Micro-Project: Create report on SJF algorithm. Give c program for SJF Turn on Screen reader support

1)Intoduction on SJF Alogorithm

The Shortest Jof First (SJF) Algorithm is a CPU scheduling algorithm that assigns the CPU to the process with the shortest burst time (i.e , the least amount of execution time). It is one of the most optimal scheduling algorithms in theory because it minimizes the average waiting time for processes.

There are two types of SJF scheduling:

1. **Non-preemptive SJF**: Once a process starts executing, it cannot be interrupted until it finishes. The system always selects the process with the least burst time from the available processes.
2. **Preemptive SJF (Shortest Remaining Time First - SRTF)**: In this version, the CPU is assigned to the process with the shortest remaining time. If a new process arrives with a shorter remaining time, the current process is preempted .

.Advantages:

. Minimizes average waiting time : As shorter processes are completed first, the overall average waiting time to decreases.

. Efficient for batch processing: Ideal for environments where short jobs need priority.

.Disadvantages:

. Starvation: Long processes might starve if there are frequent short processes arriving.

.Use Case

. Suitable for systems where task durations are known such as certain types of batch jobs or time – based simulations.

include <stdio.h>

void sortProcesses(int n, int burst\_time[], int process\_id[]) {

int i, j, temp;

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

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

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

temp = burst\_time[i];

burst\_time[i] = burst\_time[j];

burst\_time[j] = temp;

temp = process\_id[i];

process\_id[i] = process\_id[j];

process\_id[j] = temp;

}

}

}

}

int main() {

int n, i;

printf("Enter number of processes: ");

scanf("%d", &n);

int burst\_time[n], process\_id[n], wait\_time[n], turn\_around\_time[n];

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

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

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

process\_id[i] = i + 1;

}

// Sort processes by burst time

sortProcesses(n, burst\_time, process\_id);

wait\_time[0] = 0;

turn\_around\_time[0] = burst\_time[0];

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

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

turn\_around\_time[i] = wait\_time[i] + burst\_time[i];

}

// Display results

printf("\nProcess ID\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

printf("%d\t\t%d\t\t%d\t\t%d\n", process\_id[i], burst\_time[i], wait\_time[i], turn\_around\_time[i]);

}

return 0;

}

Output:-

