Slip No-1

Q.1 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults according to the LFU page replacement algorithm. Assume the memory of n frames.
Reference String : 3,4,5,4,3,4,7,2,4,5,6,7,2,4,6

Q.2 Write a C program to implement the 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
typeline +n filename :- To print first n lines in the file.
typeline -a filename :- To print all lines in the file.

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

void typeline(char *tok2,char *tok3){

    char ch,str[30];
    int lc=0,s,count=0;
    FILE *fp;
    fp = fopen(tok3,"r");


    if(fp == NULL){
        printf("File does not exist.\n");
    }else{
        printf("File exist\n");
    }

    if(strcmp(tok2,"a") == 0){
        while(!feof(fp)){
            ch = fgetc(fp);
            printf("%c",ch);
        }
    }

    else{
        int n = atoi(tok2);
        if(n>0){
            while(lc<n){
                ch = fgetc(fp);
                if(ch == '\n')
                lc++;
                printf("%c",ch);
            }
        }

        if(n<0){
            while(!feof(fp)){
                ch = fgetc(fp);
                if(ch == '\n')
                lc++;
            }

            s = lc + n;
            s++;

            fseek(fp,0,SEEK_SET);

            while(!feof(fp)){
                ch = fgetc(fp);
                    while (count!=s)
                    {
                ch = fgetc(fp);
                if(ch == '\n')
                    count++;
                    }
                    printf("%c",ch);
            }
        }
    }
}

int main(){
    char tok1[10],tok2[10],tok3[10],tok4[10],cmd[30];
    int choice;

    while(1){
        printf("\n$MYShell$:");
        gets(cmd);

        if(strcmp(cmd,"exit") == 0){
            exit(0);
        }

        int choice = sscanf(cmd,"%s%s%s%s",&tok1,&tok2,&tok3,&tok4);
        if(strcmp(tok1,"typeline") == 0){
            typeline(tok2,tok3);
            continue;
        }
    }
}
```

Slip No-2

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **FIFO** page replacement algorithm. Assume the memory of n frames.
Reference String : 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6

```
#include<stdio.h>

int page[40];
int n,frame[5],ref[50],victim=-1,pf=0;

int searchp(int p)
{
    int i;
    for(i=0;i<n;i++)
    if(frame[i]==p)
    return i;
    return -1;
}
int selectvictim()
{
    victim++;
    return victim%n;
}
int main()
{
    int i,m;
    printf("\nEnter the no. of pages -: ");
    scanf("%d",&m);

    printf("\nEnter the reference string -: ");
    for(i=0;i<m;i++)
    scanf("%d",&page[i]);

    printf("\nEnter the no. of frames -: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    frame[i]=-1;
    for(i=0;i<m;i++)
    {
        int k,j;
        k=searchp(page[i]);
        if(k==-1)
        {
            pf++;
            k=selectvictim();
            frame[k]=page[i];
        }
        printf("\n %d",page[i]);
        for(j=0;j<n;j++)
        printf("\t%d",frame[j]);
    }
    printf("\n");
    printf("\nPage Fault -: %d",pf);
}
/*
Enter the no. of pages -: 15
Enter the reference string -: 3 4 5 6 3 4 7 3 4 5 6 7 2 4 6
Enter the no. of frames -: 3
 3      3       -1      -1
 4      3       4       -1
 5      3       4       5
 6      6       4       5
 3      6       3       5
 4      6       3       4
 7      7       3       4
 3      7       3       4
 4      7       3       4
 5      7       5       4
 6      7       5       6
 7      7       5       6
 2      2       5       6
 4      2       4       6
 6      2       4       6
Page Fault -: 11
*/
```

Q.2 Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as
myshell$ list f dirname :- To print names of all the files in current directory.
myshell$ list n dirname :- To print the number of all entries in the current directory

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

 void list(char *tok2, char *tok3)
 {
   DIR *dp;
   int c =0;
   struct dirent *dir;
   dp = opendir(tok3);
    if(dp == 0)
       printf("%s directory not exit",tok3);
    else
      {
        if(strcmp(tok2,"n")==0)
          {
            while((dir=readdir(dp))!= NULL)
                c++;
            printf("no of files directory : %d ",c);
           }

        if(strcmp(tok2,"f")==0)
          {
            while((dir=readdir(dp))!= NULL)
                 printf("%s \n ",dir->d_name);
          }
      }
}

int main(int argc , char * argv[])
{
   char tok1[20],tok2[20],tok3[20];
   int choice,f;
   char cmd[40];
    while(1)
      {
        printf("\n Myshell$");
        gets(cmd);
        if(strcmp(cmd, "exit")==0)
            exit(0);
        int choice = sscanf(cmd,"%s%s%s",&tok1,&tok2,&tok3);
        if(choice==3)
           {
             if(strcmp(tok1,"list")==0)
                 list(tok2,tok3);
             continue;
           }
      }
}
```

Slip No-3

Q.1 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults according to the **LRU** (using counter method) page replacement algorithm. Assume the memory of n frames.
Reference String : 3,5,7,2,5,1,2,3,1,3,5,3,1,6,2

```
#include<stdio.h>

int pf=0,pg;
int f,frame[5],ref[50],victim=0,time[50];

int searchpage(int p)
{
    int i;
    for(i=0;i<f;i++)
    if(frame[i]==p)
    return i;
    return -1;
}

int selectvictim()
{
    int min=0,i;
    for(i=1;i<f;i++)
    {
        if(time[i]<time[min])
        min=i;
    }
    return min;
}

void main()
{
    int i;
    printf("\n Enter the no. of frames -: ");
    scanf("%d",&f);
    for(i=0;i<f;i++)
    frame[i]=-1;
    for(i=0;i<f;i++)
    time[i]=-1;

printf("\n Enter the no. of pages -: ");
scanf("%d",&pg);


printf("\n Enter the page(reference string) -: ");
for(i=0;i<pg;i++)
scanf("%d",&ref[i]);

for(i=0;i<pg;i++)
{
    int k,j;
    k=searchpage(ref[i]);
    if(k==-1)
    {
        pf++;
        k=selectvictim();
        frame[k]=ref[i];
    }
    time[k]=i;
        printf("\n pg=%d  |",ref[i]);
        for(j=0;j<f;j++)
        printf("\t%d",frame[j]);
    }
    printf("\n Page Fault is -:%d",pf);
}
/*
 Enter the no. of frames -: 3
 Enter the no. of pages -: 15
 Enter the page(reference string) -: 3 5 7 2 5 1 2 3 1 3 5 3 1 6 2
 pg=3  |        3       -1      -1
 pg=5  |        3       5       -1
 pg=7  |        3       5       7
 pg=2  |        2       5       7
 pg=5  |        2       5       7
 pg=1  |        2       5       1
 pg=2  |        2       5       1
 pg=3  |        2       3       1
 pg=1  |        2       3       1
```

```
pg=3  |        2       3       1
pg=5  |        5       3       1
pg=3  |        5       3       1
pg=1  |        5       3       1
pg=6  |        6       3       1
pg=2  |        6       2       1
Page Fault is -:9
*/
```

Q.2 Write a programto implement the toy shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.
count c filename :- To print number of characters in the file.
count w filename :- To print number of words in the file.
count l filename :- To print number of lines in the file.

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

void count(char *tok2 ,char *tok3)
{
    int l=0,w=0,c=0;
    FILE *fp;
    fp = fopen(tok3,"r");
     if(fp==NULL)
        printf("\n file not exist");
     else
       {
         while(!feof(fp))
          {
             char ch;
             ch = fgetc(fp);
             if(ch == ' ' )
                 w++;
             else if(ch == '\n')
                    {
                      w++;
                      l++;
                    }
                   else
                     c++;
          }
         if(strcmp(tok2,"w") == 0)
              printf("\n word count :%d",w);

         if(strcmp(tok2,"c") == 0)
              printf("\n character count :%d",c);

         if(strcmp(tok2,"l") == 0)
              printf("\n line count :%d",l);

       }
   }

int main(int argc , char *argv[])
 {
    char command[30],tok1[30],tok2[30],tok3[30];
    while(1)
       {
          printf("\n MyShell:");
          gets(command);

          int ch = sscanf(command,"%s%s%s",tok1,tok2,tok3);
          if(ch == 3)
            {
              if(strcmp(tok1,"count")==0)
                 count(tok2,tok3);
                 continue;
            }
       }
 }
```

Slip No-4

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **MFU** page replacement algorithm. Assume the memory of n frames.
Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2

```c
#include<stdio.h>

struct node
{
    int pno,freq;
}frames[20];

int n;
int page_found(int pno)
{
    int fno;
    for(fno=0;fno<n;fno++)
    if(frames[fno].pno==pno)
    return fno;
    return -1;
}

int get_free_frame()
{
    int fno;
    for(fno=0;fno<=n;fno++)
    if(frames[fno].pno==-1)
    return(fno);
    return(-1);
}

int get_mfu_frame()
{
    int fno;
    int selfno=0;
    for(fno=1;fno<n;fno++)
    if(frames[fno].freq>frames[selfno].freq)
    selfno=fno;
    return selfno;
}

void main()
{
    int p_request[1000];
    int size;
    int page_fault=0,i,j,fno;

    printf("\n Enter how many pages -: ");
    scanf("%d",&size);

    printf("\n Enter Reference string -: ");
    for(i=0;i<size;i++)
    scanf("%d",&p_request[i]);

    printf("\n How many frames -: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        frames[i].pno=-1;
        frames[i].freq=0;
    }
    printf("\n Page No    Page Frames      Page Fault");
    printf("\n-----------------------------------------------------");
    for(i=0;i<size;i++)
    {
        j=page_found(p_request[i]);
        if(j==-1)
        {
            j=get_free_frame();
            if(j==-1)
            j=get_mfu_frame();
            page_fault++;
            frames[j].pno=p_request[i];
            frames[j].freq=1;
            printf("\n %d\t",p_request[i]);
            for(fno=0;fno<n;fno++)
            printf("%d\t",frames[fno].pno);
```

```
            printf(" : YES ");
        }
        else
        {
            printf("\n%d\t",p_request[i]);
            frames[j].freq++;
            for(fno=0;fno<n;fno++)
            printf("%d\t",frames[fno].pno);
            printf(" : NO ");
        }
    }
    printf("\n----------------------------------------------------");
    printf("\n Number of Page_Falts -: %d",page_fault);
}
/*
Enter how many pages -: 15

 Enter Reference string -: 8 5 7 8 5 7 2 3 7 3 5 9 4 6 2

 How many frames -: 3

 Page No    Page Frames      Page Fault
----------------------------------------------------
 8      8       -1      -1       : YES
 5      8       5       -1       : YES
 7      8       5       7        : YES
8       8       5       7        : NO
5       8       5       7        : NO
7       8       5       7        : NO
 2      2       5       7        : YES
 3      2       3       7        : YES
7       2       3       7        : NO
3       2       3       7        : NO
 5      2       3       5        : YES
 9      2       9       5        : YES
 4      4       9       5        : YES
 6      6       9       5        : YES
 2      2       9       5        : YES
----------------------------------------------------
 Number of Page_Falts -: 10
*/
```

Q.2 Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.
myshell$ search a filename pattern :- To search all the occurrence of pattern in the file.
myshell$ search c filename pattern :- To count the number of occurrence
of pattern in the file.

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

void search(char *tok2,char *tok3,char *tok4)//tok2:command tok3:pattern tok4:Textfilename
{
    FILE *fp;
    int lineno=0,count=0;
    char str[50];
    fp=fopen(tok4,"r");
    if(fp==NULL)
    {
        printf("File does not exist");
        exit(0);
    }
    else
    {
        if(strcmp(tok2,"c")==0)//Total number of occurances of pattern
        {
            while(!feof(fp))
            {
                fgets(str,50,fp);
                if(strstr(str,tok3))
                {
                    count++;
                }
            }
            printf("Total number occurance of pattern(%s) is : %d\n",tok3,count);
        }

        if(strcmp(tok2,"a")==0)//Total occurances of pattern
        {
            while(!feof(fp))
            {
                fgets(str,50,fp);
                lineno++;
```

```
                if(strstr(str,tok3))
                {
                    printf("\n Total Occurance of pattern:\n %d %s\n",lineno,str);
                }
            }

        }

    }
}

void main(int argc,char *argv[])
{
    char cmd[20],tok1[20],tok2[20],tok3[20],tok4[20];
    while(1)
    {
        printf("\nMyShell$:");
        gets(cmd);

            if(strcmp(tok1,"exit")==0)
            exit(0);

        int ch=sscanf(cmd,"%s%s%s%s",tok1,tok2,tok3,tok4);
        if(ch==4)
        {

            if(strcmp(tok1,"search")==0)
            {
            search(tok2,tok3,tok4);
            continue;
            }
        }
    }
}
```

Slip No-5

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **optimal page replacement** algorithm. Assume the memory of n frames.
Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2

```
#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("\n pg=%d |",pages[i]);
                printf("%d\t",frames[j]);
            }
        }
        printf("\n Total Page Faults -: %d",faults);
        return 0;
    }
    /*
    Enter number of frames -: 3
    Enter number of pages -: 15
    Enter page reference string -: 8 5 7 8 5 7 2 3 7 3 5 9 4 6 2
     pg=8 |8
     pg=8 |-1
     pg=8 |-1

     pg=5 |8
     pg=5 |5
     pg=5 |-1

     pg=7 |8
     pg=7 |5
     pg=7 |7

     pg=8 |8
     pg=8 |5
     pg=8 |7

     pg=5 |8
     pg=5 |5
     pg=5 |7

     pg=7 |8
     pg=7 |5
     pg=7 |7

     pg=2 |2
     pg=2 |5
     pg=2 |7

     pg=3 |3
     pg=3 |5
     pg=3 |7

     pg=7 |3
     pg=7 |5
     pg=7 |7

     pg=3 |3
     pg=3 |5
     pg=3 |7

     pg=5 |3
     pg=5 |5
     pg=5 |7

     pg=9 |9
     pg=9 |5
     pg=9 |7
```

```
 pg=4 |4
 pg=4 |5
 pg=4 |7

 pg=6 |6
 pg=6 |5
 pg=6 |7

 pg=2 |2
 pg=2 |5
 pg=2 |7
 Total Page Faults -: 9
*/
```

Q.2 Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.
myshell$ search f filename pattern :- To display first occurrence of pattern in the file.
myshell$ search c filename pattern :- To count the number of occurrence
of pattern in the file.

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

void search(char *tok2,char *tok3,char *tok4)//tok2:command tok3:pattern tok4:Textfilename
{
    FILE *fp;
    int lineno=0,count=0;
    char str[50];
    fp=fopen(tok4,"r");
    if(fp==NULL)
    {
        printf("File does not exist");
                exit(0);
}
else
{

    if(strcmp(tok2,"f")==0)//First occurance of pattern
    {
        while(!feof(fp))
        {
            fgets(str,50,fp);
            lineno++;
            if(strstr(str,tok3))
            {
            printf("First occurance of '%s' is on line : %d\n",tok3,lineno);
            break;
            }
        }
    }


    if(strcmp(tok2,"c")==0)//Total number of occurances of pattern
    {
            while(!feof(fp))
            {
                fgets(str,50,fp);
                if(strstr(str,tok3))
                {
                    count++;
                }
            }
            printf("Total number occurance of pattern(%s) is : %d\n",tok3,count);
        }
    }
}

void main(int argc,char *argv[])
{
    char cmd[20],tok1[20],tok2[20],tok3[20],tok4[20];
    while(1)
    {
        printf("\nMyShell$:");
        gets(cmd);
```

```
            if(strcmp(tok1,"exit")==0)
            exit(0);

        int ch=sscanf(cmd,"%s%s%s%s",tok1,tok2,tok3,tok4);
        if(ch==4)
        {

            if(strcmp(tok1,"search")==0)
            {
            search(tok2,tok3,tok4);
            continue;
            }
        }
    }
}
```

Slip No-6

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **MRU** page replacement algorithm. Assume the memory of n frames.
Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2

```
#include<stdio.h>

int main()
{
    int ref_str[20],frame[10];
    int f,n,i,k,fcount=0,avail;

    printf("\nPls. Enter the no. of pages:");
    scanf("%d",&n);

    printf("\nEnter the Reference String:");
    for(i=1;i<=n;i++)
        scanf("%d",&ref_str[i]);

printf("\nEnter the no. of frames:");
scanf("%d",&f);

for(i=0;i<f;i++)
    frame[i] = -1;

printf("\nRef String \t Frames \t page falut or hit\n");

for(i=1;i<=n;i++)
{
    printf("%d",ref_str[i]);
    avail = 0;

       for(k=0;k<f;k++)
           if(frame[k] == ref_str[i])           // check ref_string already in frame
               {
                avail =1;
                 for(k=0;k<f;k++)
                     printf("\t%d",frame[k]);
                 printf("\tH");
     }

 else if(frame[k] == -1)
       {
         avail =1;
         fcount++;
         frame[k] =  ref_str[i];
         for(k=0; k<f; k++)
            printf("\t%d",frame[k]);
          printf("\tF");
          break;
       }


if(avail == 0)                          // if ref_string is not in frame
{
    for(k=0; k<f; k++)
     {
      if(frame[k] == ref_str[i-1] )   // check most recently used page (ref_str[i-1])
        {
                    frame[k] = ref_str[i];

                    fcount++;
                    for(k=0;k<f;k++)
                     {
                        printf("\t%d",frame[k]);
                     }
                    printf("\tF");
                }
            }
        }
        printf("\n");

    }
                printf("\nTotal number of Page Fault: %d",fcount);

}
/*
```

```
Pls. Enter the no. of pages: 15
Enter the Reference String:8 5 7 8 5 7 2 3 7 3 5 9 4 6 2
Enter the no. of frames:3

Ref String       Frames          page falut or hit
8       8       -1      -1      F
5       8       5       -1      F
7       8       5       7       F
8       8       5       7       H
5       8       5       7       H
7       8       5       7       H
2       8       5       2       F
3       8       5       3       F
7       8       5       7       F
3       8       5       3       F
5       8       5       3       H
9       8       9       3       F
4       8       4       3       F
6       8       6       3       F
2       8       2       3       F

Total number of Page Fault: 11
*/
```

Q.2 Write a programto implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.
myshell$ search f filename pattern :- To display first occurrence of pattern in the file.
myshell$ search a filename pattern :- To search all the occurrence of pattern in the file.

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

void search(char *tok2,char *tok3,char *tok4)//tok2:command tok3:pattern tok4:Textfilename
{
    FILE *fp;
    int lineno=0,count=0;
    char str[50];
    fp=fopen(tok4,"r");
    if(fp==NULL)
    {
        printf("File does not exist");
                    exit(0);
    }
    else
    {
        if(strcmp(tok2,"f")==0)//First occurance of pattern
        {
            while(!feof(fp))
            {
                fgets(str,50,fp);
                lineno++;
            if(strstr(str,tok3))
            {
            printf("First occurance of '%s' is on line : %d\n",tok3,lineno);
            break;
            }
        }
}
if(strcmp(tok2,"a")==0)//Total occurances of pattern
{
    while(!feof(fp))
    {
        fgets(str,50,fp);
        lineno++;
        if(strstr(str,tok3))
        {
            printf("\n Total Occurance of pattern:\n %d %s\n",lineno,str);
        }
    }

}
```

```
    }
}

void main(int argc,char *argv[])
{
    char cmd[20],tok1[20],tok2[20],tok3[20],tok4[20];
    while(1)
    {
        printf("\nMyShell$:");
        gets(cmd);

            if(strcmp(tok1,"exit")==0)
            exit(0);

        int ch=sscanf(cmd,"%s%s%s%s",tok1,tok2,tok3,tok4);
        if(ch==4)
        {

            if(strcmp(tok1,"search")==0)
            {
            search(tok2,tok3,tok4);
            continue;
            }
        }
    }
}
```

Slip No-7

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **Optimal page replacement** algorithm. Assume the memory of n frames.
Reference String : 7, 5, 4, 8, 5, 7, 2, 3, 1, 3, 5, 9, 4, 6, 2

```c
#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("\n pg=%d |",pages[i]);
        printf("%d\t",frames[j]);
            } }
        printf("\n Total Page Faults -: %d",faults);
        return 0;
    }
    /*
    Enter number of frames -: 3
    Enter number of pages -: 15
    Enter page reference string -: 7 5 4 8 5 7 2 3 1 3
    5 9 4 6 2
     pg=7 |7
     pg=7 |-1
     pg=7 |-1

     pg=5 |7
     pg=5 |5
     pg=5 |-1

     pg=4 |7
     pg=4 |5
     pg=4 |4

     pg=8 |7
     pg=8 |5
     pg=8 |8

     pg=5 |7
     pg=5 |5
     pg=5 |8

     pg=7 |7
     pg=7 |5
     pg=7 |8

     pg=2 |2
     pg=2 |5
     pg=2 |8

     pg=3 |2
     pg=3 |5
     pg=3 |3

     pg=1 |1
     pg=1 |5
     pg=1 |3

     pg=3 |1
     pg=3 |5
     pg=3 |3

     pg=5 |1
     pg=5 |5
     pg=5 |3

     pg=9 |9
     pg=9 |5
     pg=9 |3

     pg=4 |4
     pg=4 |5
```

```
 pg=4 |3

 pg=6 |6
 pg=6 |5
 pg=6 |3

 pg=2 |2
 pg=2 |5
 pg=2 |3
 Total Page Faults -: 11
*/
```

Q.2 Write a program to implement shell. It should display the command prompt
"myshell$". Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.
myshell$ search a filename pattern :- To search all the occurrence of pattern in the file.
myshell$ search c filename pattern :- To count the number of occurrence
of pattern in the file.

```c
#include<stdio.h>

#include<unistd.h>
#include<string.h>
#include<stdlib.h>
void search(char *tok2,char *tok3,char *tok4)//tok2:command tok3:pattern tok4:Textfilename
{
    FILE *fp;
    int lineno=0,count=0;
    char str[50];
    fp=fopen(tok4,"r");
    if(fp==NULL)
    {
        printf("File does not exist");
                    exit(0);
    }
    else
    {
        if(strcmp(tok2,"c")==0)//Total number of occurances of pattern
        {
            while(!feof(fp))
            {
                fgets(str,50,fp);
                if(strstr(str,tok3))
                {
                    count++;
                }
            }
            printf("Total number occurance of pattern(%s) is : %d\n",tok3,count);
        }
        if(strcmp(tok2,"a")==0)//Total occurances of pattern
        {
            while(!feof(fp))
            {
                fgets(str,50,fp);
                lineno++;
                if(strstr(str,tok3))
                {
                    printf("\n Total Occurance of pattern:\n %d %s\n",lineno,str);
                }
            }
        }
    }
}
void main(int argc,char *argv[])
{
    char cmd[20],tok1[20],tok2[20],tok3[20],tok4[20];
    while(1)
    {
        printf("\nMyShell$:");
        gets(cmd);

            if(strcmp(tok1,"exit")==0)
            exit(0);

        int ch=sscanf(cmd,"%s%s%s%s",tok1,tok2,tok3,tok4);
        if(ch==4)
        {

            if(strcmp(tok1,"search")==0)
            {
            search(tok2,tok3,tok4);
            continue;
            }
        }
    }
}
```

Slip No-8

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **LRU** page replacement algorithm. Assume the memory of n frames.
Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2

```c
#include<stdio.h>

int pf=0,pg;
int f,frame[5],ref[50],victim=0,time[50];

int searchpage(int p)
{
    int i;
    for(i=0;i<f;i++)
    if(frame[i]==p)
    return i;
    return -1;
}

int selectvictim()
{
    int min=0,i;
    for(i=1;i<f;i++)
    {
        if(time[i]<time[min])
        min=i;
    }
    return min;
}

void main()
{
    int i;
    printf("\n Enter the no. of frames -: ");
    scanf("%d",&f);
    for(i=0;i<f;i++)
    frame[i]=-1;
    for(i=0;i<f;i++)
    time[i]=-1;

    printf("\n Enter the no. of pages -: ");
    scanf("%d",&pg);


    printf("\n Enter the page(reference string) -: ");
    for(i=0;i<pg;i++)
    scanf("%d",&ref[i]);

    for(i=0;i<pg;i++)
    {
        int k,j;
        k=searchpage(ref[i]);
        if(k==-1)
        {
            pf++;
            k=selectvictim();
            frame[k]=ref[i];
        }
        time[k]=i;
        printf("\n pg=%d  |",ref[i]);
        for(j=0;j<f;j++)
        printf("\t%d",frame[j]);
    }
    printf("\n Page Fault is -:%d",pf);
}


/*
Enter the no. of frames -: 3
Enter the no. of pages -: 15
Enter the page(reference string) -: 8 5 7 8 5 7 2 3 7 3 5 9 4 6 2
 pg=8  |        8       -1      -1
 pg=5  |        8       5       -1
 pg=7  |        8       5       7
 pg=8  |        8       5       7
 pg=5  |        8       5       7
 pg=7  |        8       5       7
 pg=2  |        2       5       7
```

```
 pg=3  |        2       3       7
 pg=7  |        2       3       7
 pg=3  |        2       3       7
 pg=5  |        5       3       7
 pg=9  |        5       3       9
 pg=4  |        5       4       9
 pg=6  |        6       4       9
 pg=2  |        6       4       2
 Page Fault is -:10
*/
```

Q.2 Write a programto implement the shell. It should display the command prompt
“myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.
myshell$ search f filename pattern :- To display first occurrence of pattern in the file.
myshell$ search c filename pattern :- To count the number of occurrence of pattern in the file.

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

void search(char *tok2,char *tok3,char *tok4)//tok2:command tok3:pattern tok4:Textfilename
{
    FILE *fp;
    int lineno=0,count=0;
    char str[50];
    fp=fopen(tok4,"r");
    if(fp==NULL)
    {
        printf("File does not exist");
                    exit(0);
    }
    else
    {

        if(strcmp(tok2,"f")==0)//First occurance of pattern
        {
        while(!feof(fp))
        {
            fgets(str,50,fp);
            lineno++;
            if(strstr(str,tok3))
            {
            printf("First occurance of '%s' is on line : %d\n",tok3,lineno);
            break;
            }
        }
}


if(strcmp(tok2,"c")==0)//Total number of occurances of pattern
{
    while(!feof(fp))
    {
        fgets(str,50,fp);
        if(strstr(str,tok3))
        {
            count++;
        }
        printf("Total number occurance of pattern(%s) is : %d\n",tok3,count);
    }
}
}

void main(int argc,char *argv[])
{
    char cmd[20],tok1[20],tok2[20],tok3[20],tok4[20];
    while(1)
    {
        printf("\nMyShell$:");
        gets(cmd);

            if(strcmp(tok1,"exit")==0)
            exit(0);

        int ch=sscanf(cmd,"%s%s%s%s",tok1,tok2,tok3,tok4);
        if(ch==4)
        {
```

```
            if(strcmp(tok1,"search")==0)
            {
            search(tok2,tok3,tok4);
            continue;
            }
        }
    }
}
```

Slip No-9

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **FIFO** page replacement algorithm. Assume the memory of n frames.
Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2

```c
#include<stdio.h>

int page[40];
int n,frame[5],ref[50],victim=-1,pf=0;

int searchp(int p)
{
    int i;
    for(i=0;i<n;i++)
    if(frame[i]==p)
    return i;
    return -1;
}

int selectvictim()
{
    victim++;
    return victim%n;
}

int main()
{
    int i,m;
    printf("\nEnter the no. of pages -: ");
    scanf("%d",&m);

    printf("\nEnter the reference string -: ");
    for(i=0;i<m;i++)
    scanf("%d",&page[i]);

    printf("\nEnter the no. of frames -: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    frame[i]=-1;
    for(i=0;i<m;i++)
    {
        int k,j;
        k=searchp(page[i]);
        if(k==-1)
        {
            pf++;
            k=selectvictim();
            frame[k]=page[i];
        }
        printf("\n %d",page[i]);
        for(j=0;j<n;j++)
        printf("\t%d",frame[j]);
    }
    printf("\n");
    printf("\nPage Fault -: %d",pf);
}
/*
Enter the no. of pages -: 15
Enter the reference string -: 8 5 7 8 5 7 2 3 7 3 5 9 4 6 2
Enter the no. of frames -: 3

 8      8       -1      -1
 5      8       5       -1
 7      8       5       7
 8      8       5       7
 5      8       5       7
 7      8       5       7
 2      2       5       7
 3      2       3       7
 7      2       3       7
 3      2       3       7
 5      2       3       5
 9      9       3       5
 4      9       4       5
 6      9       4       6
 2      2       4       6
```

```
Page Fault -: 10
*/
```

Q.2 Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.
myshell$ search f filename pattern :- To display first occurrence of pattern in the file.
myshell$ search a filename pattern :- To search all the occurrence of
pattern in the file.

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

void search(char *tok2,char *tok3,char *tok4)//tok2:command tok3:pattern tok4:Textfilename
{
    FILE *fp;
    int lineno=0,count=0;
    char str[50];
fp=fopen(tok4,"r");
if(fp==NULL)
{
    printf("File does not exist");
                exit(0);
}
else
{

    if(strcmp(tok2,"f")==0)//First occurance of pattern
    {
        while(!feof(fp))
        {
            fgets(str,50,fp);
            lineno++;
            if(strstr(str,tok3))
            {
            printf("First occurance of '%s' is on line : %d\n",tok3,lineno);
            break;
            }
        }
    }

        if(strcmp(tok2,"a")==0)//Total occurances of pattern
        {
            while(!feof(fp))
            {
                fgets(str,50,fp);
                lineno++;
                if(strstr(str,tok3))
                {
                    printf("\n Total Occurance of pattern:\n %d %s\n",lineno,str);
                }
            }

        }

    }
}

void main(int argc,char *argv[])
{
    char cmd[20],tok1[20],tok2[20],tok3[20],tok4[20];
    while(1)
    {
        printf("\nMyShell$:");
        gets(cmd);

            if(strcmp(tok1,"exit")==0)
            exit(0);

        int ch=sscanf(cmd,"%s%s%s%s",tok1,tok2,tok3,tok4);
        if(ch==4)
        {

            if(strcmp(tok1,"search")==0)
            {
            search(tok2,tok3,tok4);
            continue;
            }
        }
   }
}
```

Slip No-10

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **FIFO** page replacement algorithm. Assume the memory of n frames.
Reference String : 2, 4, 5, 6, 9, 4, 7, 3, 4, 5, 6, 7, 2, 4, 7, 1

```c
#include<stdio.h>

int page[40];
int n,frame[5],ref[50],victim=-1,pf=0;

int searchp(int p)
{
    int i;
    for(i=0;i<n;i++)
    if(frame[i]==p)
    return i;
    return -1;
}

int selectvictim()
{
    victim++;
    return victim%n;
}

int main()
{
    int i,m;
    printf("\nEnter the no. of pages -: ");
    scanf("%d",&m);

    printf("\nEnter the reference string -: ");
    for(i=0;i<m;i++)
    scanf("%d",&page[i]);

    printf("\nEnter the no. of frames -: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    frame[i]=-1;
    for(i=0;i<m;i++)
    {
        int k,j;
        k=searchp(page[i]);
        if(k==-1)
        {
            pf++;
            k=selectvictim();
            frame[k]=page[i];
        }
        printf("\n %d",page[i]);
        for(j=0;j<n;j++)
        printf("\t%d",frame[j]);
    }
    printf("\n");
    printf("\nPage Fault -: %d",pf);
}
/*
Enter the no. of pages -: 16
Enter the reference string -: 2 4 5 6 9 4 7 3 4 5 6 7 2 4 7 1
Enter the no. of frames -: 3

 2      2       -1      -1
 4      2       4       -1
 5      2       4       5
 6      6       4       5
 9      6       9       5
 4      6       9       4
 7      7       9       4
 3      7       3       4
 4      7       3       4
 5      7       3       5
 6      6       3       5
 7      6       7       5
 2      6       7       2
 4      4       7       2
 7      4       7       2
 1      4       1       2
```

```
Page Fault -: 14
*/
```

Q.2 Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as
myshell$ list f dirname :- To print names of all the files in current directory.
myshell$ list i dirname :- To print names and inodes of the files in the current directory.

```c
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>
#include<dirent.h>

 void list(char *tok2, char *tok3)
 {
   DIR *dp;
   int c =0;
   struct dirent *dir;
   dp = opendir(tok3);
   if(dp == 0)
      printf("%s directory not exit",tok3);
   else
     {
       if(strcmp(tok2,"f")==0)
         {
           while((dir=readdir(dp))!= NULL)
                printf("%s \n ",dir->d_name);
         }

       if(strcmp(tok2,"i")==0)
         {
           while((dir=readdir(dp))!= NULL)
               printf("%s\t\t%d\n",dir->d_name,dir->d_ino);
         }
     }
}

int main(int argc , char * argv[])
{
   char tok1[20],tok2[20],tok3[20];
   int choice,f;
   char cmd[40];
    while(1)
      {
        printf("\n Myshell$");
        gets(cmd);
        if(strcmp(cmd, "exit")==0)
            exit(0);
        int choice = sscanf(cmd,"%s%s%s",&tok1,&tok2,&tok3);
        if(choice==3)
           {
             if(strcmp(tok1,"list")==0)
                 list(tok2,tok3);
             continue;
           }
      }
}
```

Slip No-11

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **LFU** page replacement algorithm. Assume the memory of n frames.
Reference String : 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6

Q.2 Write a C program to implement the shell. It should display the command prompt "myshell$". Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following 'list' commands as
myshell$ list f dirname :- To print names of all the files in current directory.
myshell$ list n dirname :- To print the number of all entries in the current directory

```c
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>
#include<dirent.h>

 void list(char *tok2, char *tok3)
 {
   DIR *dp;
   int c =0;
   struct dirent *dir;
   dp = opendir(tok3);
    if(dp == 0)
       printf("%s directory not exit",tok3);
    else
      {
        if(strcmp(tok2,"n")==0)
          {
            while((dir=readdir(dp))!= NULL)
                c++;
            printf("no of files directory : %d ",c);
           }

        if(strcmp(tok2,"f")==0)
          {
            while((dir=readdir(dp))!= NULL)
                 printf("%s \n ",dir->d_name);
          }
      }
}

int main(int argc , char * argv[])
{
   char tok1[20],tok2[20],tok3[20];
   int choice,f;
   char cmd[40];
    while(1)
      {
        printf("\n Myshell$");
        gets(cmd);
        if(strcmp(cmd, "exit")==0)
            exit(0);
        int choice = sscanf(cmd,"%s%s%s",&tok1,&tok2,&tok3);
        if(choice==3)
           {
             if(strcmp(tok1,"list")==0)
                 list(tok2,tok3);
             continue;
           }
      }
}
```

Slip No-12

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **LRU** page replacement algorithm. Assume the memory of n frames.
Reference String : 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6

```
#include<stdio.h>

int pf=0,pg;
int f,frame[5],ref[50],victim=0,time[50];

int searchpage(int p)
{
    int i;
    for(i=0;i<f;i++)
    if(frame[i]==p)
    return i;
    return -1;
}

int selectvictim()
{
    int min=0,i;
    for(i=1;i<f;i++)
    {
        if(time[i]<time[min])
        min=i;
    }
    return min;
}

void main()
{
    int i;
    printf("\n Enter the no. of frames -: ");
    scanf("%d",&f);
    for(i=0;i<f;i++)
    frame[i]=-1;
    for(i=0;i<f;i++)
    time[i]=-1;

printf("\n Enter the no. of pages -: ");
scanf("%d",&pg);


printf("\n Enter the page(reference string) -: ");
for(i=0;i<pg;i++)
scanf("%d",&ref[i]);

for(i=0;i<pg;i++)
{
    int k,j;
    k=searchpage(ref[i]);
    if(k==-1)
    {
        pf++;
        k=selectvictim();
        frame[k]=ref[i];
    }
    time[k]=i;
        printf("\n pg=%d  |",ref[i]);
        for(j=0;j<f;j++)
        printf("\t%d",frame[j]);
    }
    printf("\n Page Fault is -:%d",pf);
}
/*
 Enter the no. of frames -: 3
 Enter the no. of pages -: 15
 Enter the page(reference string) -: 3 4 5 6 3 4 7 3 4 5 6 7 2 4 6

 pg=3  |        3       -1      -1
 pg=4  |        3       4       -1
 pg=5  |        3       4       5
 pg=6  |        6       4       5
 pg=3  |        6       3       5
 pg=4  |        6       3       4
 pg=7  |        7       3       4
 pg=3  |        7       3       4
```

```
pg=4  |        7       3       4
pg=5  |        5       3       4
pg=6  |        5       6       4
pg=7  |        5       6       7
pg=2  |        2       6       7
pg=4  |        2       4       7
pg=6  |        2       4       6
Page Fault is -:13
*/
```

Q.2 Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as
myshell$ list f dirname :- To print names of all the files in current directory.
myshell$ list n dirname :- To print the number of all entries in the current directory

```c
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>
#include<dirent.h>

 void list(char *tok2, char *tok3)
 {
   DIR *dp;
   int c =0;
   struct dirent *dir;
   dp = opendir(tok3);
    if(dp == 0)
       printf("%s directory not exit",tok3);
    else
      {
        if(strcmp(tok2,"n")==0)
          {
            while((dir=readdir(dp))!= NULL)
                c++;
            printf("no of files directory : %d ",c);
           }

        if(strcmp(tok2,"f")==0)
          {
            while((dir=readdir(dp))!= NULL)
                 printf("%s \n ",dir->d_name);
          }
      }
}

int main(int argc , char * argv[])
{
   char tok1[20],tok2[20],tok3[20];
   int choice,f;
   char cmd[40];
    while(1)
      {
        printf("\n Myshell$");
        gets(cmd);
        if(strcmp(cmd, "exit")==0)
            exit(0);
        int choice = sscanf(cmd,"%s%s%s",&tok1,&tok2,&tok3);
        if(choice==3)
           {
             if(strcmp(tok1,"list")==0)
                 list(tok2,tok3);
             continue;
           }
      }
}
```

Slip No-13

Q.1 Write a C program to implement the 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 typeline -a filename :- To print all lines in the file.

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

void typeline(char *tok2,char *tok3){

    char ch,str[30];
    int lc=0,s,count=0;
    FILE *fp;
    fp = fopen(tok3,"r");


    if(fp == NULL){
        printf("File does not exist.\n");
    }else{
        printf("File exist\n");
    }

    if(strcmp(tok2,"a") == 0){
        while(!feof(fp)){
            ch = fgetc(fp);
            printf("%c",ch);
        }
}

else{
    int n = atoi(tok2);
    if(n>0){
        while(lc<n){
            ch = fgetc(fp);
            if(ch == '\n')
            lc++;
            printf("%c",ch);
        }
    }

    if(n<0){
        while(!feof(fp)){
            ch = fgetc(fp);
            if(ch == '\n')
            lc++;
        }

        s = lc + n;
        s++;

        fseek(fp,0,SEEK_SET);
            while(!feof(fp)){
                ch = fgetc(fp);
                    while (count!=s)
                    {
                ch = fgetc(fp);
                if(ch == '\n')
                    count++;
                    }
                    printf("%c",ch);
            }
        }
    }
}

int main(){
    char tok1[10],tok2[10],tok3[10],tok4[10],cmd[30];
    int choice;

    while(1){
        printf("\n$MYShell$:");
        gets(cmd);

        if(strcmp(cmd,"exit") == 0){
            exit(0);
        }

        int choice = sscanf(cmd,"%s%s%s%s",&tok1,&tok2,&tok3,&tok4);
        if(strcmp(tok1,"typeline") == 0){
            typeline(tok2,tok3);
            continue;
        }
    }
}
```

Q.2 Write the simulation program for **Round Robin** scheduling for given time quantum. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. 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.

```c
#include<stdio.h>
#include<stdlib.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);
```

```c
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;
}
```

Slip No-14

Q.1 Write a C program to implement the 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 typeline +n filename :- To print first n lines in the file.

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

void typeline(char *tok2,char *tok3){

  char ch,str[30];
  int lc=0,s,count=0;
  FILE *fp;
  fp = fopen(tok3,"r");


  if(fp == NULL){
    printf("File does not exist.\n");
  }else{
    printf("File exist\n");
  }

  if(strcmp(tok2,"a") == 0){
    while(!feof(fp)){
      ch = fgetc(fp);
      printf("%c",ch);
    }
}

else{
  int n = atoi(tok2);
  if(n>0){
    while(lc<n){
      ch = fgetc(fp);
      if(ch == '\n')
      lc++;
      printf("%c",ch);
    }
  }

  if(n<0){
    while(!feof(fp)){
      ch = fgetc(fp);
      if(ch == '\n')
      lc++;
    }

    s = lc + n;
    s++;

    fseek(fp,0,SEEK_SET);
      while(!feof(fp)){
        ch = fgetc(fp);
                  while (count!=s)
                  {
        ch = fgetc(fp);
        if(ch == '\n')
                    count++;
                  }
                  printf("%c",ch);
      }
    }
  }
}

int main(){
  char tok1[10],tok2[10],tok3[10],tok4[10],cmd[30];
  int choice;

  while(1){
    printf("\n$MYShell$:");
    gets(cmd);

    if(strcmp(cmd,"exit") == 0){
      exit(0);
    }

    int choice = sscanf(cmd,"%s%s%s%s",&tok1,&tok2,&tok3,&tok4);
    if(strcmp(tok1,"type") == 0){
      typeline(tok2,tok3);
      continue;
    }
  }
}
```

Q.2 Write a C program to simulate **Non-preemptive Shortest Job First (SJF) – scheduling**. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time

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

struct job{
    int atime; //arraival time.
    int btime; //brust time.
    int ft; //finish time.
    int tat; //Trun around time.
    int wt; // waiting time.
}p[10];

int arr[10],brust[10],n,rq[10],no_rq=0,time=0;
int addrq();
int selectionjob();
```

```c
int deleteq(int j);
int fsahll();
void main(){

    int i,j;
    printf("Enter the Process -: ");
    scanf("%d",&n);
    printf("\n");

    for(i = 0;i<n;i++){
        printf("Enter the arrival time p%d -: ",i);
        scanf("%d",&p[i].atime); //Assigning the arrival time.
        arr[i] = p[i].atime;
    }
    printf("\n");

    for(i = 0;i<n;i++){
    printf("Enter the arrival time p%d -: ",i);
    printf("%d\n",p[i].atime); //Printing the arrival time.
    arr[i] = p[i].atime;
    }
    printf("\n");

    for(i = 0;i<n;i++){
    printf("Enter the brust time p%d -: ",i);
    scanf("%d",&p[i].btime); //Assigning the brust time.
    brust[i] = p[i].btime;
    }
    printf("\n");

    for(i = 0;i<n;i++){
    printf("Enter the brust time p%d -: ",i);
    printf("%d\n",p[i].btime); //Printing the brust time.
    brust[i] = p[i].btime;
    }
    printf("\n");

    addrq();  //Adding the process.
    // Process start now.
    printf("Gnatt Chart is -: ");
    while(1){
    j = selectionjob(); //sercah the process now.

        if(j == -1){
            printf("CPU is ideal");
            time++;
            addrq();
        }else{
            while(brust[j]!=0){
                printf("\t j %d",j);
                brust[j]--;
                time++;
                addrq();
            }
            p[j].ft = time;
        }
        if(fsahll() == 1)
        break;
    }
    int Tat = 0,Twt =0;

    printf("\n");
    printf("\nProcess\t FT \t TAT \t WT");
    for(i=0;i<n;i++){
        p[i].tat = p[i].ft-p[i].atime;
        p[i].wt = p[i].tat-p[i].btime;

        printf("\n p%d \t %d \t %d \t %d",i,p[i].ft,p[i].tat,p[i].wt);
        Tat += p[i].tat;
        Twt += p[i].wt;
    }

    float avgtat = Tat /n;
    float avgwt = Twt /n;

    printf("\nAverage of wait time is -: %f",avgwt);
    printf("\nAverage of trunaround time is -: %f",avgtat);
}

int addrq(){
    int i;
    for(i=0;i<n;i++){
        if(arr[i] == time){
            rq[no_rq] = i;
            no_rq++;
        }
    }
}

int selectionjob(){
    int i,j;
    if(no_rq == 0)
        return 1;
        j = rq[0];

        for(i=1;i<no_rq;i++)
        if(brust[j]>brust[rq[i]])
        j = rq[i];

        deleteq(j);
        return j;
}

int deleteq(int j){
    int i;
    for(i=0;i<no_rq;i++)

        if(rq[i] == j)
            break;
```

```
        for(i= i+1;i<no_rq;i++)
        rq[i-1] = rq[i];
        no_rq--;
}

int fsahll(){
    int i;
    for(i=0;i<n;i++)
    if(brust[i]!=0)
    return -1;
    return 1;
}
/*
Enter the Process -: 3

Enter the arrival time p0 -: 1
Enter the arrival time p1 -: 2
Enter the arrival time p2 -: 0

Enter the arrival time p0 -: 1
Enter the arrival time p1 -: 2
Enter the arrival time p2 -: 0

Enter the brust time p0 -: 3
Enter the brust time p1 -: 2
Enter the brust time p2 -: 5

Enter the brust time p0 -: 3
Enter the brust time p1 -: 2
Enter the brust time p2 -: 5

Gnatt Chart is -:        j 2     j 2     j 2     j 2     j 2     j 1     j 1     j 0     j 0     j 0

Process  FT      TAT     WT
 p0      10      9       6
 p1      7       5       3
 p2      5       5       0
Average of wait time is -: 3.000000
Average of trunaround time is -: 6.000000
*/
```

Slip No-15

Q.1 Write a C program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as myshell$ list f dirname :- To print names of all the files in current directory.

```c
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
#include<dirent.h>

//struct dirent{
//  char d_name[20];
//  int ino;
//};

void list(char *tok2,char *tok3){
  int c = 0;
  struct dirent *dir;

//  dir -> d_name, dir->ino
  DIR *dr;

  dr = opendir(tok3);
  if(dr == NULL){
    printf("Dir does not exist.");
  }

  if(strcmp(tok2,"f") == 0){
    while(dir = readdir(dr))
    printf("\n%s",dir->d_name);
  }else if(strcmp(tok2,"n") == 0){
    while(dir = readdir(dr))
    c++;
    printf("\nttoal number of files :%d",c);
  }else if(strcmp(tok2,"i") == 0){
    while(dir = readdir(dr))
    printf("\n%s\t%d",dir->d_name,dir->d_ino);
  }

}

int main(){
  char tok1[10],tok2[10],tok3[10],tok4[10],cmd[30];
  int choice;

  while(1){
    printf("\n$shell$:");
    gets(cmd);

    if(strcmp(cmd,"exit") == 0){
      exit(0);
    }

    int choice = sscanf(cmd,"%s%s%s%s",&tok1,&tok2,&tok3,&tok4);
    if(choice == 3){
      if(strcmp(tok1,"list") == 0){
        list(tok2,tok3);
        continue;
      }
    }
  }

}
```

Q.2 Write the program to simulate **preemptive Shortest Job First (SJF) – scheduling**. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time

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

struct job
{
    bool isc;
    int at;    // Arraival time.
    int bt;    // Brust time.
    int st;    // Start time
    int wt;    // Wait time.
    int nst;   // New start time.
    int oft;   // Old finish time.
    int pcount;// Process to be count.
    int ft;    // Finish time.
    int tat;   // Trun around time.
}p[100];

int n,arr[100],tm=0,arrv=0,count=0;
int selecta();
int process(int s);
float Tat,Wt;
```

```c
void main(){
    int i,k=0,a[100];

    printf("How many Process -: ");
    scanf("%d",&n);

    printf("Enter -: \nProcess BT AT\n");
    for(i=1;i<=n;i++){
        printf("p%d\t",i);
        scanf("%d%d",&p[i].bt,&p[i].at);
        p[i].isc=false;
        p[i].pcount=0;
        count += p[i].bt;
        p[i].wt=0;
    }

    printf("Process BT AT \n");
    for(i=1;i<=n;i++)
        printf("p%d %d %d\n",i,p[i].bt,p[i].at);


    printf("\nGantt chart -: \n");
    for(i=1;i<=n;i++)
        {
        if(p[i].at==0)
            {
            a[++k]=i;
            }
    }
int minbrust(int a[],int k);
while(tm!=count)
{
    selecta();

    if(arrv==0)
        {
        printf("|id1");
        tm+=1;
        count+=1;
        }
    else
        {
        minbrust(arr,arrv);
        arrv=0;
        }

}

printf("\n");
printf("\nProcess\tBT\tAT \tST\tWT\tFT\tTAT\n");
for(i=1;i<=n;i++)
    printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n",i,p[i].bt,p[i].at,p[i].st,p[i].wt,p[i].ft,p[i].tat);

    printf("\nAvg wait time -: %f",Wt/n);
    printf("\nAvg TAT -: %f\n",Tat/n);
}


int  minbrust(int a[],int k){

    int min=p[a[1]].bt,i,m=a[1];
    for(i=1;i<=k;i++)
    {
        if(p[a[i]].bt<min)
        {
            min=p[a[i]].bt;
            m=a[i];
        }
    }
    process(m);
}


int process(int s)
{
    int k;
    p[s].pcount++;
    if(p[s].pcount==1)
    {
        p[s].wt=p[s].st-p[s].at;
        p[s].st=tm;
    }
    p[s].nst=tm;
    tm++;

    k=p[s].nst-p[s].oft;
    p[s].oft=tm;

    if(k>0)
    p[s].wt+=k;
    if(p[s].pcount==p[s].bt)
    {
        p[s].isc=true;
        p[s].ft=tm;
        p[s].tat=p[s].ft-p[s].at;
        Tat+=p[s].tat;
        Wt+=p[s].wt;
    }

    printf("p%d   ",s);
}


int selecta(){
    int i;
    for(i=1;i<=n;i++)
    {
        if(p[i].at<=tm && p[i].isc==false)
```

```
        arr[++arrv]=i;
    }
}
/*
How many Process -: 3
Enter -:
Process BT AT
p1      2 1
p2      3 2
p3      5 0
Process BT AT
p1 2 1
p2 3 2
p3 5 0

Gantt chart -:
p3   p1   p1   p2   p2   p2   p3   p3   p3   p3

Process BT      AT      ST      WT      FT      TAT
P1      2       1       1       0       3       2
P2      3       2       3       1       6       4
P3      5       0       0       5       10      10

Avg wait time -: 2.000000
Avg TAT -: 5.333333
*/
```

Slip No-16

Q.1 Write a programto implement the toy shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.

count c filename :- To print number of characters in the file.

count w filename :- To print number of words in the file.

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

void count(char *tok2 ,char *tok3)
  {
    int l=0,w=0,c=0;
    FILE *fp;
    fp = fopen(tok3,"r");
     if(fp==NULL)
        printf("\n file not exist");
     else
       {
         while(!feof(fp))
          {
             char ch;
             ch = fgetc(fp);
             if(ch == ' ' )
                 w++;
             else if(ch == '\n')
                    {
                      w++;
                      l++;
                    }
                   else
                     c++;
          }
        if(strcmp(tok2,"w") == 0)
             printf("\n word count :%d",w);

        if(strcmp(tok2,"c") == 0)
             printf("\n character count :%d",c);

       }
   }

int main(int argc , char *argv[])
 {
    char command[30],tok1[30],tok2[30],tok3[30];
    while(1)
       {
          printf("\n MyShell:");
          gets(command);

          int ch = sscanf(command,"%s%s%s",tok1,tok2,tok3);
          if(ch == 3)
            {
              if(strcmp(tok1,"count")==0)
                 count(tok2,tok3);
                 continue;
            }
       }
    }
```

Q.2 Write the program to simulate **Non preemptive priority scheduling**. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process.
Also find the average waiting time and turnaround time.

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

struct job{
    int atime; // arraival time.
    int btime; //brust time.
    int ft; //finish time.
    int tat; //Trun around time.
    int wt; // waiting time.
    int pri; //Priority varilable is add.
}p[10];

int arr[10],brust[10],n,rq[10],no_rq=0,time=0,j=-1;

void main(){

    int i,j;
    printf("Enter the job number:");
    scanf("%d",&n);
    printf("\n");

    for(i = 0;i<n;i++){
        printf("Enter the arrival time p%d:",i);
        scanf("%d",&p[i].atime); //Assigning the arrival time.
        arr[i] = p[i].atime;
    }
    printf("\n");

    for(i = 0;i<n;i++){
    printf("Enter the arrival time p%d:",i);
    printf("%d",p[i].atime); //Assigning the arrival time.
    arr[i] = p[i].atime;
    }
    printf("\n");

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

```
printf("Enter the brust time p%d:",i);
scanf("%d",&p[i].btime); //Assigning the brust time.
brust[i] = p[i].btime;
}
printf("\n");

for(i = 0;i<n;i++){
printf("Enter the brust time p%d:",i);
printf("%d",p[i].btime); //Assigning the brust time.
brust[i] = p[i].btime;
}
printf("\n");

for(i = 0;i<n;i++){
printf("Enter the Priority p%d:",i);
scanf("%d",&p[i].pri); //Assigning the Priority.
}
printf("\n");

for(i = 0;i<n;i++){
printf("Enter the Priority p%d:",i);
printf("%d",p[i].pri); //Assigning the Priority.
}
printf("\n");
```

```
addrq();  //Adding the process.
// Process start now.
while(1){
j = selectionjob(); //sercah the process now.

    if(j == -1){
        printf("CPU is ideal");
        time++;
        addrq();
    }else{
        while(brust[j]!=0){
            printf("\t j %d",j);
            brust[j]--;
            time++;
            addrq();
        }
        p[j].ft = time;
    }
    if(fsahll() == 1)
    break;
}
int Tat = 0,Twt =0;

printf("\nJob \t FT \t TAT \t WT");
for(i=0;i<n;i++){
    p[i].tat = p[i].ft-p[i].atime;
    p[i].wt = p[i].tat-p[i].btime;
```

```
        printf("\n JOb %d \t %d \t %d \t %d",i,p[i].ft,p[i].tat,p[i].wt);
        Tat += p[i].tat;
        Twt += p[i].wt;
    }

    float avgtat = Tat /n;
    float avgwt = Twt /n;

    printf("\nAverage of trun around time is:%f",avgtat);
    printf("\nAverage of wait time is:%f",avgwt);
}
int addrq(){
    int i;
    for(i=0;i<n;i++){
        if(arr[i] == time){
            if(j!=-1 && p[i].pri>p[j].pri){
            rq[no_rq] = i;
            j = i;
            }else{
                rq[no_rq++] = i;
            }
        }
    }
}
int selectionjob(){
    int i,k;
    if(no_rq == 0)
        return -1;
        k = rq[0];
```

```
    for(i=1;i<no_rq;i++)
    if(p[k].pri<p[rq[i]].pri){
        k = rq[i];
    }
    deleteq(k);
    return k;
}
int deleteq(int k){
    int i;
    for(i=0;i<no_rq;i++)

        if(rq[i] == k)
            break;

        for(i= i+1;i<no_rq;i++)
        rq[i-1] = rq[i];
        no_rq--;
}
int fsahll(){
    int i;
    for(i=0;i<n;i++)
    if(brust[i]!=0)
    return -1;
    return 1;
}
```

Slip No-17

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according **the Optimal page replacement** algorithm. Assume the memory of n frames.
Reference String : 7, 5, 4, 8, 5, 7, 2, 3, 1, 3, 5, 9, 4, 6,

```c
#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("\n pg=%d |",pages[i]);
            printf("%d\t",frames[j]);
        }
    }
    printf("\n Total Page Faults -: %d",faults);
    return 0;
}
/*
Enter number of frames -: 3
Enter number of pages -: 15
```

```
Enter page reference string -: 7 5 4 8 5 7 2 3 1 3 5 9 4 6 2


 pg=7 |7
 pg=7 |-1
 pg=7 |-1

 pg=5 |7
 pg=5 |5
 pg=5 |-1

 pg=4 |7
 pg=4 |5
 pg=4 |4

 pg=8 |7
 pg=8 |5
 pg=8 |8

 pg=5 |7
 pg=5 |5
 pg=5 |8

 pg=7 |7
 pg=7 |5
 pg=7 |8

pg=2 |2
pg=2 |5
pg=2 |8

pg=3 |2
pg=3 |5
pg=3 |3

pg=1 |1
pg=1 |5
pg=1 |3

pg=3 |1
pg=3 |5
pg=3 |3

pg=5 |1
pg=5 |5
pg=5 |3

pg=9 |9
pg=9 |5
pg=9 |3

pg=4 |4
pg=4 |5
pg=4 |3

 pg=6 |6
 pg=6 |5
 pg=6 |3

 pg=2 |2
 pg=2 |5
 pg=2 |3
 Total Page Faults -: 11
*/
```

Q.2 Write the program to simulate **FCFS CPU-scheduling**. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

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

struct job{
    int atime; // Arraival time.
    int btime; // Brust time.
    int ft; // Finish time.
    int tat; // Trun around time.
    int wt; // waiting time.
}p[10];

int arr[10],brust[10],n,rq[10],no_rq=0,time=0;
int addrq();
int selectionjob();
int deleteq(int j);
int fsahll();

void main()
{
    int i,j;
    printf("Enter the Process -: ");
    scanf("%d",&n);
    printf("\n");

    for(i = 0;i<n;i++){
        printf("Enter the Arrival time p%d -: ",i);
        scanf("%d",&p[i].atime); //Assigning the arrival time.
        arr[i] = p[i].atime;
    }
    printf("\n");

    for(i = 0;i<n;i++){
    printf("Enter the Arrival time p%d -: ",i);
    printf("%d\n",p[i].atime); //Printing the arrival time.
    arr[i] = p[i].atime;
    }
    printf("\n");
```

```c
for(i = 0;i<n;i++){
printf("Enter the Brust time p%d -: ",i);
scanf("%d",&p[i].btime); //Assigning the brust time.
brust[i] = p[i].btime;
}
printf("\n");

for(i = 0;i<n;i++){
printf("Enter the Brust time p%d -: ",i);
printf("%d\n",p[i].btime); //Printing the brust time.
brust[i] = p[i].btime;
}
printf("\n");

addrq();  //Adding the process.
// Process start now.
printf("Gnatt Chart is -: ");
while(1){
j = selectionjob(); //sercah the process now.

    if(j == -1){
        printf("CPU is ideal");
        time++;
        addrq();
    }else{
        while(brust[j]!=0){
            printf("\t j %d",j);
            brust[j]--;
            time++;
            addrq();
        }
        p[j].ft = time;
    }
    if(fsahll() == 1)
    break;
}
int Tat = 0,Twt =0;

printf("\n");
printf("\nProcess\t FT \t TAT \t WT");
for(i=0;i<n;i++){
    p[i].tat = p[i].ft-p[i].atime;
    p[i].wt = p[i].tat-p[i].btime;

    printf("\n P%d \t %d \t %d \t %d",i,p[i].ft,p[i].tat,p[i].wt);
    Tat += p[i].tat;
    Twt += p[i].wt;
}
float avgtat = Tat /n;
float avgwt = Twt /n;

printf("\nAverage of wait time is -: %f",avgwt);
printf("\nAverage of trun around time is -: %f",avgtat);
printf("\n");
}
int addrq()
{
    int i;
    for(i=0;i<n;i++){
        if(arr[i] == time){
            rq[no_rq] = i;
            no_rq++;
        }
    }
}
int selectionjob()
{
    int i,j;
    if(no_rq == 0)
        return 1;
        j = rq[0];
        deleteq(j);
        return j;
}
int deleteq(int j)
{
    int i;
    for(i=0;i<no_rq;i++)

        if(rq[i] == j)
            break;

        for(i= i+1;i<no_rq;i++)
        rq[i-1] = rq[i];
        no_rq--;
}
int fsahll()
{
    int i;
    for(i=0;i<n;i++)
    if(brust[i]!=0)
    return -1;
    return 1;
}

/*Enter the Process -: 3

Enter the Arrival time p0 -: 1
Enter the Arrival time p1 -: 2
Enter the Arrival time p2 -: 0

Enter the Arrival time p0 -: 1
Enter the Arrival time p1 -: 2
Enter the Arrival time p2 -: 0

Enter the Brust time p0 -: 2
Enter the Brust time p1 -: 3
Enter the Brust time p2 -: 5
```

```
Enter the Brust time p0 -: 2
Enter the Brust time p1 -: 3
Enter the Brust time p2 -: 5

Gnatt Chart is -:        j 2     j 2     j 2     j 2     j 2     j 0     j 0     j 1     j 1     j 1

Process  FT      TAT     WT
 P0      7       6       4
 P1      10      8       5
 P2      5       5       0
Average of wait time is -: 3.000000
Average of trun around time is -: 6.000000 */
```

Slip No-18

Q.1 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **LRU page replacement algorithm.** Assume the memory of n frames.
Reference String : 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6

Q.2 Write a C program to simulate **FCFS CPU-scheduling**. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

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

struct job{
    int atime; // Arraival time.
    int btime; // Brust time.
    int ft; // Finish time.
    int tat; // Trun around time.
    int wt; // waiting time.
}p[10];

int arr[10],brust[10],n,rq[10],no_rq=0,time=0;
int addrq();
int selectionjob();
int deleteq(int j);
int fsahll();

void main()
{
    int i,j;
    printf("Enter the Process -: ");
    scanf("%d",&n);
    printf("\n");

    for(i = 0;i<n;i++){
        printf("Enter the Arrival time p%d -: ",i);
        scanf("%d",&p[i].atime); //Assigning the arrival time.
        arr[i] = p[i].atime;
    }
    printf("\n");

    for(i = 0;i<n;i++){
    printf("Enter the Arrival time p%d -: ",i);
    printf("%d\n",p[i].atime); //Printing the arrival time.
    arr[i] = p[i].atime;
    }
    printf("\n");

    for(i = 0;i<n;i++){
    printf("Enter the Brust time p%d -: ",i);
    scanf("%d",&p[i].btime); //Assigning the brust time.
    brust[i] = p[i].btime;
    }
    printf("\n");

    for(i = 0;i<n;i++){
    printf("Enter the Brust time p%d -: ",i);
    printf("%d\n",p[i].btime); //Printing the brust time.
    brust[i] = p[i].btime;
    }
    printf("\n");

    addrq();  //Adding the process.
    // Process start now.
    printf("Gnatt Chart is -: ");
    while(1){
    j = selectionjob(); //sercah the process now.

        if(j == -1){
            printf("CPU is ideal");
            time++;
            addrq();
        }else{
            while(brust[j]!=0){
                printf("\t j %d",j);
                brust[j]--;
                time++;
                addrq();
            }
            p[j].ft = time;
        }
        if(fsahll() == 1)
        break;
    }
    int Tat = 0,Twt =0;
```

```
    printf("\n");
    printf("\nProcess\t FT \t TAT \t WT");
    for(i=0;i<n;i++){
        p[i].tat = p[i].ft-p[i].atime;
        p[i].wt = p[i].tat-p[i].btime;

        printf("\n P%d \t %d \t %d \t %d",i,p[i].ft,p[i].tat,p[i].wt);
        Tat += p[i].tat;
        Twt += p[i].wt;
    }
    float avgtat = Tat /n;
    float avgwt = Twt /n;

    printf("\nAverage of wait time is -: %f",avgwt);
    printf("\nAverage of trun around time is -: %f",avgtat);
    printf("\n");
}
int addrq()
{
    int i;
    for(i=0;i<n;i++){
        if(arr[i] == time){
            rq[no_rq] = i;
            no_rq++;
        }
    }
}
int selectionjob()
{
    int i,j;
    if(no_rq == 0)
        return 1;
        j = rq[0];
        deleteq(j);
        return j;
}
int deleteq(int j)
{
    int i;
    for(i=0;i<no_rq;i++)

        if(rq[i] == j)
            break;

        for(i= i+1;i<no_rq;i++)
        rq[i-1] = rq[i];
        no_rq--;
}
int fsahll()
{
    int i;
    for(i=0;i<n;i++)
    if(brust[i]!=0)
    return -1;
    return 1;
}

/*Enter the Process -: 3

Enter the Arrival time p0 -: 1
Enter the Arrival time p1 -: 2
Enter the Arrival time p2 -: 0

Enter the Arrival time p0 -: 1
Enter the Arrival time p1 -: 2
Enter the Arrival time p2 -: 0

Enter the Brust time p0 -: 2
Enter the Brust time p1 -: 3
Enter the Brust time p2 -: 5

Enter the Brust time p0 -: 2
Enter the Brust time p1 -: 3
Enter the Brust time p2 -: 5

Gnatt Chart is -:        j 2     j 2     j 2     j 2     j 2     j 0     j 0     j 1     j 1     j 1

Process  FT      TAT     WT
 P0      7       6       4
 P1      10      8       5
 P2      5       5       0
Average of wait time is -: 3.000000
Average of trun around time is -: 6.000000 */
```

Slip No-19

Q.1 Write a C program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as myshell$ list f dirname :- To print names of all the files in current directory.

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

 void list(char *tok2, char *tok3)
 {
   DIR *dp;
   int c =0;
   struct dirent *dir;
   dp = opendir(tok3);
    if(dp == 0)
       printf("%s directory not exit",tok3);
    else
      {
```

```
       if(strcmp(tok2,"f")==0)
         {
           while((dir=readdir(dp))!= NULL)
                printf("%s \n ",dir->d_name);
         }
     }
}

int main(int argc , char * argv[])
{
   char tok1[20],tok2[20],tok3[20];
   int choice,f;
   char cmd[40];
    while(1)
      {
        printf("\n Myshell$");
        gets(cmd);
        if(strcmp(cmd, "exit")==0)
            exit(0);
        int choice = sscanf(cmd,"%s%s%s",&tok1,&tok2,&tok3);
        if(choice==3)
           {
             if(strcmp(tok1,"list")==0)
                 list(tok2,tok3);
             continue;
           }
      }
 }
```

Q.2 Write the simulation program for **Round Robin scheduling** for given time quantum. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. 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.

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

Slip No-20

Q.1 Write a C program to implement the 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 typeline -a filename :- To print all lines in the file.

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

void typeline(char *tok2,char *tok3){

    char ch,str[30];
    int lc=0,s,count=0;
    FILE *fp;
    fp = fopen(tok3,"r");


    if(fp == NULL){
        printf("File does not exist.\n");
    }else{
        printf("File exist\n");
    }

    if(strcmp(tok2,"a") == 0){
        while(!feof(fp)){
            ch = fgetc(fp);
            printf("%c",ch);
        }
}

else{
    int n = atoi(tok2);
    if(n>0){
        while(lc<n){
            ch = fgetc(fp);
            if(ch == '\n')
            lc++;
            printf("%c",ch);
        }
    }

    if(n<0){
        while(!feof(fp)){
            ch = fgetc(fp);
            if(ch == '\n')
            lc++;
        }

        s = lc + n;
        s++;

        fseek(fp,0,SEEK_SET);
            while(!feof(fp)){
                ch = fgetc(fp);
                    while (count!=s)
                    {
                ch = fgetc(fp);
                if(ch == '\n')
                    count++;
                    }
                    printf("%c",ch);
            }
        }
    }
}

int main(){
    char tok1[10],tok2[10],tok3[10],tok4[10],cmd[30];
    int choice;

    while(1){
        printf("\n$MYShell$:");
        gets(cmd);

        if(strcmp(cmd,"exit") == 0){
            exit(0);
        }

        int choice = sscanf(cmd,"%s%s%s%s",&tok1,&tok2,&tok3,&tok4);
        if(strcmp(tok1,"typeline") == 0){
            typeline(tok2,tok3);
            continue;
        }
    }
}
```

Q.2 Write the program to simulate **Non-preemptive Shortest Job First (SJF) – scheduling**. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time

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

struct job{
    int atime; //arraival time.
    int btime; //brust time.
    int ft; //finish time.
    int tat; //Trun around time.
    int wt; // waiting time.
}p[10];

int arr[10],brust[10],n,rq[10],no_rq=0,time=0;
int addrq();
int selectionjob();
```

```c
int deleteq(int j);
int fsahll();
void main(){

    int i,j;
    printf("Enter the Process -: ");
    scanf("%d",&n);
    printf("\n");

    for(i = 0;i<n;i++){
        printf("Enter the arrival time p%d -: ",i);
        scanf("%d",&p[i].atime); //Assigning the arrival time.
        arr[i] = p[i].atime;
    }
    printf("\n");

    for(i = 0;i<n;i++){
    printf("Enter the arrival time p%d -: ",i);
    printf("%d\n",p[i].atime); //Printing the arrival time.
    arr[i] = p[i].atime;
    }
    printf("\n");

    for(i = 0;i<n;i++){
    printf("Enter the brust time p%d -: ",i);
    scanf("%d",&p[i].btime); //Assigning the brust time.
    brust[i] = p[i].btime;
    }
    printf("\n");

    for(i = 0;i<n;i++){
    printf("Enter the brust time p%d -: ",i);
    printf("%d\n",p[i].btime); //Printing the brust time.
    brust[i] = p[i].btime;
    }
    printf("\n");

    addrq();  //Adding the process.
    // Process start now.
    printf("Gnatt Chart is -: ");
    while(1){
    j = selectionjob(); //sercah the process now.

        if(j == -1){
            printf("CPU is ideal");
            time++;
            addrq();
        }else{
            while(brust[j]!=0){
                printf("\t j %d",j);
                brust[j]--;
                time++;
                addrq();
            }
            p[j].ft = time;
        }
        if(fsahll() == 1)
        break;
    }
    int Tat = 0,Twt =0;

    printf("\n");
    printf("\nProcess\t FT \t TAT \t WT");
    for(i=0;i<n;i++){
        p[i].tat = p[i].ft-p[i].atime;
        p[i].wt = p[i].tat-p[i].btime;

        printf("\n p%d \t %d \t %d \t %d",i,p[i].ft,p[i].tat,p[i].wt);
        Tat += p[i].tat;
        Twt += p[i].wt;
    }

    float avgtat = Tat /n;
    float avgwt = Twt /n;

    printf("\nAverage of wait time is -: %f",avgwt);
    printf("\nAverage of trunaround time is -: %f",avgtat);
}

int addrq(){
    int i;
    for(i=0;i<n;i++){
        if(arr[i] == time){
            rq[no_rq] = i;
            no_rq++;
        }
    }
}

int selectionjob(){
    int i,j;
    if(no_rq == 0)
        return 1;
        j = rq[0];

        for(i=1;i<no_rq;i++)
        if(brust[j]>brust[rq[i]])
        j = rq[i];

        deleteq(j);
        return j;
}

int deleteq(int j){
    int i;
    for(i=0;i<no_rq;i++)

        if(rq[i] == j)
            break;
```

```
        for(i= i+1;i<no_rq;i++)
        rq[i-1] = rq[i];
        no_rq--;
}

int fsahll(){
    int i;
    for(i=0;i<n;i++)
    if(brust[i]!=0)
    return -1;
    return 1;
}
/*
Enter the Process -: 3

Enter the arrival time p0 -: 1
Enter the arrival time p1 -: 2
Enter the arrival time p2 -: 0

Enter the arrival time p0 -: 1
Enter the arrival time p1 -: 2
Enter the arrival time p2 -: 0

Enter the brust time p0 -: 3
Enter the brust time p1 -: 2
Enter the brust time p2 -: 5

Enter the brust time p0 -: 3
Enter the brust time p1 -: 2
Enter the brust time p2 -: 5

Gnatt Chart is -:        j 2     j 2     j 2     j 2     j 2     j 1     j 1     j 0     j 0     j 0

Process  FT      TAT     WT
 p0      10      9       6
 p1      7       5       3
 p2      5       5       0
Average of wait time is -: 3.000000
Average of trunaround time is -: 6.000000
*/
```

Slip No-21

Q.1 Write a 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”.

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

int main()
{
    int p;
    p = fork();
    if(p<0)
        printf("\n program cannot executed..");
    else if(p==0)
           {
            printf("\n I'm child Process..");
            printf("\n the child process id is:%d",getpid());
           }
          else
          {
             printf("\n I'm the Parent Process..");
             printf("\n the parent process id is:%d",getppid());
          }
}

/* OUTPUT:
I'm the Parent Process..
 the parent process id is:31547
 I'm child Process..
 the child process id is:31555
*/
```

Q.2 Write a C program to simulate **Preemptive Priority scheduling**. The arrival time and first CPU-burst and priority for different n number of processes should be input to the algorithm. Assume 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.

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

struct job{
    bool isc;
    int at; //arraival time.
    int bt; //brust time.
    int st; //Start time.
    int nst; //new start time.
    int oft; //old finish time;
    int pcount; // preocess to be count.
    int ft; //finish time.
    int tat; //Trun around time.
    int wt; // waiting time.
    int pri; //Priority varilable is add.
}p[100];

int n,arr[100],tm=0,arrv=0,count=0,j=-1;
int process(int s);
int selecta();
float Tat,Wt;


void minbrust(int a[],int k){

    int min=p[a[1]].bt,i,m=a[1];
    for(i=1;i<=k;i++)
    {
        if(p[a[i]].bt<min && j!=-1 && p[i].pri>p[j].pri)
        {
            min=p[a[i]].bt;
            m=a[i];
        }
    }
    process(m);
}


int process(int s){
    int k;
    p[s].pcount++;
    if(p[s].pcount==1)
    {
        p[s].wt=p[s].st-p[s].at;
        p[s].st=tm;
    }
    p[s].nst=tm;
    tm++;

    k=p[s].nst-p[s].oft;
    p[s].oft=tm;

    if(k>0)
    p[s].wt+=k;
    if(p[s].pcount==p[s].bt)
    {
        p[s].isc=true;
        p[s].ft=tm;
        p[s].tat=p[s].ft-p[s].at;
        Tat+=p[s].tat;
        Wt+=p[s].wt;
    }

    printf("p%d   ",s);
}


int selecta(){
```

```
    int i;
    for(i=1;i<=n;i++)
    {
        if(p[i].at<=tm && p[i].isc==false)
        arr[++arrv]=i;
    }
}

void main(){
    int i,k=0,a[100];

    printf("How many Process -: ");
    scanf("%d",&n);

    printf("\nProcess BT AT Priority\n");
    for(i=1;i<=n;i++){
        printf("p%d\t",i);
        scanf("%d%d%d",&p[i].bt,&p[i].at,&p[i].pri);
        p[i].isc=false;
        p[i].pcount=0;
        count += p[i].bt;
        p[i].wt=0;
    }

    printf("Process BT AT Priority\n");
    for(i=1;i<=n;i++)
        printf("p%d \t%d \t%d \t%d\n",i,p[i].bt,p[i].at,p[i].pri);


printf("\nGantt chart -: \n");
for(i=1;i<=n;i++)
    {
    if(p[i].at==0)
        {
        a[++k]=i;
        }
    }
minbrust(a,k);
while(tm!=count)
{
    selecta();

    if(arrv==0)
        {
        printf("|idl");
        tm+=1;
        count+=1;
        }
    else
        {
        minbrust(arr,arrv);
        arrv=0;
        }

    printf("\n");
    printf("\nProcess\tBT\tAT\tPri\tST\tWT\tFT\tTAT\n");
    for(i=1;i<=n;i++)
        printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",i,p[i].bt,p[i].at,p[i].pri,p[i].st,p[i].wt,p[i].ft,p[i].tat);

    printf("\nAvg wait time -: %f",Wt/n);
    printf("\nAvg TAT -: %f\n",Tat/n);
}

/*
How many Process -: 3

Process BT AT Priority
p1      3 1 3
p2      2 2 2
p3      5 0 1
Process BT AT Priority
p1      3       1       3
p2      2       2       2
p3      5       0       1

Gantt chart -:
p3   p1   p1   p1   p2   p2   p3   p3   p3   p3

Process BT      AT      Pri     ST      WT      FT      TAT
P1      3       1       3       1       0       4       3
P2      2       2       2       4       2       6       4
P3      5       0       1       0       5       10      10

Avg wait time -: 2.333333
Avg TAT -: 5.666667
*/
```

Slip No-22

Q.1 Write a C 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.

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

int main() {
    int pid, retnice;
    printf("press DEL to stop process \n");

    pid = fork();
    for(;;)
    {
        if(pid == (0))
        {
        retnice = nice (-5);
        printf("child gets higher CPU priority %d\n", retnice);
        sleep(1);
        }
        else{
        retnice = nice (4);
        printf("Parent gets lower CPU priority %d\n", retnice);
        sleep(1);
        }
    }
}

/* OUTPUT:

press DEL to stop process
Parent gets lower CPU priority 4
child gets higher CPU priority -1
Parent gets lower CPU priority 8
child gets higher CPU priority -1
child gets higher CPU priority -1
Parent gets lower CPU priority 12
child gets higher CPU priority -1
Parent gets lower CPU priority 16
child gets higher CPU priority -1
Parent gets lower CPU priority 19
child gets higher CPU priority -1
Parent gets lower CPU priority 19
child gets higher CPU priority -1
Parent gets lower CPU priority 19
child gets higher CPU priority -1
Parent gets lower CPU priority 19
child gets higher CPU priority -1
Parent gets lower CPU priority 19
*/
```

Q.2 Write a C program to simulate **Non preemptive priority scheduling**. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

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

struct job{
    int atime; // arraival time.
    int btime; //brust time.
    int ft; //finish time.
    int tat; //Trun around time.
    int wt; // waiting time.
    int pri; //Priority varilable is add.
}p[10];

int arr[10],brust[10],n,rq[10],no_rq=0,time=0,j=-1;

void main(){

    int i,j;
    printf("Enter the job number:");
    scanf("%d",&n);
    printf("\n");

    for(i = 0;i<n;i++){
        printf("Enter the arrival time p%d:",i);
    scanf("%d",&p[i].atime); //Assigning the arrival time.
    arr[i] = p[i].atime;
}
printf("\n");

for(i = 0;i<n;i++){
printf("Enter the arrival time p%d:",i);
printf("%d",p[i].atime); //Assigning the arrival time.
arr[i] = p[i].atime;
}
printf("\n");

for(i = 0;i<n;i++){
printf("Enter the brust time p%d:",i);
scanf("%d",&p[i].btime); //Assigning the brust time.
brust[i] = p[i].btime;
}
printf("\n");

for(i = 0;i<n;i++){
printf("Enter the brust time p%d:",i);
printf("%d",p[i].btime); //Assigning the brust time.
brust[i] = p[i].btime;
}
printf("\n");
```

```c
for(i = 0;i<n;i++){
printf("Enter the Priority p%d:",i);
scanf("%d",&p[i].pri); //Assigning the Priority.
}
printf("\n");

for(i = 0;i<n;i++){
printf("Enter the Priority p%d:",i);
printf("%d",p[i].pri); //Assigning the Priority.
}
printf("\n");


addrq();  //Adding the process.
// Process start now.
while(1){
j = selectionjob(); //sercah the process now.

    if(j == -1){
        printf("CPU is ideal");
        time++;
        addrq();
    }else{
        while(brust[j]!=0){
            printf("\t j %d",j);
            brust[j]--;
            time++;
            addrq();
        }
        p[j].ft = time;
    }
    if(fsahll() == 1)
    break;
}
int Tat = 0,Twt =0;

printf("\nJob \t FT \t TAT \t WT");
for(i=0;i<n;i++){
    p[i].tat = p[i].ft-p[i].atime;
    p[i].wt = p[i].tat-p[i].btime;

    printf("\n JOb %d \t %d \t %d \t %d",i,p[i].ft,p[i].tat,p[i].wt);
    Tat += p[i].tat;
    Twt += p[i].wt;
}

float avgtat = Tat /n;
float avgwt = Twt /n;

printf("\nAverage of trun around time is:%f",avgtat);
printf("\nAverage of wait time is:%f",avgwt);
}

int addrq(){
int i;
for(i=0;i<n;i++){
    if(arr[i] == time){
        if(j!=-1 && p[i].pri>p[j].pri){
        rq[no_rq] = i;
        j = i;
        }else{
            rq[no_rq++] = i;
        }
    }
}
}

int selectionjob(){
int i,k;
if(no_rq == 0)
    return -1;
    k = rq[0];

for(i=1;i<no_rq;i++)
if(p[k].pri<p[rq[i]].pri){
    k = rq[i];
}
deleteq(k);
return k;
}

int deleteq(int k){
int i;
for(i=0;i<no_rq;i++)

    if(rq[i] == k)
        break;

    for(i= i+1;i<no_rq;i++)
    rq[i-1] = rq[i];
    no_rq--;
}

int fsahll(){
int i;
for(i=0;i<n;i++)
if(brust[i]!=0)
return -1;
return 1;
}
```

Slip No-23

Q.1 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()).

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

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

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

        sleep(10);

        // At this time parent process has finished.
        // So if u will check parent process id
        // it will show different process id
        printf("\nChild process \n");
        printf("ID: %d\n", getpid());
        printf("Parent -ID: %d\n", getppid());
    }
    else {
        printf("Failed to create child process");
    }

    return 0;
}
```

Q.2 Write the simulation program for demand paging and show the page scheduling and total number of page faults according the **Optimal page replacement algorithm**. Assume the memory of n frames.
Reference String : 7, 5, 4, 8, 5, 7, 2, 3, 1, 3, 5, 9, 4, 6,

```c
#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("\n pg=%d |",pages[i]);
            printf("%d\t",frames[j]);
        }
    }
    printf("\n Total Page Faults -: %d",faults);
    return 0;
}
/*
Enter number of frames -: 3
Enter number of pages -: 15
Enter page reference string -: 7 5 4 8 5 7 2 3 1 3 5 9 4 6 2


 pg=7 |7
 pg=7 |-1
 pg=7 |-1

 pg=5 |7
 pg=5 |5
 pg=5 |-1

 pg=4 |7
 pg=4 |5
 pg=4 |4

 pg=8 |7
 pg=8 |5
 pg=8 |8

 pg=5 |7
 pg=5 |5
 pg=5 |8

 pg=7 |7
 pg=7 |5
 pg=7 |8

 pg=2 |2
 pg=2 |5
 pg=2 |8

 pg=3 |2
 pg=3 |5
 pg=3 |3

 pg=1 |1
 pg=1 |5
 pg=1 |3

 pg=3 |1
 pg=3 |5
 pg=3 |3

 pg=5 |1
 pg=5 |5
 pg=5 |3

 pg=9 |9
 pg=9 |5
 pg=9 |3

 pg=4 |4
 pg=4 |5
 pg=4 |3

 pg=6 |6
 pg=6 |5
 pg=6 |3

 pg=2 |2
 pg=2 |5
 pg=2 |3
 Total Page Faults -: 11
*/
```

Slip No-24

Q.1 Write a 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.

```c
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
int main() {
    int pid, retnice;
    printf("press DEL to stop process \n");
    pid = fork();
    for(;;)
    {
        if(pid == (0))
        {
        retnice = nice (-5);
        printf("child gets higher CPU priority %d\n", retnice);
        sleep(1);
        }
        else{
        retnice = nice (4);
        printf("Parent gets lower CPU priority %d\n", retnice);
        sleep(1);
        }
    }
}
/* OUTPUT:
press DEL to stop process
Parent gets lower CPU priority 4
child gets higher CPU priority -1
Parent gets lower CPU priority 8
child gets higher CPU priority -1
child gets higher CPU priority -1
Parent gets lower CPU priority 12
child gets higher CPU priority -1
Parent gets lower CPU priority 16
child gets higher CPU priority -1
Parent gets lower CPU priority 19
child gets higher CPU priority -1
Parent gets lower CPU priority 19
child gets higher CPU priority -1
Parent gets lower CPU priority 19
child gets higher CPU priority -1
Parent gets lower CPU priority 19
child gets higher CPU priority -1
Parent gets lower CPU priority 19*/
```

Q.2 Write a C program to implement the toy shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands. count c filename :- To print number of characters in the file. count w filename :- To print number of words in the file. count l filename :- To print number of lines in the file.

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <string.h>
void count(char *tok2 ,char *tok3)
  {
    int l=0,w=0,c=0;
    FILE *fp;
    fp = fopen(tok3,"r");
     if(fp==NULL)
        printf("\n file not exist");
     else
       {
         while(!feof(fp))
          {
             char ch;
             ch = fgetc(fp);
             if(ch == ' ' )
                 w++;
             else if(ch == '\n')
                    {
                      w++;
                      l++;
                    }
                   else
                     c++;
          }
         if(strcmp(tok2,"w") == 0)
              printf("\n word count :%d",w);

        if(strcmp(tok2,"c") == 0)
             printf("\n character count :%d",c);

        if(strcmp(tok2,"l") == 0)
             printf("\n line count :%d",l);
       }
   }
int main(int argc , char *argv[])
 {
    char command[30],tok1[30],tok2[30],tok3[30];
    while(1)
       {
          printf("\n MyShell:");
          gets(command);

          int ch = sscanf(command,"%s%s%s",tok1,tok2,tok3);
          if(ch == 3)
            {
              if(strcmp(tok1,"count")==0)
                 count(tok2,tok3);
                 continue;
            }
       }
   }
```

Slip No-25

Q.1 Write a C program that accepts an integer array. Main function forks child process. Parent process sorts an integer array and passes the sorted array to child process through the command line arguments of execve() system call. The child process uses execve() system call to load new program that uses this sorted array for performing the binary search to search the particular item in the array.

Q.2 Write a programto implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.
myshell$ search f filename pattern :- To display first occurrence of pattern in the file.

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

void search(char *tok2,char *tok3,char *tok4)//tok2:command tok3:pattern tok4:Textfilename
{
    FILE *fp;
    int lineno=0,count=0;
    char str[50];
    fp=fopen(tok4,"r");
    if(fp==NULL)
    {
        printf("File does not exist");
                    exit(0);
    }
    else
    {

        if(strcmp(tok2,"f")==0)//First occurance of pattern
        {
            while(!feof(fp))
            {
                fgets(str,50,fp);
                lineno++;
                if(strstr(str,tok3))
                {
                printf("First occurance of '%s' is on line : %d\n",tok3,lineno);
                break;
                }
            }
        }

    }
}

void main(int argc,char *argv[])
{
    char cmd[20],tok1[20],tok2[20],tok3[20],tok4[20];
    while(1)
    {
        printf("\nMyShell$:");
        gets(cmd);

            if(strcmp(tok1,"exit")==0)
            exit(0);

        int ch=sscanf(cmd,"%s%s%s%s",tok1,tok2,tok3,tok4);
        if(ch==4)
        {

            if(strcmp(tok1,"search")==0)
            {
            search(tok2,tok3,tok4);
            continue;
            }
        }
   }
}
```