**VAISHNAVI E 231901059**

**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.

1. Calculate the average waiting time and average turnaround time.
2. 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];

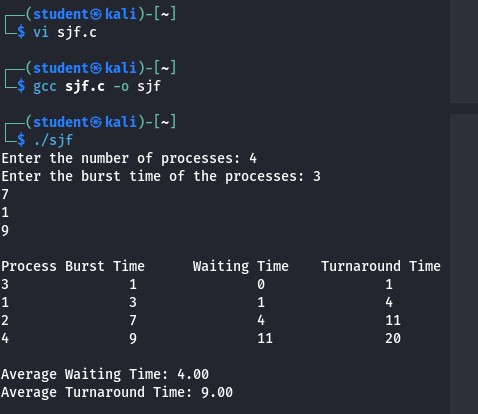
}

// 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;

}



**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];

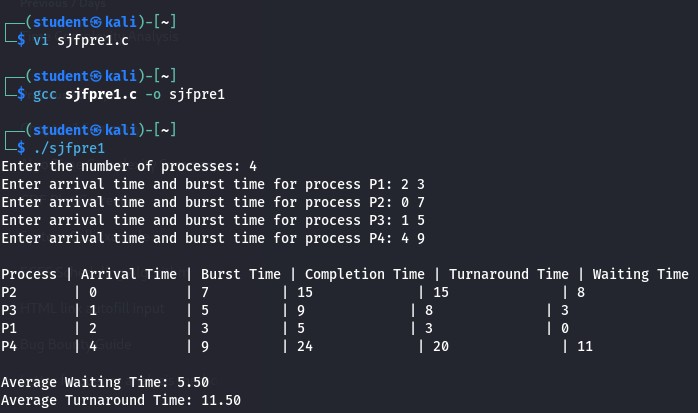
for (int i = 0; i < n; i++) { processes[i].pid = i + 1; // Assign process ID printf("Enter arrival time and burst time for process P%d: ", i + 1); scanf("%d %d", &processes[i].arrival\_time, &processes[i].burst\_time); processes[i].remaining\_time = processes[i].burst\_time; // Initialize remaining time

}

sjf\_preemptive(processes, n);

return 0;

}



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