WEEK 3 OS LAB 1BM21CS247

Q) Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.  
  
  1. SJF (pre-emptive &; Non-pre-emptive)  
2. Priority (pre-emptive &; Non-pre-emptive)  
3. Round Robin (Experiment with different quantum sizes for RR algorithm)  
  
  
Write a single program and inside the same program write different functions for different scheduling algorithms.

SOURCE CODE:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_PROCESSES 10

struct Process {

int pid;

int arrival\_time;

int burst\_time;

int priority;

int remaining\_time;

int turnaround\_time;

int waiting\_time;

};

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

// Sort the processes based on burst time in ascending order

int i,j,count=0,m;

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

{

if(processes[i].arrival\_time==0)

count++;

}

if(count==n||count==1)

{

if(count==n)

{

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

for (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;

}

}

}

}

else

{

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

for (j = 1; 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;

}

}

}

}

}

int total\_time = 0;

double total\_turnaround\_time = 0;

double total\_waiting\_time = 0;

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

total\_time += processes[i].burst\_time;

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

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

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

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

}

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

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

printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround\_time, processes[i].waiting\_time);

}

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

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

}

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

int total\_time = 0,i;

int completed = 0;

while (completed < n) {

int shortest\_burst = -1;

int next\_process = -1;

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

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

if (shortest\_burst == -1 || processes[i].remaining\_time < shortest\_burst) {

shortest\_burst = processes[i].remaining\_time;

next\_process = i;

}

}

}

if (next\_process == -1) {

total\_time++;

continue;

}

processes[next\_process].remaining\_time--;

total\_time++;

if (processes[next\_process].remaining\_time == 0) {

completed++;

processes[next\_process].turnaround\_time = total\_time - processes[next\_process].arrival\_time;

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

}

}

double total\_turnaround\_time = 0;

double total\_waiting\_time = 0;

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

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

printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround\_time, processes[i].waiting\_time);

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

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

}

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

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

}

void priority\_nonpreemptive(struct Process processes[], int n) {

// Sort the processes based on priority in ascending order

int i,j,count=0,m;

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

{

if(processes[i].arrival\_time==0)

count++;

}

if(count==n||count==1)

{

if(count==n)

{

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

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

if (processes[j].priority > processes[j + 1].priority) {

struct Process temp = processes[j];

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

processes[j + 1] = temp;

}

}

}

}

else

{

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

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

if (processes[j].priority > processes[j + 1].priority) {

struct Process temp = processes[j];

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

processes[j + 1] = temp;

}

}

}

}

}

int total\_time = 0;

double total\_turnaround\_time = 0;

double total\_waiting\_time = 0;

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

total\_time += processes[i].burst\_time;

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

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

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

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

}

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

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

printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround\_time, processes[i].waiting\_time);

}

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

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

}

void priority\_preemptive(struct Process processes[], int n) {

int total\_time = 0,i;

int completed = 0;

while (completed < n) {

int highest\_priority = -1;

int next\_process = -1;

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

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

if (highest\_priority == -1 || processes[i].priority < highest\_priority) {

highest\_priority = processes[i].priority;

next\_process = i;

}

}

}

if (next\_process == -1) {

total\_time++;

continue;

}

processes[next\_process].remaining\_time--;

total\_time++;

if (processes[next\_process].remaining\_time == 0) {

completed++;

processes[next\_process].turnaround\_time = total\_time - processes[next\_process].arrival\_time;

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

}

}

double total\_turnaround\_time = 0;

double total\_waiting\_time = 0;

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

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

printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround\_time, processes[i].waiting\_time);

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

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

}

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

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

}

void round\_robin(struct Process processes[], int n, int quantum) {

int total\_time = 0,i;

int completed = 0;

while (completed < n) {

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

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

if (processes[i].remaining\_time <= quantum) {

total\_time += processes[i].remaining\_time;

processes[i].remaining\_time = 0;

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

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

completed++;

} else {

total\_time += quantum;

processes[i].remaining\_time -= quantum;

}

}

}

}

double total\_turnaround\_time = 0;

double total\_waiting\_time = 0;

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

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

printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround\_time, processes[i].waiting\_time);

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

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

}

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

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

}

int main() {

int n, quantum,i,choice;

struct Process processes[MAX\_PROCESSES];

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

scanf("%d", &n);

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

printf("Process %d\n", i + 1);

printf("Enter arrival time: ");

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

printf ("Enter burst time: ");

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

printf("Enter priority: ");

scanf("%d", &processes[i].priority);

processes[i].pid = i + 1;

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

processes[i].turnaround\_time = 0;

processes[i].waiting\_time = 0;

}

printf("\nSelect a scheduling algorithm:\n");

printf("1. SJF Non-preemptive\n");

printf("2. SJF Preemptive\n");

printf("3. Priority Non-preemptive\n");

printf("4. Priority Preemptive\n");

printf("5. Round Robin\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

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

sjf\_nonpreemptive(processes, n);

break;

case 2:

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

sjf\_preemptive(processes, n);

break;

case 3:

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

priority\_nonpreemptive(processes, n);

break;

case 4:

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

priority\_preemptive(processes, n);

break;

case 5:

printf("\nEnter the quantum size for Round Robin: ");

scanf("%d", &quantum);

printf("\nRound Robin Scheduling (Quantum: %d):\n", quantum);

round\_robin(processes, n, quantum);

break;

default:

printf("Invalid choice!\n");

return 1;

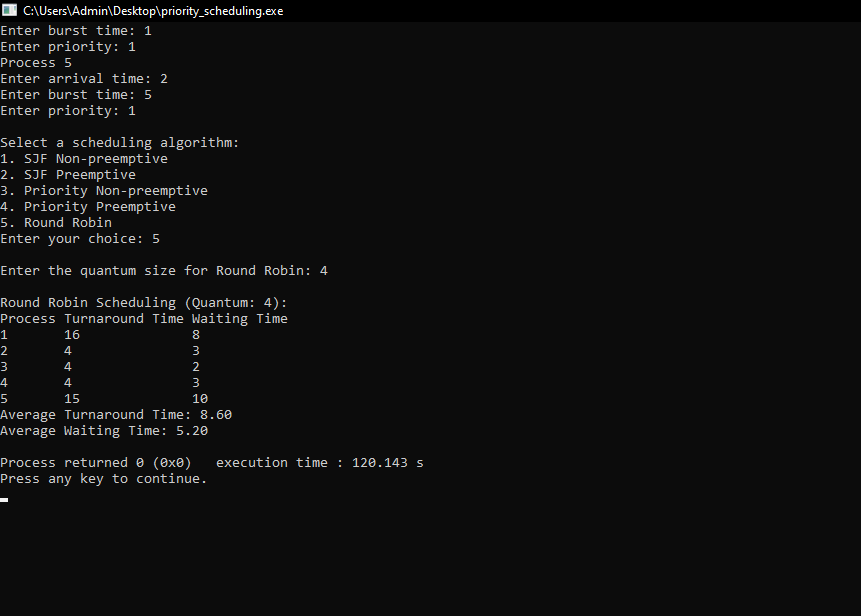
}

return 0;

}

OUTPUT:-

ROUND ROBIN



SJF PRE-EMPTIVE



