**CODE:**

#include<iostream>

#include<algorithm>

#include<vector>

#define MAX 10

using namespace std;

int N,TOTbt=0,space=0;

typedef struct PROCESSES{

int index;

int BT;

int AT;

int PRIO;

int TAT;

int WT;

int CT;

int CountBT;

};

typedef struct ARRANGE{vector<PROCESSES> P;};

ARRANGE ALIST[50];

void processInput(PROCESSES Table[])

{

int temp;

cout<<"ENTER NUMBER OF ELEMENTS: ";

cin>>N;

cout<<"ENTER THE PROCESS DETAILS"<<endl;

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

{

cout<<"P"<<i+1<<": \n";

Table[i].index=i+1;

cout<<"ENTER BURST TIME: ";

cin>>Table[i].BT; TOTbt+=Table[i].BT;

cout<<"ENTER ARRIVAL TIME: ";

cin>>Table[i].AT;

cout<<"ENTER PRIORITY: ";

cin>>Table[i].PRIO;

cout<<endl;

Table[i].CountBT=0;

}

}

bool CRITERIA2(PROCESSES A,PROCESSES B) {return(A.index<B.index);}

void ProcessInfo(PROCESSES Table[])

{

int TotalTAT=0,TotalWT=0;

sort(Table,Table+N,CRITERIA2);

cout<<"\n\nPROCESS "<<"BT "<<"AT "<<"CT "<<"PRIORITY "<<"TAT "<<"WT "<<endl;

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

{

cout.setf(ios::left,ios::adjustfield);

cout.width(12);

cout<<Table[i].index;

cout.width(7);

cout<<Table[i].BT;

cout.width(7);

cout<<Table[i].AT;

cout.width(7);

cout<<Table[i].CT;

cout.width(11);

cout<<Table[i].PRIO;

cout.width(8);

cout<<Table[i].TAT;

TotalTAT+=Table[i].TAT;

cout.width(7);

cout<<Table[i].WT;

TotalWT+=Table[i].WT;

cout<<endl;

}

cout<<"\nATAT ="<<(float)TotalTAT/N;

cout<<"\nAWT ="<<(float)TotalWT/N<<endl;

}

bool CRITERIA1(PROCESSES A,PROCESSES B) { return(A.PRIO<B.PRIO); }

void processArrange(PROCESSES Table[])

{

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

{

ALIST[Table[i].AT].P.push\_back(Table[i]);

}

}

void PROCESS\_execute(int i,vector<PROCESSES>& WaitQ)

{

if(WaitQ.empty()!=true)

{

cout<<"| P"<<WaitQ[0].index<<" ";

WaitQ[0].CountBT++;

}

else{

cout<<" ";

space++;

}

}

void PRIORITY\_SCHEDULING(PROCESSES Table[])

{

vector<PROCESSES>WaitQ;

vector<PROCESSES>::iterator QT;

int i;

cout<<endl<<"GHANTT CHART\n";

for(i=0;i<TOTbt+space;i++)

{

if(ALIST[i].P.empty()==false)

{

for(QT=ALIST[i].P.begin();QT!=ALIST[i].P.end();++QT)

{

WaitQ.push\_back(\*QT);

}

sort(WaitQ.begin(),WaitQ.end(),CRITERIA1);

}

PROCESS\_execute(i,WaitQ);

if(WaitQ.empty()!=true)

if(WaitQ[0].CountBT==WaitQ[0].BT)

{

Table[WaitQ[0].index-1].CT=i+1;

Table[WaitQ[0].index-1].TAT=(Table[WaitQ[0].index-1].CT) - (Table[WaitQ[0].index-1].AT);

Table[WaitQ[0].index-1].WT=(Table[WaitQ[0].index-1].TAT) - (Table[WaitQ[0].index-1].BT);

WaitQ.erase(WaitQ.begin()); //If process BT done then delete from waiting queue

}

}

cout<<"|"<<endl;

for(int j=0;j<=TOTbt+space;j++)

{

cout.setf(ios::left,ios::adjustfield);

cout.width(5);

cout<<j;

}

}

int main()

{

PROCESSES Table[MAX];

processInput(Table);

processArrange(Table);

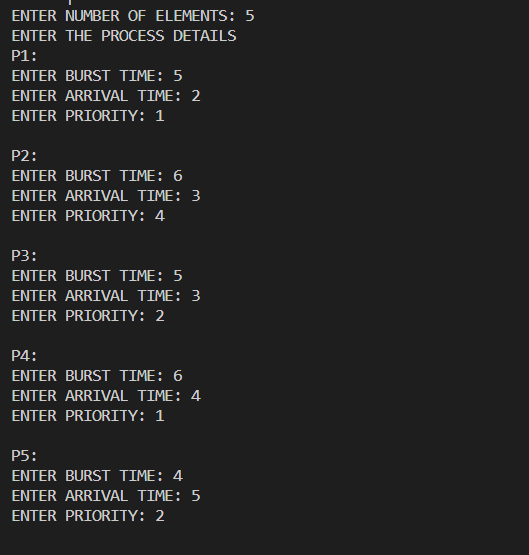
cout<<endl<<endl;

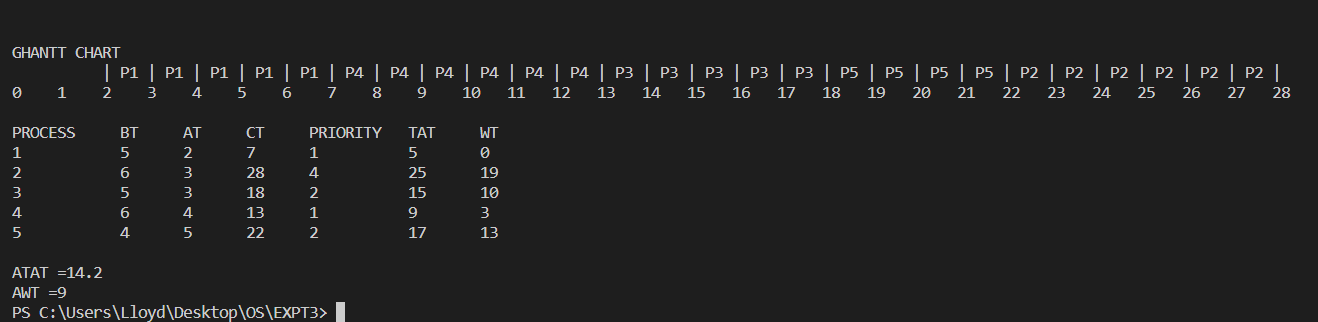
PRIORITY\_SCHEDULING(Table);

ProcessInfo(Table);

}

**OUTPUT:**

****

****

**CONCLUSION**

The pre-emptive priority scheduling algorithm was comprehended and was successfully implemented.

**CODE:**

#include<iostream>

#include<queue>

using namespace std;

#define MAX\_SIZE 10

#define WAIT s--

#define SIGNAL s++

#define FULL EMPTY\_SLOTS==X

#define SLOT\_OCCUPIED EMPTY\_SLOTS--

int s=0;

int EMPTY\_SLOTS=MAX\_SIZE;

int X=0;

queue<int> BUFFER;

void DISPLAY()

{

queue<int> DISPLAY\_BUFF=BUFFER;

cout<<"CURRENT BUFFER CONTENT: ";

if(DISPLAY\_BUFF.empty()==true)

{

cout<<"QUEUE IS EMPTY"<<endl;

return;

}

while(DISPLAY\_BUFF.empty()==false)

{

cout<<DISPLAY\_BUFF.front()<<" ";

DISPLAY\_BUFF.pop();

}

}

void PRODUCE()

{

int r=rand()%100;

if(FULL)

{

cout<<"\n\nQUEUE IS FULL";

return;

}

WAIT; cout<<"\n\nSEMAPHORE WAIT: "<<s<<endl;

BUFFER.push(r);

SLOT\_OCCUPIED;

SIGNAL; cout<<"SEMAPHORE SIGNAL: "<<s<<endl;

DISPLAY();

}

void CONSUME()

{

if(BUFFER.empty()==true)

{

cout<<"\n\nQUEUE IS EMPTY"<<endl;

return;

}

WAIT; cout<<"\n\nSEMAPHORE WAIT: "<<s<<endl;

cout<<"PROCESS CONSUMED: "<<BUFFER.front()<<endl;

BUFFER.pop();X--;

SIGNAL; cout<<"SEMAPHORE SIGNAL: "<<s<<endl;

DISPLAY();

}

int main()

{

int ch;

do

{

cout<<"\n\n1: PRODUCE\n2: CONSUME\n3: DISPLAY\n0: EXIT\n";

cout<<"ENTER YOUR CHOICE ";

cin>>ch;

switch(ch)

{

case 1: PRODUCE();

break;

case 2: CONSUME();

break;

case 3: DISPLAY();

break;

case 0: break;

default : cout<<"INVALID CHOICE ";

break;

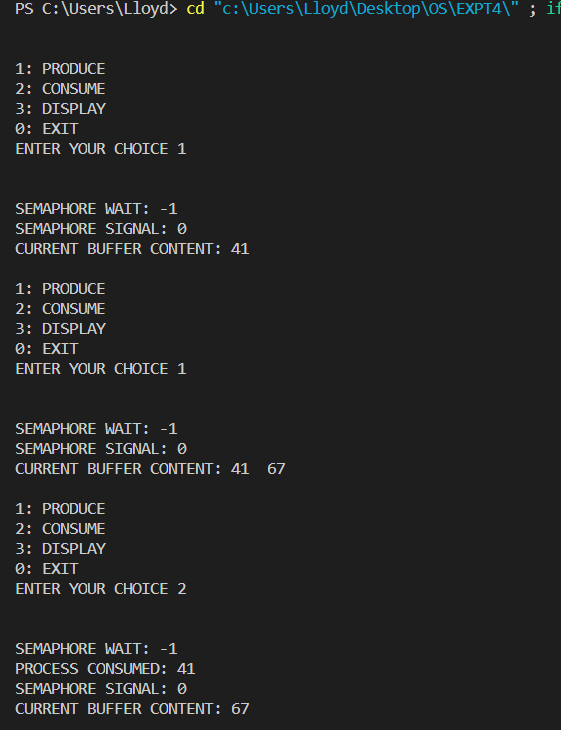
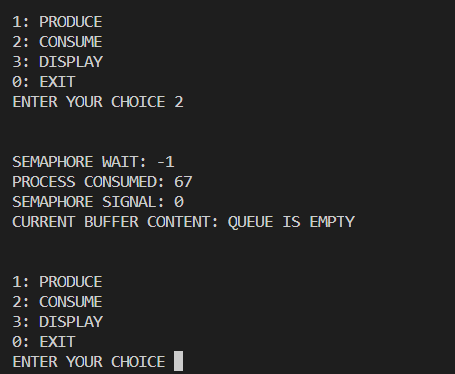
}

}while(ch);

return 0;

}

**OUTPUT:**

****