OPERATING SYSTEMS LAB

## CYCLE - 1

1. Write a program to simulate the following CPU scheduling algorithm
2. FCFS

Source Code

#include<stdio.h>

#include<stdlib.h>

void swap(int \*i, int \*j) {

int temp = \*j;

\*j = \*i;

\*i = temp;

}

float avg(int a[],int n){

float sum=0;

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

sum+=a[i];

}

return sum=sum/n;

}

void bubblesort(int a[], int p[], int bt[], int n) {

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

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

if (a[j] < a[i]) {

swap(&a[i], &a[j]);

swap(&p[i], &p[j]);

swap(&bt[i], &bt[j]);

}

}

}

}

int main(){

int n;

printf("ENTER no of ELEMENT : ");

scanf("%d",&n);

int a[n];

int p[n];

int bt[n];

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

p[i]=i+1;

}

printf("ENTER arrival time :\n ");

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

scanf("%d",&a[i]);

}

printf("ENTER burst time time :\n ");

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

scanf("%d",&bt[i]);

}

printf("arrival time : ");

bubblesort(a,p,bt,n);

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

printf("%d\t",a[i]);

}

printf("burst time :");

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

printf("%d\t",bt[i]);

}

printf("prcs :");

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

printf("%d\t",p[i]);

}\*/

int c[n];

c[0]=a[0]+bt[0];

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

c[i]=bt[i]+c[i-1];

}

int tt[n];

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

tt[i]=c[i]-a[i];

}

int wt[n];

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

wt[i]=tt[i]-bt[i];

}

/\*

printf("wait time :");

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

printf("%d\t",wt[i]);

}

printf("complete time :");

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

printf("%d\t",c[i]);

}

printf("total time :");

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

printf("%d\t",tt[i]);

}\*/

printf("\n%-8s%-15s%-15s%-18s%-15s\n", "Process", "Arrival Time", "Burst Time", "Turnaround Time", "Waiting Time");

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

printf("%-8d%-15d%-15d%-18d%-15d\n", p[i], a[i], bt[i], tt[i], wt[i]);

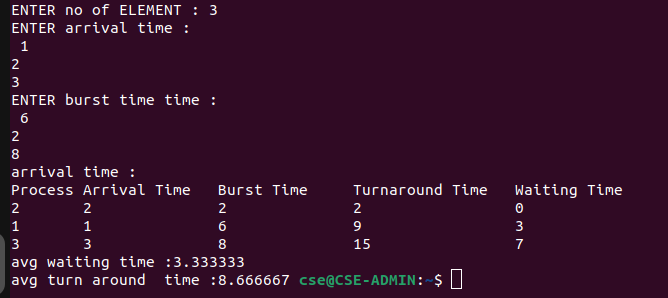
}

printf("avg waiting time :%f\n",avg(wt,n));

printf("avg turn around time :%f ",avg(tt,n));

}

Output



1. SJF

Source Code

#include<stdio.h>

#include<stdlib.h>

void swap(int \*i, int \*j) {

int temp = \*j;

\*j = \*i;

\*i = temp;

}

float avg(int a[],int n){

float sum=0;

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

sum+=a[i];

}

return sum=sum/n;

}

void bubblesort(int a[], int p[], int bt[], int n) {

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

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

if (bt[j] < bt[i]) {

swap(&a[i], &a[j]);

swap(&p[i], &p[j]);

swap(&bt[i], &bt[j]);

}

else if(bt[i]==bt[j]){

if (a[j] < a[i]) {

swap(&a[i], &a[j]);

swap(&p[i], &p[j]);

swap(&bt[i], &bt[j]);

}

}

}

}

}

int main(){

int n;

printf("ENTER no of ELEMENT : ");

scanf("%d",&n);

int a[n];

int p[n];

int bt[n];

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

p[i]=i+1;

}

printf("ENTER arrival time :\n ");

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

scanf("%d",&a[i]);

}

printf("ENTER burst time time :\n ");

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

scanf("%d",&bt[i]);

}

printf("arrival time : ");

bubblesort(a,p,bt,n);

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

printf("%d\t",a[i]);

}

printf("burst time :");

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

printf("%d\t",bt[i]);

}

printf("prcs :");

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

printf("%d\t",p[i]);

}\*/

int c[n];

c[0]=a[0]+bt[0];

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

c[i]=bt[i]+c[i-1];

}

int tt[n];

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

tt[i]=c[i]-a[i];

}

int wt[n];

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

wt[i]=tt[i]-bt[i];

}

/\*

printf("wait time :");

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

printf("%d\t",wt[i]);

}

printf("complete time :");

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

printf("%d\t",c[i]);

}

printf("total time :");

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

printf("%d\t",tt[i]);

}\*/

printf("\n%-8s%-15s%-15s%-18s%-15s\n", "Process", "Arrival Time", "Burst Time", "Turnaround Time", "Waiting Time");

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

printf("%-8d%-15d%-15d%-18d%-15d\n", p[i], a[i], bt[i], tt[i], wt[i]);

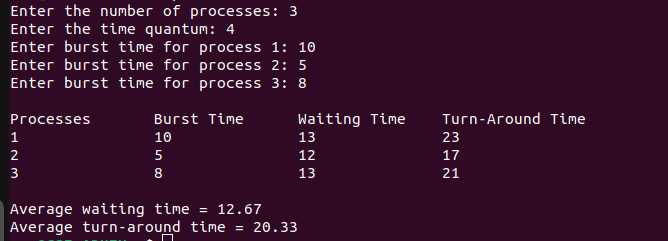
}

printf("avg waiting time :%f\n",avg(wt,n));

printf("avg turn around time :%f ",avg(tt,n));

}

Output



1. Round Robin

Source Code

#include <stdio.h>

int main() {

int n, quantum;

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

scanf("%d", &n);

printf("Enter the time quantum: ");

scanf("%d", &quantum);

int pid[n], burstTime[n], waitingTime[n], turnAroundTime[n], remainingTime[n];

int currentTime = 0, completed = 0;

float totalWaitingTime = 0, totalTurnAroundTime = 0;

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

pid[i] = i + 1;

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

scanf("%d", &burstTime[i]);

remainingTime[i] = burstTime[i];

waitingTime[i] = 0;

}

while (completed < n) {

int executed = 0;

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

if (remainingTime[i] > 0) {

executed = 1;

if (remainingTime[i] > quantum) {

currentTime += quantum;

remainingTime[i] -= quantum;

} else {

currentTime += remainingTime[i];

waitingTime[i] = currentTime - burstTime[i];

turnAroundTime[i] = waitingTime[i] + burstTime[i];

remainingTime[i] = 0;

completed++;

totalWaitingTime += waitingTime[i];

totalTurnAroundTime += turnAroundTime[i];

}

}

}

if (!executed) {

currentTime++;

}

}

printf("\nProcesses\tBurst Time\tWaiting Time\tTurn-Around Time\n");

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

printf("%d\t\t%d\t\t%d\t\t%d\n", pid[i], burstTime[i], waitingTime[i], turnAroundTime[i]);

}

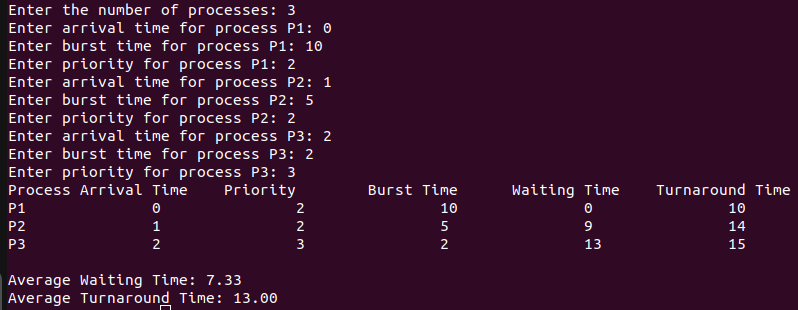
printf("\nAverage waiting time = %.2f\n", totalWaitingTime / n);

printf("Average turn-around time = %.2f\n", totalTurnAroundTime / n);

return 0;

}

Output



1. Priority

Source Code

#include <stdio.h>

void calculateWaitingTime(int n, int arrivalTime[], int burstTime[], int waitingTime[]) {

int currentTime = 0;

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

if (currentTime < arrivalTime[i]) {

currentTime = arrivalTime[i];

}

waitingTime[i] = currentTime - arrivalTime[i];

if (waitingTime[i] < 0) {

waitingTime[i] = 0;

}

currentTime += burstTime[i];

}

}

void calculateTurnaroundTime(int n, int burstTime[], int waitingTime[], int turnaroundTime[]) {

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

turnaroundTime[i] = waitingTime[i] + burstTime[i];

}

}

void sortByPriorityAndArrival(int n, int process[], int arrivalTime[], int burstTime[], int priority[]) {

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

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

if (priority[i] > priority[j] || (priority[i] == priority[j] && arrivalTime[i] > arrivalTime[j])) {

int temp = priority[i];

priority[i] = priority[j];

priority[j] = temp;

temp = arrivalTime[i];

arrivalTime[i] = arrivalTime[j];

arrivalTime[j] = temp;

temp = burstTime[i];

burstTime[i] = burstTime[j];

burstTime[j] = temp;

temp = process[i];

process[i] = process[j];

process[j] = temp;

}

}

}

}

void displayResults(int n, int process[], int arrivalTime[], int burstTime[], int priority[], int waitingTime[], int turnaroundTime[]) {

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

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

printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", process[i], arrivalTime[i], priority[i], burstTime[i], waitingTime[i], turnaroundTime[i]);

}

}

void displayAverages(int n, int waitingTime[], int turnaroundTime[]) {

double totalWaitingTime = 0, totalTurnaroundTime = 0;

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

totalWaitingTime += waitingTime[i];

totalTurnaroundTime += turnaroundTime[i];

}

printf("\nAverage Waiting Time: %.2lf\n", totalWaitingTime / n);

printf("Average Turnaround Time: %.2lf\n", totalTurnaroundTime / n);

}

int main() {

int n;

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

scanf("%d", &n);

int process[n], arrivalTime[n], burstTime[n], priority[n];

int waitingTime[n], turnaroundTime[n];

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

process[i] = i + 1;

printf("Enter arrival time for process P%d: ", process[i]);

scanf("%d", &arrivalTime[i]);

printf("Enter burst time for process P%d: ", process[i]);

scanf("%d", &burstTime[i]);

printf("Enter priority for process P%d: ", process[i]);

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

}

sortByPriorityAndArrival(n, process, arrivalTime, burstTime, priority);

calculateWaitingTime(n, arrivalTime, burstTime, waitingTime);

calculateTurnaroundTime(n, burstTime, waitingTime, turnaroundTime);

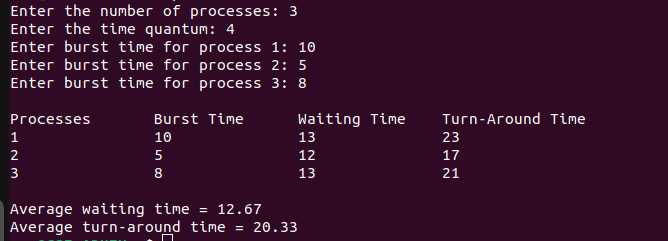
displayResults(n, process, arrivalTime, burstTime, priority, waitingTime, turnaroundTime);

displayAverages(n, waitingTime, turnaroundTime);

return 0;

}

Output



2. a. Write a program to implement process management system calls viz, fork, exist,wait, waitpid, exec.

Source Code

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

exit(1);

}

if (pid == 0) {

char \*args[] = {"./fc", NULL};

execvp(args[0], args);

perror("exec failed");

exit(1);

} else {

printf("Parent Process: PID = %d, Child PID = %d\n", getpid(), pid);

int status;

wait(&status);

printf("Parent Process: Child has finished with status %d\n", status);

exit(0);

}

return 0;

}

Output

