A picture containing orange, drawing

Description automatically generated OPERAING SYSTEM ASSIGNMENT

CSE – 316 (CA - 3)

Name :- Sai Ram Ponnaganti

Roll No :- A22

Section : - K18GT

Regd No :- 11806910

Submitted to :- Priyanka Mittal (Asst Professor)

SCHOOL OF COMPUTER SCIENCE

Q1.

Ans :-

Shortest job first (SJF) is a scheduling Algoritm that selects the waiting process with the smallest execution time to execute next. SJN is a Non pre-emptive algorithm.

* Comparing to the Scheduling Algorithms Shortest Job First has the minimum average waiting time.
* It is a Greedy Algorithm.
* It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.
* It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available

Algorithm :-

1.Sorting all the Processes according to the Arrival Time .

2.After that select the Process which has minimum Arrival time and minimum Burst Time.

3.After the Process Completing make a process pool (Gantt Chart) after the completion of every process select that process select that process into the pool having minimum Burst Time.

Demonstrating Algorithm Using an Example :-

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process No | Arrival  Time | Burst  Time | Completion  Time | Turn Around  Time | Waiting Time |
| P1 | 1 | 3 | 6 | 5 | 2 |
| P2 | 2 | 4 | 10 | 8 | 4 |
| P3 | 1 | 2 | 3 | 2 | 0 |
| P4 | 4 | 4 | 14 | 10 | 6 |

Gantt Chart :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| XX | P3 | P1 | P2 | P4 |

Turn Around Time = Completion Time – Arrival Time

Waiting Time = Turn Around Time – Burst Time .

Code in C :- This Code is implemented in C language by Using the Operating Systems concepts such as System Calls .

#include<stdio.h>

int n;

struct process

{

int p\_no;

int arrival\_t,burst\_t,ct,wait\_t,taround\_time,p;

int flag;

}p\_list[100];

void Sorting()

{

struct process p;

int i, j;

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

{

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

{

if(p\_list[i].arrival\_t > p\_list[j].arrival\_t)

{

p = p\_list[i];

p\_list[i] = p\_list[j];

p\_list[j] = p;

}

}

}

}

int main()

{

int i,t=0,b\_t=0,peak;

int a[10];

float wait\_time = 0, taround\_time = 0, avg\_w\_t=0, avg\_taround\_time=0;

printf("enter the no. of processes: ");

scanf("%d",&n);

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

{

p\_list[i].p\_no = i+1;

printf("\nEnter Details For P%d process:-\n", p\_list[i].p\_no);

printf("Enter Arrival Time: ");

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

printf("Enter Burst Time: ");

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

p\_list[i].flag = 0;

b\_t = b\_t + p\_list[i].burst\_t;

}

Sorting();

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

{

a[i]=p\_list[i].burst\_t;

}

p\_list[9].burst\_t = 9999;

for(t = p\_list[0].arrival\_t; t <= b\_t+1;)

{

peak = 9;

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

{

if(p\_list[i].arrival\_t <= t && p\_list[i].burst\_t < p\_list[peak].burst\_t && p\_list[i].flag != 1)

{

peak = i;

}

if(p\_list[peak].burst\_t==0 && p\_list[i].flag != 1)

{

p\_list[i].flag = 1;

p\_list[peak].ct=t;p\_list[peak].burst\_t=9999;

printf("P%d completes in %d\n",p\_list[i].p\_no,p\_list[peak].ct);

}

}

t++;

(p\_list[peak].burst\_t)--;

}

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

{

p\_list[i].taround\_time=(p\_list[i].ct)-(p\_list[i].arrival\_t);

avg\_taround\_time=avg\_taround\_time+p\_list[i].taround\_time;

p\_list[i].wait\_t=((p\_list[i].taround\_time)-a[i]);

avg\_w\_t=avg\_w\_t+p\_list[i].wait\_t;

}

printf("PNO\tAT\tCT\tTA\tWTt\n");

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

{

printf("P%d\t%d\t%d\t%d\t%d\n",p\_list[i].p\_no,p\_list[i].arrival\_t,p\_list[i].ct,p\_list[i].taround\_time

,p\_list[i].wait\_t);

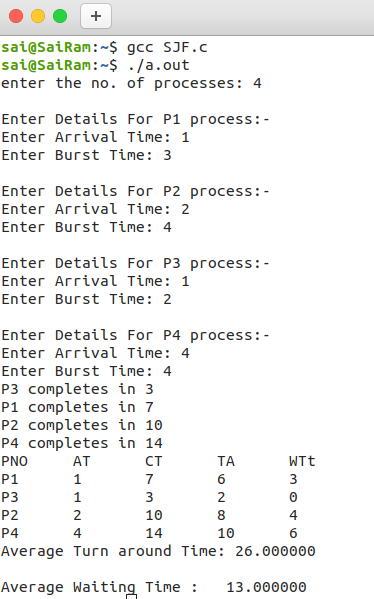
}

printf("Average Turn around Time: %f\t\n\n",avg\_taround\_time);

printf("Average Waiting Time :\t %f\t\n",avg\_w\_t);}

Time Complexity is : n\*2\*log n => ~(n log n)

Result :- This code was run on Linux Terminal and the code given this output .

Input of Code was same which were given in the Example of Algorithm.

Picture Showing the Output of Code

* The Code giving the Accurate values of given input to the Desired Output.

Q2.

Ans :-

If some Processes are given and we have to find the Average waiting time and average Turn Around time using FCFS scheduling algorithm .

First in First Out also known as First Come First Serve(FCFS), is the Simplest scheduling algorithm. In this method we use the Ready Queue . In this Process the First will be executed next process will start only after the fully execution of Previous Process.

Algorithm :-

Criteria : Arrival Time Mode : Non Pre-emptive

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process  No | Arrival  Time | Burst  Time | Completion  Time | Turn Around  Time | Waiting  Time |
| P1 | 0 | 2 | 2 | 2 | 0 |
| P2 | 1 | 2 | 4 | 3 | 1 |
| P3 | 5 | 3 | 8 | 3 | 0 |
| P4 | 6 | 4 | 12 | 6 | 2 |

Gantt Chart :-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | XX | P3 | P4 |

Turn Around Time = Completion Time – Arrival Time

Waiting Time = Turn Around Time – Burst Time .

Note :-

1. This Algorithm is Non Pre-emptive
2. Average waiting time is Not Optimal
3. It doesn’t use the Resources parallelly because it results in Convoy effect .
4. In this Algorithm Priority Queue was used so that lowest CPU burst was given highest Priority .

Code in C :-

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#define FILE\_NAME "CPU\_BURST.txt"

struct Process{

int at,bt,wt,tat;

char name[4];

};

struct Process initialize(int at,int bt,int name){

struct Process X;

X.bt = bt;

X.at = at;

sprintf(X.name,"P%d",name+1);

return X;

}

int main(){

FILE \*fp = fopen(FILE\_NAME,"r");

if(!fp)

return -1\*printf("FILE OPEN ERROR!\n");

int d,i,j,count=0;

int \*queue = (int\*)malloc(sizeof(int));

while(EOF != fscanf(fp,"%d ",&d )){

printf("%d ",d);

queue = (int\*)realloc(queue,(count+1)\*sizeof(int));

queue[count++] = d;

}

fclose(fp);

//int queue[] = {3,1,3,2,4,5};

struct Process P[count];

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

P[i] = initialize(0,queue[i],i);

for(i=1; i<count; i++){

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

if(P[j].bt>P[j+1].bt){

struct Process temp = P[j];

P[j] = P[j+1];

P[j+1] = temp;

}

}

}

printf("\nOrder : ");

int elapsed\_time=0;

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

P[i].wt = elapsed\_time;

P[i].tat= P[i].wt+P[i].bt;

elapsed\_time += P[i].bt;

printf("%s ",P[i].name);

} for(i=1; i<count; i++){

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

if(P[j].name[1]>P[j+1].name[1]){

struct Process temp = P[j];

P[j] = P[j+1];

P[j+1] = temp;

}

}

}

printf("\n\n%7s|%8s|%6s|%5s|%s\n","PROCESS","ARRIVAL","BURST","WAIT","TURNAROUND");

int total\_wt=0,total\_tt=0;

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

total\_wt+=P[i].wt;

total\_tt+=P[i].tat;

printf("%7s|%8d|%6d|%5d|%9d\n",P[i].name,P[i].at,P[i].bt,P[i].wt,P[i].tat);

}

printf("\nAverage Waiting Time : %.2f\n",total\_wt\*1.0/count);

printf("\nAverage Turn Around Time : %.2f\n",total\_tt\*1.0/count);

return 0\*printf("\nSUCCESSFUL EXIT\n");

}

Time Complexity is : O(log n)

Result :- This code was run on Linux Terminal and the code given this output .

A screenshot of a cell phone

Description automatically generatedInput of Code was same which were given in the Example of Algorithm.

Picture Showing Output of CPU BURST

* The Code giving the Accurate values of given input to the Desired Output.