THENMOZHI R **231901064**

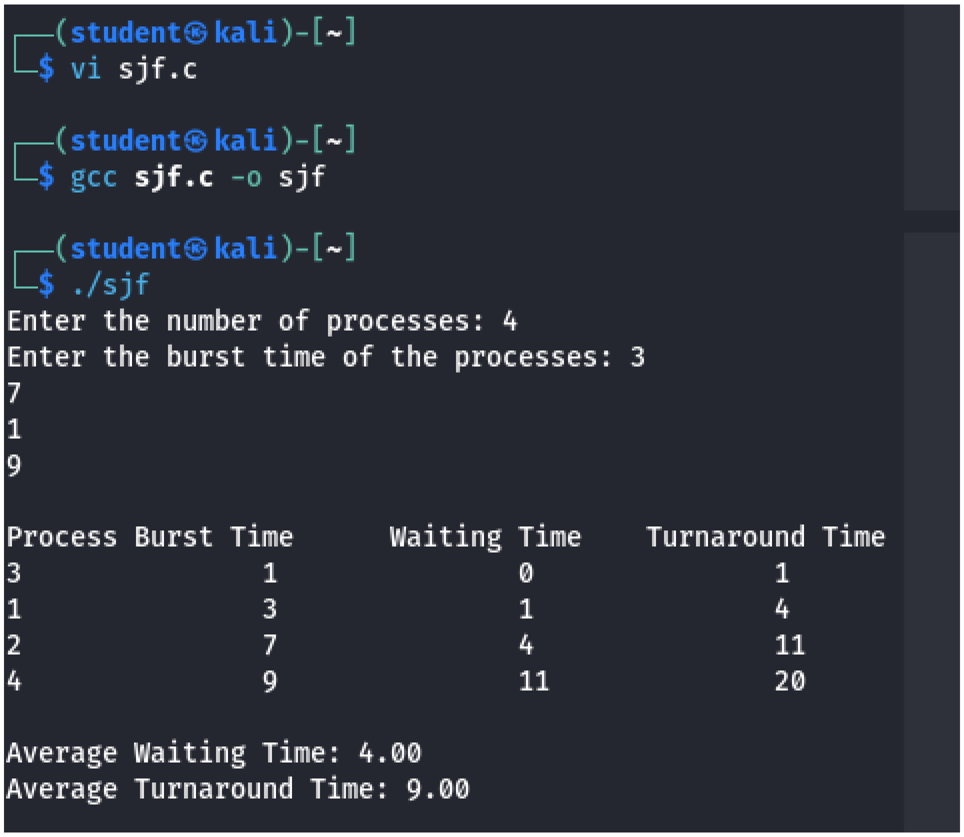
EX.NO:6B
DATE:05.03.2024

SHORTEST JOB FIRST

Aim: To implement the Shortest Job First (SJF) scheduling technique

Algorithm:
1. Declare the structure and its elements.
2. Get a number of processes as input from the user.
3. Read the process name, arrival time and burst time
4. Initialize waiting time, turnaround time & flag of read processes to zero.
5.Sort based on burst time of all processes in ascending order
6. Calculate the waiting time and turnaround time for each process.
7. Calculate the average waiting time and average turnaround time.
8. Display the results.

Program Code:
NON - PREEMPTIVE:
```
#include <stdio.h>
int main() {
int n;
// Step 1: Get the number of processes
printf("Enter the number of processes: ");
scanf("%d", &n);
int burst_time[n], waiting_time[n], turnaround_time[n], process_order[n];
// Step 2: Read the burst time for each process
printf("Enter the burst time of the processes: ");
for (int i = 0; i < n; i++) {
scanf("%d", &burst_time[i]);
process_order[i] = i + 1; // Store the process number for display
}
// Step 3: Sort burst time in ascending order (SJF algorithm)
for (int i = 0; i < n - 1; i++) {
for (int j = i + 1; j < n; j++) {
if (burst_time[i] > burst_time[j]) {
// Swap burst times
int temp = burst_time[i];
burst_time[i] = burst_time[j];
burst_time[j] = temp;
// Swap process order to maintain correct process sequence
temp = process_order[i];
process_order[i] = process_order[j];
process_order[j] = temp;
}
}
}
// Initialize waiting time and turnaround time
waiting_time[0] = 0;
turnaround_time[0] = burst_time[0];
// Step 4: Calculate waiting time and turnaround time for each process
int total_waiting_time = 0;
int total_turnaround_time = 0;
// Calculate waiting time for each process
for (int i = 1; i < n; i++) {
waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];
}
// Calculate turnaround time for each process
for (int i = 0; i < n; i++) {
turnaround_time[i] = burst_time[i] + waiting_time[i];
}
// Step 5: Display the results
printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
for (int i = 0; i < n; i++) {
printf("%d\t\t%d\t\t%d\t\t%d\n", process_order[i], burst_time[i], waiting_time[i],
turnaround_time[i]);
total_waiting_time += waiting_time[i];
total_turnaround_time += turnaround_time[i];
}
// Step 6: Calculate and display average waiting time and turnaround time
float avg_waiting_time = (float)total_waiting_time / n;
float avg_turnaround_time = (float)total_turnaround_time / n;
printf("\nAverage Waiting Time: %.2f\n", avg_waiting_time);
printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);
return 0;
}
```



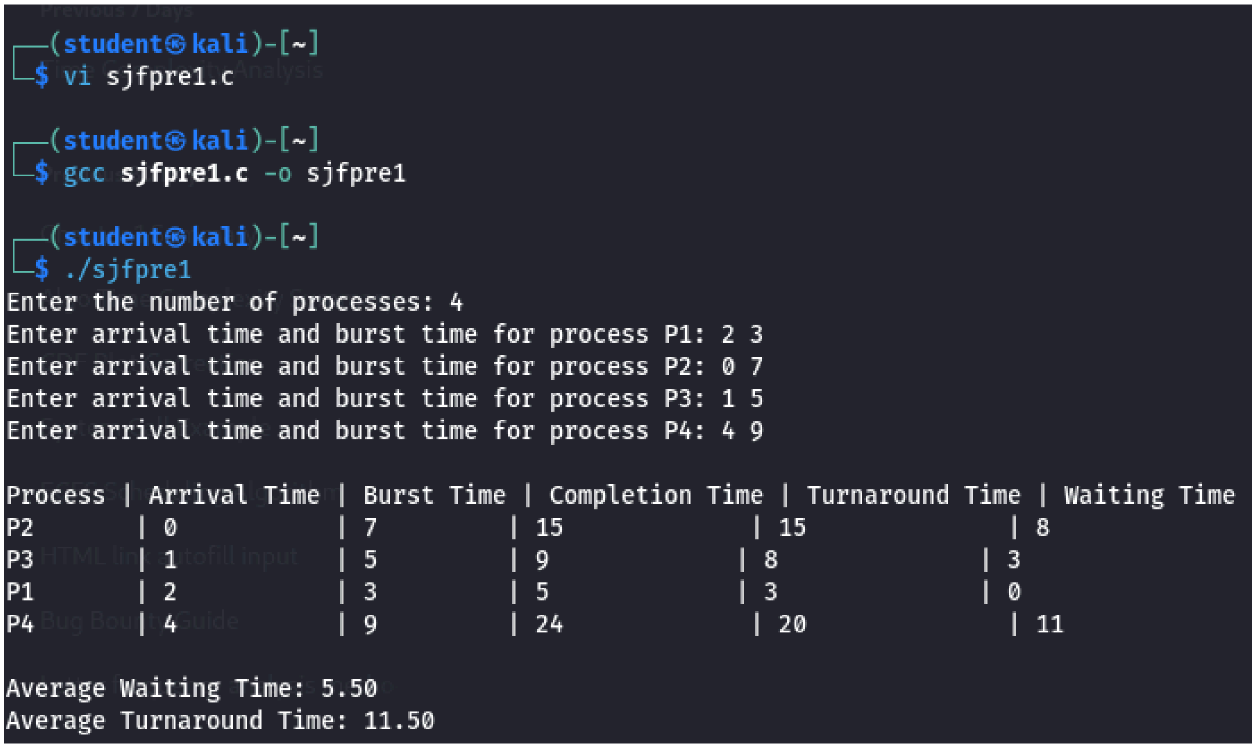
```
┌──(student㉿kali)-[~]
└─$ vi sjf.c

┌──(student㉿kali)-[~]
└─$ gcc sjf.c -o sjf

┌──(student㉿kali)-[~]
└─$ ./sjf
Enter the number of processes: 4
Enter the burst time of the processes: 3
7
1
9

Process Burst Time      Waiting Time    Turnaround Time
3               1               0               1
1               3               1               4
2               7               4               11
4               9               11              20

Average Waiting Time: 4.00
Average Turnaround Time: 9.00
```

PREEMPTIVE:

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

struct Process {
        int pid;            // Process ID
        int arrival_time;   // Arrival time
        int burst_time;           // Burst time (CPU time)
        int remaining_time; // Remaining time to complete
        int completion_time;
        int turnaround_time;
        int waiting_time;
};

void sjf_preemptive(struct Process processes[], int n) {
        int time = 0, completed = 0;
        int current_process = -1;
        int min_burst_time, idx;

        // Sort processes by arrival time
        for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
        if (processes[i].arrival_time > processes[j].arrival_time) {
                struct Process temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
        }
        }
        }

        while (completed < n) {
        min_burst_time = 99999;
        idx = -1;

        // Find the process with the smallest remaining time that has arrived
        for (int i = 0; i < n; i++) {
        if (processes[i].arrival_time <= time && processes[i].remaining_time > 0 &&
processes[i].remaining_time < min_burst_time) {
                min_burst_time = processes[i].remaining_time;
                idx = i;
        }
        }

        if (idx != -1) {
        processes[idx].remaining_time--;
        time++;

        if (processes[idx].remaining_time == 0) {
                processes[idx].completion_time = time;
                processes[idx].turnaround_time = processes[idx].completion_time -
processes[idx].arrival_time;
                processes[idx].waiting_time = processes[idx].turnaround_time -
processes[idx].burst_time;
                completed++;
        }
        } else {
        time++;
        }
        }
```

```
	// Calculate average waiting time and turnaround time
	int total_waiting_time = 0;
	int total_turnaround_time = 0;

	// Print results
	printf("\nProcess | Arrival Time | Burst Time | Completion Time | Turnaround Time |
Waiting Time\n");
	for (int i = 0; i < n; i++) {
	total_waiting_time += processes[i].waiting_time;
	total_turnaround_time += processes[i].turnaround_time;
	printf("P%d    | %d            | %d            | %d            | %d            | %d\n",
	processes[i].pid, processes[i].arrival_time, processes[i].burst_time,
	processes[i].completion_time, processes[i].turnaround_time, processes[i].waiting_time);
	}

	// Calculate and print averages
	double avg_waiting_time = (double) total_waiting_time / n;
	double avg_turnaround_time = (double) total_turnaround_time / n;

	printf("\nAverage Waiting Time: %.2f\n", avg_waiting_time);
	printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);
}

int main() {
	int n;
	printf("Enter the number of processes: ");
	scanf("%d", &n);

	struct Process processes[n];

	for (int i = 0; i < n; i++) {
	processes[i].pid = i + 1; // Assign process ID
	printf("Enter arrival time and burst time for process P%d: ", i + 1);
	scanf("%d %d", &processes[i].arrival_time, &processes[i].burst_time);
	processes[i].remaining_time = processes[i].burst_time; // Initialize remaining time
	}

	sjf_preemptive(processes, n);

	return 0;
```

}



```
┌──(student㉿kali)-[~]
└─$ vi sjfpre1.c

┌──(student㉿kali)-[~]
└─$ gcc sjfpre1.c -o sjfpre1

┌──(student㉿kali)-[~]
└─$ ./sjfpre1
Enter the number of processes: 4
Enter arrival time and burst time for process P1: 2 3
Enter arrival time and burst time for process P2: 0 7
Enter arrival time and burst time for process P3: 1 5
Enter arrival time and burst time for process P4: 4 9

Process | Arrival Time | Burst Time | Completion Time | Turnaround Time | Waiting Time
P2      | 0            | 7          | 15               | 15               | 8
P3      | 1            | 5          | 9               | 8               | 3
P1      | 2            | 3          | 5               | 3               | 0
P4      | 4            | 9          | 24               | 20               | 11

Average Waiting Time: 5.50
Average Turnaround Time: 11.50
```

Result: Hence, average waiting time and average turnaround time has been calculated using SJF Algorithm in both preemptive and non preemptive technique.