1. Program for First Come First Serve.

#include <iostream>

using namespace std;

int main()

{

int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

cout<<"Enter total number of processes(maximum 20):";

cin>>n;

cout<<"\nEnter Process Burst Time\n";

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

{

cout<<"P["<<i+1<<"]:";

cin>>bt[i];

}

wt[0]=0; //waiting time for first process is 0

//calculating waiting time

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

{

wt[i]=0;

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

wt[i]+=bt[j];

}

cout<<"\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time";

//calculating turnaround time

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

{

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

avwt+=wt[i];

avtat+=tat[i];

cout<<"\nP["<<i+1<<"]"<<"\t\t"<<bt[i]<<"\t\t"<<wt[i]<<"\t\t"<<tat[i];

}

avwt/=i;

avtat/=i;

cout<<"\n\nAverage Waiting Time:"<<avwt;

cout<<"\nAverage Turnaround Time:"<<avtat;

return 0;

}

Output –

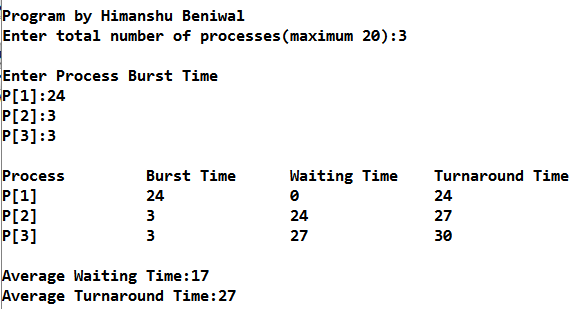


Figure 1.1 Screen-shot for Output of FCFS

1. Program for Priority Scheduling.

#include <iostream>

using namespace std;

int main()

{

int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg\_wt,avg\_tat;

cout<<"Enter Total Number of Process:";

cin>>n;

cout<<"\nEnter Burst Time and Priority\n";

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

{

cout<<"\nP["<<i+1<<"]\n";

cout<<"Burst Time:";

cin>>bt[i];

cout<<"Priority:";

cin>>pr[i];

p[i]=i+1; //contains process number

}

//sorting burst time, priority and process number in ascending order using selection sort

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

{

pos=i;

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

{

if(pr[j]<pr[pos])

pos=j;

}

temp=pr[i];

pr[i]=pr[pos];

pr[pos]=temp;

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0; //waiting time for first process is zero

//calculate waiting time

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=total/n; //average waiting time

total=0;

cout<<"\nProcess\t Burst Time \tWaiting Time\tTurnaround Time";

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

{

tat[i]=bt[i]+wt[i]; //calculate turnaround time

total+=tat[i];

cout<<"\nP["<<p[i]<<"]\t\t "<<bt[i]<<"\t\t "<<wt[i]<<"\t\t\t"<<tat[i];

}

avg\_tat=total/n; //average turnaround time

cout<<"\n\nAverage Waiting Time="<<avg\_wt;

cout<<"\nAverage Turnaround Time="<<avg\_tat;

return 0;

}

Output –

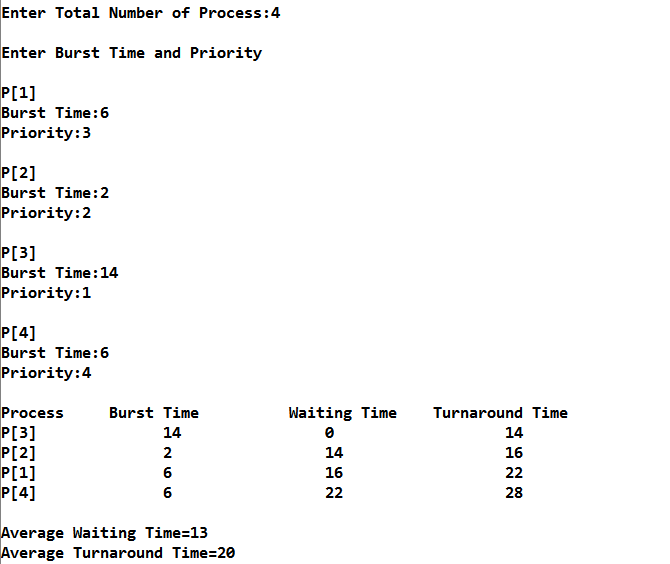


Figure 2.1 Screen-shot for Output of Priority Scheduling.

1. C++ Program for Shortest Job First (SJF) Scheduling Algorithm.

#include <iostream>

using namespace std;

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

cout<<"Enter number of process:";

cin>>n;

cout<<"\nEnter Burst Time:\n";

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

{

cout<<"p:";

cin>>bt[i];

p[i]=i+1; //contains process number

}

//sorting burst time in ascending order using selection sort

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

{

pos=i;

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

{

if(bt[j]<bt[pos])

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0; //waiting time for first process will be zero

//calculate waiting time

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n; //average waiting time

total=0;

cout<<"\nProcess\t Burst Time \tWaiting Time\tTurnaround Time";

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

{

tat[i]=bt[i]+wt[i]; //calculate turnaround time

total+=tat[i];

cout<<"\np"<<p[i]<<"\t\t "<<bt[i]<<"\t\t "<<wt[i]<<"\t\t\t"<<tat[i];

}

avg\_tat=(float)total/n; //average turnaround time

cout<<"\n\nAverage Waiting Time="<<avg\_wt<<endl;

cout<<"\nAverage Turnaround Time="<<avg\_tat<<endl;

return 0;

}

Output –

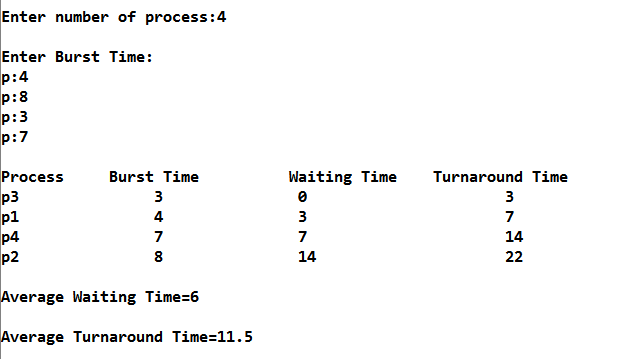


Figure 3.1 Screen-shot for Output of Shortest Job First Scheduling.

1. Program for Round Robin Scheduling.

#include <iostream>

using namespace std;

int main()

{

int count,j,n,time,remain,flag=0,time\_quantum;

int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];

cout<<"Enter Total Process:\t ";

cin>>n;

remain=n;

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

{

cout<<"\nEnter Arrival Time and Burst Time for Process Process Number:"<<count+1<<" ";

cin>>at[count];

cin>>bt[count];

rt[count]=bt[count];

}

cout<<"\nEnter Time Quantum:\t";

cin>>time\_quantum;

cout<<"\n\nProcess\t|Turnaround Time|Waiting Time\n\n";

for(time=0,count=0;remain!=0;)

{

if(rt[count]<=time\_quantum && rt[count]>0)

{

time+=rt[count];

rt[count]=0;

flag=1;

}

else if(rt[count]>0)

{

rt[count]-=time\_quantum;

time+=time\_quantum;

}

if(rt[count]==0 && flag==1)

{

remain--;

cout<<"P["<<count+1<<"]\t|\t"<<time-at[count]<<"\t|\t"<<time-at[count]-bt[count]<<"\n";

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if(count==n-1)

count=0;

else if(at[count+1]<=time)

count++;

else

count=0;

}

cout<<"\nAverage Waiting Time= "<<wait\_time\*1.0/n;

cout<<"\nAvg Turnaround Time = "<<turnaround\_time\*1.0/n;

return 0;

}

Output –

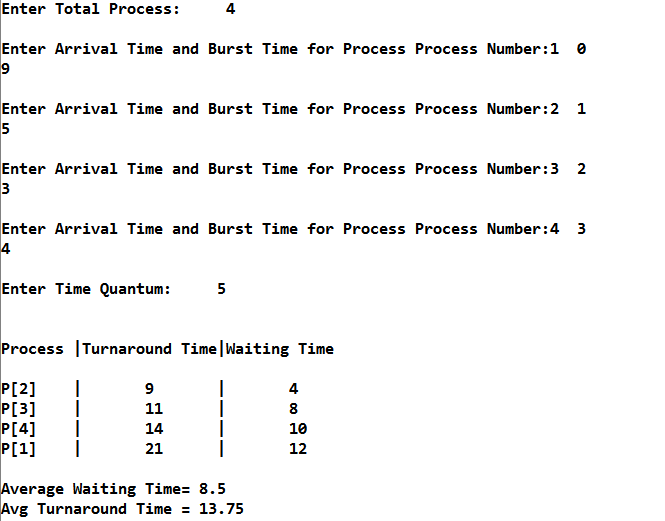


Figure 4.1 Screen-shot for Output of Round Robin Scheduling.

1. Program for Peterson Solution for two Problems in C++.

#include <iostream>

#include<conio.h>

#include<process.h>

using namespace std;

int flag[2]={0,0};

int turn;

void process0()

{

flag[0]=1;

turn=1;

while((turn==1)&&(flag[1]==1));

{

cout<<"\nCritical Section for Process 0 .... ";

}

flag[0]=0;

}

void process1()

{

flag[1]=1;

turn=0;

while((turn==0)&&(flag[0]==1));

{

cout<<"\nCritical Section for Process 1 ....";

}

flag[1]=0;

}

int main()

{

int c;

while(1)

{

cout<<"\n------Menu-------";

cout<<"\n1.Process 0 \n2.Process 1 \n3.Exit ";

cout<<"\nPlease Enter your Choice ";

cin>>c;

switch(c)

{

case 1:

process0();

break;

case 2:

process1();

break;

case 3:

break;

}

}

return 0;

}

Output –

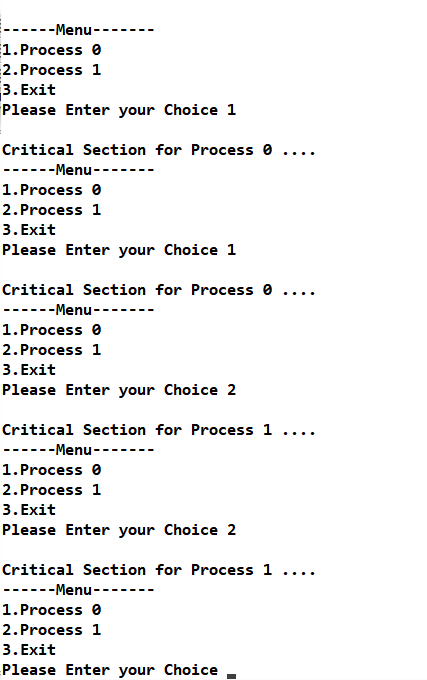


Figure 5.1 Screen-shot for Output of Peteson solution Problem.

1. Program for Producer Consumer solution in C++.

#include <iostream>

#include<process.h>

#include<conio.h>

using namespace std;

int mutex=1,full=0,empty=3,x=0;

int main()

{

int n;

void producer();

void consumer();

int wait(int);

int singal(int);

cout<<"\n1.Producer \n2.Consumer \n3.Exit";

while(1)

{

cout<<"\n Enter your choice ........ : ";

cin>>n;

switch(n)

{

case 1:

if((mutex==1)&&(empty!=0))

{ producer(); }

else

{ cout<<"\nBuffer is Full........ "; }

break;

case 2:

if((mutex==1)&&(full!=0))

{ consumer(); }

else

{ cout<<"\nBuffer is Empty ......."; }

break;

case 3:

break;

break;

}

}

return 0;

}

int wait(int s)

{ return (--s); }

int signal(int s)

{ return (++s); }

void producer()

{ mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

cout<<"\nProducer Produces the item ------ "<<x;

mutex=signal(mutex);

}

void consumer()

{ mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

cout<<"\nConsumer Consumes the item ------- "<<x;

x--;

mutex=signal(mutex);

}

Output –

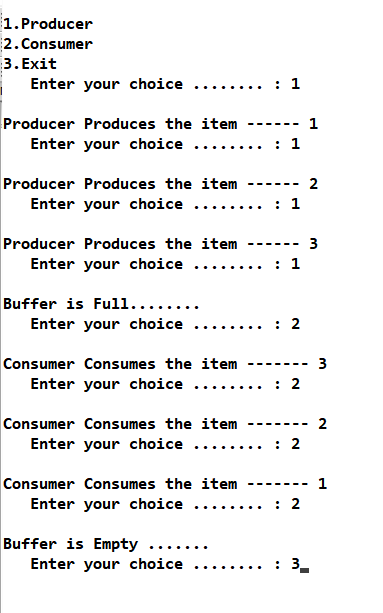


Figure 6.1 Screen-shot for Output of Producer-Consumer Problem.

1. Program for FIFO Page Replacement in C++.

#include <iostream>

#include<conio.h>

using namespace std;

int top=0,pf=0;

int main()

{

int flag;

int i,j,k;

int page[12]={7,0,1,2,0,3,0,4,2,3,0,3};

int frame[3]={-1,-1,-1};

cout<<"\nPages to be entered are : {7,0,1,2,0,3,0,4,2,3,0,3} ";

for(i=0;i<12;i++)

{

flag=0;

for(j=0;j<3;j++)

{

if(frame[j]==page[i])

{

flag=1;

break;

}

}

if(flag==0)

{

frame[top]=page[i];

top++;

cout<<"\n PAge Fault on "<<pf<<" is Encountered.";

pf++;

if(top>=3)

{

top=0;

}

}

for(k=0;k<3;k++)

{

cout<<" "<<frame[k];

}

}

cout<<"\nNumber of page fault : "<<pf;

getch();

return 0;

}

Output –

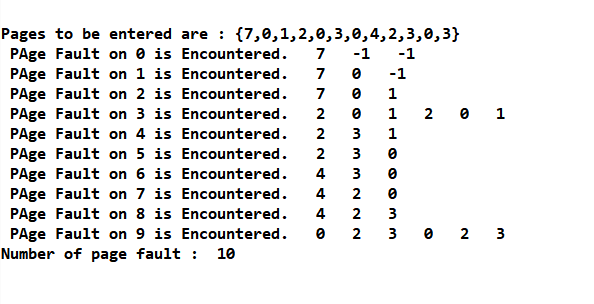


Figure 7.1 Screen-shot for Output of FIFO Page Replacement.

1. Program for LRU Page Replacement in C++.

#include<iostream>

using namespace std;

#include<conio.h>

int main()

{ int nop,nof,page[20],i,count=0;

cout<<"\n\tEnter the No. of Pages:";

cin>>nop; //Store the no of Pages

cout<<"\n\t Enter the Reference String:";

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

{ cout<<"\t";

cin>>page[i]; //Store the pages

}

cout<<"\n\t Enter the No of frames:-";

cin>>nof;

int frame[nof],fcount[nof];

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

{

frame[i]=-1; //Store the frames

fcount[i]=0; //Track when the page is last used

}

i=0;

while(i<nop)

{ int j=0,flag=0;

while(j<nof) {

if(page[i]==frame[j])

{ //Checking whether page already exist in frames or not

flag=1;

fcount[j]=i+1;

}

j++;

}

j=0;

cout<<"\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout<<"\t"<<page[i]<<"-->";

if(flag==0)

{

int min=0,k=0;

while(k<nof-1)

{

if(fcount[min]>fcount[k+1]) //Calculating the page which is least recently used

min=k+1;

k++;

}

frame[min]=page[i]; //Replacing

fcount[min]=i+1; //Increasing the time

count++; //counting Page Fault

while(j<nof)

{

cout<<"\t|"<<frame[j]<<"|";

j++; }

} i++;

}

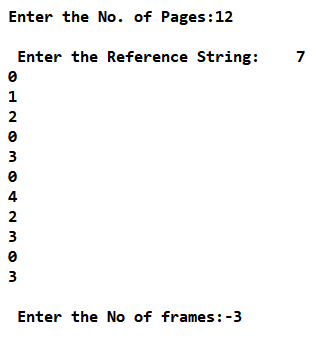
cout<<"\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout<<"\n\tPage Fault is:"<<count;

return 0;

}

Output –



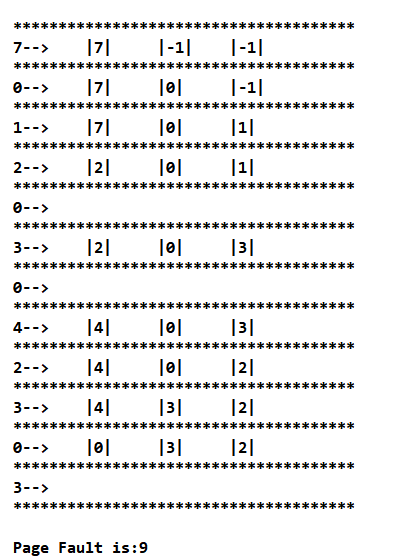


Figure 8: Screen shot of Output of LRU page replacement.

1. Program for OPR Page Replacement in C++.

#include<iostream>

using namespace std;

#include<conio.h>

int main()

{ int nop,nof,page[20],i,count=0;

cout<<"\n\tEnter the No. of Pages:";

cin>>nop; //Store the no of pages

cout<<"\n\t Enter the Reference String:";

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

{ cout<<"\t";

cin>>page[i]; //Array for Storing Reference String

}

cout<<"\n\t Enter the No of frames:-";

cin>>nof;

int frame[nof],fcount[nof];

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

{frame[i]=-1; //Frame Array

fcount[i]=0; // Track the next Availability of frames

}

i=0;

while(i<nop) {

int j=0,flag=0;

while(j<nof)

{if(page[i]==frame[j]){ // Checking Whether the Page is Already in frame or not

flag=1; }

j++;

}

j=0;

cout<<"\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout<<"\t"<<page[i]<<"-->";

if(flag==0)

{if(i>=nof)

{ int max=0,k=0;

while(k<nof)

{int dist=0,j1=i+1;

while(j1<nop)

{ if(frame[k]!=page[j1]) //Calculating Distances of pages

dist++;

else

{break; }

j1++;

} fcount[k]=dist; //Storing Distances into array

k++;

}k=0;

while(k<nof-1)

{

if(fcount[max]<fcount[k+1]) //Finding out the maxximum distance

max=k+1;

k++;

}frame[max]=page[i];

}

else { frame[i%nof]=page[i]; }

count++; // Increasing Page Fault.

while(j<nof)

{cout<<"\t|"<<frame[j]<<"|";

j++; }

}i++;

}

cout<<"\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout<<"\n\tPage Fault is:"<<count;

getch();

return 0;

}

Output –

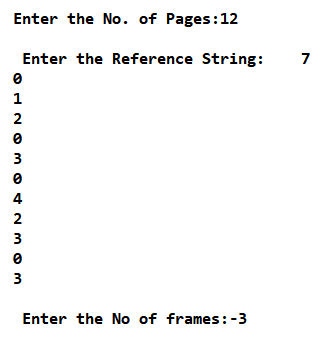
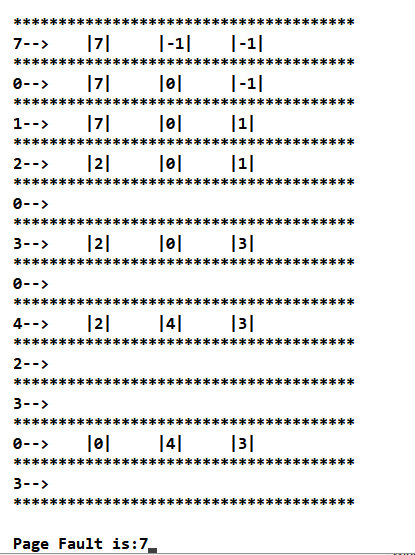
 

Figure 9.1: Screenshot for inputting pages string. Figure 9.2: Screenshot of Output of OPR.

1. Program for Reader Writer Problem in C++.

#include <iostream>

using namespace std;

#include<stdlib.h>

int wrt=1,readcount=0,mutex=1;

int wait(int);

int signal(int);

void write()

{ wrt=wait(wrt);

cout<<"\nThis is Write Operation in Critical Section...";

wrt=signal(wrt);

}

void read()

{ mutex=wait(mutex);

readcount++;

if(readcount==1)

{ wrt=wait(wrt); }

mutex=signal(mutex);

cout<<"\nNow We are in Reader section of Critical Section... ";

mutex=wait(mutex);

readcount--;

if(readcount==0)

{ wrt=signal(wrt); }

mutex=signal(mutex);

}

int wait(int s)

{ return (--s); }

int signal(int s)

{ return (++s); }

int main()

{ int choice;

while(1)

{ cout<<"\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

cout<<"\n1. Write\n2. Read \n3.Exit\n";

cout<<"\nMake your Choice .......... : ";

cin>>choice;

switch(choice)

{

case 1:

write();

break;

case 2:

read();

break;

case 3:

exit(1);

break;

}

}

return 0; }

Output –

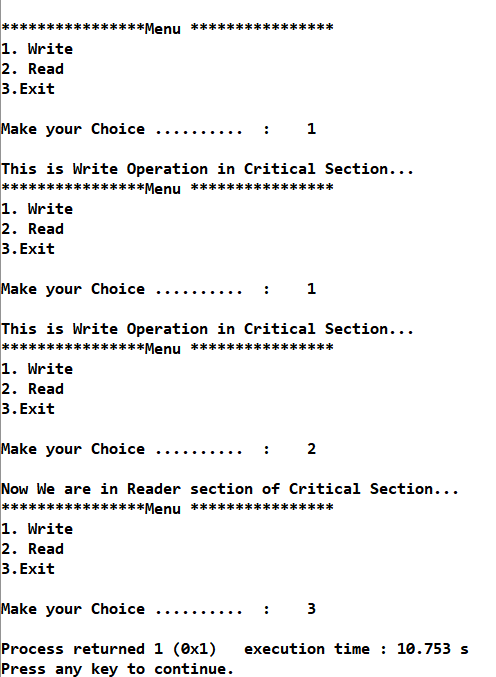


Figure 10: Screenshot of Reader-Writer Problem.