

# Experiment 1

## Simulate the following CPU scheduling algorithms: (a) FCFS (b) SJF

1. **FCFS(First Come First Serve) Program:**

#include<stdio.h> int main()

{

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

printf("Enter total number of processes(maximum 20):"); scanf("%d",&n);

printf("nEnter Process Burst Timen"); for(i=0;i<n;i++)

{

printf("P[%d]:",i+1);

scanf("%d",&bt[i]);

}

wt[0]=0;

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

{

wt[i]=0; for(j=0;j<i;j++)

wt[i]+=bt[j];

}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

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

{

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

printf("\n%d\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

}

avwt/=i; avtat/=i;

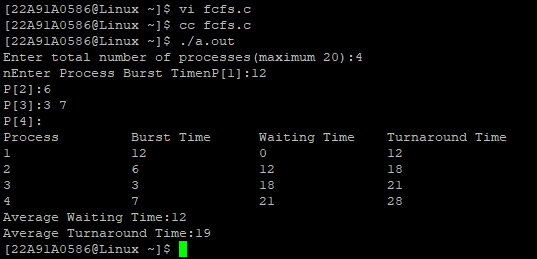
printf("\nAverage Waiting Time:%d",avwt); printf("\nAverage Turnaround Time:%d\n",avtat);

return 0;

}



**Output:**

****



## SJF (Shortest Job First) Program:

#include<stdio.h> int main()

{

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

printf("Enter number of process:"); scanf("%d",&n);

printf("\nEnter Burst Time:n"); for(i=0;i<n;i++)

{

printf("%d:",i+1);

scanf("%d",&bt[i]); p[i]=i+1;

}

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;

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; total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time"); for(i=0;i<n;i++)

{



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

printf("\n%d\t\t %d\t\t %d\t\t%d",p[i],bt[i],wt[i],tat[i]);

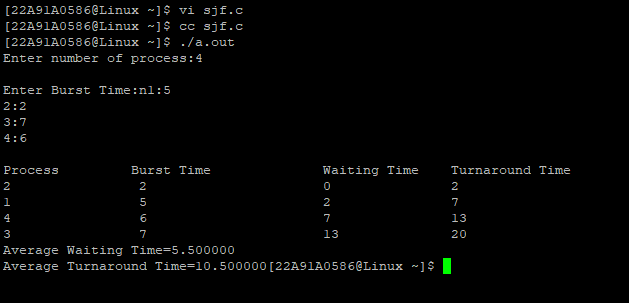
}

avg\_tat=(float)total/n;

printf("\nAverage Waiting Time=%f",avg\_wt); printf("\nAverage Turnaround Time=%f",avg\_tat);

}

**Output:**

****



# Experiment 2

## Simulate the following CPU scheduling algorithms: (a) Priority (b) Round Robin

1. **Priority**

**Program:**

#include <stdio.h> void swap(int \*a,int \*b)

{

int temp=\*a;

\*a=\*b;

\*b=temp;

}

int main()

{

int n,i,j,t=0,m,a[20];

printf("Enter Number of Processes: "); scanf("%d",&n);

int b[n],p[n],index[n]; for(i=0;i<n;i++)

{

printf("Enter Burst Time and Priority Value for Process %d: ",i+1); scanf("%d %d",&b[i],&p[i]);

index[i]=i+1;

}

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

{

int a=p[i],m=i; for(j=i;j<n;j++)

{

if(p[j] > a)

{

a=p[j]; m=j;

}

}

swap(&p[i], &p[m]);

swap(&b[i], &b[m]);

swap(&index[i],&index[m]);

}

printf("Order of process Execution is\n"); for(i=0;i<n;i++)

{

printf("%d is executed from %d to %d\n",index[i],t,t+b[i]); t+=b[i];

}

printf("\nProcess Id\t priority\tBurst Time\tWait Time\tTurnAround Time\n"); int wait\_time=0;

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

{

printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\n",index[i],p[i],b[i],wait\_time,wait\_time + b[i]); wait\_time += b[i];

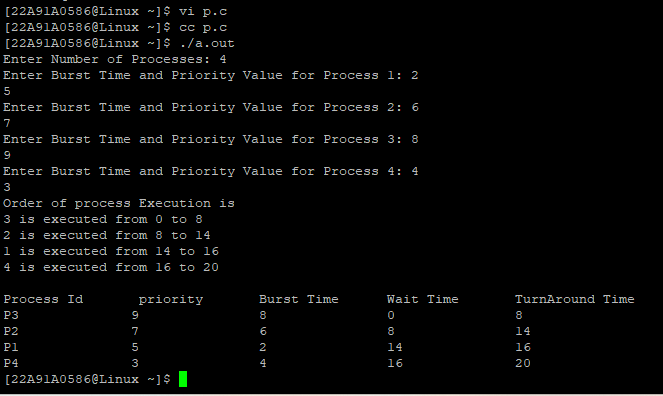
}

return 0;

}



**Output:**





## Round Robin

**Program:** #include<stdio.h> int main()

{

int i, limit, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10]; float average\_wait\_time, average\_turnaround\_time;

printf("\nEnter Total Number of Processes:"); scanf("%d", &limit);

x = limit;

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

{

printf("\nEnter Details of Process[%d]\n", i + 1); printf("Arrival Time:");

scanf("%d", &arrival\_time[i]); printf("Burst Time:"); scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

printf("Enter Time Quantum:"); scanf("%d", &time\_quantum);

printf("\nProcess ID\tBurst Time\tTurnaround Time\tWaiting Time"); for(total = 0, i = 0; x != 0;)

{

if(temp[i] <= time\_quantum && temp[i] > 0)

{

total = total + temp[i]; temp[i] = 0;

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum; total = total + time\_quantum;

}

if(temp[i] == 0 && counter == 1)

{

x--;

printf("\nP%d\t\t%d\t\t%d\t\t%d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i]; turnaround\_time = turnaround\_time + total - arrival\_time[i]; counter = 0;

}

if(i == limit - 1)

{

i = 0;



}

else if(arrival\_time[i + 1] <= total)

{

i++;

}

else

{

i = 0;

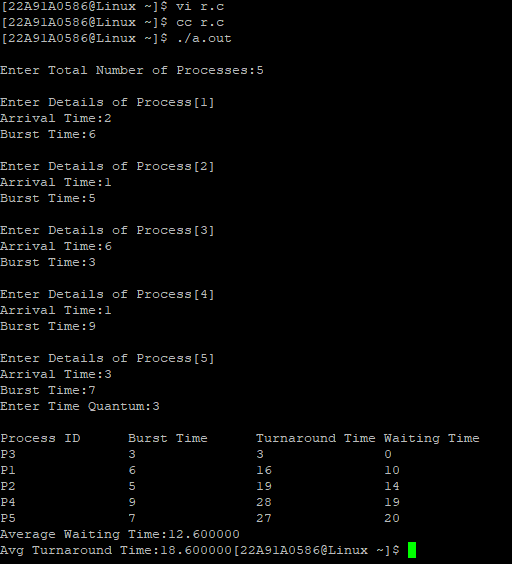
}

}

average\_wait\_time = wait\_time \* 1.0 / limit; average\_turnaround\_time = turnaround\_time \* 1.0 / limit; printf("\nAverage Waiting Time:%f", average\_wait\_time); printf("\nAvg Turnaround Time:%f", average\_turnaround\_time); return 0;

}

**Output:**

****



# Experiment 3

## Multiprogramming-Memory management-Implementation of fork (), wait (), exec() and exit (), System calls

**Program:** #include<unistd.h> #include<stdio.h> #include<stdlib.h> #include<sys/wait.h> int main()

{

pid\_t p; p=fork();

if(p==-1)

{

printf("Fork Error"); exit(0);

}

else if(p==0)

{

int i;

printf("Child PID is %d and PPID is %d\n",getpid(),getppid()); for(i=1;i<6;i++)

{

printf("Child i is %d\n",i);

}

\_exit(0);

}

else

{

int i;

printf("Parent PID is %d and PPID is %d\n",getpid(),getppid()); pid\_t p1=wait(0);

printf("PID=%d child ended",p1); for(i=1;i<6;i++)

{

printf("Parent i is %d\n",i);

}

exit(0);

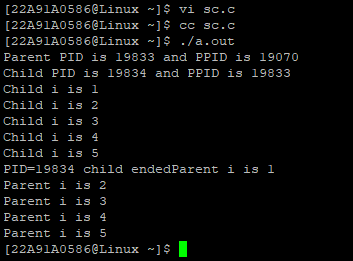
}

return 0;

}



**Output:**

****



# Experiment 4

## Simulate the Multiprogramming with a fixed number of tasks (MFT)

**Program:**

#include<stdio.h> #include<math.h> void main()

{

int np,nb,mm,bs,i,j,ps[100],nba[100],ifm[100],sb=0,flag=0; float x;

printf("Enter the Memory size: "); scanf("%d",&mm);

printf("Enter the no of Blocks: "); scanf("%d",&nb);

printf("Enter the no of processes: "); scanf("%d",&np);

bs=mm/nb;

for(i=1;(i<=np)&&(sb<nb);i++)

{

printf("Enter the size of p[%d]: ",i); scanf("%d",&ps[i]);

if(ps[i]<=bs)

nba[i]=1;

else

{

x=ps[i]/(float)bs; nba[i]=(ceil)(x);

}

ifm[i]=nba[i]\*bs-ps[i]; sb=sb+nba[i];

if(sb>nb)

{

i=i-1; flag=1;

}

}

j=i; printf("Process\tSize\tnba\tifm\n");

for(i=1;i<j;i++) printf("%d\t%d\t%d\t%d\n",i,ps[i],nba[i],ifm[i]);

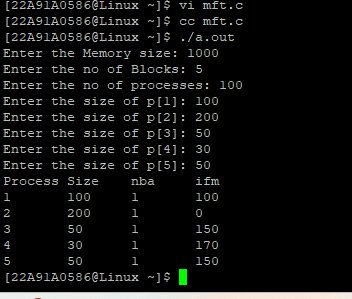
if(flag==1)

printf("Memory space is unavailable");

}



**Output:**

****



# Experiment 5

## Simulate the Multiprogramming with a variable number of tasks (MVT)

**Program:**

#include<stdio.h> void main()

{

int mm,np,ps[100],rm[100],am=0,flag=0,i,j; printf("Enter the memory size"); scanf("%d",&mm);

printf("enter no of processes"); scanf("%d",&np); for(i=0;(i<np)&&(am<mm);i++)

{

printf("Enter the size of p[%d]:",i+1); scanf("%d",&ps[i]);

am=am+ps[i]; if(am>=mm)

{

flag=1; break;

}

rm[i]=mm-am;

}

j=i; printf("Process\tsize\trm\n"); for(i=0;i<j;i++)

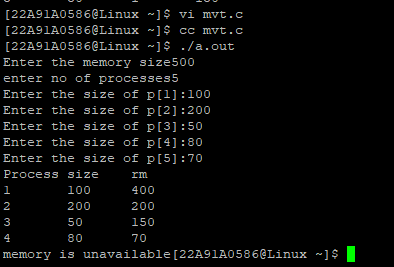
printf("%d\t%d\t%d\n",i+1,ps[i],rm[i]); if(flag==1)

printf("memory is unavailable");

}



**Output:**

****



# Experiment 6

## Simulate Bankers Algorithm for Dead Lock Avoidance

**Program:** #include <stdio.h> #include <stdlib.h> int main()

{

int Max[10][10],need[10][10],alloc[10][10],avail[10],completed[10], safeSequence[10] int p, r, i, j, process, count;

count = 0;

printf("Enter the no of processes : "); scanf("%d", &p);

for(i = 0; i< p; i++) completed[i] = 0;

printf("\nEnter the no of resources : "); scanf("%d", &r);

printf("\nEnter the Max Matrix for each process\n : "); for(i = 0; i < p; i++)

{

printf("For process %d : ", i + 1); for(j = 0; j < r; j++)

scanf("%d", &Max[i][j]);

}

printf("\nEnter the allocation for each process \n: "); for(i = 0; i < p; i++)

{

printf("For process %d : ",i + 1); for(j = 0; j < r; j++)

scanf("%d", &alloc[i][j]);

}

printf("\nEnter the Available Resources : "); for(i = 0; i < r; i++)

scanf("%d", &avail[i]); for(i = 0; i < p; i++)

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

need[i][j] = Max[i][j] - alloc[i][j]; do

{

printf("\n Max matrix:\tAllocation matrix:\n");

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

{

for( j = 0; j < r; j++) printf("%d ", Max[i][j]);

printf("\t\t");

for( j = 0; j < r; j++) printf("%d ", alloc[i][j]);

printf("\n");

}

process = -1;

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



{

if(completed[i] == 0)//if not completed

{

process = i ;

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

{

if(avail[j] < need[i][j])

{

process = -1; break;

}

}

}

if(process != -1) break;

}

if(process != -1)

{

printf("\nProcess %d runs to completion!", process + 1); safeSequence[count] = process + 1;

count++;

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

{

avail[j] += alloc[process][j]; alloc[process][j] = 0;

Max[process][j] = 0;

completed[process] = 1;

}

}

}

while(count != p && process != -1);

if(count == p)

{

printf("\nThe system is in a safe state!!\n"); printf("Safe Sequence : < ");

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

printf("%d ", safeSequence[i]); printf(">\n");

}

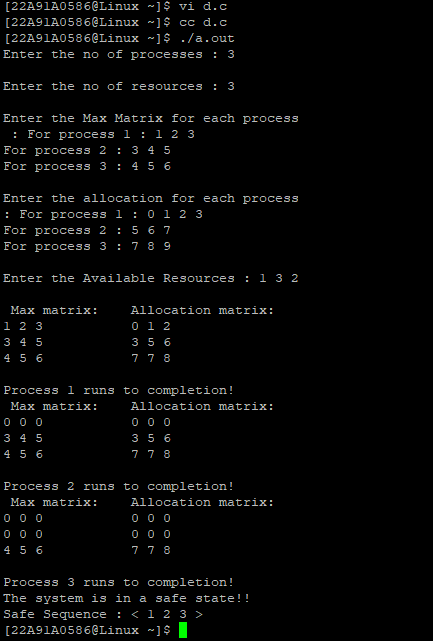
else

printf("\nThe system is in an unsafe state!!");

}



**Output:**

****



# Experiment 7

## Simulate the FIFO page replacement algorithm

**Program:** #include<stdio.h> #include<conio.h> void main()

{

int a[5],b[50],p=0,q=0,m=0,h,k,i,q1=1,j,u,r; char f='F';

printf("Enter the range: "); scanf("%d",&r); printf("Enter numbers:"); for(i=0;i<r;i++) scanf("%d",&b[i]);

printf("\nRefString PageFrame\n"); for(i=0;i<r;i++)

{

if(p==0)

{

if(q>=3) q=0;

a[q]=b[i]; q++;

if(q1<3)

{

q1=q;

}

}

printf("\n%d",b[i]); printf("\t"); for(h=0;h<q1;h++) printf("\t%d",a[h]);

if((p==0)&&(q1==3))

{ m++;

} p=0;

for(k=0;k<q-1;k++)

{

if(b[i+1]==a[k]) p=1;

}

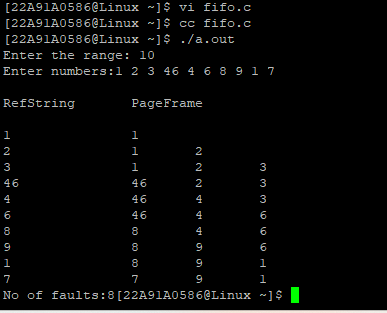
}

printf("\nNo of faults:%d",m);

}



**Output:**

****



# Experiment 8

## Simulate the LRU page replacement algorithm Program:

#include<stdio.h>

void main()

{

int g=0,r,a[5],b[20],p=0,q=0,m=0,h,k,i,q1=1,j,u; char f='F';

printf("Enter the range: "); scanf("%d",&r);

printf("Enter no: "); for(i=0;i<r;i++) scanf("%d",&b[i]); for(i=0;i<r;i++)

{

if(p==0)

{

if(q>=3) q=0;

a[q]=b[i]; q++;

if(q1<3)

{

q1=q; g=1;

}

}

printf("\n%d",b[i]); printf("\t"); for(h=0;h<q1;h++) printf("%d\t",a[h]);

if((p==0)&&(q1==3)&&(g!=1))

{

printf("-->%c",f); m++;

} p=0; g=0;

if(q1==3)

{

for(k=0;k<q-1;k++)

{

if(b[i+1]==a[k]) p=1;

}

for(j=0;j<q1;j++)

{ u=0;

k=i;

while(k>(i-2)&&(k>=0))

{

if(b[k]==a[j]) u++;



k--;

}

if(u==0) q=j;

}

}

else

{

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

{

if(b[i+1]==a[k]) p=1;

}

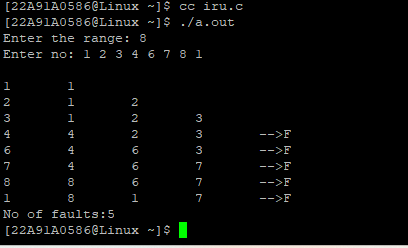
}

}

printf("\nNo of faults:%d\n",m);

}

**Output:**

****