

Ankit kumar

Roll -23

11806646

Sub -cse 316 Simulation based Assignment

Q1 Write a C program to solve the following problem: Suppose that a disk drive has 5,000

cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder143,

and the previous request was at cylinder 125. The queue of pending requests, in FIFO

order,is:

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

Starting from the current head position, what is the total distance (in cylinders)that the disk arm

moves to satisfy all the pending requests for each of the FCFS disk-scheduling algorithms?

CODE::

#include<stdio.h>

#include<conio.h>

void main()

{

int queue[20],n,head,i,j,k,seek=0,max,diff;

float aver;

clrscr();

printf("enter the max range of disk");

scanf("%d",&max);

printf("enter the size of queue request");

scanf("%d",&n);

printf("enter the queue");

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

{scanf("%d",&queue[i]);}

printf("enter the initial head position");

scanf("%d",&head);

queue[0]=head;

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

{

diff=abs(queue[j+1]-queue[j]);

seek+=diff;

printf("move is from %d to %d with seek %d\n",queue[j],queue[j+1],diff);

}

printf("total seek time is%d\n",seek);

aver=seek/(float)n;

printf("avrage seek time is %f\n",aver);

getch();

}

**Q2.** Design a scheduling program to implements a Queue with two levels:

Level 1 : Fixed priority preemptive Scheduling Level 2: Round Robin Scheduling

For a Fixed priority preemptive Scheduling (Queue1), the Priority 0 is highest priority. If one process P1 is scheduled and running, another process P2 with higher priority comes. The New process (high priority) process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in the multiples of 2. All the processes in second level queue will complete their execution according to round robin scheduling

Consider:

1. Queue 2 will be processed after Queue 1 becomes empty.

2. Priority of Queue 2 has lower priority than in Queue 1.

**Description:**

Multilevel feedback queue scheduling:

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.

In general, a multilevel feedback queue scheduler is defined by the following parameters:

* The number of queues.
* The scheduling algorithm for each queue.
* The method used to determine when to upgrade a process to a higher-priority queue.
* The method used to determine when to demote a process to a lower-priority queue.
* The method used to determine which queue a process will enter when that process needs service.

The program implements Multilevel Feedback Queue with two Levels: fixed priority preemptive Scheduling and Round Robin Scheduling

Queue 1 : In Fixed Priority Preemptive Scheduling Priority 0 is highest priority. If p1 is scheduling and running , now another process p2 with highest priority arrives, then p2 preempts currently running process p1 and p1 will go to second level queue

Queue 2 : In Round Robin Scheduling , the time quantum is 4 unit time.

**Code Snippet:**

#include <bits/stdc++.h>

using namespace std;

struct p\_data

{

int numer;

int process\_id;

int arr\_time;

int burst\_time;

int prior;

int finish\_time;

int remain\_time;

int wait\_time;

int start\_time;

int response\_time;

};

struct p\_data curr;

typedef struct p\_data P\_d ;

bool sort\_id(const P\_d& x , const P\_d& y)

{

return x.process\_id < y.process\_id;

}

bool sort\_arr( const P\_d& x ,const P\_d& y)

{

if(x.arr\_time < y.arr\_time)

return true;

else if(x.arr\_time > y.arr\_time)

return false;

if(x.prior < y.prior)

return true;

else if(x.prior > y.prior)

return false;

if(x.process\_id < y.process\_id)

return true;

return false;

}

bool sort\_num( const P\_d& x ,const P\_d& y)

{

return x.numer < y.numer;

}

struct comp

{

bool operator()(const P\_d& x ,const P\_d& y)

{

if( x.prior > y.prior )

return true;

else if( x.prior < y.prior )

return false;

if( x.process\_id > y.process\_id )

return true;

return false;

}

};

void my\_check(vector<P\_d> mv)

{

for(unsigned int i= 0; i < mv.size() ;i++)

{

cout<<" process\_id :"<<mv[i].process\_id<<" \_time : "<<mv[i].arr\_time<<" burst\_time : "<<mv[i].burst\_time<<" prior : "<<mv[i].prior<<endl;

}

}

int main()

{

int i;

vector< P\_d > input;

vector<P\_d> input\_copy;

P\_d temp;

int pq\_process = 0;

int rq\_process = 0;

int arr\_time;

int burst\_time;

int process\_id;

int prior;

int n;

int clock;

int total\_exection\_time = 0;

cout<<"Enter the number of processes : ";

cin>>n;

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

{

cout<<"Enter process\_id , Arrival\_time , Burst\_time , prior for process "<<i+1<<" : ";

cin>>process\_id>>arr\_time>>burst\_time>>prior;

temp.numer = i+1;

temp.arr\_time = arr\_time;

temp.burst\_time = burst\_time;

temp.remain\_time = burst\_time;

temp.process\_id = process\_id;

temp.prior = prior;

input.push\_back(temp);

}

input\_copy = input;

sort( input.begin(), input.end(), sort\_arr );

total\_exection\_time = total\_exection\_time + input[0].arr\_time;

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

{

if( total\_exection\_time >= input[i].arr\_time )

{

total\_exection\_time = total\_exection\_time +input[i].burst\_time;

}

else

{

int diff = (input[i].arr\_time - total\_exection\_time);

total\_exection\_time = total\_exection\_time + diff + burst\_time;

}

}

int Ghant[total\_exection\_time]={0};

for( i= 0; i< total\_exection\_time; i++ )

{

Ghant[i]=-1;

}

priority\_queue < P\_d ,vector<p\_data> ,comp> pq;

queue< P\_d > rq;

int cpu\_state = 0;

int quantum = 4 ;

curr.process\_id = -2;

curr.prior = 999999;

for ( clock = 0; clock< total\_exection\_time; clock++ )

{

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

{

if(clock == input[j].arr\_time)

{

pq.push(input[j]);

}

}

if(cpu\_state == 0)

{

if(!pq.empty())

{

curr = pq.top();

cpu\_state = 1;

pq\_process = 1;

pq.pop();

quantum = 4;

}

else if(!rq.empty())

{

curr = rq.front();

cpu\_state = 1;

rq\_process = 1;

rq.pop();

quantum = 4;

}

}

else if(cpu\_state == 1)

{

if(pq\_process == 1 && (!pq.empty()))

{

if(pq.top().prior < curr.prior )

{

rq.push(curr);

curr = pq.top();

pq.pop();

quantum = 4;

}

}

else if(rq\_process == 1 && (!pq.empty()))

{

rq.push(curr);

curr = pq.top();

pq.pop();

rq\_process = 0;

pq\_process = 1;

quantum = 4 ;

}

}

if(curr.process\_id != -2)

{

curr.remain\_time--;

quantum--;

Ghant[clock] = curr.process\_id;

if(curr.remain\_time == 0)

{

cpu\_state = 0 ;

quantum = 4 ;

curr.process\_id = -2;

curr.prior = 999999;

rq\_process = 0;

pq\_process = 0;

}

else if(quantum == 0 )

{

rq.push(curr);

curr.process\_id = -2;

curr.prior = 999999;

rq\_process = 0;

pq\_process = 0;

cpu\_state=0;

}

}

}

sort( input.begin(), input.end(), sort\_id );

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

{

for(int k=total\_exection\_time;k>=0;k--)

{

if(Ghant[k]==i+1)

{

input[i].finish\_time=k+1;

break;

}

}

}

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

{

for(int k=0;k<total\_exection\_time;k++)

{

if(Ghant[k]==i+1)

{

input[i].start\_time=k;

break;

}

}

}

sort( input.begin(), input.end(), sort\_num );

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

{

input[i].response\_time=input[i].start\_time-input[i].arr\_time;

input[i].wait\_time=(input[i].finish\_time-input[i].arr\_time)-input[i].burst\_time;

}

cout<<endl<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;

cout<<"process\_id\tRes\_time F\_time\tW\_time"<<endl;

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

{

cout<<input[i].process\_id<<"\t"<<input[i].response\_time<<"\t"<<input[i].finish\_time<<"\t"<<input[i].wait\_time

<<endl;

}

return 0;

}

.