Practical Number: 03

TITLE: **Implementation of Multiprogramming operating system Stage III:**

**i. Multiprogramming**

**ii. Virtual Memory**

**iii. Process Scheduling and Synchronization**

**iv. Inter-Process Communication**

**v. I/O Handling, Spooling and Buffering**

NAME: **Rohan Doshi** ROLLNO: **41**

CLASS: **TY C** BRANCH: **COMPUTER** **SCIENCE** BATCH: **3**

DATE OF PERFORMANCE:

/\*OS PHASE 3

Rohan Doshi C-41

Prasad Ghangal C-46

Shubham Dod C-37

Vaibhav Dod C-38

PROGRAM:\*/

#include<sstream>

#include<iostream>

#include<string.h>

#include<string>

#include<fstream>

#include<string>

#include<cstdlib>

#include<vector>

#include<queue>

using namespace std;

ifstream fin;

ofstream fout;

int SI,PI,TI;

bool occupied\_pages[30];

int IOI,ch[4]={0,5,5,2};

class PCB

{

private:

int TLL,TTL,LLC,TLC,page\_table\_ptr,terminate\_code;

string p\_id;

public:

void initialize\_PCB(string limits)

{

//initialize the contents of PCB

p\_id=limits.substr(0,4);

TTL=s\_to\_i(limits.substr(4,2))\*100+s\_to\_i(limits.substr(6,2));

TLL=s\_to\_i(limits.substr(8,2))\*100+s\_to\_i(limits.substr(10,2));

LLC=0;TLC=0;terminate\_code=0;

}

int s\_to\_i(string operand)//ok

{

//return the integer no for the given string

if(operand[0]>='0' && operand[0]<='9' && operand[1]>='0' && operand[1]<='9')

return ((int)operand[0]-48)\*10+((int)operand[1]-48);

return -1;

}

int get\_page\_table\_ptr()

{

//reterieve page table pointer

return page\_table\_ptr;

}

int set\_page\_table\_ptr(int ptr)

{

//set the page table pointer

page\_table\_ptr=ptr;

}

void increment\_TLC()

{

//increment the TLC

TLC++;

}

void increment\_LLC()

{

//increment LLC

LLC++;

}

int get\_TLC()

{

//get the time limit counter

return TLC;

}

string get\_pid()

{

//get the process id

return p\_id;

}

int get\_terminate\_code()

{

//get the terminate code

return terminate\_code;

}

void set\_terminate\_code(int t\_c)

{

//set the terminate code

terminate\_code=t\_c;

}

int get\_LLC()

{

//get the LLC

return LLC;

}

int TLC\_is\_greater\_than\_TTL()

{

//compare TLC anf TTL

if(TLC>TTL)

return 1;

else

return 0;

}

int LLC\_is\_greater\_than\_TLL()

{

//compare LLC and TLL

if(LLC>TLL)

return 1;

else

return 0;

}

};

PCB process;

class memory

{

private:

char mem[300][4];

char ch;

int page\_table\_ptr;

public:

void reset()//ok

{

//reset the memory by replacing every symbol in 2D array by $

memset(mem,'$',sizeof(char)\*300\*4);

memset(occupied\_pages,false,sizeof(bool)\*30);

page\_table\_ptr=process.get\_page\_table\_ptr();

page\_table\_ptr=rand()%30;

occupied\_pages[page\_table\_ptr]=true;

page\_table\_ptr\*=10;

process.set\_page\_table\_ptr(page\_table\_ptr);

SI=PI=TI=0;

}

string get\_mem(int pos)//get word form memory

{

//get the memory contents of given position

string temp="";

for(int i=0;i<4;i++)

temp+=mem[pos][i];

return temp;

}

void set\_mem(string s, int pos)//store word in mem

{

//set the memory for the recieved value at postion

for(int i=0;i<4;i++)

mem[pos][i]=s[i];

}

void set\_page\_table(int row\_num,int page\_no)

{

//set page table

page\_table\_ptr=process.get\_page\_table\_ptr();

ostringstream temp;

temp << page\_no;

string table\_entry;

if(page\_no<10)

table\_entry="$10"+temp.str();

else

table\_entry="$1"+temp.str();

set\_mem(table\_entry,page\_table\_ptr+row\_num);

}

int allocate\_page()

{

//allocate page logic goes here

int page\_no=rand()%30;

while(occupied\_pages[page\_no]==true)

page\_no=rand()%30;

occupied\_pages[page\_no]=true;

return page\_no;

}

void store\_card(string s,int mem\_cnt)

{

//extract the words and call the setmem function

string word="";

int page\_no=allocate\_page();

set\_page\_table(mem\_cnt, page\_no);

page\_no\*=10;

for(int i=0;i<s.length();i+=4)

{

for(int j=0;j<4;j++)

{

word+=s[i+j];

}

set\_mem(word,page\_no);

page\_no++;

word="";

}

}

void print\_mem()

{

int flag=0;

for(int i=0;i<300;i++)

{

if(i%10==0)

cout<<"----------Block "<<i/10<<"----------\n";

for(int j=0;j<4;j++)

{

cout<<mem[i][j];

}

cout<<endl;

}

}

}m\_obj;

class supervisory\_storage

{

private:

queue < string > ibq,obq;

int ebc,ibc,obc;

public:

supervisory\_storage()

{

ebc=10;

}

void inc\_ebc()

{

ebc++;

}

void dec\_ebc()

{

ebc--;

}

int get\_ebc()

{

return ebc;

}

void fill\_ibq(string temp)

{

ibq.push(temp);

}

string get\_cards\_from\_ib()

{

string s=ibq.front();

ibq.pop();

return s;

}

}super;

class aux\_drum

{

private:

string drum[100];

int curr;

public:

aux\_drum()

{

curr=0;

}

void set\_drum(string s)

{

drum[curr]=s;

++curr;

}

string get\_drum\_entry(int i)

{

return drum[i];

}

int get\_curr()

{

return curr;

}

}drum\_obj;

class cpu

{

private:

int fetched\_IC,pos,flag,program\_card\_cnt;

bool terminate,fetched\_C;

string fetched\_IR,operand,opreator,fetched\_R,compare\_string;

char IR[4],R[4],IC[2];

bool C,run\_mos;

string s;

public:

//set and reset function of all the register

int s\_to\_i(string operand)//ok

{

//return the integer no for the given string

if(operand[0]>='0' && operand[0]<='9' && operand[1]>='0' && operand[1]<='9')

return ((int)operand[0]-48)\*10+((int)operand[1]-48);

return -1;

}

void set\_IC()

{

IC[0]='0';

IC[1]='0';

}

void set\_IC(int pos)//ok

{

IC[1]=((char)pos%10)+48;

pos=pos/10;

IC[0]=((char)pos%10)+48;

}

int get\_IC()

{

int val;

val=((int)IC[0]-48)\*10+((int)IC[1]-48);

return val;

}

void inc\_IC()//ok

{

int val;

val=get\_IC();

val++;

set\_IC(val);

}

void set\_IR(int IC)//ok

{

string returned\_value="";

returned\_value=m\_obj.get\_mem(IC);

for(int i=0;i<4;i++)

IR[i]=returned\_value[i];

}

string get\_IR()//ok

{

string ret\_IR="";

for(int i=0;i<4;i++)

ret\_IR+=IR[i];

return ret\_IR;

}

void set\_R(int pos)//ok

{

string returned\_value="";

returned\_value=m\_obj.get\_mem(pos);

for(int i=0;i<4;i++)

R[i]=returned\_value[i];

}

string get\_R()//ok

{

string ret\_R="";

for(int i=0;i<4;i++)

{

ret\_R+=R[i];

}

return ret\_R;

}

void set\_C(bool value)//ok

{

C=value;

}

bool get\_C()//ok

{

return C;

}

int address\_tranlation(int virtual\_add)

{

//address translation logic

int page=process.get\_page\_table\_ptr()+(virtual\_add/10);//get the page table pointer and the virtual address tens place

string value\_page=m\_obj.get\_mem(page);//get the required page from the memory

if(value\_page[1]=='$')//check for the page fault

{

PI=3;

return -1;

}

value\_page=value\_page.substr(2,2);//reterieve the page value

return (s\_to\_i(value\_page)\*10+(virtual\_add%10));//return the real by adding the unit place to retrieve the specific instruction in the that page

}

int address\_tranlation(string op)

{

//check for operan error

if(s\_to\_i(op)==-1)

{

PI=2;

return -2;

}

else

address\_tranlation(s\_to\_i(op));//no operand error go for tranlation logic

}

void IR1()

{

if(fin.eof())

flag=4;

else

{

getline(fin,s);

cout<<"IN CH1....reading from file\nRead value : "<<s<<endl;

if(s.find("$AMJ")!=-1)

{

//reset the memory

process.initialize\_PCB(s.substr(4,12));

flag=1;

program\_card\_cnt=0;

}

else if(s.find("$DTA")!=-1)

{

flag=2;

}

else if(s.find("$END")!=-1)

{

//proceed to the next job

flag=3;

}

else if(flag==1 || flag==2)

{

cout<<"Pushing data in IFQ"<<endl;

//load the program card as soon as it comes

super.fill\_ibq(s);

super.dec\_ebc();

}

ch[1]=5;

}

}

void IR3()

{

cout<<"IN CH3\n";

string temp;

if(flag==4)

{

cout<<"EOF found...Memory printing\n";

m\_obj.print\_mem();

exit(0);

}

if(super.get\_ebc()!=10)

{

temp=super.get\_cards\_from\_ib();

cout<<"Found data in IFQ....Data is "<<temp<<endl;

super.inc\_ebc();

drum\_obj.set\_drum(temp);

if(flag==1)

program\_card\_cnt++;

}

else if(flag==3)

{

cout<<"Moving from drum to memory\n";

//int lim\_cnt=drum\_obj.get\_curr();

for(int i=0;i<program\_card\_cnt;i++)

{

s=drum\_obj.get\_drum\_entry(i);

m\_obj.store\_card(s,i);

}

program\_card\_cnt=0;

}

ch[3]=2;

}

void startexe()

{

//fetch decode execute cycle given below

set\_IC(0);

terminate=false;

while(!terminate)

{

//fetch ic

//fetch ir form the location specified by ic

//get the value of operator and operand

run\_mos=false;

fetched\_IC=address\_tranlation(get\_IC());

inc\_IC();

set\_IR(fetched\_IC);

fetched\_IR=get\_IR();

if((fetched\_IR.compare("H"))==3)

//handle the error of instruction single H , suppose if the user writes in the program H instead of H123

fetched\_IR="Hrrr";

opreator=fetched\_IR.substr(0,2);

operand=fetched\_IR.substr(2,2);

pos=address\_tranlation(s\_to\_i(operand));

if(address\_tranlation(operand)!=-2 || !(operand.compare("rr")))

{

if(!(opreator.compare("LR")))//ok

{

//set the contents of the register R from the given location of memory specified in the operand

//cout<<"LR";

if(pos==-1)

{

run\_mos=true;//if operand error runmos = true

}

else

{

set\_R(pos);

process.increment\_TLC();

}

}

else if (!(opreator.compare("SR")))//ok

{

//get the contents of register R

//store them at the location of memory specified in the operand

//cout<<"SR";

//process.increment\_TLC();

fetched\_R=get\_R();

if(pos==-1)

{

run\_mos=true;

}

else

{

m\_obj.set\_mem(fetched\_R, pos);

process.increment\_TLC();

}

}

else if (!(opreator.compare("CR")))//ok

{

//get the contents of register r

//compare with given memory location

//if the values of above two matches then set toggle register to true

//else set the toggle register to false

//cout<<"CR";

fetched\_R=get\_R();

if(pos==-1)

{

run\_mos=true;

}

else

{

process.increment\_TLC();

compare\_string=m\_obj.get\_mem(pos);

if(fetched\_R.compare(compare\_string)==0)

set\_C(true);

else

set\_C(false);

}

}

else if (!(opreator.compare("BT")))//ok

{

//if the value of toggle register is true

//we change the value of ic

//cout<<"BT";

//fetched\_C=get\_C();

if(fetched\_C)

{

fetched\_C=get\_C();

//int get\_physical\_add=address\_tranlation();

set\_IC(s\_to\_i(operand));

}

process.increment\_TLC();

}

else if (!(opreator.compare("GD")))//ok

{

if(pos!=-1)

{

process.increment\_TLC();

SI=1;

}

run\_mos=true;

}

else if (!(opreator.compare("PD")))//ok

{

if(pos!=-1)

{

process.increment\_TLC();

SI=2;

}

run\_mos=true;

}

else if (!(opreator.compare("Hr")))//ok

{

process.increment\_TLC();

fetched\_IR="H";

SI=3;

run\_mos=true;

}

else

{

PI=1;

run\_mos=true;

}

}

else

{

run\_mos=true;

}

if(process.TLC\_is\_greater\_than\_TTL())

{

TI=2;

run\_mos=true;

terminate=true;

process.set\_terminate\_code(3);

}

if(run\_mos)

MOS();

}

}

void MOS()

{

//acording to ti si and pi set here we will handle the interrupt

if(TI==0 && SI==1)

{

//get the instrution of the file in terms of 4 words

//store it one by one into memory

//at start store the program card from memory location with unit place 0

string s;

pos=(pos/10)\*10;

getline(fin,s);

if(s.find("$END")!=-1)// if the $END encountered before hand then out of data

{

terminate=true;

process.set\_terminate\_code(1);

}

else

{

if(!s.empty() && s[s.size()-1]=='\r')

s.erase(s.size()-1);

int len=s.length(),start=0,i;

string s1;

for(i=pos;start<len;i++)

{

if((len-start)<4)

s1=s.substr(start,(len-start));

else

s1=s.substr(start,4);

start+=4;

m\_obj.set\_mem(s1,i);

}

}

SI=0;

}

else if(TI==2 && SI==1)

{

terminate=true;

process.set\_terminate\_code(3);

}

else if((TI==0 || TI==2) && SI==2)

{

process.increment\_LLC();

if(process.LLC\_is\_greater\_than\_TLL())//lime limit exceeded

{

terminate=true;

process.set\_terminate\_code(2);

}

//put the data from memory into the file specified

else

{

int pos=address\_tranlation(s\_to\_i(operand)),flag=0;

pos=(pos/10)\*10;

string ans="",temp="";

for(int i=pos;i<pos+10;i++)

{

temp=m\_obj.get\_mem(i);

for(int j=0;j<4;j++)

{

if(temp[j]=='\0' || temp[j]=='$')

{

break;

flag=1;

}

ans+=temp[j];

}

if(flag)

break;

}

fout<<ans<<endl;

if(TI==2)//time limit exceeded

{

terminate=true;

process.set\_terminate\_code(3);

}

}

SI=0;

}

else if((TI==0 || TI==2) && SI==3)//no error has ocurred terminated the program normally

{

//make terminate true to end the program

//as we encountered the halt instruction

terminate=true;

process.set\_terminate\_code(0);

}

else if(TI==0 && PI==1)//operation code error has occured

{

process.set\_terminate\_code(4);

terminate=true;

}

else if(TI==0 && PI==2)//operand code error has ocurred

{

process.set\_terminate\_code(5);

terminate=true;

}

else if(TI==0 && PI==3)//handle the valid page fault or terminate with invaild page faults

{

if(!(opreator.compare("GD")) || !(opreator.compare("SR")))

{

process.increment\_TLC();

int page\_no=m\_obj.allocate\_page();

m\_obj.set\_page\_table((s\_to\_i(operand))/10,page\_no);

set\_IC(get\_IC()-1);

}

else

{

process.set\_terminate\_code(6);

terminate=true;

}

}

else if(TI==2 && PI==1)//time limit exceeded with operation code error

{

process.set\_terminate\_code(7);

terminate=true;

}

else if(TI==2 && PI==2)//time limit exceeded with operand error

{

process.set\_terminate\_code(8);

terminate=true;

}

else if(TI==2 && PI==3)//time limit exceeded

{

process.set\_terminate\_code(3);

terminate=true;

}

switch(IOI)

{

case 0:

break;

case 1:

IR1();

break;

case 2:

// IR2();

break;

case 3:

// IR2();

IR1();

break;

case 4:

IR3();

break;

case 5:

IR3();

IR1();

break;

case 6:

IR3();

//IR2();

break;

case 7:

IR3();

//IR2();

IR1();

break;

}

IOI=0;

if(terminate)

{

fout<<process.get\_pid()<<" ";

switch(process.get\_terminate\_code())

{

case 0:fout<<"NO ERROR\n";

break;

case 1:fout<<"OUT OF DATA\n";

break;

case 2:fout<<"LINE LIMIT EXCEEDED\n";

break;

case 3:fout<<"TIME LIMIT EXCEEDED\n";

break;

case 4:fout<<"OPERATION CODE ERROR\n";

break;

case 5:fout<<"OPERAND ERROR\n";

break;

case 6:fout<<"INVALID PAGE FAULT\n";

break;

case 7:fout<<"TIME LIMIT EXCEEDED with OPERATION CODE ERROR\n";

break;

case 8:fout<<"TIME LIMIT EXCEEDED with OPERAND ERROR\n";

break;

}

fout<<get\_IC()<<" "<<fetched\_IR<<" "<<process.get\_TLC()<<" "<<process.get\_LLC()<<endl;

fout<<endl<<endl;

}

}

}exe;

void simulation()

{

for(int i=1;i<4;i++)

{

if(ch[i]!=0)

ch[i]--;

else

{

IOI+=i;

if(i==3)

IOI+=1;

}

}

}

int main()

{

//open two files one input and one output

//create memory and cpu object

fin.open("job1.txt");

fout.open("job2\_Output2.txt");

string s,s1;

int mem\_cnt=0;

m\_obj.reset();

while(true)

{

simulation();

exe.MOS();

//s\_m();

}

fin.close();

fout.close();

return 0;

}

**Input File :**

$AMJ010200210001

GD30LR36SR40LR35SR41LR34SR42LR33SR43LR32

SR44LR31SR45LR30SR46LR39SR47SR38SR49PD40

H

$DTA

AMA PANANAL A CPLANN A A MA

$END0102

$AMJ020100120003

GD20LR20GD30CR33BT07GD40PD40PD20PD30GD40

PD40H

$DTA

HOPE FOR IT

THERE IS NO HOPE

BUT STILL HOPE

$END0201

**Output for Input File :**

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[1, 2, 3, 4, 5, 6, 7, 8, 9] IFBQ :[0]

RQ1 :0 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[2, 3, 4, 5, 6, 7, 8, 9] IFBQ :[0, 1]

RQ1 :0 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[2, 3, 4, 5, 6, 7, 8, 9, 0] IFBQ :[1]

RQ1 :0 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[3, 4, 5, 6, 7, 8, 9, 0] IFBQ :[1, 2]

RQ1 :0 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[3, 4, 5, 6, 7, 8, 9, 0, 1] IFBQ :[2]

RQ1 :0 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[4, 5, 6, 7, 8, 9, 0, 1] IFBQ :[2, 3]

RQ1 :0 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[4, 5, 6, 7, 8, 9, 0, 1, 2] IFBQ :[3]

RQ1 :0 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[5, 6, 7, 8, 9, 0, 1, 2] IFBQ :[3, 4]

RQ1 :0 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[5, 6, 7, 8, 9, 0, 1, 2, 3] IFBQ :[4]

RQ1 :0 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[6, 7, 8, 9, 0, 1, 2, 3] IFBQ :[4, 5]

RQ1 :0 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[6, 7, 8, 9, 0, 1, 2, 3, 4] IFBQ :[5]

RQ1 :0 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[7, 8, 9, 0, 1, 2, 3, 4] IFBQ :[5, 6]

RQ1 :0 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[7, 8, 9, 0, 1, 2, 3, 4, 5] IFBQ :[6]

RQ1 :0 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[8, 9, 0, 1, 2, 3, 4, 5] IFBQ :[6, 7]

RQ1 :0 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[8, 9, 0, 1, 2, 3, 4, 5, 6] IFBQ :[7]

RQ1 :0 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[9, 0, 1, 2, 3, 4, 5, 6] IFBQ :[7, 8]

RQ1 :0 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[9, 0, 1, 2, 3, 4, 5, 6] IFBQ :[7, 8]

RQ1 :1 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[0, 1, 2, 3, 4, 5, 6] IFBQ :[7, 8, 9]

RQ1 :1 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[0, 1, 2, 3, 4, 5, 6, 7] IFBQ :[8, 9]

RQ1 :1 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[1, 2, 3, 4, 5, 6, 7] IFBQ :[8, 9, 0]

RQ1 :1 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[1, 2, 3, 4, 5, 6, 7, 8] IFBQ :[9, 0]

RQ1 :1 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[2, 3, 4, 5, 6, 7, 8] IFBQ :[9, 0, 1]

RQ1 :1 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[2, 3, 4, 5, 6, 7, 8, 9] IFBQ :[0, 1]

RQ1 :1 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[3, 4, 5, 6, 7, 8, 9] IFBQ :[0, 1, 2]

RQ1 :1 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[3, 4, 5, 6, 7, 8, 9, 0] IFBQ :[1, 2]

RQ1 :1 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[4, 5, 6, 7, 8, 9, 0] IFBQ :[1, 2, 3]

RQ1 :1 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[4, 5, 6, 7, 8, 9, 0, 1] IFBQ :[2, 3]

RQ1 :1 RQ2 :0

In IR1

CH1: CH1 activated CH3: CH3 not active

EBQ:[5, 6, 7, 8, 9, 0, 1] IFBQ :[2, 3, 4]

RQ1 :1 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 activated

EBQ:[5, 6, 7, 8, 9, 0, 1, 2] IFBQ :[3, 4]

RQ1 :1 RQ2 :0

In IR1

IN IR3

CH1: CH1 not active CH3: CH3 not active

EBQ:[5, 6, 7, 8, 9, 0, 1, 2, 3] IFBQ :[4]

RQ1 :1 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 not active

EBQ:[5, 6, 7, 8, 9, 0, 1, 2, 3, 4] IFBQ :[]

RQ1 :1 RQ2 :0

IN IR3

CH1: CH1 not active CH3: CH3 not active

EBQ:[5, 6, 7, 8, 9, 0, 1, 2, 3, 4] IFBQ :[]

RQ1 :1 RQ2 :2

IN IR3