Assignment No.1: Operations on Processes

Practical Assignments:
Set A
(1) Implement the C Program to create a child process using fork(), display parent and child process id. Child process will display the message "I am Child Process" and the parent process should display "I am Parent Process".

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
   // fork() Create a child process

   int pid = fork();
   if (pid > 0) {
      printf("I am Parent process\n");
      printf("ID : %d\n\n", getpid());
   }
   else if (pid == 0) {
      printf("I am Child process\n");
      // getpid() will return process id of child process
      printf("ID: %d\n", getpid());

   }
   else {
      printf("Failed to create child process");
   }

   return 0;
}
```

2)Write a program that demonstrates the use of nice() system call. After a child process is started using fork(), assign higher priority to the child using nice() system call.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
pid_t pid;
pid = fork();
if (pid == 0)
{
printf("\nI am child process, id=%d\n",getpid());
printf("\nPriority :%d,id=%d\n",nice (-7),getpid());
}
else
{
printf("\nI am parent process, id=%d\n",getpid());
nice(1);
printf("\nPriority :%d,id=%d\n",nice (15),getpid());
}
return 0;
}
```

Set B:

1) **Implement the C program to accept n integers to be sorted. Main function creates child process using fork system call. Parent process sorts the integers using bubble sort and waits for child process using wait system call. Child process sorts the integers using insertion sort.**

Solution:

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
```

```
#include<stdlib.h>

void bubblesort(int arr[30],int n)
{
        int i,j,temp;
        for(i=0;i<n;i++)
        {
            for(j=0;j<n-1;j++)
            {
                    if(arr[j]>arr[j+1])
                    {
                            temp=arr[j];
                            arr[j]=arr[j+1];
                            arr[j+1]=temp;
                    }
            }
        }
}

void insertionsort(int arr[30], int n)
{
   int i, j, temp;
   for (i = 1; i < n; i++) {
      temp = arr[i];
      j = i - 1;

      while(j>=0 && temp <= arr[j])
      {
         arr[j+1] = arr[j];
         j = j-1;
      }
      arr[j+1] = temp;
   }
}
void fork1()
```

```
{
        int arr[25],arr1[25],n,i,status;
        printf("\nEnter the no of values in array :");
        scanf("%d",&n);
        printf("\nEnter the array elements :");
        for(i=0;i<n;i++)
            scanf("%d",&arr[i]);
        int pid=fork();
        if(pid==0)
        {
            sleep(10);
            printf("\nchild process\n");
            printf("child process id=%d\n",getpid());
            insertionsort(arr,n);
            printf("\nElements Sorted Using insertionsort:");
            printf("\n");
            for(i=0;i<n;i++)
                    printf("%d,",arr[i]);
            printf("\b");
            printf("\nparent process id=%d\n",getppid());
            system("ps -x");
     }
    else
     {
            printf("\nparent process\n");
            printf("\nparent process id=%d\n",getppid());
            bubblesort(arr,n);
            printf("Elements Sorted Using bubblesort:");
            printf("\n");
            for(i=0;i<n;i++)
                    printf("%d,",arr[i]);
            printf("\n\n\n");
    }
 }
int main()
```

```
{
        fork1();
        return 0;
}
```

2) Write a C program to illustrate the concept of orphan process. Parent process creates a child and terminates before child has finished its task. So child process becomes orphan process. (Use fork(), sleep(), getpid(), getppid()).

Solution:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
 int pid;
 pid=getpid();
 printf("Current Process ID is : %d\n",pid);
 printf("\n[Forking Child Process ... ] \n");
 pid=fork();
 if(pid < 0)
 {
  printf("\nProcess can not be created ");
 }
 else
 {
        if(pid==0)
  {
   printf("\nChild Process is Sleeping ...");
   sleep(5);
   printf("\nOrphan Child's Parent ID : %d",getppid());
  }

  else
```

```
  { /* Parent Process */
   printf("\nParent Process Completed ...");
  }
 }
 return 0;
}
```

Assignment No.2: Operations on Processes

Set A

Write a C program that behaves like a shell which displays the command prompt 'myshell$'. It accepts the command, tokenize the command line and execute it by creating the child process. Also implement the additional command 'count' as

myshell$ count c filename: It will display the number of characters in given file
myshell$ count w filename: It will display the number of words in given file
myshell$ count l filename: It will display the number of lines in given file

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void make_toks(char *s, char *tok[])
{
int i=0;
char *p;
p = strtok(s," ");
while(p!=NULL)
{
 tok[i++]=p;
 p=strtok(NULL," ");
}
tok[i]=NULL;
}
void count(char *fn, char op)
```

```
{
int fh,cc=0,wc=0,lc=0;
char c;
fh = open(fn,O_RDONLY);
if(fh==-1)
{
 printf("File %s not found.\n",fn);
return;
}
while(read(fh,&c,1)>0)
{
if(c==' ') wc++;
else if(c=='\n')
 {
 wc++;
 lc++;
 }
 cc++;
}
close(fh);
switch(op)
{
case 'c':
 printf("No.of characters:%d\n",cc-1);
break;
case 'w':
 printf("No.of words:%d\n",wc);
break;
case 'l':
 printf("No.of lines:%d\n",lc+1);
break;
}
}
int main()
{
char buff[80],*args[10];
int pid;
while(1)
{
 printf("myshell$ ");
```

```
 fflush(stdin);
fgets(buff,80,stdin);
buff[strlen(buff)-1]='\0';
make_toks(buff,args);
if(strcmp(args[0],"count")==0)
count(args[2],args[1][0]);
else
 {
 pid = fork();
 if(pid>0)
 wait();
 else
 {
 if(execvp(args[0],args)==-1)
 printf("Bad command.\n");
 }
 }
}
return 0;
}
```

SET B: Write a program to implement a toy shell (Command Interpreter). It has its own prompt say "MyShell $". Any normal shell command is executed from this shell (MyShell$) by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following commands:
list f dirname : To print names of all the files in current directory
list n dirname : To print the number of all entries in the current directory.
list i dirname : To print names and inodes of the files in the current directory.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>

void make_toks(char *s, char *tok[])
{
 int i=0;
```

```
 char *p;

 p = strtok(s," ");
 while(p!=NULL)
 {
  tok[i++]=p;
  p=strtok(NULL," ");
 }

 tok[i]=NULL;
}

void list(char *dn, char op)
{
 DIR *dp;
 struct dirent *entry;
 int dc=0,fc=0;

 dp = opendir(dn);
 if(dp==NULL)
 {
  printf("Dir %s not found.\n",dn);
  return;
 }

 switch(op)
 {
 case 'f':
  while(entry=readdir(dp))
  {
   if(entry->d_type==DT_REG)
    printf("%s\n",entry->d_name);
  }
  break;
 case 'n':
  while(entry=readdir(dp))
  {
   if(entry->d_type==DT_DIR) dc++;
   if(entry->d_type==DT_REG)  fc++;
  }
```

```
 printf("%d Dir(s)\t%d File(s)\n",dc,fc);
 break;
case 'i':
 while(entry=readdir(dp))
 {
  if(entry->d_type==DT_REG)
   printf("%s\t%d\n",entry->d_name,entry->d_fileno);
 }
}

closedir(dp);
}

int main()
{
 char buff[80],*args[10];
 int pid;

 while(1)
 {
  printf("myshell$");
  fflush(stdin);
  fgets(buff,80,stdin);
  buff[strlen(buff)-1]='\0';
  make_toks(buff,args);
  if(strcmp(args[0],"list")==0)
   list(args[2],args[1][0]);
  else
  {
   pid = fork();
   if(pid>0)
   wait();
   else
   {
    if(execvp(args[0],args)==-1)
     printf("Bad command.\n");
   }
  }
 }
```

```
 return 0;
}
```

2) Write a C program that behaves like a shell which displays the command prompt 'myshell$'. It accepts the command, tokenize the command line and execute it by creating the child process.
Also implement the additional command 'typeline' as
myshell$ typeline n filename: It will display first n lines of the file.
myshell$ typeline -n filename: It will display last n lines of the file.
myshell$ typeline a filename: It will display all the lines of the file.

Solution:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <unistd.h>
int make_toks(char *s, char *tok[]) {
int i = 0;
char *p;
p = strtok(s, " ");
while(p != NULL) {
tok[i++] = p;
p = strtok(NULL, " ");
}
tok[i] = NULL;
return i;
}
void typeline(char *op, char *fn) {
int fh,
i,
j,
n;
char c;
fh = open(fn, O_RDONLY);
if(fh == -1) {
printf("File %s not found.\n", fn);
return;
```

```
}
if(strcmp(op, "a") == 0) {
while(read(fh, &c, 1) > 0)
printf("%c", c);
close(fh);
return;
}
n = atoi(op);
if(n > 0) {
i = 0;
while(read(fh, &c, 1) > 0) {
printf("%c", c);
if(c == '\n') i++;
if(i == n) break;
}
}
if(n < 0) {
i = 0;
while(read(fh, &c, 1) > 0) {
if(c == '\n') i++;
}
lseek(fh, 0, SEEK_SET);
j = 0;
while(read(fh, &c, 1) > 0) {
if(c == '\n') j++;
if(j == i+n+1) break;
}
while(read(fh, &c, 1) > 0) {
printf("%c", c);
}
}
close(fh);
}
int main() {
char buff[80],
*args[10];
while(1) {
printf ("\n");
printf("\nmyshell$ ");
fgets(buff, 80, stdin);
```

```
buff[strlen(buff)-1] = '\0';
int n = make_toks(buff, args);
switch (n) {
case 1:
if(strcmp(args[0], "exit") == 0)
exit(1);
if (!fork())
execlp (args [0], args[0], NULL);
break;
case 2:
if (!fork ())
execlp (args [0], args[0], args[1], NULL);
break;
case 3:
if (strcmp(args[0], "typeline") == 0)
typeline (args[1], args[2]);
else {
if (!fork ())
execlp (args [0], args[0], args[1], args[2], NULL);
}
break;
case 4:
if (!fork ())
execlp (args [0], args [0], args [1], args [2], args [3], NULL);
break;
}
}
return 0;
}
```

SET C

Q.2) Write a program to implement a toy shell (Command Interpreter). It has its own prompt say "MyShell $". Any normal shell command is executed from this shell (MyShell$) by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following commands:

search f file name :- To search first occurrence of the pattern in the file

search a file name :- To search all the occurrence of the pattern in the file

search c file name :- To count the number of occurrence of the pattern in the file.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void make_toks(char *s, char *tok[])
{
 int i=0;
 char *p;

 p = strtok(s," ");
 while(p!=NULL)
 {
  tok[i++]=p;
  p=strtok(NULL," ");
 }

 tok[i]=NULL;
}

void search(char *fn, char op, char *pattern)
{
 int fh,count=0,i=0,j=0;
 char buff[255],c,*p;

 fh = open(fn,O_RDONLY);
 if(fh==-1)
 {
  printf("File %s Not Found\n",fn);
  return;
 }

 switch(op)
 {
 case 'f':
```

```
 while(read(fh,&c,1))
 {
  buff[j++]=c;
  if(c=='\n')
  {
   buff[j]='\0';
   j=0;
   i++;
   if(strstr(buff,pattern))
   {
    printf("%d: %s",i,buff);
    break;
   }
  }
 }
 break;
case 'c':
 while(read(fh,&c,1))
 {
  buff[j++]=c;
  if(c=='\n')
  {
   buff[j]='\0';
   j=0;
   p = buff;
   while(p=strstr(p,pattern))
   {
    count++;
    p++;
   }
  }
 }
 printf("Total No.of Occurrences = %d\n",count);
 break;
case 'a':
 while(read(fh,&c,1))
 {
  buff[j++]=c;
  if(c=='\n')
  {
```

```
    buff[j]='\0';
    j = 0;
    i++;
    if(strstr(buff,pattern))
     printf("%d: %s",i,buff);
   }
  }
 }//switch
 close(fh);
}//search

int main()
{
 char buff[80],*args[10];
 int pid;

 while(1)
 {
  printf("myshell$");
  fflush(stdin);
  fgets(buff,80,stdin);
  buff[strlen(buff)-1]='\0';
  make_toks(buff,args);
  if(strcmp(args[0],"search")==0)
   search(args[3],args[1][0],args[2]);
  else
  {
   pid = fork();
   if(pid>0)
   wait();
   else
   {
    if(execvp(args[0],args)==-1)
     printf("Bad command.\n");
   }
  }
 }

 return 0;
}
```

Assignment 3- CPU Scheduling
SET A

1) **Write the program to simulate FCFS CPU-scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Assume that the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly.The output should give Gantt chart, turnaround time and waiting time for each process.Also find the average waiting time and turnaround time.**

```
#include<stdio.h>

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n,

                                        int bt[], int wt[])

{

        // waiting time for first process is 0

        wt[0] = 0;


        // calculating waiting time

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

                wt[i] = bt[i-1] + wt[i-1] ;

}


// Function to calculate turn around time
```

```
void findTurnAroundTime( int processes[], int n,
                         int bt[], int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}

//Function to calculate average time
void findavgTime( int processes[], int n, int bt[])
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    //Function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt);

    //Function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);

    //Display processes along with all details
```

```
        printf("Processes Burst time Waiting time Turn around time\n");

        // Calculate total waiting time and total turn
        // around time
        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                printf(" %d ",(i+1));
                printf("        %d ", bt[i] );
                printf("        %d",wt[i] );
                printf("        %d\n",tat[i] );
        }
        int s=(float)total_wt / (float)n;
        int t=(float)total_tat / (float)n;
        printf("Average waiting time = %d",s);
        printf("\n");
        printf("Average turn around time = %d ",t);
}

// Driver code
```

```
int main()
{
    //process id's
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof processes[0];

    //Burst time of all processes
    int burst_time[] = {10, 5, 8};

    findavgTime(processes, n, burst_time);
    return 0;
}
```

Write a simulation program to implement non-pre-emptive Priority CPU scheduling algorithm. Accept the number of Processes and arrival time, CPU burst time and priority for each process as input. Priorities should in High to Low order (1 is High). The output should give the Gantt chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.

Solution:

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

typedef struct process_info
{
 char pname[20];
 int at,bt,ct,bt1,p;
 struct process_info *next;
```

```
}NODE;

int n;
NODE *first,*last;

void accept_info()
{
 NODE *p;
 int i;

 printf("Enter no.of process:");
 scanf("%d",&n);

 for(i=0;i<n;i++)
 {
  p = (NODE*)malloc(sizeof(NODE));

  printf("Enter process name:");
  scanf("%s",p->pname);

  printf("Enter arrival time:");
  scanf("%d",&p->at);

  printf("Enter first CPU burst time:");
  scanf("%d",&p->bt);

  printf("Enter priority:");
  scanf("%d",&p->p);

  p->bt1 = p->bt;

  p->next = NULL;

  if(first==NULL)
   first=p;
  else
```

```
   last->next=p;

  last = p;
 }
}

void print_output()
{
 NODE *p;
 float avg_tat=0,avg_wt=0;

 printf("pname\tat\tbt\tp\ttct\ttat\twt\n");

 p = first;
 while(p!=NULL)
 {
  int tat = p->ct-p->at;
  int wt = tat-p->bt;

  avg_tat+=tat;
  avg_wt+=wt;

  printf("%s\t%d\t%d\t%d\t%d\t%d\t%d\n",
   p->pname,p->at,p->bt,p->p,p->ct,tat,wt);

  p=p->next;
 }

 printf("Avg TAT=%f\tAvg WT=%f\n",
   avg_tat/n,avg_wt/n);
}

void print_input()
{
 NODE *p;
```

```
 p = first;

 printf("pname\tat\tbt\tp\n");
 while(p!=NULL)
 {
  printf("%s\t%d\t%d\t%d\n",
   p->pname,p->at,p->bt1,p->p);
  p = p->next;
 }
}

void sort()
{
 NODE *p,*q;
 int t;
 char name[20];

 p = first;
 while(p->next!=NULL)
 {
  q=p->next;
  while(q!=NULL)
  {
   if(p->at > q->at)
   {
    strcpy(name,p->pname);
    strcpy(p->pname,q->pname);
    strcpy(q->pname,name);

    t = p->at;
    p->at = q->at;
    q->at = t;

    t = p->bt;
    p->bt = q->bt;
    q->bt = t;
```

```
    t = p->ct;
    p->ct = q->ct;
    q->ct = t;

    t = p->bt1;
    p->bt1 = q->bt1;
    q->bt1 = t;

    t = p->p;
    p->p = q->p;
    q->p = t;
   }

   q=q->next;
  }

  p=p->next;
 }
}

int time;

NODE * get_p()
{
 NODE *p,*min_p=NULL;
 int min=9999;

 p = first;
 while(p!=NULL)
 {
  if(p->at<=time && p->bt1!=0 &&
   p->p<min)
  {
   min = p->p;
   min_p = p;
```

```
  }
  p=p->next;
 }

 return min_p;
}

struct gantt_chart
{
 int start;
 char pname[30];
 int end;
}s[100],s1[100];

int k;

void pnp()
{
 int prev=0,n1=0;
 NODE *p;

 while(n1!=n)
 {
  p = get_p();

  if(p==NULL)
  {
   time++;
   s[k].start = prev;
   strcpy(s[k].pname,"*");
   s[k].end = time;

   prev = time;
   k++;
  }
  else
```

```
  {
   time+=p->bt1;
   s[k].start = prev;
   strcpy(s[k].pname, p->pname);
   s[k].end = time;

   prev = time;
   k++;

   p->ct = time;
   p->bt1 = 0;

   n1++;
  }

  print_input();
  sort();
 }
}

void print_gantt_chart()
{
 int i,j,m;

 s1[0] = s[0];

 for(i=1,j=0;i<k;i++)
 {
  if(strcmp(s[i].pname,s1[j].pname)==0)
   s1[j].end = s[i].end;
  else
   s1[++j] = s[i];
 }

 printf("%d",s1[0].start);
 for(i=0;i<=j;i++)
```

```
 {
  m = (s1[i].end - s1[i].start);

  for(k=0;k<m/2;k++)
   printf("-");

  printf("%s",s1[i].pname);

  for(k=0;k<(m+1)/2;k++)
   printf("-");

  printf("%d",s1[i].end);
 }
}

int main()
{
 accept_info();
 sort();
 pnp();
 print_output();
 print_gantt_chart();

 return 0;
}
```

1. Write a simulation program to implement a Non-Pre-emptive Shortest Job First (SJF) – CPU scheduling algorithm. Accept the number of Processes and arrival time and CPU burst time for each process as input. The output should give the Gantt chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.

Solution:

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

```
#include<string.h>

typedef struct process_info
{
 char pname[20];
 int at,bt,ct,bt1;
 struct process_info *next;
}NODE;

int n;
NODE *first,*last;

void accept_info()
{
 NODE *p;
 int i;

 printf("Enter no.of process:");
 scanf("%d",&n);

 for(i=0;i<n;i++)
 {
  p = (NODE*)malloc(sizeof(NODE));

  printf("Enter process name:");
  scanf("%s",p->pname);

  printf("Enter arrival time:");
  scanf("%d",&p->at);

  printf("Enter first CPU burst time:");
  scanf("%d",&p->bt);


  p->bt1 = p->bt;
```

```
  p->next = NULL;

  if(first==NULL)
   first=p;
  else
   last->next=p;

  last = p;
 }
}

void print_output()
{
 NODE *p;
 float avg_tat=0,avg_wt=0;

 printf("pname\tat\tbt\tct\ttat\twt\n");

 p = first;
 while(p!=NULL)
 {
  int tat = p->ct-p->at;
  int wt = tat-p->bt;

  avg_tat+=tat;
  avg_wt+=wt;

  printf("%s\t%d\t%d\t%d\t%d\t%d\n",
   p->pname,p->at,p->bt,p->ct,tat,wt);

  p=p->next;
 }

 printf("Avg TAT=%f\tAvg WT=%f\n",
   avg_tat/n,avg_wt/n);
```

```
}

void print_input()
{
 NODE *p;

 p = first;

 printf("pname\tat\tbt\n");
 while(p!=NULL)
 {
  printf("%s\t%d\t%d\n",
   p->pname,p->at,p->bt1);
  p = p->next;
 }
}

void sort()
{
 NODE *p,*q;
 int t;
 char name[20];

 p = first;
 while(p->next!=NULL)
 {
  q=p->next;
  while(q!=NULL)
  {
   if(p->at > q->at)
   {
    strcpy(name,p->pname);
    strcpy(p->pname,q->pname);
    strcpy(q->pname,name);
```

```
    t = p->at;
    p->at = q->at;
    q->at = t;

    t = p->bt;
    p->bt = q->bt;
    q->bt = t;

    t = p->ct;
    p->ct = q->ct;
    q->ct = t;

    t = p->bt1;
    p->bt1 = q->bt1;
    q->bt1 = t;

   }

   q=q->next;
  }

  p=p->next;
 }
}

int time;

NODE * get_sjf()
{
 NODE *p,*min_p=NULL;
 int min=9999;

 p = first;
 while(p!=NULL)
 {
```

```
  if(p->at<=time && p->bt1!=0 &&
   p->bt1<min)
  {
   min = p->bt1;
   min_p = p;
  }
  p=p->next;
 }

 return min_p;
}

struct gantt_chart
{
 int start;
 char pname[30];
 int end;
}s[100],s1[100];

int k;

void sjfnp()
{
 int prev=0,n1=0;
 NODE *p;

 while(n1!=n)
 {
  p = get_sjf();

  if(p==NULL)
  {
   time++;
   s[k].start = prev;
   strcpy(s[k].pname,"*");
```

```
   s[k].end = time;

   prev = time;
   k++;
  }
  else
  {
   time+=p->bt1;
   s[k].start = prev;
   strcpy(s[k].pname, p->pname);
   s[k].end = time;

   prev = time;
   k++;

   p->ct = time;
   p->bt1 = 0;

   n1++;
  }

  print_input();
  sort();
 }
}

void print_gantt_chart()
{
 int i,j,m;

 s1[0] = s[0];

 for(i=1,j=0;i<k;i++)
 {
  if(strcmp(s[i].pname,s1[j].pname)==0)
```

```
   s1[j].end = s[i].end;
  else
   s1[++j] = s[i];
 }

 printf("%d",s1[0].start);
 for(i=0;i<=j;i++)
 {
  m = (s1[i].end - s1[i].start);

  for(k=0;k<m/2;k++)
   printf("-");

  printf("%s",s1[i].pname);

  for(k=0;k<(m+1)/2;k++)
   printf("-");

  printf("%d",s1[i].end);
 }
}

int main()
{
 accept_info();
 sort();
 sjfnp();
 print_output();
 print_gantt_chart();

 return 0;
}
```

Write a simulation program to implement Round Robin CPU scheduling algorithm for the given time quantum as input. Also accept the number of

processes and arrival time and CPU burst time for each process as input. The output should give the Gant Chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.

Solution:

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

typedef struct process_info
{
 char pname[20];
 int at,bt,ct,bt1;
 struct process_info *next;
}NODE;

int n,ts;
NODE *first,*last;

void accept_info()
{
 NODE *p;
 int i;

 printf("Enter no.of process:");
 scanf("%d",&n);

 for(i=0;i<n;i++)
 {
  p = (NODE*)malloc(sizeof(NODE));

  printf("Enter process name:");
  scanf("%s",p->pname);
```

```
  printf("Enter arrival time:");
  scanf("%d",&p->at);

  printf("Enter first CPU burst time:");
  scanf("%d",&p->bt);

  p->bt1 = p->bt;

  p->next = NULL;

  if(first==NULL)
   first=p;
  else
   last->next=p;

  last = p;
 }

 printf("Enter time slice:");
 scanf("%d",&ts);
}

void print_output()
{
 NODE *p;
 float avg_tat=0,avg_wt=0;

 printf("pname\tat\tbt\tct\ttat\twt\n");

 p = first;
 while(p!=NULL)
 {
  int tat = p->ct-p->at;
  int wt = tat-p->bt;

  avg_tat+=tat;
```

```
  avg_wt+=wt;

  printf("%s\t%d\t%d\t%d\t%d\t%d\n",
   p->pname,p->at,p->bt,p->ct,tat,wt);

  p=p->next;
 }

 printf("Avg TAT=%f\tAvg WT=%f\n",
   avg_tat/n,avg_wt/n);
}

void print_input()
{
 NODE *p;

 p = first;

 printf("pname\tat\tbt\n");
 while(p!=NULL)
 {
  printf("%s\t%d\t%d\n",
   p->pname,p->at,p->bt1);
  p = p->next;
 }
}

void sort()
{
 NODE *p,*q;
 int t;
 char name[20];

 p = first;
 while(p->next!=NULL)
 {
```

```
  q=p->next;
  while(q!=NULL)
  {
   if(p->at > q->at)
   {
    strcpy(name,p->pname);
    strcpy(p->pname,q->pname);
    strcpy(q->pname,name);

    t = p->at;
    p->at = q->at;
    q->at = t;

    t = p->bt;
    p->bt = q->bt;
    q->bt = t;

    t = p->ct;
    p->ct = q->ct;
    q->ct = t;

    t = p->bt1;
    p->bt1 = q->bt1;
    q->bt1 = t;
   }

   q=q->next;
  }

  p=p->next;
 }
}

int time;

int is_arrived()
```

```
{
 NODE *p;

 p = first;
 while(p!=NULL)
 {
  if(p->at<=time && p->bt1!=0)
   return 1;

  p=p->next;
 }

 return 0;
}

NODE * delq()
{
 NODE *t;

 t = first;
 first = first->next;
 t->next=NULL;

 return t;
}

void addq(NODE *t)
{
 last->next = t;
 last = t;
}

struct gantt_chart
{
 int start;
 char pname[30];
```

```
 int end;
}s[100],s1[100];

int k;

void rr()
{
 int prev=0,n1=0;
 NODE *p;

 while(n1!=n)
 {
  if(!is_arrived())
  {
   time++;
   s[k].start = prev;
   strcpy(s[k].pname,"*");
   s[k].end = time;
   k++;
   prev=time;
  }
  else
  {
   p = first;
   while(1)
   {
    if(p->at<=time && p->bt1!=0)
     break;

    p = delq();
    addq(p);
    p = first;
   }

   if(p->bt1<=ts)
   {
```

```
    time+=p->bt1;
    p->bt1=0;
   }
   else
   {
    time+=ts;
    p->bt1-=ts;
   }

   p->ct = time;

   s[k].start = prev;
   strcpy(s[k].pname,p->pname);
   s[k].end = time;

   k++;
   prev = time;

   if(p->bt1==0) n1++;

   p = delq();
   addq(p);
  }

  print_input();
 }
}

void print_gantt_chart()
{
 int i,j,m;

 s1[0] = s[0];

 for(i=1,j=0;i<k;i++)
 {
```

```
  if(strcmp(s[i].pname,s1[j].pname)==0)
   s1[j].end = s[i].end;
  else
   s1[++j] = s[i];
 }

 printf("%d",s1[0].start);
 for(i=0;i<=j;i++)
 {
  m = (s1[i].end - s1[i].start);

  for(k=0;k<m/2;k++)
   printf("-");

  printf("%s",s1[i].pname);

  for(k=0;k<(m+1)/2;k++)
   printf("-");

  printf("%d",s1[i].end);
 }
}

int main()
{
 accept_info();
 sort();
 rr();
 print_output();
 print_gantt_chart();

 return 0;
}
```

Write a simulation program to implement a Pre-emptive Shortest Job First (SJF) – CPU scheduling algorithm. Accept the number of Processes as input.

Also accept arrival time and CPU burst time for each process as input. The output should give the Gantt chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.

Solution:

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

typedef struct process_info
{
 char pname[20];
 int at,bt,ct,bt1;
 struct process_info *next;
}NODE;

int n;
NODE *first,*last;

void accept_info()
{
 NODE *p;
 int i;

 printf("Enter no.of process:");
 scanf("%d",&n);

 for(i=0;i<n;i++)
 {
  p = (NODE*)malloc(sizeof(NODE));

  printf("Enter process name:");
  scanf("%s",p->pname);
```

```
  printf("Enter arrival time:");
  scanf("%d",&p->at);

  printf("Enter first CPU burst time:");
  scanf("%d",&p->bt);

  p->bt1 = p->bt;

  p->next = NULL;

  if(first==NULL)
   first=p;
  else
   last->next=p;

  last = p;
 }
}

void print_output()
{
 NODE *p;
 float avg_tat=0,avg_wt=0;

 printf("pname\tat\tbt\tct\ttat\twt\n");

 p = first;
 while(p!=NULL)
 {
  int tat = p->ct-p->at;
  int wt = tat-p->bt;

  avg_tat+=tat;
  avg_wt+=wt;

  printf("%s\t%d\t%d\t%d\t%d\t%d\n",
```

```
   p->pname,p->at,p->bt,p->ct,tat,wt);

  p=p->next;
 }

 printf("Avg TAT=%f\tAvg WT=%f\n",
   avg_tat/n,avg_wt/n);
}

void print_input()
{
 NODE *p;

 p = first;

 printf("pname\tat\tbt\n");
 while(p!=NULL)
 {
  printf("%s\t%d\t%d\n",
   p->pname,p->at,p->bt1);
  p = p->next;
 }
}

void sort()
{
 NODE *p,*q;
 int t;
 char name[20];

 p = first;
 while(p->next!=NULL)
 {
  q=p->next;
  while(q!=NULL)
  {
```

```
   if(p->at > q->at)
   {
    strcpy(name,p->pname);
    strcpy(p->pname,q->pname);
    strcpy(q->pname,name);

    t = p->at;
    p->at = q->at;
    q->at = t;

    t = p->bt;
    p->bt = q->bt;
    q->bt = t;

    t = p->ct;
    p->ct = q->ct;
    q->ct = t;

    t = p->bt1;
    p->bt1 = q->bt1;
    q->bt1 = t;
   }

   q=q->next;
  }

  p=p->next;
 }
}

int time;

NODE * get_sjf()
{
 NODE *p,*min_p=NULL;
 int min=9999;
```

```
 p = first;
 while(p!=NULL)
 {
  if(p->at<=time && p->bt1!=0 &&
   p->bt1<min)
  {
   min = p->bt1;
   min_p = p;
  }
  p=p->next;
 }

 return min_p;
}

struct gantt_chart
{
 int start;
 char pname[30];
 int end;
}s[100],s1[100];

int k;

void sjfp()
{
 int prev=0,n1=0;
 NODE *p;

 while(n1!=n)
 {
  p = get_sjf();

  if(p==NULL)
  {
```

```
   time++;
   s[k].start = prev;
   strcpy(s[k].pname,"*");
   s[k].end = time;

   prev = time;
   k++;
  }
  else
  {
   time++;
   s[k].start = prev;
   strcpy(s[k].pname, p->pname);
   s[k].end = time;

   prev = time;
   k++;

   p->ct = time;
   p->bt1--;

   if(p->bt1==0)
    n1++;
  }

  print_input();
  sort();
 }
}

void print_gantt_chart()
{
 int i,j,m;

 s1[0] = s[0];
```

```
 for(i=1,j=0;i<k;i++)
 {
  if(strcmp(s[i].pname,s1[j].pname)==0)
   s1[j].end = s[i].end;
  else
   s1[++j] = s[i];
 }

 printf("%d",s1[0].start);
 for(i=0;i<=j;i++)
 {
  m = (s1[i].end - s1[i].start);

  for(k=0;k<m/2;k++)
   printf("-");

  printf("%s",s1[i].pname);

  for(k=0;k<(m+1)/2;k++)
   printf("-");

  printf("%d",s1[i].end);
 }
}

int main()
{
 accept_info();
 sort();
 sjfp();
 print_output();
 print_gantt_chart();

 return 0;
}
```

Write a simulation program to implement Pre-emptive Priority CPU scheduling algorithm. Accept the number of processes, arrival time, CPU burst time and priority for each process as input. Priorities should in High to Low order (1 is High). The output should give the Gantt chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.

Solution:

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

typedef struct process_info
{
 char pname[20];
 int at,bt,ct,bt1,p;
 struct process_info *next;
}NODE;

int n;
NODE *first,*last;

void accept_info()
{
 NODE *p;
 int i;

 printf("Enter no.of process:");
 scanf("%d",&n);

 for(i=0;i<n;i++)
 {
  p = (NODE*)malloc(sizeof(NODE));

  printf("Enter process name:");
  scanf("%s",p->pname);
```

```
  printf("Enter arrival time:");
  scanf("%d",&p->at);

  printf("Enter first CPU burst time:");
  scanf("%d",&p->bt);

  printf("Enter priority:");
  scanf("%d",&p->p);

  p->bt1 = p->bt;

  p->next = NULL;

  if(first==NULL)
   first=p;
  else
   last->next=p;

  last = p;
 }
}

void print_output()
{
 NODE *p;
 float avg_tat=0,avg_wt=0;

 printf("pname\tat\tbt\tp\ttct\ttat\twt\n");

 p = first;
 while(p!=NULL)
 {
  int tat = p->ct-p->at;
  int wt = tat-p->bt;
```

```
  avg_tat+=tat;
  avg_wt+=wt;

  printf("%s\t%d\t%d\t%d\t%d\t%d\t%d\n",
   p->pname,p->at,p->bt,p->p,p->ct,tat,wt);

  p=p->next;
 }

 printf("Avg TAT=%f\tAvg WT=%f\n",
   avg_tat/n,avg_wt/n);
}

void print_input()
{
 NODE *p;

 p = first;

 printf("pname\tat\tbt\tp\n");
 while(p!=NULL)
 {
  printf("%s\t%d\t%d\t%d\n",
   p->pname,p->at,p->bt1,p->p);
  p = p->next;
 }
}

void sort()
{
 NODE *p,*q;
 int t;
 char name[20];
```

```
p = first;
while(p->next!=NULL)
{
 q=p->next;
 while(q!=NULL)
 {
  if(p->at > q->at)
  {
   strcpy(name,p->pname);
   strcpy(p->pname,q->pname);
   strcpy(q->pname,name);

   t = p->at;
   p->at = q->at;
   q->at = t;

   t = p->bt;
   p->bt = q->bt;
   q->bt = t;

   t = p->ct;
   p->ct = q->ct;
   q->ct = t;

   t = p->bt1;
   p->bt1 = q->bt1;
   q->bt1 = t;

   t = p->p;
   p->p = q->p;
   q->p = t;

  }

  q=q->next;
```

```
  }

  p=p->next;
 }
}

int time;

NODE * get_p()
{
 NODE *p,*min_p=NULL;
 int min=9999;

 p = first;
 while(p!=NULL)
 {
  if(p->at<=time && p->bt1!=0 &&
   p->p<min)
  {
   min = p->p;
   min_p = p;
  }
  p=p->next;
 }

 return min_p;
}

struct gantt_chart
{
 int start;
 char pname[30];
 int end;
}s[100],s1[100];
```

```
int k;

void pnp()
{
 int prev=0,n1=0;
 NODE *p;

 while(n1!=n)
 {
  p = get_p();

  if(p==NULL)
  {
   time++;
   s[k].start = prev;
   strcpy(s[k].pname,"*");
   s[k].end = time;

   prev = time;
   k++;
  }
  else
  {
   time++;
   s[k].start = prev;
   strcpy(s[k].pname, p->pname);
   s[k].end = time;

   prev = time;
   k++;

   p->ct = time;
   p->bt1--;

   if(p->bt1==0) n1++;
```

```
  }

  print_input();
  sort();
 }
}

void print_gantt_chart()
{
 int i,j,m;

 s1[0] = s[0];

 for(i=1,j=0;i<k;i++)
 {
  if(strcmp(s[i].pname,s1[j].pname)==0)
   s1[j].end = s[i].end;
  else
   s1[++j] = s[i];
 }

 printf("%d",s1[0].start);
 for(i=0;i<=j;i++)
 {
  m = (s1[i].end - s1[i].start);

  for(k=0;k<m/2;k++)
   printf("-");

  printf("%s",s1[i].pname);

  for(k=0;k<(m+1)/2;k++)
   printf("-");

  printf("%d",s1[i].end);
```

```
 }
}

int main()
{
 accept_info();
 sort();
 pnp();
 print_output();
 print_gantt_chart();

 return 0;
}
```

Assignment No. 4: Demand Paging

SET A

1.Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string.
Give input n as the number of memory frames.
Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

1) Implement FIFO

2) Implement LRU

FIFO

```
#include<stdio.h>
#define MAX 20
int frames[MAX],ref[MAX],mem[MAX][MAX],faults,sp,m,n;
void accept()
{
int i;
```

```
printf("Enter no.of frames:");
scanf("%d", &n);
printf("Enter no.of references:");
scanf("%d", &m);
printf("Enter reference string:\n");
for(i=0;i<m;i++)
{
printf("[%d]=",i);
scanf("%d",&ref[i]);
}
}
void disp()
{
int i,j;
for(i=0;i<m;i++)
printf("%3d",ref[i]);
printf("\n\n");
for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
{
if(mem[i][j])
printf("%3d",mem[i][j]);
else
printf(" ");

}
printf("\n");
}
printf("Total Page Faults: %d\n",faults);
}
int search(int pno)
{
int i;
for(i=0;i<n;i++)
{
if(frames[i]==pno)
return i;
```

```
}
return -1;
}
void fifo()
{
int i,j;
for(i=0;i<m;i++)
{
if(search(ref[i])==-1)
{
frames[sp] = ref[i];
sp = (sp+1)%n;
faults++;
for(j=0;j<n;j++)
mem[j][i] = frames[j];
}
}
}
int main()
{
accept();
fifo();
disp();
return 0;
}
```

LRU

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,
sp,m,n,time[MAX];
void accept()
{
int i;
printf("Enter no.of frames:");
scanf("%d", &n);
```

```
printf("Enter no.of references:");
scanf("%d", &m);
printf("Enter reference string:\n");
for(i=0;i<m;i++)
{
printf("[%d]=",i);
scanf("%d",&ref[i]);
}
}
void disp()
{
int i,j;
for(i=0;i<m;i++)
printf("%3d",ref[i]);
printf("\n\n");
for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
{
if(mem[i][j])
printf("%3d",mem[i][j]);
else
printf(" ");
}
printf("\n");
}
printf("Total Page Faults: %d\n",faults);
}
int search(int pno)
{
int i;
for(i=0;i<n;i++)
{

if(frames[i]==pno)
return i;
}
return -1;
```

```
}
int get_lru()
{
int i,min_i,min=9999;
for(i=0;i<n;i++)
{
if(time[i]<min)
{
min = time[i];
min_i = i;
}
}
return min_i;
}

void lru()
{
int i,j,k;
for(i=0;i<m && sp<n;i++)
{
k=search(ref[i]);
if(k==-1)
{
frames[sp]=ref[i];
time[sp]=i;
faults++;
sp++;
for(j=0;j<n;j++)
mem[j][i]=frames[j];
}
else
time[k]=i;
}
for(;i<m;i++)
{
k = search(ref[i]);
if(k==-1)
{
```

```
sp = get_lru();

frames[sp] = ref[i];
time[sp] = i;
faults++;
for(j=0;j<n;j++)
mem[j][i] = frames[j];
}
else
time[k]=i;
}
}

int main()
{
accept();
lru();
disp();
return 0;
}
```

SET B

1.Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string.
Give input n as the number of memory frames
Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

1) Implement OPT
2) Implement MFU

OPT

```
#include<stdio.h>
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2,
flag3, i, j, k, pos, max, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);

   printf("Enter number of pages: ");
   scanf("%d", &no_of_pages);

   printf("Enter page 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]){
               flag1 = flag2 = 1;
               break;
            }
      }

      if(flag1 == 0){
         for(j = 0; j < no_of_frames; ++j){
            if(frames[j] == -1){
               faults++;
               frames[j] = pages[i];
               flag2 = 1;
```

```
            break;
          }
       }
    }

if(flag2 == 0){
 flag3 =0;

    for(j = 0; j < no_of_frames; ++j){
     temp[j] = -1;

     for(k = i + 1; k < no_of_pages; ++k){
     if(frames[j] == pages[k]){
     temp[j] = k;
     break;
     }
     }
    }

    for(j = 0; j < no_of_frames; ++j){
     if(temp[j] == -1){
     pos = j;
     flag3 = 1;
     break;
     }
    }

    if(flag3 ==0){
     max = temp[0];
     pos = 0;

     for(j = 1; j < no_of_frames; ++j){
     if(temp[j] > max){
     max = temp[j];
     pos = j;
     }
     }
    }
```

```
frames[pos] = pages[i];
faults++;
        }

        printf("\n");

        for(j = 0; j < no_of_frames; ++j){
            printf("%d\t", frames[j]);
        }
    }

    printf("\n\nTotal Page Faults = %d", faults);

    return 0;
}
```

MFU

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,
 sp,m,n,count[MAX];

void accept()
{
 int i;

 printf("Enter no.of frames:");
 scanf("%d", &n);

 printf("Enter no.of references:");
 scanf("%d", &m);

 printf("Enter reference string:\n");
```

```
 for(i=0;i<m;i++)
 {
  printf("[%d]=",i);
  scanf("%d",&ref[i]);
 }
}

void disp()
{
 int i,j;

 for(i=0;i<m;i++)
  printf("%3d",ref[i]);

 printf("\n\n");

 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   if(mem[i][j])
    printf("%3d",mem[i][j]);
   else
    printf("   ");
  }
  printf("\n");
 }

 printf("Total Page Faults: %d\n",faults);
}

int search(int pno)
{
 int i;

 for(i=0;i<n;i++)
 {
  if(frames[i]==pno)
```

```
   return i;
 }

 return -1;
}

int get_mfu(int sp)
{
 int i,max_i,max=-9999;

 i=sp;
 do
 {
  if(count[i]>max)
  {
   max = count[i];
   max_i = i;
  }
  i=(i+1)%n;
 }while(i!=sp);

 return max_i;
}

void mfu()
{
 int i,j,k;

 for(i=0;i<m && sp<n;i++)
 {
  k=search(ref[i]);
  if(k==-1)
  {
   frames[sp]=ref[i];
   count[sp]++;
   faults++;
   sp++;
```

```
   for(j=0;j<n;j++)
    mem[j][i]=frames[j];
  }
  else
   count[k]++;

 }

 sp=0;
 for(;i<m;i++)
 {
  k = search(ref[i]);
  if(k==-1)
  {
   sp = get_mfu(sp);
   frames[sp] = ref[i];
   count[sp]=1;
   faults++;
   sp = (sp+1)%n;

   for(j=0;j<n;j++)
    mem[j][i] = frames[j];
  }
  else
   count[k]++;
 }
}


int main()
{
 accept();
 mfu();
 disp();

 return 0;
}
```

********** SET A
****
1.Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames.
Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

1) Implement FIFO

2) Implement LRU

**********FIFO ANSWER**********

```
#include<stdio.h>
#define MAX 20
int frames[MAX],ref[MAX],mem[MAX][MAX],faults,sp,m,n;
void accept()
{
int i;
printf("Enter no.of frames:");
scanf("%d", &n);
printf("Enter no.of references:");
scanf("%d", &m);
printf("Enter reference string:\n");
for(i=0;i<m;i++)
{
printf("[%d]=",i);
scanf("%d",&ref[i]);
}
}
void disp()
{
int i,j;
for(i=0;i<m;i++)
printf("%3d",ref[i]);
printf("\n\n");
for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
{
if(mem[i][j])
printf("%3d",mem[i][j]);
else
printf(" ");

}
printf("\n");
}
printf("Total Page Faults: %d\n",faults);
}
int search(int pno)
{
```

```
int i;
for(i=0;i<n;i++)
{
if(frames[i]==pno)
return i;
}
return -1;
}
void fifo()
{
int i,j;
for(i=0;i<m;i++)
{
if(search(ref[i])==-1)
{
frames[sp] = ref[i];
sp = (sp+1)%n;
faults++;
for(j=0;j<n;j++)
mem[j][i] = frames[j];
}
}
}
int main()
{
accept();
fifo();
disp();
return 0;
}
```

********** LRU ANSWER **********

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,
sp,m,n,time[MAX];
void accept()
{
int i;
printf("Enter no.of frames:");
scanf("%d", &n);
printf("Enter no.of references:");
scanf("%d", &m);
printf("Enter reference string:\n");
for(i=0;i<m;i++)
{
printf("[%d]=",i);
scanf("%d",&ref[i]);
}
}
void disp()
```

```
{
int i,j;
for(i=0;i<m;i++)
printf("%3d",ref[i]);
printf("\n\n");
for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
{
if(mem[i][j])
printf("%3d",mem[i][j]);
else
printf(" ");
}
printf("\n");
}
printf("Total Page Faults: %d\n",faults);
}
int search(int pno)
{
int i;
for(i=0;i<n;i++)
{

if(frames[i]==pno)
return i;
}
return -1;
}
int get_lru()
{
int i,min_i,min=9999;
for(i=0;i<n;i++)
{
if(time[i]<min)
{
min = time[i];
min_i = i;
}
}
return min_i;
}

void lru()
{
int i,j,k;
for(i=0;i<m && sp<n;i++)
{
k=search(ref[i]);
if(k==-1)
{
frames[sp]=ref[i];
time[sp]=i;
faults++;
sp++;
```

```
for(j=0;j<n;j++)
mem[j][i]=frames[j];
}
else
time[k]=i;
}
for(;i<m;i++)
{
k = search(ref[i]);
if(k==-1)
{
sp = get_lru();

frames[sp] = ref[i];
time[sp] = i;
faults++;
for(j=0;j<n;j++)
mem[j][i] = frames[j];
}
else
time[k]=i;
}
}

int main()
{
accept();
lru();
disp();
return 0;
}
```

********** SET B **********

1.Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string.
Give input n as the number of memory frames
Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

1) Implement OPT
2) Implement MFU

********** OPT ANSWER **********

```
#include<stdio.h>
int main()
{
    int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max,
faults = 0;
    printf("Enter number of frames: ");
```

```
scanf("%d", &no_of_frames);

printf("Enter number of pages: ");
scanf("%d", &no_of_pages);

printf("Enter page 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]){
              flag1 = flag2 = 1;
              break;
          }
    }

    if(flag1 == 0){
        for(j = 0; j < no_of_frames; ++j){
            if(frames[j] == -1){
                faults++;
                frames[j] = pages[i];
                flag2 = 1;
                break;
            }
        }
    }

    if(flag2 == 0){
     flag3 =0;

        for(j = 0; j < no_of_frames; ++j){
         temp[j] = -1;

         for(k = i + 1; k < no_of_pages; ++k){
         if(frames[j] == pages[k]){
         temp[j] = k;
         break;
         }
         }
        }

        for(j = 0; j < no_of_frames; ++j){
         if(temp[j] == -1){
         pos = j;
         flag3 = 1;
         break;
```

```
            }
           }

           if(flag3 ==0){
            max = temp[0];
            pos = 0;

            for(j = 1; j < no_of_frames; ++j){
            if(temp[j] > max){
            max = temp[j];
            pos = j;
            }
            }
           }
frames[pos] = pages[i];
faults++;
       }

       printf("\n");

       for(j = 0; j < no_of_frames; ++j){
           printf("%d\t", frames[j]);
       }
   }

   printf("\n\nTotal Page Faults = %d", faults);

   return 0;
}
```

********** MFU ANSWER **********

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,
 sp,m,n,count[MAX];

void accept()
{
 int i;

 printf("Enter no.of frames:");
 scanf("%d", &n);

 printf("Enter no.of references:");
 scanf("%d", &m);

 printf("Enter reference string:\n");
 for(i=0;i<m;i++)
 {
```

```
  printf("[%d]=",i);
  scanf("%d",&ref[i]);
 }
}

void disp()
{
 int i,j;

 for(i=0;i<m;i++)
  printf("%3d",ref[i]);

 printf("\n\n");

 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   if(mem[i][j])
    printf("%3d",mem[i][j]);
   else
    printf("   ");
  }
  printf("\n");
 }

 printf("Total Page Faults: %d\n",faults);
}

int search(int pno)
{
 int i;

 for(i=0;i<n;i++)
 {
  if(frames[i]==pno)
   return i;
 }

 return -1;
}

int get_mfu(int sp)
{
 int i,max_i,max=-9999;

 i=sp;
 do
 {
  if(count[i]>max)
  {
   max = count[i];
   max_i = i;
  }
  i=(i+1)%n;
```

```
 }while(i!=sp);

 return max_i;
}

void mfu()
{
 int i,j,k;

 for(i=0;i<m && sp<n;i++)
 {
  k=search(ref[i]);
  if(k==-1)
  {
   frames[sp]=ref[i];
   count[sp]++;
   faults++;
   sp++;

   for(j=0;j<n;j++)
    mem[j][i]=frames[j];
  }
  else
   count[k]++;

 }

 sp=0;
 for(;i<m;i++)
 {
  k = search(ref[i]);
  if(k==-1)
  {
   sp = get_mfu(sp);
   frames[sp] = ref[i];
   count[sp]=1;
   faults++;
   sp = (sp+1)%n;

   for(j=0;j<n;j++)
    mem[j][i] = frames[j];
  }
  else
   count[k]++;
 }
}


int main()
{
 accept();
 mfu();
 disp();

 return 0;
```

}