* First come first serve (fcfs)

#include <stdio.h>

struct Process {

int pid;

int arrival;

int burst;

int waiting;

int turnaround;

int completion;

};

void fcfs(struct Process p[], int n) {

int current\_time = 0;

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

if (current\_time < p[i].arrival) {

current\_time = p[i].arrival;

}

p[i].completion = current\_time + p[i].burst;

p[i].turnaround = p[i].completion - p[i].arrival;

p[i].waiting = p[i].turnaround - p[i].burst;

current\_time = p[i].completion;

}

}

int main() {

int n;

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

scanf("%d", &n);

struct Process p[n];

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

p[i].pid = i + 1;

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

scanf("%d", &p[i].arrival);

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

scanf("%d", &p[i].burst);

}

// Sort processes by arrival time

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

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

if (p[j].arrival > p[j + 1].arrival) {

struct Process temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

}

}

fcfs(p, n);

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

float avg\_wait = 0, avg\_turn = 0;

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

avg\_wait += p[i].waiting;

avg\_turn += p[i].turnaround;

printf("%d\t%d\t%d\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].completion, p[i].turnaround, p[i].waiting);

}

avg\_wait /= n;

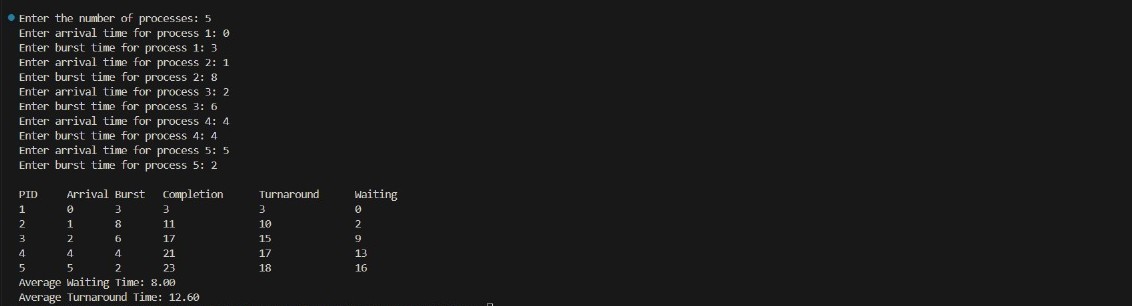
avg\_turn /= n;

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

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

return 0;

}



* Shortest job first (sjf)

#include <stdio.h>

struct Process {

int pid;

int arrival;

int burst;

int waiting;

int turnaround;

int completion;

int completed;

};

void sjf(struct Process p[], int n) {

int current\_time = 0;

int total\_completed = 0;

while (total\_completed < n) {

int shortest = -1;

int min\_burst = 9999;

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

if (p[i].arrival <= current\_time && !p[i].completed) {

if (p[i].burst < min\_burst) {

min\_burst = p[i].burst;

shortest = i;

}

}

}

if (shortest == -1) {

current\_time++;

continue;

}

p[shortest].completion = current\_time + p[shortest].burst;

p[shortest].turnaround = p[shortest].completion - p[shortest].arrival;

p[shortest].waiting = p[shortest].turnaround - p[shortest].burst;

p[shortest].completed = 1;

current\_time = p[shortest].completion;

total\_completed++;

}

}

int main() {

int n;

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

scanf("%d", &n);

struct Process p[n];

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

p[i].pid = i + 1;

p[i].completed = 0;

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

scanf("%d", &p[i].arrival);

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

scanf("%d", &p[i].burst);

}

sjf(p, n);

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

float avg\_wait = 0, avg\_turn = 0;

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

avg\_wait += p[i].waiting;

avg\_turn += p[i].turnaround;

printf("%d\t%d\t%d\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].completion, p[i].turnaround, p[i].waiting);

}

avg\_wait /= n;

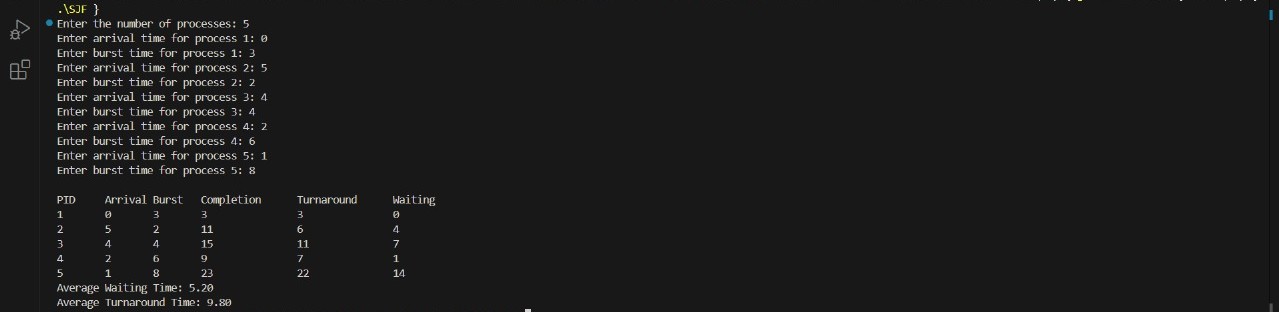
avg\_turn /= n;

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

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

return 0;

}



* Round robin(rr)

#include <stdio.h>

struct Process {

int pid;

int arrival;

int burst;

int remaining;

int waiting;

int turnaround;

int completion;

};

void roundRobin(struct Process p[], int n, int quantum) {

int current\_time = 0;

int completed = 0;

int queue[1000];

int front = 0, rear = -1;

int remaining[n];

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

remaining[i] = p[i].burst;

}

int arrived[n];

for (int i = 0; i < n; i++) arrived[i] = 0;

// Sort processes by arrival time

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

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

if (p[j].arrival > p[j + 1].arrival) {

struct Process temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

}

}

// Enqueue processes arriving at current\_time

int i = 0;

while (i < n && p[i].arrival <= current\_time) {

queue[++rear] = i;

arrived[i] = 1;

i++;

}

while (completed < n) {

if (front > rear) {

current\_time++;

while (i < n && p[i].arrival <= current\_time) {

queue[++rear] = i;

arrived[i] = 1;

i++;

}

} else {

int idx = queue[front++];

int exec\_time = (remaining[idx] < quantum) ? remaining[idx] : quantum;

remaining[idx] -= exec\_time;

current\_time += exec\_time;

// Check for new arrivals

while (i < n && p[i].arrival <= current\_time) {

if (!arrived[i]) {

queue[++rear] = i;

arrived[i] = 1;

}

i++;

}

if (remaining[idx] > 0) {

queue[++rear] = idx;

} else {

p[idx].completion = current\_time;

p[idx].turnaround = p[idx].completion - p[idx].arrival;

p[idx].waiting = p[idx].turnaround - p[idx].burst;

completed++;

}

}

}

}

int main() {

int n, quantum;

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

scanf("%d", &n);

struct Process p[n];

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

p[i].pid = i + 1;

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

scanf("%d", &p[i].arrival);

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

scanf("%d", &p[i].burst);

}

printf("Enter time quantum: ");

scanf("%d", &quantum);

roundRobin(p, n, quantum);

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

float avg\_wait = 0, avg\_turn = 0;

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

avg\_wait += p[i].waiting;

avg\_turn += p[i].turnaround;

printf("%d\t%d\t%d\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].arrival, p[i].burst, p[i].completion, p[i].turnaround, p[i].waiting);

}

avg\_wait /= n;

avg\_turn /= n;

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

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

return 0;

}

