**1. Simulate the following CPU scheduling algorithms**

**a) Round Robin b) SJF c) FCFS d) Priority**

**SJF**

#include <stdio.h>

int main()

{

int arrival\_time[10], burst\_time[10], temp[10];

int i, smallest, count = 0, time, limit;

double wait\_time = 0, turnaround\_time = 0, end;

float average\_waiting\_time, average\_turnaround\_time;

printf("Enter the Total Number of Processes:\t");

scanf("%d", &limit);

printf("\nEnter Details of %d Processes\n", limit);

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

{

printf("\n enter P%d Arrival Time:",i+1);

scanf("%d", &arrival\_time[i]);

printf("\n enter P%d Burst Time:",i+1);

scanf("%d", &burst\_time[i]);

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

}

burst\_time[9] = 9999;

for(time = 0; count != limit; time++)

{

smallest = 9;

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

{

if(arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] && burst\_time[i] > 0)

{

smallest = i;

}

}

burst\_time[smallest]--;

if(burst\_time[smallest] == 0)

{

count++;

end = time + 1;

wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

}

}

average\_waiting\_time = wait\_time / limit;

average\_turnaround\_time = turnaround\_time / limit;

printf("\nAverage Waiting Time:t%lf\nn", average\_waiting\_time);

printf("Average Turnaround Time:t%lf\n", average\_turnaround\_time);

return 0;

}

**FCFS**

#include<stdio.h>

main()

{

int n,a[10],b[10],t[10],w[10],g[10],i,m;

float att=0,awt=0;

for(i=0;i<10;i++)

{

a[i]=0; b[i]=0; w[i]=0; g[i]=0;

}

printf("enter the number of process");

scanf("%d",&n);

printf("\nenter the arrival times\n");

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

{

printf("for P%d\t",i+1);

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

}

printf("enter the burst times\n");

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

{

printf("For P%d\t",i+1);

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

}

g[0]=0;

for(i=0;i<10;i++)

g[i+1]=g[i]+b[i];

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

{

w[i]=g[i]-a[i];

t[i]=g[i+1]-a[i];

awt=awt+w[i];

att=att+t[i];

}

awt =awt/n;

att=att/n;

printf("\n\tprocess\twaiting time\tturn arround time\n");

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

{

printf("\tp%d\t\t%d\t\t%d\n",i,w[i],t[i]);

}

printf("the average waiting time is %f\n",awt);

printf("the average turn around time is %f\n",att);

return 0;

}

**Output:**

**Round Robin algorithm:**

#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:\t");

scanf("%d", &limit);

x = limit;

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

{

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

printf("Arrival Time:\t");

scanf("%d", &arrival\_time[i]);

printf("Burst Time:\t");

scanf("%d", &burst\_time[i]);

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

}

printf("\nEnter Time Quantum:\t");

scanf("%d", &time\_quantum);

printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

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("\nProcess[%d]\t\t%d\t\t %d\t\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("\n\nAverage Waiting Time:\t%f", average\_wait\_time);

printf("\nAvg Turnaround Time:\t%f\n", average\_turnaround\_time);

return 0;

}

**Output:**

**Priority:**

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n;

int total=0,pos,temp,avg\_wt,avg\_tat;

printf("Enter Total Number of Process:");

scanf("%d",&n);

printf("\nEnter Burst Time and Priority\n");

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

{

printf("\nP[%d]\n",i+1);

printf("Burst Time:");

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

printf("Priority:");

scanf("%d",&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;

printf("\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];

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

}

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

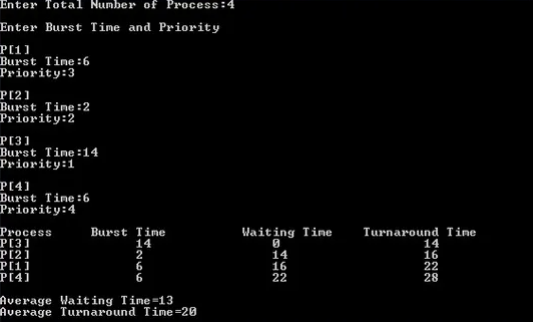
printf("\n\nAverage Waiting Time=%d",avg\_wt);

printf("\nAverage Turnaround Time=%d\n",avg\_tat);

return 0;

}

Output:



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

🡪Fork and Wait

#include<unistd.h>

#include<sys/types.h>

#include<stdio.h>

#include<sys/wait.h>

int main()

{

int p;

printf("before fork\n");

p=fork();

if(p==0)

{

pirntf("child process id is %d", getpid());

pirntf("parent process id is %d", getppid());

}

else

{

wait(NULL)

printf("chiid id is %d", p);

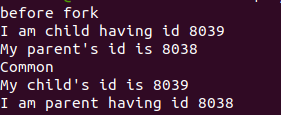
printf("parent id %d",getpid());

}

return 0;

}

Output:



🡪SYSTEM CALLS

There are four system calls for file management,

open ()

read ()

write ()

close ()

**1. open ()**

open() system call is used to know the file descriptor of user-created files. Since read and write use file descriptor as their 1st parameter so to know the file descriptor open() system call is used.

🡪Syntax:

fd = open (file\_name, mode, permission);

Example:

fd = open ("file", O\_CREAT | O\_RDWR, 0777);

Here,

file\_name is the name to the file to open.

mode is used to define the file opening modes such as create, read, write modes.

permission is used to define the file permissions.

Return value: Function returns the file descriptor.

**2. read ()**

read() system call is used to read the content from the file. It can also be used to read the input from the keyboard by specifying the 0 as file descriptor (see in the program given below).

🡪Syntax:

length = read(file\_descriptor , buffer, max\_len);

Example:

n = read(0, buff, 50);

Here,

file\_descriptor is the file descriptor of the file.

buffer is the name of the buffer where data is to be stored.

max\_len is the number specifying the maximum amount of that data can be read

Return value: If successful read returns the number of bytes actually read.

**3. write ()**

write() system call is used to write the content to the file.

🡪Syntax:

length = write(file\_descriptor , buffer, len);

Example:

n = write(fd, "Hello world!", 12);

Here,

file\_descriptor is the file descriptor of the file.

buffer is the name of the buffer to be stored.

len is the length of the data to be written.

Return value: If successful write() returns the number of bytes actually written.

**4. close ()**

close() system call is used to close the opened file, it tells the operating system that you are done with the file and close the file.

🡪Syntax:

int close(int fd);

Here,

fd is the file descriptor of the file to be closed.

Return value: If file closed successfully it returns 0, else it returns -1.

🡪C code to demonstrate example of system calls AND exit()

#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<stdio.h>

int main()

{

int n,fd;

char buff[50]; // declaring buffer

printf("Enter text to write in the file:\n");

//read from keyboard, specifying 0 as fd for std input device

//Here, n stores the number of characters

n= read(0, buff, 50);

// creating a new file using open.

fd=open("file",O\_CREAT | O\_RDWR, 0777);

//writting input data to file (fd)

write(fd, buff, n);

//Write to display (1 is standard fd for output device)

write(1, buff, n);

//closing the file

int close(int fd);

return 0;

}

**Output**

Enter text to write in the file:

Hello world, welcome

Hello world, welcome

System call: exec

#include<stdio.h>

void main(int argc, char \*argv[])

{

int pid;

/\* Fork another process \*/

pid = fork();

if(pid < 0)

{

//Error occurred

fprintf(stderr, "Fork Failed");

exit(-1);

}

else if (pid == 0)

{

//Child process

execlp("/bin/ls","ls",NULL);

}

else

{

//Parent process

//Parent will wait for the child to complete

wait(NULL);

printf("Child complete");

exit(0);

}

}

**Output:**

**3. Simulate the following**

**a) Multiprogramming with a fixed number of tasks (MFT)**

**b) Multiprogramming with a variable number of tasks (MVT)**

**MFT MEMORY MANAGEMENT TECHNIQUE**

#include<stdio.h>

#include<conio.h>

main()

{

int ms, bs, nob, ef,n, mp[10],tif=0;

int i,p=0;

clrscr();

printf("Enter the total memory available (in Bytes) -- ");

scanf("%d",&ms);

printf("Enter the block size (in Bytes) -- ");

scanf("%d", &bs);

nob=ms/bs;

ef=ms - nob\*bs;

printf("\nEnter the number of processes -- ");

scanf("%d",&n);

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

{

printf("Enter memory required for process %d (in Bytes)-- ",i+1);

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

}

printf("\nNo. of Blocks available in memory -- %d",nob);

printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL FRAGMENTATION");

for(i=0;i<n && p<nob;i++)

{

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

if(mp[i] > bs)

printf("\t\tNO\t\t---");

else

{

printf("\t\tYES\t%d",bs-mp[i]);

tif = tif + bs-mp[i];

p++;

}

}

if(i<n)

printf("\nMemory is Full, Remaining Processes cannot be accomodated");

printf("\n\nTotal Internal Fragmentation is %d",tif);

printf("\nTotal External Fragmentation is %d",ef);

getch();

}

**Output:**

Enter the total memory available (in Bytes) -- 1000

Enter the block size (in Bytes)-- 300

Enter the number of processes – 5

Enter memory required for process 1 (in Bytes) -- 275

Enter memory required for process 2 (in Bytes) -- 400

Enter memory required for process 3 (in Bytes) -- 290

Enter memory required for process 4 (in Bytes) -- 293

Enter memory required for process 5 (in Bytes) -- 100

No. of Blocks available in memory -- 3

PROCESS MEMORY-REQUIRED ALLOCATED INTERNAL-FRAGMENTATION

1 275 YES 25

2 400 NO -----

3 290 YES 10

4 293 YES 7

Memory is Full, Remaining Processes cannot be accommodated

Total Internal Fragmentation is 42

Total External Fragmentation is 100

**MVT MEMORY MANAGEMENT TECHNIQUE**

#include<stdio.h>

#include<conio.h>

main()

{

int ms,mp[10],i, temp,n=0;

char ch = 'y';

clrscr();

printf("\nEnter the total memory available (in Bytes)-- ");

scanf("%d",&ms);

temp=ms;

for(i=0;ch=='y';i++,n++)

{

printf("\nEnter memory required for process %d (in Bytes) -- ",i+1);

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

if(mp[i]<=temp)

{

printf("\nMemory is allocated for Process %d ",i+1);

temp = temp - mp[i];

}

else

{

printf("\nMemory is Full");

break;

}

printf("\nDo you want to continue(y/n) -- ");

scanf(" %c", &ch);

}

printf("\n\nTotal Memory Available -- %d", ms);

printf("\n\n\tPROCESS\t\t MEMORY ALLOCATED ");

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

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

printf("\n\nTotal Memory Allocated is %d",ms-temp);

printf("\nTotal External Fragmentation is %d",temp);

getch();

}

**OUTPUT**

Enter the total memory available (in Bytes) -- 1000

Enter memory required for process 1 (in Bytes) -- 400

Memory is allocated for Process 1

Do you want to continue(y/n) -- y

Enter memory required for process 2 (in Bytes) -- 275

Memory is allocated for Process 2

Do you want to continue(y/n) -- y

Enter memory required for process 3 (in Bytes) -- 550

Memory is Full

Total Memory Available -- 1000

PROCESS MEMORY-ALLOCATED

1 400

2 275

Total Memory Allocated is 675

Total External Fragmentation is 325

**6. Simulate the following page replacement algorithms.**

**a) FIFO b) LRU c) LFU**

**FIFO**

#include<stdio.h>

int main()

{

int i, j, k, f, pf=0, count=0, rs[25], m[10], n;

printf("\n Enter the length of reference string -- ");

scanf("%d",&n);

printf("\n Enter the reference string -- ");

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

printf("\n Enter no. of frames -- ");

scanf("%d",&f);

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

m[i]=-1;

printf("\n The Page Replacement Process is -- \n");

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

{

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

{

if(m[k]==rs[i]) break;

}

if(k==f)

{ m[count++]=rs[i];

pf++;

}

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

printf("\t%d",m[j]);

if(k==f)

printf("\tPF No. %d",pf);

printf("\n");

if(count==f)

count=0;

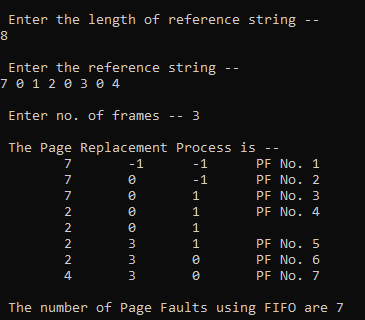
}

printf("\n The number of Page Faults using FIFO are %d",pf);

return 0;

}

**Output:**



**LRU:**

#include<stdio.h>

int main()

{

int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;

printf("Enter the length of reference string -- ");

scanf("%d",&n);

printf("Enter the reference string -- ");

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

{

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

flag[i]=0;

}

printf("Enter the number of frames -- ");

scanf("%d",&f);

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

{

count[i]=0;

m[i]=-1;

}

printf("\nThe Page Replacement process is -- \n");

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

{

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

{

if(m[j]==rs[i])

{

flag[i]=1;

count[j]=next;

next++;

}

}

if(flag[i]==0)

{

if(i<f)

{

m[i]=rs[i];

count[i]=next;

next++;

}

else

{

min=0;

for(j=1;j<f;j++)

if(count[min] > count[j])

min=j;

m[min]=rs[i];

count[min]=next;

next++;

}

pf++;

}

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

printf("%d\t", m[j]);

if(flag[i]==0)

printf("PF No. -- %d" , pf);

printf("\n");

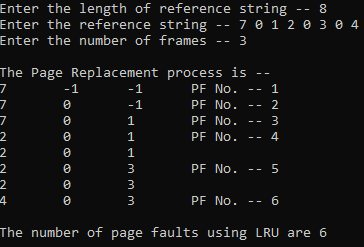
}

printf("\nThe number of page faults using LRU are %d",pf);

return 0;

}

**Output:**



**LFU**

#include<stdio.h>

main()

{

int rs[50], i, j, k, m, f, cntr[20], a[20], min, pf=0;

printf("\nEnter number of page references -- ");

scanf("%d",&m);

printf("\nEnter the reference string -- ");

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

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

printf("\nEnter the available no. of frames -- ");

scanf("%d",&f);

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

{

cntr[i]=0;

a[i]=-1;

}

printf("\nThe Page Replacement Process is – \n");

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

{

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

if(rs[i]==a[j])

{

cntr[j]++;

break;

}

if(j==f)

{

min = 0;

for(k=1;k<f;k++)

if(cntr[k]<cntr[min])

min=k;

a[min]=rs[i];

cntr[min]=1;

pf++;

}

printf("\n");

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

printf("\t%d",a[j]);

if(j==f)

printf("\tPF No. %d",pf);

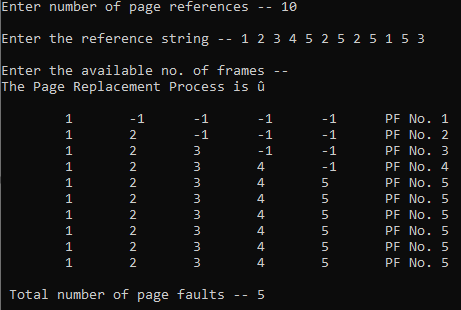
}

printf("\n\n Total number of page faults -- %d",pf);

return 0;

}

**Output:**

****

**4. Simulate Bankers Algorithm for Dead Lock Avoidance**

//C program for Banker's Algorithm

#include <stdio.h>

int main()

{

// P0, P1, P2, P3, P4 are the names of Process

int n, r, i, j, k;

n = 5; // Indicates the Number of processes

r = 3; //Indicates the Number of resources

int alloc[5][3] = { { 0, 0, 1 }, // P0 // This is Allocation Matrix

{ 3, 0, 0 }, // P1

{ 1, 0, 1 }, // P2

{ 2, 3, 2 }, // P3

{ 0, 0, 3 } }; // P4

int max[5][3] = { { 7, 6, 3 }, // P0 // MAX Matrix

{ 3, 2, 2 }, // P1

{ 8, 0, 2 }, // P2

{ 2, 1, 2 }, // P3

{ 5, 2, 3 } }; // P4

int avail[3] = { 2, 3, 2 }; // These are Available Resources

int f[n], ans[n], ind = 0;

for (k = 0; k < n; k++) {

f[k] = 0;

}

int need[n][r];

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

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

need[i][j] = max[i][j] - alloc[i][j];

}

int y = 0;

for (k = 0; k < 5; k++) {

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

if (f[i] == 0) {

int flag = 0;

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

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

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

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

printf("Th SAFE Sequence is as follows\n");

for (i = 0; i < n - 1; i++)

printf(" P%d ->", ans[i]);

printf(" P%d", ans[n - 1]);

return (0);

}

**Output:**

****

### [5. Write a C program to simulate the contiguous memory allocation    technique.](http://studice.blogspot.in/p/blog-page_7.html)

### First Fit b. BEST-FIT

### First Fit

### #include<stdio.h>

### #define max 25

### int main()

### {

### int frag[max],b[max],f[max],i,j,nb,nf,temp;

### static int bf[max],ff[max];

### printf("\n\tMemory Management Scheme - First Fit");

### printf("\nEnter the number of blocks:");

### scanf("%d",&nb);

### printf("Enter the number of files:");

### scanf("%d",&nf);

### printf("\nEnter the size of the blocks:-\n");

### for(i=1;i<=nb;i++)

### {

### printf("Block %d:",i);

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

### }

### printf("Enter the size of the files :-\n");

### for(i=1;i<=nf;i++)

### {

### printf("File %d:",i);

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

### }

### for(i=1;i<=nf;i++)

### {

### for(j=1;j<=nb;j++)

### {

### if(bf[j]!=1)

### {

### temp=b[j]-f[i];

### if(temp>=0)

### {

### ff[i]=j;

### break;

### }

### }

### }

### frag[i]=temp; bf[ff[i]]=1;

### }

### printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");

### for(i=1;i<=nf;i++)

### printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

### return 0;

### }

### Output:

### 

### Best Fit

### #include<stdio.h>

### #define max 25

### int main()

### {

### int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;

### static int bf[max],ff[max];

### printf("\n\tMemory Management Scheme – Best Fit");

### printf("\nEnter the number of blocks:");

### scanf("%d",&nb);

### printf("Enter the number of files:");

### scanf("%d",&nf);

### printf("\nEnter the size of the blocks:-\n");

### for(i=1;i<=nb;i++)

### {

### printf("Block %d:",i);

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

### }

### printf("Enter the size of the files :-\n");

### for(i=1;i<=nf;i++)

### {

### printf("File %d:",i);

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

### }

### for(i=1;i<=nf;i++)

### {

### for(j=1;j<=nb;j++)

### {

### if(bf[j]!=1)

### {

### temp=b[j]-f[i];

### if(temp>=0)

### if(lowest>temp)

### {

### ff[i]=j;

### lowest=temp;

### }

### }

### }

### frag[i]=lowest;

### bf[ff[i]]=1;

### lowest=10000;

### }

### printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");

### for(i=1;i<=nf && ff[i]!=0;i++)

### printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

### return 0;

### }

### Output:

### 