1) SJF(non-preemptive)

#include <stdio.h>

#include <stdlib.h>

// Process structure

struct Process {

int pid; // Process ID

int burst\_time; // Burst time

int arrival\_time; // Arrival time

};

// Function to perform SJF scheduling (non-preemptive)

void sjf\_non\_preemptive(struct Process \*processes, int n) {

int total\_waiting\_time = 0, total\_turnaround\_time = 0;

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

// Sort processes based on burst time

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

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

if (processes[j].burst\_time > processes[j + 1].burst\_time) {

struct Process temp = processes[j];

processes[j] = processes[j + 1];

processes[j + 1] = temp;

}

}

}

completion\_time[0] = processes[0].burst\_time;

turnaround\_time[0] = completion\_time[0] - processes[0].arrival\_time;

waiting\_time[0] = turnaround\_time[0] - processes[0].burst\_time;

total\_waiting\_time += waiting\_time[0];

total\_turnaround\_time += turnaround\_time[0];

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

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

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

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

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

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

}

// Print result

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

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].burst\_time,

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

}

// Print average waiting time and average turnaround time

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

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

}

int main() {

int n;

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

scanf("%d", &n);

struct Process processes[n];

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

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

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

processes[i].pid = i + 1;

}

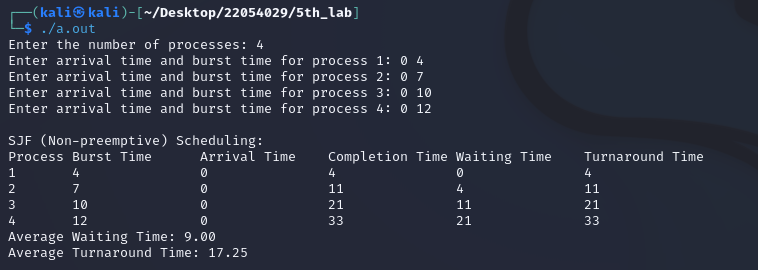
printf("\nSJF (Non-preemptive) Scheduling:\n");

sjf\_non\_preemptive(processes, n);

return 0;

}

Output



2) SJF (preemptive)

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

// Process structure

struct Process {

int pid; // Process ID

int burst\_time; // Burst time

int remaining\_time; // Remaining burst time

int arrival\_time; // Arrival time

};

// Function to perform Preemptive SJF scheduling

void sjf\_preemptive(struct Process \*processes, int n) {

int current\_time = 0, completed = 0;

int total\_waiting\_time = 0, total\_turnaround\_time = 0;

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

// Initialize remaining time and waiting time

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

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

waiting\_time[i] = 0;

turnaround\_time[i] = 0;

}

// Schedule processes until all are completed

while (completed != n) {

int min\_remaining\_time = INT\_MAX;

int min\_remaining\_time\_index = -1;

// Find process with minimum remaining time

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

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

min\_remaining\_time = processes[i].remaining\_time;

min\_remaining\_time\_index = i;

}

}

// If no process is found, increment current time

if (min\_remaining\_time\_index == -1) {

current\_time++;

continue;

}

// Execute the process for 1 unit of time

processes[min\_remaining\_time\_index].remaining\_time--;

current\_time++;

// If a process is completed

if (processes[min\_remaining\_time\_index].remaining\_time == 0) {

completed++;

int completion\_time = current\_time;

waiting\_time[min\_remaining\_time\_index] = completion\_time - processes[min\_remaining\_time\_index].burst\_time - processes[min\_remaining\_time\_index].arrival\_time;

turnaround\_time[min\_remaining\_time\_index] = completion\_time - processes[min\_remaining\_time\_index].arrival\_time;

total\_waiting\_time += waiting\_time[min\_remaining\_time\_index];

total\_turnaround\_time += turnaround\_time[min\_remaining\_time\_index];

}

}

// Print result

printf("Process\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n");

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

printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].burst\_time,

processes[i].arrival\_time, waiting\_time[i], turnaround\_time[i]);

}

// Print average waiting time and average turnaround time

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

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

}

int main() {

int n;

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

scanf("%d", &n);

struct Process processes[n];

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

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

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

processes[i].pid = i + 1;

}

printf("\nSJF (Preemptive) Scheduling:\n");

sjf\_preemptive(processes, n);

return 0;

}

Output

A screenshot of a computer program

Description automatically generated