

Name : komma Pavan Kumar Reddy

Registration:-11718447

Rollno:-A70

Section:-EE032

Email Id:kommapavanreddy230@gmail.com

Github link:-

Submitted

To: Dr. Aarti

Sudesh Sharma is a Linux expert who wants to have an online system where he can handle student queries. Since there can be multiple requests at any time he wishes to dedicate a fixed amount of time to every request so that everyone gets a fair share of his time. He will log into the system from 10am to 12am only. He wants to have separate requests queues for students and faculty. Implement a strategy for the same. The summary at the end of the session should include the total time he spent on handling queries and average query time.

Algorithm:-

Program for implementing Scheduling Algorithm

* **This repository contains program used to implement following problem:**  
  Sudesh Sharma is a Linux expert who wants to have an online system where he can handle student queries. Since there can be multiple requests at any time he wishes to dedicate a fixed amount of time to every request so that everyone gets a fair share of his time. He will log into the system from 10am to 12am only. He wants to have separate requests queues for students and faculty, where faculty queue is given a higher priority. Implement a strategy for the same. The summary at the end of the session should include the total time he spent on handling queries and average query time.
* **Solution Proposed for the problem :**  
  The given problem is based upon solving queries of persons of different classes i.e. Faculty and Students. Thus, these queries can be compared to different processes in terms of operating system where each process has its demands and needs resources and time for its execution. And this demands of processes are handled by the CPU. In the given scenario, Mr. Sudesh Sharma, Linux expert, can be considered as a CPU, who solves the queries of either Faculty or Student by allocating proper resources to their individual demands and processing them by allocating them time accordingly. Now, Mr. Sharma, wants to provide priority for each query based upon its class, as well as, he wants to dedicate a fixed amount of time to every request. Thus in Operating System, if we divide the requests into two separate queues i.e. Faculty and Student such that the first queue contains faculty queries has higher priority and the second contains student queries which has lower priority, then we can resolve the problem, by allocating them required resources based upon their priorities as done in the scheduling algorithm in operating systems.

Code:-

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#define NEW 0

#define READY 1

#define WAITING 2

#define RUNNING 3

#define EXIT 4

#define SRT 0

#define RR 1

int system\_time=0,runningtime=0,AllEx=0,Context\_Switch\_Permitted=1,AlreadyRuns=0,n,Cur\_Mode=RR,TymQuanta;

//allex is used for when all the program exited from both the queues Cur\_mode tells which queue is running at the time

struct Proc\_Det

{ int pid;

int state;

int more\_to\_run;

int at;

int waitingtime,turnAroundTime,ct,exect,qno;

struct Proc\_Det \*prev;

struct Proc\_Det \*next;

} \*process\_initial;

struct Queue

{

struct Proc\_Det \*front ,\*rear;

}\*RQ1,\*RQ2;

void enqueuep(struct Proc\_Det \*pros)if

(RQ1->front==NULL)

{

RQ1->front=pros;

RQ1->rear=pros;

pros->next=NULL;

}

else

{

if(pros->more\_to\_run<RQ1->front->more\_to\_run)

{

pros->next=RQ1->front;

RQ1->front->prev=pros;

RQ1->front=pros;

}

else if(pros->more\_to\_run==RQ1->front->more\_to\_run)

{

pros->next=RQ1->front->next;

pros->prev=RQ1->front;

RQ1->front->next=pros;

if(pros->next!=NULL)

{

pros->next->prev=pro

}

}

else if(pros->more\_to\_run>RQ1->rear->more\_to\_run)

{

pros->next=NULL;

RQ1->rear->next=pros;

pros->prev=RQ1->rear;

RQ1->rear=pros;

}

else

{

struct Proc\_Det \*start=RQ1->front->next;

while(start->more\_to\_run<pros->more\_to\_run)

{

start=start->next;

}

if(start!=NULL&& pros->more\_to\_run==start->more\_to\_run)

{

pros->next=start->next;

start->next=pros;

pros->prev=start;

}

else

{

(start->prev)->next=pros;

pros->next=start;

pros->prev=start->prev;

start->prev=pros;

}

}

}

}

struct Proc\_Det \* deQueuep()

{

if(RQ1->front==NULL)

{

return NULL;

}

struct Proc\_Det \* temp=RQ1->front;

RQ1->front=RQ1->front->next;

temp->next=NULL;

if(RQ1->front==NULL)

{

RQ1->rear=NULL;

}

return temp;

}

void enqueue(struct Proc\_Det \*pros)

{

if(RQ2->front==NULL)

{

pros->prev=NULL;

pros->next=NULL;

RQ2->front=pros;

RQ2->rear=pros;

return;

}

pros->prev=RQ2->rear;

RQ2->rear->next=pros;

RQ2->rear=pros;

}

struct Proc\_Det \* deQueue()

{

if(RQ2->front==NULL)

{

return NULL;

}

struct Proc\_Det \* temp=RQ2->front;

RQ2->front=RQ2->front->next;

temp->next=NULL;

if(RQ2->front==NULL)

{

RQ2->rear=NULL;

}

return temp;

}

int main()

{

RQ1 =(struct Queue\*) malloc(sizeof(struct Queue));

RQ2 =(struct Queue\*) malloc(sizeof(struct Queue));

printf("Please enter No of processes to schedule \*\* \n");

scanf("%d",&n);

printf("\nPlease Enter Time Quantum \*\* \n");

scanf("%d",&TymQuanta);

process\_initial=(struct Proc\_Det \*)malloc(sizeof(struct Proc\_Det)\*n);

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

{

printf("\n\nEnter Process Id For %d Process \*\* \n",(i+1));

scanf("%d",&(process\_initial[i].pid));

printf("\nEnter arrival time For %d Process \*\*\n",(i+1));

scanf("%d",&(process\_initial[i].at));

printf("\nEnter Execution time For %d Process \*\* \n",(i+1));

scanf("%d",&(process\_initial[i].more\_to\_run));

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

scanf("%d",&(process\_initial[i].qno));

process\_initial[i].exect=process\_initial[i].more\_to\_run;

process\_initial[i].state=NEW;

}

struct Proc\_Det key;

int i,j;

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

key = process\_initial[i];

j = i - 1;

while (j >= 0 && process\_initial[j].at > key.at) {

process\_initial[j + 1] = process\_initial[j];

j = j - 1;

}

process\_initial[j + 1] = key;

}

struct Proc\_Det \*runing\_pros=NULL;

while(1)

{

int cnt=0,i;

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

{

if(process\_initial[i].state == NEW && system\_time>=process\_initial[i].at)

{

if(process\_initial[i].qno==1)

{

enqueuep(&process\_initial[i]);

process\_initial[i].state=READY;

}

else if(process\_initial[i].qno==2)

{

enqueue(&process\_initial[i]);

process\_initial[i].state=READY;

}

}

if(process\_initial[i].state==EXIT)

{

cnt++;

}

}

if(cnt==n)

{

AllEx=1;

}

if(AllEx)

{

break;

}

if(RQ1->front!=NULL && AlreadyRuns==0)

{

if(runing\_pros!=NULL && runing\_pros->qno==2)

{

enqueue(runing\_pros);

runing\_pros->state=READY;

runing\_pros=NULL;

}

Cur\_Mode=SRT;

runningtime=1;

runing\_pros=deQueuep();

runing\_pros->state=RUNNING;

runing\_pros->more\_to\_run--;

system\_time++;

if(runningtime==runing\_pros->exect)

{

Context\_Switch\_Permitted=1;

AlreadyRuns =0;

runing\_pros->state=EXIT;

runing\_pros->ct=system\_time;

runing\_pros->turnAroundTime=system\_time-runing\_pros->at;

runing\_pros->waitingtime=runing\_pros->turnAroundTime-runing\_pros->exect;

runing\_pros=NULL;

}

else

{

Context\_Switch\_Permitted=0;

AlreadyRuns=1;

}

}

else if(RQ2->front!=NULL&& RQ1->front==NULL && runing\_pros==NULL && Context\_Switch\_Permitted==1)

{

Cur\_Mode=RR;

runningtime=1;

runing\_pros=deQueue();

runing\_pros->state=RUNNING;

runing\_pros->more\_to\_run--;

system\_time++;

if(runningtime==runing\_pros->exect)

{

Context\_Switch\_Permitted=1;

runing\_pros->state=EXIT;

runing\_pros->ct=system\_time;

runing\_pros->turnAroundTime=system\_time-runing\_pros->at;

runing\_pros->waitingtime=runing\_pros->turnAroundTime-runing\_pros->exect;

runing\_pros=NULL;

}

else

{

Context\_Switch\_Permitted=0;

}

}

else if(runing\_pros!=NULL && runing\_pros->qno==1 && Context\_Switch\_Permitted==0&&runing\_pros->state==RUNNING)

{

if(runing\_pros->more\_to\_run==0)

{

Context\_Switch\_Permitted=1;

AlreadyRuns=0;

runing\_pros->state=EXIT;

runing\_pros->ct=system\_time;

runing\_pros->turnAroundTime=system\_time-(runing\_pros->at);

runing\_pros->waitingtime=(runing\_pros->turnAroundTime)-(runing\_pros->exect);

runing\_pros=NULL;

continue;

}

if(RQ1->front!=NULL)

{

if(runing\_pros->more\_to\_run>RQ1->front->more\_to\_run)

{

runing\_pros->state=READY;

enqueuep(runing\_pros);

AlreadyRuns=0;

runing\_pros=NULL;

Context\_Switch\_Permitted=1;

continue;

}

}

runningtime++;

runing\_pros->more\_to\_run--;

system\_time++

if(runing\_pros->more\_to\_run==0)

{

Context\_Switch\_Permitted=1;

AlreadyRuns=0;

runing\_pros->state=EXIT;

runing\_pros->ct=system\_time;

runing\_pros->turnAroundTime=system\_time-(runing\_pros->at);

runing\_pros->waitingtime=(runing\_pros->turnAroundTime)-(runing\_pros->exect);

runing\_pros=NULL;

}

else if(RQ1->front!=NULL)

{

if(runing\_pros->more\_to\_run>RQ1->front->more\_to\_run)

{

runing\_pros->state=READY;

enqueuep(runing\_pros);

AlreadyRuns=0;

runing\_pros=NULL;

Context\_Switch\_Permitted=1;

}

else

{

Context\_Switch\_Permitted=0;

AlreadyRuns=1;

}

}

else

{

Context\_Switch\_Permitted=0;

AlreadyRuns=1;

}

}

else if(runing\_pros!=NULL && runing\_pros->qno==2&& runing\_pros->state==RUNNING && Context\_Switch\_Permitted==0)

{

if(runing\_pros->more\_to\_run==0)

{

Context\_Switch\_Permitted=1;

runing\_pros->state=EXIT;

runing\_pros->ct=system\_time;

runing\_pros->turnAroundTime=system\_time-(runing\_pros->at);

runing\_pros->waitingtime=(runing\_pros->turnAroundTime)-(runing\_pros->exect);

continue;

}

else if(runningtime==TymQuanta)

{

runing\_pros->state=READY;

enqueue(runing\_pros);

runing\_pros=NULL;

Context\_Switch\_Permitted=1;

continue;

}

runningtime++;

runing\_pros->more\_to\_run--;

system\_time++;

if(runing\_pros->more\_to\_run==0)

{

Context\_Switch\_Permitted=1;

runing\_pros->state=EXIT;

runing\_pros->ct=system\_time;

runing\_pros->turnAroundTime=system\_time-(runing\_pros->at);

runing\_pros->waitingtime=(runing\_pros->turnAroundTime)-(runing\_pros->exect);

}

else if(runningtime==TymQuanta)

{

runing\_pros->state=READY;

enqueue(runing\_pros);

runing\_pros=NULL;

Context\_Switch\_Permitted=1;

}

else

{

Context\_Switch\_Permitted=0;

}

}

else

{

system\_time++

} }

int sumwt=0,sumtat=0;

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

{

printf("\tprocess id=%d\t\t\tat=%d\t\t\tbt=%d\t\t\tct=%d\t\t\tTAT=%d\t\t\tWT=%d\n",process\_initial[i].pid,process\_initial[i].at,process\_initial[i].exect,process\_initial[i].ct,process\_initial[i].turnAroundTime,process\_initial[i].waitingtime);

sumwt+=process\_initial[i].waitingtime;

sumtat+=process\_initial[i].turnAroundTime

}

printf("\n\n Avergae TurnAroundTime=%f \t AverageWaitingTime=%f\n\n",(sumtat/(n\*1.0)),(sumwt/(n\*1.0)));

printf("\n\n\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n");

}



