Fork () system call

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main() {
   pid_t pid;
  // Create a child process
   pid = fork();

   if (pid < 0) {
      printf( "Fork failed\n");
      return 1;
   } else if (pid == 0) {
      // Child process
      printf("This is the child process (PID: %d)\n", getpid());
   } else {
      // Parent process
      printf("This is the parent process (PID: %d)\n", getpid());
      printf("Child's PID: %d\n", pid);
   }  return 0;
}
```

Output

```
This is the parent process (PID: 12345)
Child's PID: 12346
This is the child process (PID: 12346)
```

opendir,readdir , closedir system calls

```
#include<stdio.h>
#include<dirent.h>
#include<string.h>
struct dirent *file;
void main(){char str[50];
DIR * directory;
printf("Enter name of directory");
gets(str);
directory=opendir(str);
if(directory==NULL){
printf("The directory does not exsist");
}
else{
file=readdir(directory);
while(file!=NULL){
puts(file->d_name);
file=readdir(directory);
}
}
```

```
closedir(directory);
}
```

Output

```
Enter name of directory     /home
directory.c
hello.c
new.txt
new2.txt
forkachild.c
a.out
combined.txt
```

execv() system call

//execv.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main()
{      printf("Making execv () call from execv.c program \n");
                        char *args[]={"./hello",NULL};
                        execv(args[0],args);
                        printf("Ending-----");
}
```

//Hello.c

```
#include <stdio.h>
void main(){
   printf("Printing ' Hello World ' from hello.c\n");
}
```

Output

```
runner20@Check:/home$ gcc execv.c -o execv
runner20@Check:/home$ gcc hello.c -o hello
runner20@Check:/home$ ./execv
Making execv () call from execv.c program

Printing ' Hello World ' from hello.c
```

Grep command simulation

```
#include<stdio.h>
#include<fcntl.h>
#include<string.h>

void main()
{
char fn[100],pat[10],temp[1000];
FILE *fp;
printf("\n Enter file name : ");
gets(fn);
printf("Enter the pattern to be searched : ");
gets(pat);
fp=fopen(fn,"r");
while(!feof(fp)){
fgets(temp,1000,fp);
if(strstr(temp,pat))
printf("%s",temp);}
fclose(fp);
}
```

cp command

```
#include<fcntl.h>
#include<unistd.h>
#include<stdio.h>
#include<string.h>
void main()
{
FILE *fp,*fp2;
char ch;
int sc=0;
printf("Enter the path of file to copy from");
char f1[100],f2[100];
gets(f1);
printf("Enter the path of file to copy to");
gets(f2);
fp=fopen(f1,"r");
fp2=fopen(f2,"w");
if(fp==NULL)
printf("unable to open a file");
else
{
while(!feof(fp))
{
ch=fgetc(fp);
fputc(ch,fp2);
if(ch==' ')
```

```
sc++;
}}



printf("no of spaces %d",sc);
printf("\n");
fclose(fp);
}
```

Output

```
Enter the path of file to copy fromnew.txt
Enter the path of file to copy tonew2.txt
no of spaces 49
```

ls command

```
#include<stdio.h>
#include<dirent.h>
#include<string.h>
struct dirent *file;
void main(){char str[50];
DIR * directory;
printf("Enter name of directory");
gets(str);
directory=opendir(str);
if(directory==NULL){
printf("The directory does not exsist");
}
else{
file=readdir(directory);
while(file!=NULL){
puts(file->d_name);
file=readdir(directory);
}
}
closedir(directory);
}
```

Output

```
Enter name of directory     /home
directory.c
hello.c
new.txt
new2.txt
forkachild.c
a.out
combined.txt
```

Shell programming

To check if a number is odd or even

```
echo "Enter the Number"
read n
r=`expr $n % 2`
if [ $r -eq 0 ]
then
echo "$n is Even number"
else
echo "$n is Odd number"
fi
```

Output

```
Enter the Number
453
453 is Odd number
```

Find factorial of a number

```
echo "Enter a Number"
read n
i=`expr $n - 1`
p=1
while [ $i -ge 1 ]
do
n=`expr $n \* $i`
i=`expr $i - 1`
done
echo "The Factorial of the given Number is $n"
```

Output

```
Enter a Number
5
The Factorial of the given Number is 120
```

Swap two numbers

```
echo "Enter Two Numbers"
read a b
temp=$a
a=$b
b=$temp
echo "after swapping"
echo $a $b
```

Output

```
Enter Two Numbers    7 34
after swapping
34 7
```

Priority scheduling

```
#include<stdio.h>
struct Process {
   int pid;       // Process ID
   int burst_time; // Burst time
   int priority;
   int waittime;
};

void priorityScheduling(struct Process proc[], int n) {
   int i, j;
   struct Process temp;
// Sort the processes based on priority
   for (i = 0; i < n - 1; i++) {
      for (j = 0; j < n - i - 1; j++) {
         if (proc[j].priority > proc[j + 1].priority) {
            // Swap the processes
            temp = proc[j];
            proc[j] = proc[j + 1];
            proc[j + 1] = temp;
         } }  }
// Calculate waiting time, turnaround time, and average waiting time
   int total_waiting_time = 0;
   int total_turnaround_time = proc[0].burst_time;
 proc[0].waittime=0;
   for (i = 1; i < n; i++) {
      proc[i].waittime=proc[i-1].waittime+proc[i-1].burst_time;
      total_turnaround_time+=proc[i].waittime+proc[i].burst_time;
      total_waiting_time+=proc[i].waittime;
      }
   printf("Process\tBurst Time\tPriority\tWaiting Time \t Turn around time\n");
   for (i = 0; i < n; i++) {
      printf("%d\t%d\t\t%d\t%d\t%d\n", proc[i].pid, proc[i].burst_time,
proc[i].priority,proc[i].waittime,proc[i].waittime+proc[i].burst_time);
   }
   printf("\nAverage Waiting Time: %.2f\n", (float) total_waiting_time / n);
   printf("Average Turnaround Time: %.2f\n", (float) total_turnaround_time / n);
}

void main() {
   int n, i;
   printf("Enter the number of processes: ");
   scanf("%d", &n);
   struct Process proc[n];
   printf("Enter process details (Process ID, Burst Time, Priority):\n");
   for (i = 0; i < n; i++) {
      printf("Process %d: ", i + 1);
```

```
        scanf("%d %d %d", &proc[i].pid, &proc[i].burst_time, &proc[i].priority);
    }
    // Call the priority scheduling function
    priorityScheduling(proc, n);
}
```

Output

```
Enter the number of processes: 5
Enter process details (Process ID, Burst Time, Priority):
Process 1: 1
10
3
Process 2: 2
1
1
Process 3: 3
2
4
Process 4: 4
1
5
Process 5: 5
5
2
Process Burst Time      Priority       Waiting Time     Turn around time
2           1                  1           0                  1
5           5                  2          1                 6
1           10                  3           6                 16
3           2                  4         16                  18
4            1                  5          18                19

Average Waiting Time: 8.20
Average Turnaround Time: 12.00
```

Round robin

```
#include<stdio.h>
 int main()
{   int  n;
    printf("Enter Total Number of Processes:");
    scanf("%d", &n);
    int wait_time = 0, ta_time = 0, arr_time[n], burst_time[n], temp_burst_time[n];
    int x = n;
    for(int i = 0; i < n; i++)
    {
       printf("Enter Details of Process %d \n", i + 1);
       printf("Arrival Time:  ");
       scanf("%d", &arr_time[i]);
       printf("Burst Time:   ");
       scanf("%d", &burst_time[i]);
       temp_burst_time[i] = burst_time[i];
    }
    int time_slot;
    printf("Enter Time Slot:");
    scanf("%d", &time_slot);
    int total = 0,  counter = 0,i;
    printf("Process ID      Burst Time      Turnaround Time      Waiting Time\n");
    for(total=0, i = 0; x!=0; )
    {
       if(temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0)
       {
          total = total + temp_burst_time[i];
          temp_burst_time[i] = 0;
          counter=1;
       }
       else if(temp_burst_time[i] > 0)
       {
          temp_burst_time[i] = temp_burst_time[i] - time_slot;
          total  += time_slot;
       }
       if(temp_burst_time[i]==0 && counter==1)
       {   x--; //decrement the process no.
          printf("\nProcess No %d  \t\t %d\t\t\t\t %d\t\t\t %d", i+1, burst_time[i],
                total-arr_time[i], total-arr_time[i]-burst_time[i]);
          wait_time = wait_time+total-arr_time[i]-burst_time[i];
          ta_time += total -arr_time[i];
          counter =0;
       }
       if(i==n-1)
       {  i=0;   }
       else if(arr_time[i+1]<=total)
       {  i++;
```

```
      }
      else
      {  i=0;
      }
   }
   float average_wait_time = wait_time * 1.0 / n;
   float average_turnaround_time = ta_time * 1.0 / n;
   printf("\nAverage Waiting Time:%f", average_wait_time);
   printf("\nAvg Turnaround Time:%f", average_turnaround_time);
   return 0;
}
```

Output

```
Enter Total Number of Processes:3
Enter Details of Process 1
Arrival Time:  0
Burst Time:   10
Enter Details of Process 2
Arrival Time:  1
Burst Time:   8
Enter Details of Process 3
Arrival Time:  2
Burst Time:   7
Enter Time Slot:5
Process ID      Burst Time      Turnaround Time      Waiting Time

Process No 1            10                      20                      10
Process No 2            8                       22                      14
Process No 3            7                       23                      16
Average Waiting Time:13.333333
Avg Turnaround Time:21.666666
```

First Come first serve

```
#include<stdio.h>
struct Process {
   int pid;       // Process ID
   int burst_time; // Burst time
   int at;
   int waittime;
};
void priorityScheduling(struct Process proc[],
int n) {
   int i, j;
   struct Process temp;
   for (i = 0; i < n - 1; i++) {
      for (j = 0; j < n - i - 1; j++) {
         if (proc[j].at > proc[j + 1].at) {
            // Swap the processes
            temp = proc[j];
            proc[j] = proc[j + 1];
            proc[j + 1] = temp;
         } } }
   int total_waiting_time = 0;
   int total_turnaround_time =
proc[0].burst_time;
   proc[0].waittime=0;

   for (i = 1; i < n; i++) {
      proc[i].waittime=proc[i-
1].waittime+proc[i-1].burst_time;

total_turnaround_time+=proc[i].waittime+pro
c[i].burst_time;
      total_waiting_time+=proc[i].waittime;
   }
   printf("Process\tBurst Time\tArrival
Time\tWaiting Time \t Turn around time\n");
   for (i = 0; i < n; i++) {
      printf("%d\t%d\t\t%d\t%d\t%d\n",
proc[i].pid, proc[i].burst_time,
proc[i].at,proc[i].waittime,proc[i].waittime+pr
oc[i].burst_time);
   }
   printf("\nAverage Waiting Time: %.2f\n",
(float) total_waiting_time / n);
   printf("Average Turnaround Time: %.2f\n",
(float) total_turnaround_time / n);
}
void main() {
   int n, i;
   printf("Enter the number of processes: ");
   scanf("%d", &n);
 struct Process proc[n];
   printf("Enter process details (Process ID,
Burst Time, Arrival Time):\n");
   for (i = 0; i < n; i++) {
      printf("Process %d: ", i + 1);
      scanf("%d %d %d", &proc[i].pid,
&proc[i].burst_time, &proc[i].at);
   }
   priorityScheduling(proc, n);
}
```

Output

```
Enter the number of processes: 4
Enter process details (Process ID, Burst Time,
Arrival Time):
Process 1: 1
3
0
Process 2: 2
6
1
Process 3: 3
4
4
Process 4: 4
2
6
Process Burst Time      Arrival Time
        Waiting Time    Turn around time
1       3               0       0       3
2       6               1       3       9
3       4               4       9       13
4       2               6       13      15

Average Waiting Time: 6.25
Average Turnaround Time: 10.00
```

Shortest job first

```
#include<stdio.h>

struct Process {
   int pid;       // Process ID
   int burst_time; // Burst time
   int waittime;
};

void sjfScheduling(struct Process proc[], int n)
{
   int i, j;
   struct Process temp;
   for (i = 0; i < n - 1; i++) {
      for (j = 0; j < n - i - 1; j++) {
         if (proc[j].burst_time > proc[j +
1].burst_time) {
            // Swap the processes
            temp = proc[j];
            proc[j] = proc[j + 1];
            proc[j + 1] = temp;
         } } }
   int total_waiting_time = 0;
   int total_turnaround_time =
proc[0].burst_time;
   proc[0].waittime=0;

   for (i = 1; i < n; i++) {
      proc[i].waittime=proc[i-
1].waittime+proc[i-1].burst_time;

total_turnaround_time+=proc[i].waittime+pro
c[i].burst_time;
      total_waiting_time+=proc[i].waittime;
   }
   printf("Process\tBurst Time\tWaiting Time
\t Turn around time\n");
   for (i = 0; i < n; i++) {
      printf("%d\t%d\t\t\t%d\t%d\n",
proc[i].pid,
proc[i].burst_time,proc[i].waittime,proc[i].wai
ttime+proc[i].burst_time);
   }
   printf("\nAverage Waiting Time: %.2f\n",
(float) total_waiting_time / n);
   printf("Average Turnaround Time: %.2f\n",
(float) total_turnaround_time / n);
}

void main() {
   int n, i;
   printf("Enter the number of processes: ");
   scanf("%d", &n);
 struct Process proc[n];
   printf("Enter process details (Process ID,
Burst Time):\n");
   for (i = 0; i < n; i++) {
      printf("Process %d: ", i + 1);
      scanf("%d %d", &proc[i].pid,
&proc[i].burst_time);
   }
   sjfScheduling(proc, n);
}
```

Output

```
Enter the number of processes: 4
Enter process details (Process ID, Burst Time):
Process 1: 1
6
Process 2: 2
8
Process 3: 3
7
Process 4: 4
3
Process Burst Time        Waiting Time     Turn
around time
4        3                        0        3
1        6                        3        9
3        7                        9        16
2        8                        16       24

Average Waiting Time: 7.00
Average Turnaround Time: 13.00
```

IPC

Write.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{    int i;             void *shared_memory;
char buff[100];  int shmid;
shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);
printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0);
printf("Process attached at %p\n",shared_memory);
printf("Enter some data to write to shared memory\n");
gets(buff);
strcpy(shared_memory,buff);
printf("You wrote : %s\n",(char *)shared_memory);
}
```

Read.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{    int i;    void *shared_memory;  char buff[100];
int shmid;
shmid=shmget((key_t)2345, 1024, 0666);
printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0); //process attached to shared memory segment
printf("Process attached at %p\n",shared_memory);
printf("Data read from shared memory is : %s\n",(char *)shared_memory);
}
```

Output

```
Key of shared memory is 0
Process attached at 0x7ff559962000
Enter some data to write to shared memory
My name is Nanditha
//Write.c
You wrote : My name is Nanditha
Key of shared memory is 0
Process attached at 0x7f6b576f8000
Data read from shared memory is : My name is Nanditha
```

Producer consumer using shared memory

```
#include<stdio.h>
#include<stdlib.h>

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

int main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.Producer\n2.Consumer\n3.Exit");
while(1)
{
printf("\nEnter your choice:");
scanf("%d",&n);
switch(n)
{
case 1: if((mutex==1)&&(empty!=0))
producer();
else
printf("Buffer is full!!");
break;
case 2: if((mutex==1)&&(full!=0))
consumer();
else
printf("Buffer is empty!!");
break;
case 3:
exit(0);
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++;
printf("\nProducer produces the item %d",x);
mutex=signal(mutex);
}
void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\nConsumer consumes item %d",x);
x--;
mutex=signal(mutex);
}
```

OUTPUT :

```
1 for producer
2 for consumer
3 exit
Enter your choice :2
No Buffer to consume
Enter your choice :1
New item produced 1
Enter your choice :1
New item produced 2
Enter your choice :1
New item produced 3
Enter your choice :1
Buffer full
Enter your choice :2
Item consumed 3
Enter your choice :2
Item consumed 2
Enter your choice :2
Item consumed 1
Enter your choice :2
No Buffer to consume
Enter your choice :3
Exited
```

Memory allocation

Best fit

```
#include <stdio.h>

void implimentBestFit(int blockSize[], int
blocks, int processSize[], int processes)
{
   // This will store the block id of the
allocated block to a process
   int allocation[processes];

   // initially assigning -1 to all allocation
indexes
   // means nothing is allocated currently
   for(int i = 0; i < processes; i++){
      allocation[i] = -1;
   }

   // pick each process and find suitable blocks
   // according to its size ad assign to it
   for (int i=0; i < processes; i++)
   {

      int indexPlaced = -1;
      for (int j=0; j < blocks; j++)
      {
         if (blockSize[j] >= processSize[i])
         {
            // place it at the first block fit to
accomodate process
            if (indexPlaced == -1)
               indexPlaced = j;

            // if any future block is better that is
            // any future block with smaller size
encountered
            // that can accomodate the given
process
            else if (blockSize[j] <
blockSize[indexPlaced])
               indexPlaced = j;
         }
      }

      // If we were successfully able to find
block for the process
      if (indexPlaced != -1)
      {
         // allocate this block j to process p[i]
         allocation[i] = indexPlaced;
      // Reduce available memory for the block
         blockSize[indexPlaced] -=
processSize[i];
      }
   }

   printf("\nProcess No.\tProcess Size\tBlock
no.\n");
   for (int i = 0; i < processes; i++)
   {
      printf("%d \t\t\t %d \t\t\t", i+1,
processSize[i]);
      if (allocation[i] != -1)
         printf("%d\n",allocation[i] + 1);
      else
         printf("Not Allocated\n");
   }
}

// Driver code
int main()
{
   int blockSize[] = {50, 20, 100, 90};
   int processSize[] = {10, 30, 60, 30};
   int blocks =
sizeof(blockSize)/sizeof(blockSize[0]);
   int processes =
sizeof(processSize)/sizeof(processSize[0]);
implimentBestFit(blockSize, blocks,
processSize, processes);

   return 0 ;
}
```

Output

Process No.	Process Size	Block no.
1	10	2
2	30	1
3	60	4
4	30	4

Worst fit

```
#include <stdio.h>
void implimentWorstFit(int blockSize[], int blocks, int processSize[], int processes)
{
   int allocation[processes];
      for(int i = 0; i < processes; i++){
      allocation[i] = -1;    }
    for (int i=0; i<processes; i++)
   {int indexPlaced = -1;
      for (int j=0; j<blocks; j++)
      {
         if (blockSize[j] >= processSize[i])
         {   if (indexPlaced == -1)
                indexPlaced = j;
            else if (blockSize[indexPlaced] < blockSize[j])
                indexPlaced = j;
         } }
      if (indexPlaced != -1)
      {   allocation[i] = indexPlaced;
         blockSize[indexPlaced] -= processSize[i];
      }   }
   printf("\nProcess No.\tProcess Size\tBlock no.\n");
   for (int i = 0; i < processes; i++)
   {
      printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
      if (allocation[i] != -1)
         printf("%d\n",allocation[i] + 1);
      else
         printf("Not Allocated\n");
   }}
Void  main()
{
   int blockSize[] = {5, 4, 3, 6, 7};
   int processSize[] = {1, 3, 5, 3};
   int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
   int processes = sizeof(processSize)/sizeof(processSize[0]);

   implimentWorstFit(blockSize, blocks, processSize, processes);
}
```

Output

Process No.	Process Size	Block no.
1	1	5
2	3	4
3	5	5
4	3	1

First fit

```
#include<stdio.h>
void firstFit(int blockSize[], int m, int
processSize[], int n)
{
	int i, j;
	int allocation[n];
	for(i = 0; i < n; i++)
	{allocation[i] = -1;}
	for (i = 0; i < n; i++)
	{for (j = 0; j < m; j++)	//here, m ->
number of blocks
		{if (blockSize[j] >=
processSize[i])
			{allocation[i] = j;
				blockSize[j] -=
processSize[i];
break; //go to the next process in the queue
			}	}	}

	printf("\nProcess No.\tProcess
Size\tBlock no.\n");
	for (int i = 0; i < n; i++)
	{
		printf(" %i\t\t\t", i+1);
		printf("%i\t\t\t\t",
processSize[i]);
		if (allocation[i] != -1)
			printf("%i",
allocation[i] + 1);
		else
			printf("Not
Allocated");
		printf("\n");
	}
}
int main()
{		int m,n;
	printf("Enter the number of blocks");
	scanf("%d",&m);
	printf("Enter the number of
processes\n");
	scanf("%d",&n);
	int blockSize[m];
	int processSize[n];
	printf("Enter the block  size");
   for (int i=0;i<m;i++){
      scanf("%d",&blockSize[i]);
	}
	printf("Enter the process size");
	for (int i=0;i<n;i++){
      scanf("%d",&processSize[i]);
	}
firstFit(blockSize, m, processSize, n);

	return 0 ;
}
```

Output

```
Enter the number of blocks5
Enter the number of processes
4
Enter the block  size100
500
200
300
600
Enter the process size212
417
112
426

Process No.	Process Size	Block no.
 1			212		2
 2			417		5
 3			112		2
 4			426		NotAllocated
```

Bankers algorithm

```
#include <stdio.h>
int main()
{   int n, m, i, j, k;
   printf("Enter the number of processes");
   scanf("%d",&n);
    printf("Enter the number of resources");
   scanf("%d",&m);
     int alloc[n][m];
   printf("Enter the values of allocation matrix
");
   for(i=0;i<n;i++){
      for(j=0;j<m;j++){
         scanf("%d",&alloc[i][j]);
      }
   }
   int max[n][m];
    printf("Enter the values of max matrix ");
   for(i=0;i<n;i++){
      for(j=0;j<m;j++){
         scanf("%d",&max[i][j]);
      }
   }
   int avail[m];
   printf("Enter the number of available
instances of each resource type");
   for(i=0;i<m;i++)
   scanf("%d",&avail[i]);

   int f[n], ans[n], ind = 0;
   for (k = 0; k < n; k++)
   {
      f[k] = 0;
   }
   int need[n][m];
   printf("The need matrix is :\n");
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < m; j++) {
         need[i][j] = max[i][j] - alloc[i][j];
         printf("%d",need[i][j]);}
         printf("\n"); }
   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 < m; j++)
            {
               if (need[i][j] > avail[j])
               {
                  flag = 1;
                  break;
               }
            }
            if (flag == 0)
            {
               ans[ind++] = i;
               for (y = 0; y < m; y++)
                  avail[y] += alloc[i][y];
               f[i] = 1;
            }    }  }   }
   int flag = 1;
   for (int i = 0; i < n; i++)
   {
      if (f[i] == 0)
      {
         flag = 0;
         printf("The following system is not
safe");
         break;
      }
   }
   if (flag == 1)
   {
      printf("Following is the SAFE
Sequence\n");
      for (i = 0; i < n - 1; i++)
         printf(" P%d ->", ans[i]);
      printf(" P%d", ans[n - 1]);
   }
   return (0);
}
```

Output

```
Enter the number of processes5
Enter the number of resources4
Enter the values of allocation matrix 0
0
1
2
```

1
0
0
0
1
3
5
4
0
6
3
2
0
0
1
4
Enter the values of max matrix 0
0
1
2
1
7
5
0
2
3
5
6
0
6
5
2
0
6
5
6
Enter the number of available instances of each resource type1
5
2
0
The need matrix is :
0 0 0 0
0 7 5 0
1 0 0 2
0 0 2 0
0 6 4 2
Following is the SAFE Sequence
P0 -> P2 -> P3 -> P4 -> P1

Page replacement algorithm-FIFO

```
#include < stdio.h >
void main()
{
   int incomingStream[] = {4 , 1 , 2 , 4 , 5};
   int pageFaults = 0;
   int frames = 3;
   int m, n, s, pages;
   pages = sizeof(incomingStream)/sizeof(incomingStream[0]);
   printf(" Incoming \ t Frame 1 \ t Frame 2 \ t Frame 3 ");
   int temp[ frames ];
   for(m = 0; m < frames; m++)
   {   temp[m] = -1;
   }
   for(m = 0; m < pages; m++)
   {    s = 0;
      for(n = 0; n < frames; n++)
      {
         if(incomingStream[m] == temp[n])
         {   s++;
            pageFaults--;
         }
      }
      pageFaults++;
      if((pageFaults <= frames) && (s == 0))
      {
         temp[m] = incomingStream[m];
      }
      else if(s == 0)
      {
         temp[(pageFaults - 1) % frames] = incomingStream[m];
      }
      printf("\n");
      printf("%d\t\t\t",incomingStream[m]);
      for(n = 0; n < frames; n++)
      {
         if(temp[n] != -1)
            printf(" %d\t\t\t", temp[n]);
         else
            printf(" - \t\t\t");
   }
   printf("\nTotal Page Faults:\t%d\n", pageFaults);  }
```

Output:

```
Incoming   Frame1  Frame Frame3
4          4       -       -
1          4       1       -
2          4       1       2
4          4       1       2
5          5       1       2
Total Page Faults: 4
```

Page replacement algorithm-LRU

```
#include<stdio.h>

int findLRU(int time[], int n){
int i, minimum = time[0], pos = 0;

for(i = 1; i < n; ++i){
if(time[i] < minimum){
minimum = time[i];
pos = i;
}
}
return pos;
}
void main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
printf("Enter reference string: ");
   for(i = 0; i < no_of_pages; ++i){
    scanf("%d", &pages[i]);
   }
for(i = 0; i < no_of_frames; ++i){
    frames[i] = -1;
   }
   for(i = 0; i < no_of_pages; ++i){
    flag1 = flag2 = 0;
  for(j = 0; j < no_of_frames; ++j){
   if(frames[j] == pages[i]){
   counter++;
   time[j] = counter;
  flag1 = flag2 = 1;
  break;
  }
    if(flag1 == 0){
```

```
for(j = 0; j < no_of_frames; ++j){
    if(frames[j] == -1){
    counter++;
    faults++;
    frames[j] = pages[i];
    time[j] = counter;
    flag2 = 1;
    break;
    }
    }
    }
   if(flag2 == 0){
    pos = findLRU(time, no_of_frames);
    counter++;
    faults++;
    frames[pos] = pages[i];
    time[pos] = counter;
    }
      printf("\n");
      for(j = 0; j < no_of_frames; ++j){
    printf("%d\t", frames[j]);
    }
}
printf("\n\nTotal Page Faults = %d", faults);
}
```

Output

```
Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 5 7 5 6 7 3
5 -1 -1
5 7 -1
5 7 -1
5 7 6
5 7 6
3 7 6
Total Page Faults = 4
```

Page replacement algorithm-LFU

```
#include<stdio.h>
#include<conio.h>
main()
{
  int rs[50], i, j, k, m, f, cntr[20], a[20], min,
pf=0;
  clrscr();
  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);
   getch();
}
```

OUTPUT

```
Enter number of page references -- 10
Enter the reference string -- 1 2 3 4 5 2 5 2 5 1
4 3
 Enter the available no. of frames -- 3
The Page Replacement Process is –

1 -1 -1 PF No. 1
1 2 -1 PF No. 2
```

1 2 3 PF No. 3
4 2 3 PF No. 4
5 2 3 PF No. 5
5 2 3
5 2 3
5 2 1 PF No. 6
5 2 4 PF No. 7
5 2 3 PF No. 8

Total number of page faults – 8

Disk scheduling -FCFS

Program

```
#include <stdio.h>
#include <math.h>
int size = 8;
void FCFS(int arr[],int head)
{       int seek_count = 0;
        int cur_track, distance;
        for(int i=0;i<size;i++)
        {               cur_track = arr[i];
distance = fabs(head - cur_track);
seek_count += distance;
                head = cur_track;
        }
        printf("Total number of seek operations: %d\n",seek_count);
        printf("Seek Sequence is\n");
for (int i = 0; i < size; i++) {
                printf("%d\n",arr[i]);
        }
}
int main()
{int arr[8] = { 176, 79, 34, 60, 92, 11, 41, 114 };
        int head = 50;
        FCFS(arr,head);
return 0;
}
```

Output

```
Total number of seek operations = 510
Seek Sequence is
176
79
34
60
92
11
41
114
```

Disk scheduling -SCAN

Program

```
#include<stdio.h>
int n,m,i,j,h,p,temp,k,total=0;
int t[100],a[100],diff;
void main()
{
 printf("ENTER THE NUMBER OF TRACKS : ");
 scanf("%d",&n);
 printf("ENTER THE HEAD POINTER POSITION :
");
 scanf("%d",&h);
 printf("ENTER THE TRACKS TO BE TRAVERSED
: ");
 for(i=0;i<n;i++)
 {
 scanf("%d",&t[i]);
 }
 t[n+2] = 199;
 t[n+1] = 0;
 t[n] = h;
 n=n+3;
 for(i=0;i<n;i++) {
 for(j=0;j<n-i-1;j++){
 if(t[j]>t[j+1]){
 temp=t[j];
 t[j]=t[j+1];
 t[j+1]=temp;
 } } }
 for(i=0;i<n;i++){
 if(t[i]==h){
 k=i;
 break; }
 }
 if(h<(199-h)){
 for(i=k;i>=0;i--,p++){
 a[p]=t[i];}
 for(i=k+1;i<n-1;i++,p++){
 a[p]=t[i];
 } }
 else {
 for(i=k;i<n;i++,p++){
 a[p]=t[i];}
 for(i=k-1;i>=0;i--,p++){
 a[p]=t[i];}
 }
 printf("TRAVERSED ORDER : ");
 for(i=0;i<p;i++){
 printf("%d => ",a[i]);
 }
 for(total=0,j=0;j<p-1;j++){
 diff=0;
 if(a[j]>a[j+1]){
 diff=a[j]-a[j+1]; }
 else{
 diff=a[j+1]-a[j]; }
 total=total+diff;
 }
 printf("\b\b\b. \nTOTAL HEAD MOVEMENTS :
%d\n",total);
}
```

OUTPUT :

```
ENTER THE NUMBER OF TRACKS : 8
ENTER THE HEAD POINTER POSITION : 50
ENTER THE TRACKS TO BE TRAVERSED : 176
79
34
60
92
11
41
114
TRAVERSED ORDER : 50 => 41 => 34 => 11 => 0
=> 60 => 79 => 92 => 114 => 176 =>
TOTAL HEAD MOVEMENTS : 226
```

Disk scheduling -C-SCAN

Program

```
#include<stdio.h>
int main()
{
 int
queue[20],n,head,i,j,k,seek=0,max,diff,temp,q
ueue1[20],queue2[20],
 temp1=0,temp2=0;
 float avg;
 printf("Enter the max range of disk\n");
 scanf("%d",&max);
 printf("Enter the initial head position\n");
 scanf("%d",&head);
 printf("Enter the size of queue request\n");
 scanf("%d",&n);
 printf("Enter the queue of disk positions to be
read\n");
 for(i=1;i<=n;i++) {
 scanf("%d",&temp);
 if(temp>=head){
 queue1[temp1]=temp;
 temp1++;}
 else{
 queue2[temp2]=temp;
 temp2++; }
 }
 for(i=0;i<temp1-1;i++) {
 for(j=i+1;j<temp1;j++){
 if(queue1[i]>queue1[j]){
 temp=queue1[i];
 queue1[i]=queue1[j];
 queue1[j]=temp; } }
 }
 for(i=0;i<temp2-1;i++){
 for(j=i+1;j<temp2;j++){
 if(queue2[i]>queue2[j]){
 temp=queue2[i];
 queue2[i]=queue2[j];
 queue2[j]=temp; }
}
 }
 for(i=1,j=0;j<temp1;i++,j++)
 queue[i]=queue1[j];
 queue[i]=max;
 queue[i+1]=0;
 for(i=temp1+3,j=0;j<temp2;i++,j++)
 queue[i]=queue2[j];
 queue[0]=head;
 for(j=0;j<=n+1;j++){
 diff=abs(queue[j+1]-queue[j]);
 seek+=diff;
 printf(" %d -> ",queue[j]);
 }
 printf("\n Total seek time is %d\n",seek);
 avg=seek/(float)n;
 printf("Average seek time is %f\n",avg);
 return 0;
}
```

OUTPUT :

```
Enter the max range of disk
199
Enter the initial head position
50
Enter the size of queue request
8
Enter the queue of disk positions to be read
176 79 34 60 92 11 41 114
50 -> 60 -> 79 -> 92 -> 114 -> 176 -> 199 -> 0 -
> 11 -> 34
Total seek time is 389
Average seek time is 48.62500
```