: 17.00

Average Turnaround Time is: 19.00

Result:

The program implements the FCFS scheduling technique by calculating the waiting time and turnaround time for each process based on their burst times. The output shows the burst time, waiting time, and turnaround time for each process, along with the average waiting time and average turnaround time, which confirms the correctness of the scheduling algorithm.  
  
**Ex No. 6b:**

**Shortest Job First (SJF) Scheduling**

Aim:

To implement the Shortest Job First (SJF) scheduling technique.

Algorithm:

Declare a structure to hold process details (process name, burst time, arrival time, waiting time, and turnaround time).

Get the number of processes from the user.

Read the process name, arrival time, and burst time for each process.

Initialize waiting time, turnaround time, and a flag for all processes to zero.

Sort processes based on burst time in ascending order.

Calculate the waiting time and turnaround time for each process:

Waiting time for the first process is 0.

For subsequent processes, the waiting time is the sum of burst times of all previous processes.

Turnaround time is the sum of waiting time and burst time for each process.

Calculate the average waiting time and average turnaround time.

Display the results: process name, burst time, waiting time, and turnaround time for each process.

Program Code:

#include <stdio.h>

struct Process {

int name;

int burst\_time;

int waiting\_time;

int turnaround\_time;

};

void sortByBurstTime(struct Process processes[], int n) {

struct Process temp;

for (int i = 0; i < n - 1; i++) {

for (int j = i + 1; j < n; j++) {

if (processes[i].burst\_time > processes[j].burst\_time) {

temp = processes[i];

processes[i] = processes[j];

processes[j] = temp;

}

}

}

}

int main() {

int n;

float total\_wt = 0, total\_tat = 0;

// Get the number of processes

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process processes[n];

// Get process details: burst time for each process

printf("Enter the burst time of the processes: ");

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

processes[i].name = i + 1; // Process name (ID)

scanf("%d", &processes[i].burst\_time);

processes[i].waiting\_time = 0;

processes[i].turnaround\_time = 0;

}

// Sort processes based on burst time

sortByBurstTime(processes, n);

// Calculate waiting time and turnaround time for each process

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

if (i == 0) {

processes[i].waiting\_time = 0;

} else {

processes[i].waiting\_time = processes[i - 1].waiting\_time + processes[i - 1].burst\_time;

}

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

}

// Display process details

printf("\nProcess Burst Time Waiting Time Turn Around Time\n");

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

printf("%d\t\t%d\t\t%d\t\t%d\n", processes[i].name, processes[i].burst\_time, processes[i].waiting\_time, processes[i].turnaround\_time);

}

// Calculate total waiting time and total turnaround time

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

total\_wt += processes[i].waiting\_time;

total\_tat += processes[i].turnaround\_time;

}

// Calculate average waiting time and average turnaround time

float avg\_wt = total\_wt / n;

float avg\_tat = total\_tat / n;

// Display average waiting time and average turnaround time

printf("\nAverage waiting time is: %.2f", avg\_wt);

printf("\nAverage Turnaround Time is: %.2f\n", avg\_tat);

return 0;

}

Sample Output:

Enter the number of processes:

4

Enter the burst time of the processes:

8 4 9 5

Process Burst Time Waiting Time Turn Around Time

2 4 0 4

4 5 4 9

1 8 9 17

3 9 17 26

Average waiting time is: 7.50

Average Turnaround Time is: 13.00

Result:

The program successfully implements the Shortest Job First (SJF) scheduling algorithm, which schedules processes in increasing order of their burst times. The output shows the burst time, waiting time, and turnaround time for each process, along with the average waiting time and average turnaround time, confirming the correct implementation of the SJF scheduling technique.  
  
**Ex. No.: 6c**

**Priority Scheduling**

Aim:

To implement the Priority Scheduling technique.

Algorithm:

Get the number of processes from the user.

Read the process name, burst time, and priority for each process.

Sort the processes based on priority (ascending order or descending depending on the requirement).

Calculate the waiting time and turnaround time for each process:

The waiting time for the first process is 0.

For subsequent processes, the waiting time is the sum of burst times of all previous processes.

Turnaround time is the sum of waiting time and burst time for each process.

Display the process name and burst time for each process.

Display the total waiting time, average waiting time, and turnaround time.

Program Code:

#include <stdio.h>

struct Process {

int name;

int burst\_time;

int priority;

int waiting\_time;

int turnaround\_time;

};

void sortByPriority(struct Process processes[], int n) {

struct Process temp;

for (int i = 0; i < n - 1; i++) {

for (int j = i + 1; j < n; j++) {

if (processes[i].priority > processes[j].priority) {

temp = processes[i];

processes[i] = processes[j];

processes[j] = temp;

}

}

}

}

int main() {

int n;

float total\_wt = 0, total\_tat = 0;

// Get the number of processes

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process processes[n];

// Get process details: burst time and priority for each process

printf("Enter the burst time and priority of the processes: \n");

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

processes[i].name = i + 1; // Process name (ID)

printf("Enter burst time and priority for process %d: ", i + 1);

scanf("%d %d", &processes[i].burst\_time, &processes[i].priority);

processes[i].waiting\_time = 0;

processes[i].turnaround\_time = 0;

}

// Sort processes based on priority

sortByPriority(processes, n);

// Calculate waiting time and turnaround time for each process

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

if (i == 0) {

processes[i].waiting\_time = 0;

} else {

processes[i].waiting\_time = processes[i - 1].waiting\_time + processes[i - 1].burst\_time;

}

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

}

// Display process details

printf("\nProcess Burst Time Priority Waiting Time Turn Around Time\n");

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

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].name, processes[i].burst\_time, processes[i].priority, processes[i].waiting\_time, processes[i].turnaround\_time);

}

// Calculate total waiting time and total turnaround time

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

total\_wt += processes[i].waiting\_time;

total\_tat += processes[i].turnaround\_time;

}

// Calculate average waiting time and average turnaround time

float avg\_wt = total\_wt / n;

float avg\_tat = total\_tat / n;

// Display average waiting time and average turnaround time

printf("\nAverage waiting time is: %.2f", avg\_wt);

printf("\nAverage Turnaround Time is: %.2f\n", avg\_tat);

return 0;

}

Sample Output:

Enter the number of processes:

4

Enter the burst time and priority of the processes:

Enter burst time and priority for process 1: 6 3

Enter burst time and priority for process 2: 8 1

Enter burst time and priority for process 3: 5 4

Enter burst time and priority for process 4: 7 2

Process Burst Time Priority Waiting Time Turn Around Time

2 8 1 0 8

4 7 2 8 15

1 6 3 15 21

3 5 4 21 26

Average waiting time is: 11.00

Average Turnaround Time is: 17.50

Result:

The program successfully implements the Priority Scheduling technique, where processes are scheduled based on their priority. The output includes the burst time, priority, waiting time, and turnaround time for each process, along with the average waiting time and average turnaround time.  
  
**Ex No.6d:  
 Round Robin Technique**

Aim:

To implement the Round Robin (RR) scheduling technique.

Algorithm:

Declare the structure and its elements.

Get the number of processes and Time Quantum as input from the user.

Read the process name, arrival time, and burst time.

Create an array rem\_bt[] to keep track of the remaining burst time of processes, which is initially a copy of bt[] (burst times array).

Create another array wt[] to store waiting times of processes. Initialize this array as 0.

Initialize time: t = 0.

Keep traversing all processes while all processes are not done. Do the following for each process if it is not done yet:

If rem\_bt[i] > quantum

t = t + quantum

rem\_bt[i] -= quantum

Else (last cycle for this process)

t = t + rem\_bt[i]

wt[i] = t - bt[i]

rem\_bt[i] = 0 // This process is over

Calculate the waiting time and turnaround time for each process.

Calculate the average waiting time and average turnaround time.

Display the results.

Program Code:

#include <stdio.h>

struct Process {

int processID;

int bt; // Burst Time

int at; // Arrival Time

int wt; // Waiting Time

int tat; // Turnaround Time

int rem\_bt; // Remaining Burst Time

};

void roundRobin(struct Process proc[], int n, int quantum) {

int t = 0; // Current time

int done = 0; // Flag to check if all processes are done

int i;

while (done < n) {

done = 0;

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

if (proc[i].rem\_bt > 0) {

done++;

if (proc[i].rem\_bt > quantum) {

t += quantum;

proc[i].rem\_bt -= quantum;

} else {

t += proc[i].rem\_bt;

proc[i].wt = t - proc[i].bt;

proc[i].rem\_bt = 0;

}

}

}

}

}

void calculateTurnaroundTime(struct Process proc[], int n) {

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

proc[i].tat = proc[i].bt + proc[i].wt;

}

}

void calculateAverageTimes(struct Process proc[], int n) {

int total\_wt = 0, total\_tat = 0;

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

total\_wt += proc[i].wt;

total\_tat += proc[i].tat;

}

printf("Average Waiting Time: %.2f\n", (float)total\_wt / n);

printf("Average Turnaround Time: %.2f\n", (float)total\_tat / n);

}

int main() {

int n, quantum;

printf("Enter number of processes: ");

scanf("%d", &n);

struct Process proc[n];

printf("Enter time quantum: ");

scanf("%d", &quantum);

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

proc[i].processID = i + 1;

printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);

scanf("%d %d", &proc[i].at, &proc[i].bt);

proc[i].rem\_bt = proc[i].bt;

proc[i].wt = 0;

}

roundRobin(proc, n, quantum);

calculateTurnaroundTime(proc, n);

calculateAverageTimes(proc, n);

printf("\nProcess ID Arrival Time Burst Time Waiting Time Turnaround Time\n");

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

printf("%d %d %d %d %d\n",

proc[i].processID, proc[i].at, proc[i].bt, proc[i].wt, proc[i].tat);

}

return 0;

}

Sample Output:

Enter number of processes: 3

Enter time quantum: 4

Enter Arrival Time and Burst Time for Process 1: 0 5

Enter Arrival Time and Burst Time for Process 2: 1 3

Enter Arrival Time and Burst Time for Process 3: 2 6

Average Waiting Time: 3.33

Average Turnaround Time: 6.00

Process ID Arrival Time Burst Time Waiting Time Turnaround Time

1 0 5 4 9

2 1 3 4 7

3 2 6 2 8

Result:

The program successfully implements the Round Robin scheduling algorithm, calculates waiting and turnaround times for each process, and computes the average waiting and turnaround times.