**FCFS**

#include "stdio.h"

typedef struct {

int pid;

int arrival\_time;

int burst\_time;

int complition\_time;

int turnaround\_time;

int waiting\_time;

} Process;

void fcfs(Process processes[], int n) {

int current\_time = 0;

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

Process \*p = &processes[i];

if (p->arrival\_time > current\_time) {

current\_time = p->arrival\_time;

}

p->complition\_time = current\_time + p->burst\_time;

p->turnaround\_time = p->complition\_time - p->arrival\_time;

p->waiting\_time = p->turnaround\_time - p->burst\_time;

current\_time = p->complition\_time;

}

}

int main() {

int n;

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

scanf("%d", &n);

Process processes[n];

printf("Enter process details (PID, Arrival Time, Burst Time)\n");

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

Process \*p = &processes[i];

printf("Process %d: ", i);

scanf("%d %d %d", &p->pid, &p->arrival\_time, &p->burst\_time);

}

fcfs(processes, n);

printf("\n");

printf("Process scheduling result (FCFS)\n");

printf("\n");

printf("PID\tAT\tBT\tCT\tTAT\tWT\n");

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

Process \*p = &processes[i];

printf("%d\t%d\t%d\t%d\t%d\t%d\n", p->pid, p->arrival\_time, p->burst\_time, p->complition\_time, p->turnaround\_time, p->waiting\_time);

}

printf("\n");

float avg\_turnaround\_time = 0, avg\_waiting\_time = 0;

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

avg\_turnaround\_time += processes[i].turnaround\_time;

avg\_waiting\_time += processes[i].waiting\_time;

}

avg\_turnaround\_time /= n;

avg\_waiting\_time /= n;

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

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

return 0;

}

**SJF**

#include "stdio.h"

typedef struct {

int pid;

int arrival\_time;

int burst\_time;

int complition\_time;

int turnaround\_time;

int waiting\_time;

} Process;

int compare\_process(Process \*p1, Process \*p2) {}

void sjf(Process processes[], int n) {

int current\_time = 0;

int completed\_processes = 0;

for (int i=0; i<n; i++) processes[i].complition\_time = 0;

while (completed\_processes < n) {

int sortest\_job\_index = -1;

int sortest\_burst\_time = -1;

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

Process \*p = &processes[i];

if (p->complition\_time == 0 && p->arrival\_time <= current\_time) {

if (sortest\_job\_index == -1 || p->burst\_time < sortest\_burst\_time) {

sortest\_job\_index = i;

sortest\_burst\_time = p->burst\_time;

}

}

}

if (sortest\_job\_index == -1) {

current\_time++;

continue;

}

}

}