Ex.No : 1

BASICS of UNIX COMMANDS

Date: 29.07.2024

Aim:

To study basic UNIX commands.

1) Command : cat

a) **To Create a New File:** *Syntax: cat > filename*

```
[srm@srmlinux]$ cat >
one.txtHi
Welcome to CSE Lab
SRM Institute of Science and Technology
Tiruchirappalli
```

Pressing Ctrl+z to save the file

```
[2]+  Stopped              cat
>one.txt[srm@srmlinux]$
```

b) To Display the Contents of the File

:*Syntax : cat filename*

```
[srm@srmlinux]$ cat one.txt
Hi
Welcome to CSE Lab
SRM Institute of Science and Technology
Tiruchirappalli
[srm@srmlinux]$
```

---

2) Command : mkdir (Make directory)*Syntax : mkdir filename*

```
[srm@srmlinux]$ mkdir
programs[srm@srmlinux]$ ls
a.out          college.txt  fir.sh           new.txt
second.shcol1.txt          coll.txt           firs.sh
               one          pgmscollege2.txt  file.txt
               one.txt      programs [srm@srmlinux]$
```

---

3) Command : cd (change directory) *Syntax : cd directory name*

```
[srm@srmlinux]$cd programs [srm@srmlinux programs]$ cd  ..
[srm@srmlinux]$ cd programs
[srm@srmlinux programs]$ mkdir
shellprogram[srm@srmlinux programs]$ cd
shellprogram [srm@srmlinux shellprogram]$
cd [srm@srmlinux]$
```

4) Command : pwd (Print Working Directory)

```
[srm@srmlinux shellprogram]$ pwd
/home/srm/programs/shellprogra
m                    [srm@srmlinux
shellprogram]$                    cd
[srm@srmlinux]$ pwd
/home/srm
[srm@srmlinux]
$
```

---

5) Command : ls (List Files and Directories)

a) ls

```
[srm@srmlinux]$ ls
111    2.prn    cpgm.c      HelloWorld.class  mac.c     print.c
1.ccc   a.out    duplicate.c  HelloWorld.java  Makefile
sample.c1.docx  array.c  fib.c            inc.c     OS
        streverse.c
1.prn   CP       fun.c       JAVA            pr.c
                 vin[srm@srmlinux]$
```

b) ls –l : Lists Files and Directories in Long Listing Mode

```
[srm@srmlinux]$ ls
-ltotal 112
-rw-rw-r-- 1 srm srm  35 Jul 21 16:31 111 -rw-r--r--
1 srm srm    0 Sep 3 13:12 1.ccc
-rw-rw-r-- 1 srm srm 6857 Aug  9 16:30 1.docx
-rw-rw-r-- 1 srm srm  785 Jul 21 16:32 1.prn
-rw-rw-r-- 1 srm srm  166 Jul 21 16:33 2.prn
-rwxrwxr-x 1 srm srm 4777 Nov 22 11:00 a.out -rw-rw-
r-- 1 srm srm  346 Jul 15  2016 array.c drwxrwxr-x 2
srm srm 4096 Sep  2 13:22 CP -rw-rw-r-- 1 srm srm
81 Aug  3 16:47 cpgm.c
-rw-rw-r-- 1 srm srm 842 Aug  6 12:21 duplicate.c
-rw-rw-r-- 1 srm srm  265 Oct 31 13:26 fib.c
[srm@srmlinux]$
```

c) [srm@srmlinux]$ ls –a (To show Hidden files)

```
.      2.prn        .bashrc      fun.c           mac.c     sample.c
..     a.out        CP          .gnome2          Makefile .ssh
111    array.c       cpgm.c       HelloWorld.class  .mozilla
streverse.c1.ccc   .bash_history  duplicate.c  HelloWorld.java   OS
        .vim 1.docx  .bash_logout   .emacs                  inc.c  pr.c
```

```
    .viminfo
1.prn   .bash_profile  fib.c        JAVA            print.c  vin [srm@srmlinux]$
```

d) [srm@srmlinux]$ ls –x

```
111             1.ccc           1.docx   1.prn       2.prn
                a.out array.cCP                     cpgm.c
duplicate.c  fib.c              fun.c
HelloWorld.class  HelloWorld.java  inc.c   JAVA       mac.c       Makefile
OS              pr.c           print.c  sample.c     streverse.c vin
[srm@srmlinux]$
```

---

6) Command : mv (Move)

a) **mv (Move a File to Directory)** <u>*Syntax :*</u> *mv SourceFileDestinationDirectory*

```
[srm@srmlinux]$ mv coll.txt
programs[srm@srmlinux]$
cd programs [srm@srmlinux
programs]$ ls
coll.txt
                    she
llprogram
[srm@srmlinux
programs]$
```

b) mv (Move Contents of one File to Another File)<u>*Syntax :*</u> *mv SourceFile DestinationFile*

```
[srm@srmlinux]$ cat
one.txt
Hi, Welcome to CSE Lab
SRM Institute of Science and Technology
[srm@srmlinux]$ mv one.txt new.txt
[srm@srmlinux]$ cat new.txt
Hi, Welcome to CSE Lab
SRM Institute of Science and Technology
[srm@srmlinux]$
```

---

7) Command : rm

(Remove a File)<u>*Syntax :*</u>

rm filename

```
[srm@srmlinux]$ ls
a.out     college2.txt file.txt firs.sh one
programs col1.txt  college.txt   fir.sh
          new.txt pgms second.sh
[srm@srmlinux]$ rm new.txt
[srm@srmlinux]$ ls
a.out     college2.txt  file.txt  firs.sh  pgms
          second.shcol1.txt  college.txt   fir.sh
          one                      programs
```

```
[srm@srmlinux]$
```

---

8) Command : rmdir (Remove a Diretory)<u>Syntax :</u> *rmdir Directoryname*

```
[srm@srmlinux programs]$ ls
coll.txt  shellprogram
[srm@srmlinux programs]$ rmdir shellprogram
[srm@srmlinux programs]$ ls
coll.txt
[srm@srmlinux programs]$
```

---

9) Command : rm -rf (Remove a Diretory which is not Empty)

<u>Syntax :</u> *rm* ***–rf*** *Directoryname*

```
[srm@srmlinux programs]$ ls coll.txt
[srm@srmlinux]$ rm -rf programs
[srm@srmlinux]$ ls
a.out     college2.txt file.txt firs.sh
pgms col1.txt  college.txt   fir.sh
          one
          second.sh[srm@srmlinux]$
```

---

10) Command : echo (Prints the Given String)

a) *Syntax: echo "String"*

```
[srm@srmlinux]$ echo "SRM Institute of Science and Technology"
SRM Institute of Science and Technology
[srm@srmlinux]$
```

b) *Syntax: echo* ***–n*** *"String"* (Don't Print a New Line)

```
[srm@srmlinux]$ echo -n "SRM Institute of Science and Technology" SRMInstitute of Science and Technology
[srm@srmlinux]$
```

---

11) **Command : head (Prints required no. of lines in the File Counting from theBeginning of the File)** <u>Syntax :</u> *head* ***–n*** *filename* ***n*** *:* Number of Lines to be displayed

```
[srm@srmlinux]$ cat one.txt
Welcome to SRM Institute of Science and
TechnologyTiruchirappalli
CSE Dept
Computer Practices
```

```
Laboratory
[srm@srmlinux]$ head -2
one.txt
Welcome to SRM Institute of Science and
TechnologyTiruchirappalli
[srm@srmlinux]$
```

---

12) **Command : tail (Prints required no. of lines in the File Counting fromthe End of the File)** <u>*Syntax :*</u> *tail –**n** filename*

n *:* Number of Lines to be displayed

```
[srm@srmlinux]$ cat
one.txt
Welcome to SRM Institute of Science and
TechnologyTiruchirappalli
CSE Dept
Computer     Practices
Laboratory
[srm@srmlinux]$  tail
-2 one.txtCSE Dept
Computer Practices
 Laboratory
 [srm@srmlinux]$
```

---

13) Command : who (Displays the Users Who Logged into the System)

Syntax: who [srm@srmlinux]$ who

```
root            tty7       2015-01-06 02:14 (:0) srm      pts/20       2016-
      12-01 16:07 (172.16.67.30) [srm@srmlinux]$
```

Command : who am i (Displays the Name of the Current User of thisSystem)

Syntax: who am i

```
[srm@srmlinux]$ who am i
      srm   pts/20       2016-12-01 16:07 (172.16.67.30)
       [srm@srmlinux]$
```

---

14) Command : date (Displays the Current Date and Time)

Syntax: date

```
[srm@srmlinux]$ date
 Thu Dec 1 16:07:49 IST 2016
 [srm@srmlinux]$
```

---

15) Command : cal (Displays the Calendar)

Syntax: cal

```
            [srm@srmlinux
            ]$ cal
            [srm@srmlinux
            ]$ calDecember
            2016
            Su Mo Tu We Th Fr Sa
                        1  2  3
 4    5   6   7   8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31


             [srm@srmlinux]$ cal 5 2009
              May 2009
      Su Mo Tu We Th Fr Sa
                                  1  2
       3    4    5     6    7  8   9
      10  11  12   13  14 15 16
      17  18  19   20  21 22 23
      24  25  26   27  28 29 30
      31
```

16) Command : grep (Displays a Line from the file Containing the Given String)

<u>Syntax :</u> grep "String" filename *<u>Syntax :</u> grep* ***–i*** *"String" filename* ***i*** *– Ignore Case of the Given String*

```
[srm@srmlinux]$ cat one.txt
Welcome to SRM Institute of Science and
TechnologyTiruchirappalli
CSE Dept
Computer Practices Laboratory
[srm@srmlinux]$ grep "SRM"
one.txt
Welcome to SRM Institute of Science and Technology
```

Ex. No. 2 **IMPLEMENTATION OF SYSTEM CALLS IN UNIX**

Date : 05.08.2024

AIM:

To write a program for implementing process management using the following system callsof UNIX operating system: fork, exec, getpid, exit, wait, close.

ALGORITHM:

1. Start the program.
2. Read the input from the command line.
3. Use fork() system call to create process, getppid() system call used to get the parent process ID and getpid() system call used to get the current process ID
4. execvp() system call used to execute that command given on that command line argument
5. execlp() system call used to execute specified command.
6. Open the directory at specified in command line input.
7. Display the directory contents. 8. Stop the program.

PROGRAM:

```
#include<stdio.h> main(int arc,char*ar[])

{ int pid; char s[100];

pid=fork(); if(pid<0)

printf("error"); else

if(pid>0)

{

wait(NULL);

  printf("\n Parent Process:\n"); printf("\n\tParent Process
  id:%d\t\n",getpid()); execlp("cat","cat",ar[1],(char*)0);
  error("can't executecat %s,",ar[1]);

  } else {

  printf("\

  nChild

  process:

  ");
printf("\n\tChildprocess parent id:\t %d",getppid()); sprintf(s,"\n\tChild process id
:\t%d",getpid()); wsrme(1,s,strlen(s)); printf(" "); printf(" ");

printf(" "); execvp(ar[2],&ar[2]); error("can’t execute %s",ar[2]);

}
}
```

SAMPLE OUTPUT:

```
[root@localhost ~]# ./a.out tst date Child process:
Child process id :
3137 Sat Apr 10 02:45:32 IST 2010
Parent Process:
Parent Process id:3136 sd dsaASD[root@localhost ~]# cat tst sddsaASD
```

RESULT:

Thus the program for process management was written and successfully executed.

EX.NO:3 SHELL PROGRAMMING

Date : 12.08.2024

3.1 SUM OF “n” NATURAL NUMBERS

AIM: To write a shell script to find the sum of “N” natural numbers.

ALGORITHM:

Step1: Start the program.
Step2: Enter the number.
Step3: Assign “S” is equal to zero and i is even to one.
Step4: Using while loop calculate the sum, display the sum.
Step5: Stop the program.

PROGRAM:

```
echo "Enter n:"
read n
i=1
sum=0
while [ $i -le $n ]
do
    sum=`expr $sum + $i`
    i=`expr $i + 1`
done
echo "The sum is: $sum"
```

OUTPUT:

```
enter n 10
the sum is : 55
```

RESULT:

Thus a shell script to find the sum of “N” natural numbers was executed successfully.

3.2 FIBONACCI SERIES

AIM:

To write a shell script to generate Fibonacci series.

ALGORITHM:

Step1: Start the program.
Step2: Enter the number.
Step3: Initialize “b” and “d” is equal to zero.
Step4: Using while loop, the Fibonacci series is generated.
Step5: Terminate the program.

PROGRAM:

```
#!/bin/bash
echo "Enter the number of terms in the Fibonacci series:"
read n
a=0
b=1
count=0
echo "Fibonacci series up to $n terms:"
while [ $count -lt $n ]; do
   echo $a
   fn=$((a + b))
   a=$b
   b=$fn
   count=$((count + 1))
done
echo "Program terminated."
```

OUTPUT:

```
$ ./fibonacci.sh
Enter the number of terms in the Fibonacci series:
10
Fibonacci series up to 10 terms:
0
1
1
2
3
5
8
13
21
34
Program terminated.
```

RESULT:

Thus a shell script to generate Fibonacci series was executed successfully.

3.3. CALCULATE EMPLOYEE'S PAYROLL

AIM:

To write the shell script to find net and gross pay.

ALGORITHM:

```
Step1: Start the program.
Step2: Enter the basic pay.
Step3: Calculate HRA, PF and DA.
Step4: Calculate gross pay, using HRA, PF and DA.
Step5: Calculate the net pay.
Step6: Stop the program.
```

PROGRAM:

```
#!/bin/bash
echo "Enter basic pay:"
read basic_pay
echo "Enter DA (Dearness Allowance):"
read da
echo "Enter HRA (House Rent Allowance):"
read hra
echo "Enter PF (Provident Fund):"
read pf
gross_pay=$(expr $basic_pay + $da + $hra)
net_pay=$(expr $gross_pay - $pf)
echo "Net Pay: $net_pay"
echo "Gross Pay: $gross_pay"
```

OUTPUT:

```
$ ./pay_calculator.sh
Enter basic pay:
10000
Enter DA (Dearness Allowance):
500
Enter HRA (House Rent Allowance):
2000
Enter PF (Provident Fund):
1800
Net Pay: 10700
Gross Pay: 12500
```

RESULT:

Thus a shell script to find net and gross pay was executed successful.

3.4 REVERSE THE STRING AND TO CALCULATE THE LENGTH OF THE STRING.

AIM:

To write the shell script to find the length of the string and reverse the string.

ALGORITHM:

Step1: Start the program.
Step2: Enter the string.
Step3: Calculate the length of the string using “wc” command.
Step4: Using while loop, check whether the length of string is greater than zero.
Step5: Using the test command, cut each other character and store it in a temporary variable.
Step6: Stop the program.

PROGRAM:

```
echo " enter the string " read str
len=`echo $str | wc -c` len=`expr $len - 1`
echo " the length of the string is $len " while [ $len -gt 0 ]
do temp=`echo $str | cut -c
$len` rev=`echo $rev$temp` len=`expr $len - 1`
done
echo " the reversed sytring is $rev "

enter the string welcome the length of the string is 7
the reversed sytring is emoclew
```

OUTPUT:

```
Enter the string:
welcome
The length of the string is 7
The reversed string is emoclew
```

RESULT:

Thus the shell script to find the length of the string and reverse the string was executed successfully.

3.5 FIND GIVEN STRING IS PALINDROME OR NOT

AIM:

To write the shell script to find given string is palindrome or not.

ALGORITHM:

Step1: Start the program.
Step2: Enter the string.
Step3: Calculate the length of the string using “wc” command.
Step4: Using while loop, check whether the length of string is greater than zero.
Step5: Using the test command, cut each other character and store it in a temporary variable.
Step6: Check reverse of the string and entered string are equal,if it is equal print string is
palindrome else print string is not a palindrome.
Step7: Stop the program.

PROGRAM:

```
echo "Enter the string: "
read str
len=$(echo -n "$str" | wc -c)
rev=""
while [ $len -gt 0 ]; do
   temp=$(echo "$str" | cut -c $len)
   rev="$rev$temp"
   len=$((len - 1))
done

echo "The reversed string is $rev"
if [ "$rev" = "$str" ]; then
   echo "$str is a palindrome"
else
   echo "$str is not a palindrome"
fi
```

OUTPUT:

```
Enter the string: madam
The length of the string is 5
The reversed string is madam
madam is a palindrome
```

RESULT:

Thus, the shell script to determine if the given string is a palindrome was executed successfully.

EX.NO: 4 **IMPLEMENTATION OF SEMAPHORES**
Date : 12.08.2024
AIM:

To write a C program to implement PCP (Producer Consumer Problem) using semaphores.

ALGORITHM:

```
Step1: Read size of buffer
Step2: Producer process produces and buffers the items using shmget() and shmctl()
Step3: Consumer process consumes item from buffer using semop() and semrel()
Step4: Producer process waits if buffer is full and consumer process waits if buffer is empty.
```

PROGRAM:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdlib.h>

union semun
{
   int val;
   struct semid_ds *buf;
   unsigned short *array;
};
void semaphoreOperation(int semid, int sem_num, int op)
{
   struct sembuf sb;
   sb.sem_num = sem_num;
   sb.sem_op = op;
   sb.sem_flg = 0;
   semop(semid, &sb, 1);
}
int main() {
   int n, sid, k;
   int *buffer;
   int in = 0, out = 0;
   printf("Enter the size of the buffer: ");
   scanf("%d", &n);
   sid = semget(IPC_PRIVATE, 3, IPC_CREAT | 0666);
   if (sid == -1) {
      perror("Semaphore creation failed");
      exit(1);
   }
```

```
    union semun arg;
    arg.array = (unsigned short *)malloc(3 * sizeof(unsigned short));
    arg.array[0] = n; // Semaphore for empty slots
    arg.array[1] = 0; // Semaphore for full slots
    arg.array[2] = 1; // Mutex semaphore
    semctl(sid, 0, SETALL, arg);
    k = shmget(IPC_PRIVATE, n * sizeof(int), IPC_CREAT | 0666);
    buffer = (int *)shmat(k, NULL, 0);
    if (fork() > 0) { // Producer Process
        for (int i = 0; i < n; i++) {
            int item;
            printf("Enter an item to produce: ");
            scanf("%d", &item);
            semaphoreOperation(sid, 0, -1);
            semaphoreOperation(sid, 2, -1);
            buffer[in] = item;
            in = (in + 1) % n;
            semaphoreOperation(sid, 2, 1);
            semaphoreOperation(sid, 1, 1);
        }
        shmdt(buffer);
        shmctl(k, IPC_RMID, NULL);
        semctl(sid, 0, IPC_RMID, 0);
    }
else
{
        for (int j = 0; j < n; j++)
{
            int item;
            semaphoreOperation(sid, 1, -1);
            semaphoreOperation(sid, 2, -1);
            item = buffer[out];
            out = (out + 1) % n;
            semaphoreOperation(sid, 2, 1);
            semaphoreOperation(sid, 0, 1);
            printf("Consumer consumed %d\n", item);
        }
        shmdt(buffer);
        shmctl(k, IPC_RMID, NULL);
        semctl(sid, 0, IPC_RMID, 0);
    }

    free(arg.array);
    return 0;
}
```

OUTPUT:

```
Enter the size of the buffer: 3
Enter an item to produce: 5
Enter an item to produce: 10
Enter an item to produce: 15
Consumer consumed 5
Consumer consumed 10
Consumer consumed 15
```

RESULT:

Thus the program to implement PCP (Producer Consumer Problem) using semaphores was executed successfully.

EX.NO: 5 IMPLEMENTATION OF THREADING & SYNCHRONIZATION APPLICATIONS

Date:02.09.2024

AIM:

To write a C program to implement thread and synchronization application.

ALGORITHM:

```
Step1: Start the program.
Step2: i is initialized in the beginning of the main function.
Step3: Pthread function used to create thread by calling myThread().The variable is locked
      using mutex variable i.
Step4: The threads wait for 30ms then the thread is completed one by one in the order which
      will completed execution.
Step5: At the end of the main function the mutex is destroyed.
Step6: Stop the program.
```

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *myThreadFun(void *vargp)
{
   int thread_id = (int)(intptr_t)vargp;
   printf("Thread %d is processing\n", thread_id);
   for (volatile long i = 0; i < 100000000; i++);
   printf("Thread %d is completed\n", thread_id);
   pthread_exit(NULL);
}
int main() {
   int i;
   pthread_t a[5];
   printf("Before Thread\n");
   for (i = 0; i < 5; i++)
{
      if (pthread_create(&a[i], NULL, myThreadFun, (void *)(intptr_t)i) != 0)
{
         printf("Thread not created\n");
      }
   }
   for (i = 0; i < 5; i++) {
      pthread_join(a[i], NULL);
   }
   printf("All threads completed.\n");
   return 0;
}
```

OUTPUT:

```
Before Thread
Thread 0 is processing
Thread 1 is processing
Thread 2 is processing
Thread 3 is processing
Thread 4 is processing
Thread 0 is completed
Thread 1 is completed
Thread 2 is completed
Thread 3 is completed
Thread 4 is completed
All threads completed.
```

RESULT:

Thus, the program to implement thread synchronization using mutexes was executed successfully. Each thread processed its task without conflicts, ensuring safe access to the shared resource.

EX.NO: 6 **CPU SCHEDULING ALGORITHMS**
Date : 09.09.2024

Round Robin ALGORITHM

AIM:

To write a C program to implement round robin scheduling ALGORITHM.

ALGORITHM:

1. Read no. of processes and time quantum (TQ).
2. Read process name and burst time (BT) for each process.
3. Ready queue is treated as circular queue. CPU schedules all processes (according to their of
   Order of arrival) only up to given time quantum.
4. A timer is set to interrupt the scheduling if time quantum expires for a process.
5. If BT of process is greater than TQ then after executing upto TQ, it gets added to tail of
   ready queue.
6. If BT of process is less than TQ then CPU gets released from it and schedules next process
   in ready queue.
7. Set waiting time (WT) of first process as zero and turnaround time (TAT) as burst time.
8. Calculate waiting time and turnaround time of other processes as follows:
   Pi (WT) = Pi-1(WT) +P i-1(BT)
   P i (TAT) = P i (BT) + P i (WT)
9. Calculate and display average WT and TAT.
10. Display order of execution of processes ie. Process name, burst time, WT and TAT.

PROGRAM:

```
#include<stdio.h>
int main()
{
   int  n;
   printf("Enter Total Number of Processes:");
   scanf("%d", &n);
   int wait_time = 0, ta_time = 0, arr_time[n], burst_time[n], temp_burst_time[n];
   int x = n;
   for(int i = 0; i < n; i++)
   {
      printf("Enter Details of Process %d \n", i + 1);
      printf("Arrival Time:  ");
      scanf("%d", &arr_time[i]);
      printf("Burst Time:   ");
      scanf("%d", &burst_time[i]);
      temp_burst_time[i] = burst_time[i];
   }
```

```
    int time_slot;
    printf("Enter Time Slot:");
    scanf("%d", &time_slot);
    int total = 0,  counter = 0,i;
    printf("Process ID       Burst Time       Turnaround Time      Waiting Time\n");
    for(total=0, i = 0; x!=0; )
    {
        if(temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0)
        {
total = total + temp_burst_time[i];
            temp_burst_time[i] = 0;
            counter=1;
        }
        else if(temp_burst_time[i] > 0)
        {
            temp_burst_time[i] = temp_burst_time[i] - time_slot;
            total  += time_slot;
        }
        if(temp_burst_time[i]==0 && counter==1)
        {
            x--;
            printf("\nProcess No %d  \t\t %d\t\t\t\t %d\t\t\t %d", i+1, burst_time[i],
                   total-arr_time[i], total-arr_time[i]-burst_time[i]);
            wait_time = wait_time+total-arr_time[i]-burst_time[i];
            ta_time += total -arr_time[i];
            counter =0;
        }
        if(i==n-1)
        {
            i=0;
        }
        else if(arr_time[i+1]<=total)
        {
            i++;
        }
        else
        {
            i=0;
        }
    }
    float average_wait_time = wait_time * 1.0 / n;
    float average_turnaround_time = ta_time * 1.0 / n;
    printf("\nAverage Waiting Time:%f", average_wait_time);
    printf("\nAvg Turnaround Time:%f", average_turnaround_time);
    return 0;
}
```

OUTPUT:

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

Process ID      Burst Time      Turnaround Time     Waiting Time

Process No 1      10              20            10
Process No 2      8               22                    14
Process No 3      7               23                    16

Average Waiting Time: 13.333333
Avg Turnaround Time: 21.666666
```

EX.NO: 6(b) **Shortest Job First Algorithm**

AIM:

To write a C program to implement SJF (Shortest Job First) scheduling ALGORITHM.

ALGORITHM:

```
Step1: Read no. of processes.
Step2: Read process name and burst time for each process.
Step3: Sort the processes in ready queue according to burst time. CPU schedules
        process with shortest burst time first followed by other processes.
Step4: Set waiting time(WT) of first process as zero and turnaround time(TAT) as
        burst time.
Step5: Calculate waiting time and turnaround time of other processes as follows:
        Pi (WT) = P i-1(WT) +P i-1(BT)
        P i (TAT) = P i ( BT) + P i (WT)
Step6: Calculate and display average WT and TAT.
Step7: Display order of execution of processes ie. Process name, burst time, WT and
        TAT.
```

PROGRAM:

```
#include<stdio.h>
int main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,totalT=0,pos,temp;
   float avg_wt,avg_tat;
   printf("Enter number of process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time:\n");
   for(i=0;i<n;i++)
   {
      printf("p%d:",i+1);
      scanf("%d",&bt[i]);
      p[i]=i+1;
   }

   //sorting of burst times
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
```

```
        {
           if(bt[j]<bt[pos])
               pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;

    //finding the waiting time of all the processes
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
             //individual WT by adding BT of all previous completed processes
            wt[i]+=bt[j];

        //total waiting time
        total+=wt[i];
    }

    //average waiting time
    avg_wt=(float)total/n;

    printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        //turnaround time of individual processes
        tat[i]=bt[i]+wt[i];

        //total turnaround time
        totalT+=tat[i];
        printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
     //average turnaround time
    avg_tat=(float)totalT/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f",avg_tat);
}
```

OUTPUT:

```
Enter number of process:4

Enter Burst Time:
p1:5
p2:4
p3:12
p4:7

Process      Burst Time     Waiting Time    Turnaround Time
p2            4              0               4
p1            5              4               9
p4            7              9               16
p3            12             16              28

Average Waiting Time=7.250000
Average Turnaround Time=14.250000
```

EX.NO: 6(C) First Come First Serve ALGORITHM (FCFS)

AIM:

To write a C program to implement FCFS scheduling ALGORITHM.

ALGORITHM:

```
Step1: Read no. of processes.
Step2: Read process name and burst time for each process.
Step3: CPU schedules processes according to their order of arrival in read queue (ie It
first
        executes process which is at head of ready queue)
Step4: Set waiting time(WT) of first process as zero and turnaround time(TAT) as
burst time.
Step5: Calculate waiting time and turnaround time of other processes as follows:
        Pi (WT) = P i-1(WT) +P i-1(BT)
        P i (TAT) = P i ( BT) + P i (WT)
Step6: Calculate and display average WT and TAT.
Step7: Display order of execution of processes ie. Process name, burst time, WT and
TAT.
```

PROGRAM:

```
#include <stdio.h>
int main()
{
   int pid[15];
   int bt[15];
   int n;
   printf("Enter the number of processes: ");
   scanf("%d",&n);

   printf("Enter process id of all the processes: ");
   for(int i=0;i<n;i++)
   {
      scanf("%d",&pid[i]);
   }

   printf("Enter burst time of all the processes: ");
   for(int i=0;i<n;i++)
   {
      scanf("%d",&bt[i]);
   }
    int i, wt[n];
   wt[0]=0;
    //for calculating waiting time of each process
   for(i=1; i<n; i++)
   {
      wt[i]= bt[i-1]+ wt[i-1];
    }

    printf("Process ID     Burst Time     Waiting Time     TurnAround Time\n");
    float twt=0.0;
    float tat= 0.0;
for(i=0; i<n; i++)
    {
        printf("%d\t\t", pid[i]);
        printf("%d\t\t", bt[i]);
        printf("%d\t\t", wt[i]);

        //calculating and printing turnaround time of each process
        printf("%d\t\t", bt[i]+wt[i]);
        printf("\n");

        //for calculating total waiting time
        twt += wt[i];

        //for calculating total turnaround time
        tat += (wt[i]+bt[i]);
    }
    float att,awt;

    //for calculating average waiting time
    awt = twt/n;

    //for calculating average turnaround time
    att = tat/n;
    printf("Avg. waiting time= %f\n",awt);
    printf("Avg. turnaround time= %f",att);
}
```

OUTPUT:

```
Enter the number of processes: 3
Enter process id of all the processes: 1 2 3
Enter burst time of all the processes: 5 11 11
Process ID     Burst Time     Waiting Time     TurnAround Time
1              5              0                5
2              11             5                16
3              11             16               27
Avg. waiting time= 7.000000
Avg. turnaround time= 16.000000
```

EX.NO: 6 (D) Priority Scheduling ALGORITHM

AIM:

To write a C program to implement priority scheduling ALGORITHM.

ALGORITHM:

```
Step1: Read no. of processes.
Step2: Read process name, burst time and priority for each process.
Step3: Sort the processes in ready queue according to priority. (i.e. Process with high
priority get placed at head of ready queue) CPU schedules process with high priority
first followed by other processes.
Step4: Set waiting time (WT) of first process as zero and turnaround time (TAT) as
burst time.
Step5: Calculate waiting time and turnaround time of other processes as follows:
        Pi (WT) = P i-1(WT) +P i-1(BT)
        P i (TAT) = P i (BT) + P i (WT)
Step6: Calculate and display average WT and TAT.
Step7: Display order of execution of processes ie. Process name, burst time, priority,
WT and
        TAT.
Step8: Stop the program.
```

PROGRAM:

```
#include <stdio.h>
//Function to swap two variables
void swap(int *a,int *b)
{
   int temp=*a;
   *a=*b;
   *b=temp;
}
int main()
{
   int n;
   printf("Enter Number of Processes: ");
   scanf("%d",&n);
   int b[n],p[n],index[n];
   for(int i=0;i<n;i++)
   {
      printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
      scanf("%d %d",&b[i],&p[i]);
     index[i]=i+1;
   }
   for(int i=0;i<n;i++)
   {
      int a=p[i],m=i;
      for(int j=i;j<n;j++)
```

```
        {
           if(p[j] > a)
           {
              a=p[j];
              m=j;
           }
        }
        swap(&p[i], &p[m]);
        swap(&b[i], &b[m]);
        swap(&index[i],&index[m]);
     }
     int t=0;
     printf("Order of process Execution is\n");
     for(int i=0;i<n;i++)
     {
        printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);
        t+=b[i];
     }
     printf("\n");
     printf("Process Id     Burst Time   Wait Time    TurnAround Time\n");
     int wait_time=0;
     for(int i=0;i<n;i++)
     {
        printf("P%d          %d          %d
  %d\n",index[i],b[i],wait_time,wait_time + b[i]);
        wait_time += b[i];
     }
     return 0;
  }
```

OUTPUT:

```
Enter Number of Processes: 3
Enter Burst Time and Priority Value for Process 1: 10 2
Enter Burst Time and Priority Value for Process 2: 5 0
Enter Burst Time and Priority Value for Process 3: 8 1
Order of process Execution is
P1 is executed from 0 to 10
P3 is executed from 10 to 18
P2 is executed from 18 to 23

Process Id     Burst Time   Wait Time    TurnAround Time
   P1          10           0            10
   P3          8           10            18
   P2          5           18            23
```

RESULT:

Thus the Process Scheduling algorithms programs were executed successfully.

EX.NO: 7 Implement Bankers ALGORITHM for Deadlock Detection
Date :23.09.2024
AIM:

To write a C program to implement the concept of deadlock detection.

ALGORITHM:

1. Get the number of processes and number of resource instances.
2. Get the allocation matrix and Available matrix from the user.
3. Calculate need matrix.
4. Using banker’s ALGORITHM allocate resources to processes.
5. Print deadlock occurred or not.
6. Stop the program.

PROGRAM:

```
#include <stdio.h>
#define MAX 100

int max[MAX][MAX], alloc[MAX][MAX], need[MAX][MAX], avail[MAX], n, r;

void input() {
    printf("Enter the number of Processes: ");
    scanf("%d", &n);
    printf("Enter the number of Resource Instances: ");
    scanf("%d", &r);

    printf("Enter the Max Matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < r; j++)
            scanf("%d", &max[i][j]);

    printf("Enter the Allocation Matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < r; j++)
            scanf("%d", &alloc[i][j]);

    printf("Enter the Available Resources:\n");
    for (int j = 0; j < r; j++)
        scanf("%d", &avail[j]);
}

void calculateNeed() {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < r; j++)
            need[i][j] = max[i][j] - alloc[i][j];
}
```

```
int isSafe() {
    int finish[MAX] = {0}, work[MAX], count = 0;
    for (int i = 0; i < r; i++)
        work[i] = avail[i];

    while (count < n) {
. int found = 0;
        for (int i = 0; i < n; i++) {
            if (!finish[i]) {
                int j;
                for (j = 0; j < r; j++)
                    if (need[i][j] > work[j]) break;

                if (j == r) {
                    for (j = 0; j < r; j++)
                        work[j] += alloc[i][j];
                    finish[i] = 1;
                    found = 1;
                    count++;
                }
            }
        }
        if (!found) {
            printf("System is in Deadlock. The Deadlocked Processes are:\n");
            for (int i = 0; i < n; i++)
                if (!finish[i])
                    printf("P%d ", i + 1);
            printf("\n");
            return 0; // Deadlock occurred
        }
    }
    printf("No Deadlock Occurred.\n");
    return 1; // No deadlock
}

int main() {
    printf("********** Deadlock Detection Algo ************\n");
    input();
    calculateNeed();
    if (isSafe()) {
        printf("Process\tAllocation\tMax\tAvailable\n");
        for (int i = 0; i < n; i++) {
            printf("P%d\t", i + 1);
            for (int j = 0; j < r; j++)
                printf("%d ", alloc[i][j]);
            printf("\t");
            for (int j = 0; j < r; j++)
                printf("%d ", max[i][j]);
            if (i == 0) {
                printf("\t");
                for (int j = 0; j < r; j++)
                    printf("%d ", avail[j]);
            }
            printf("\n");
        }
    }
    return 0;
}
```

OUTPUT:

```
Enter the number of Processes: 5
Enter the number of Resource Instances: 3
Enter the Max Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the Available Resources:
3 3 2

********** Deadlock Detection Algo ************
No Deadlock Occurred.
Process         Allocation      Max     Available
P1      0 1 0   P1: 7 5 3       Available: 3 3 2
P2      2 0 0
P3      3 0 2
P4      2 1 1
P5      0 0 2
```

RESULT:

Thus the banker algorithm was implemented successfully for Deadlock Detection.

EX.NO: 8 Implement BANKERS ALGORITHM for Deadlock Avoidance
Date:23.9.2024

AIM:

A program to simulate the BANKERS ALGORITHM for Deadlock Avoidance.

ALGORITHM:

```
Step1: Get the number of processes and number of resource instances.
Step2: Get the allocation matrix and Available matrix from the user.
Step3: Calculate need matrix.
Step4: Using banker’s ALGORITHM allocate resources to processes.
Step5: Print safe sequence of processes.
Step6: Stop the program.
```

PROGRAM:

```
#include <stdio.h>
#define MAX 100
int max[MAX][MAX], alloc[MAX][MAX], need[MAX][MAX], avail[MAX];
int n, r;
void input() {
   printf("Enter the number of Processes: ");
   scanf("%d", &n);
   printf("Enter the number of Resource Instances: ");
   scanf("%d", &r);
   printf("Enter the Max Matrix:\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < r; j++)
         scanf("%d", &max[i][j]);

   printf("Enter the Allocation Matrix:\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < r; j++)
         scanf("%d", &alloc[i][j]);

   printf("Enter the Available Resources:\n");
   for (int j = 0; j < r; j++)
      scanf("%d", &avail[j]);
}
void show() {
   printf("Process\tAllocation\tMax\tAvailable\n");
   for (int i = 0; i < n; i++) {
      printf("P%d\t", i + 1);
      for (int j = 0; j < r; j++)
printf("%d ", alloc[i][j]);
      printf("\t");
```

```
        for (int j = 0; j < r; j++)
            printf("%d ", max[i][j]);
        if (i == 0) {
            printf("\t");
            for (int j = 0; j < r; j++)
                printf("%d ", avail[j]);
        }
        printf("\n");
    }
}

void cal() {
    int finish[MAX] = {0}, safeSeq[MAX], count = 0;

    for (int i = 0; i < n; i++)
        for (int j = 0; j < r; j++)
            need[i][j] = max[i][j] - alloc[i][j];

    while (count < n) {
        int found = 0;
        for (int i = 0; i < n; i++) {
            if (!finish[i]) {
                int j;
                for (j = 0; j < r; j++)
                    if (need[i][j] > avail[j]) break;

                if (j == r) {
                    for (j = 0; j < r; j++)
                        avail[j] += alloc[i][j];
                    finish[i] = 1;
                    safeSeq[count++] = i;
                    found = 1;
                    printf("P%d->", i);
                }
            }
        }
        if (!found) break;  // No more processes can be finished
    }
    printf("\n");
    if (count == n) {
        printf("The system is in a safe state.\n");
        printf("Safe sequence is: ");
        for (int i = 0; i < n; i++)
            printf("P%d ", safeSeq[i] + 1);
        printf("\n");
    } else {
        printf("Processes are in deadlock.\n");
        printf("The system is in unsafe state.\n");
    }
}
int main() {
    printf("********** Banker's ALGORITHM ************\n");
    input();
    show();
    cal();
    return 0;
}
```

OUTPUT:

```
Enter the number of Processes: 5
Enter the number of Resource Instances: 3
Enter the Max Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the Available Resources:
3 3 2

********** Banker's ALGORITHM ************
Process         Allocation      Max     Available
P1      0 1 0   7 5 3   3 3 2
P2      2 0 0   3 2 2
P3      3 0 2   9 0 2
P4      2 1 1   2 2 2
P5      0 0 2   4 3 3
P1->P3->P4->P2->

The system is in a safe state.
Safe sequence is: P1 P3 P4 P2 P5
```

RESULT:

Thus the program for implementing deadlock avoidance ALGORITHM was implemented has been executed successfully.

EX.NO: 9(a) IMPLEMENTATION OF MEMORY MANAGEMENT-PAGING

Date: 30.09.2024

AIM:

To write a C program to implement paging concept for memory management.

ALGORITHM:

```
Step 1: Start the program.
Step 2: Enter the logical memory address i.e no of pages in memory.
Step 3: Enter the page table which has offset and page frame.
Step 4: The corresponding physical address can be calculate by, PA = [
pageframe* No. of page size ] + Page offset.
Step 5: Print the physical address for the corresponding logical address. Step
6: Terminate the program.
```

PROGRAM:

```
#include <stdio.h>
#define MAX 50

int main() {
    int page[MAX], i, n, f, ps, off, pno;

    printf("Enter the number of pages in memory: ");
    scanf("%d", &n);
    printf("Enter page size: ");
    scanf("%d", &ps);
    printf("Enter number of frames: ");
    scanf("%d", &f);

    // Initialize page table
    for (i = 0; i < n; i++) {
        page[i] = -1;
    }

    printf("\nEnter the page table (Enter frame no as -1 if that page is not present in any
frame)\n\n");
    printf("Page No\tFrame No\n-------\t-------\n");
    for (i = 0; i < n; i++) {
        printf("%d\t\t", i);
        scanf("%d", &page[i]);
    }
    printf("\nEnter the logical address (i.e., page no and offset): ");
    scanf("%d%d", &pno, &off);

    // Check if the page is present
    if (pno < 0 || pno >= n || page[pno] == -1) {
        printf("\nThe required page is not available in any of the frames.\n");
    } else {
        printf("\nPhysical address (i.e., frame no and offset): %d, %d", page[pno], off);
printf("\nPhysical Address is %d\n", (page[pno] * ps) + off);
    }
    return 0;
}
```

OUTPUT:

```
Enter the number of pages in memory: 4
Enter page size: 2
Enter number of frames: 4
Enter the page table (Enter frame no as -1 if that page is not present in any frame)

Page No Frame No
------- -------
0      3
1      5
2      1
3      7

Enter the logical address (i.e., page no and offset): 2 19

Physical address (i.e., frame no and offset): 1, 19
Physical Address is 21
```

RESULT:

Thus C program for implementing paging concept for memory management has beenexecuted successfully.

EX.NO: 9 (b) PAGE REPLACEMENT ALGORITHM - (First In First Out)

AIM:

To write a C program to implement FIFO page replacement ALGORITHM.

ALGORITHM:

Step1: Read the size of the frame, no. of elements and elements one by one.
Step2: Initialize the frames with value -1.
Step3: Insert each element into frame, if it’s already not present.
Step4: If the frame is full and the new element is not already present then replace the oldest element by the new element.
Step5: Increment no. of page faults by one while inserting each element into the frames.
Step6: Display the contents of frames during processing and the total no. of page faults.

PROGRAM:

```
#include <stdio.h>
int main() {
    int reference_string[10], page_faults = 0, m, n, s, pages, frames;
    printf("Enter Total Number of Pages: ");
    scanf("%d", &pages);
    printf("Enter values of Reference String:\n");
    for (m = 0; m < pages; m++) {
        printf("Value No. [%d]: ", m + 1);
        scanf("%d", &reference_string[m]);
    }

    printf("Enter Total Number of Frames: ");
    scanf("%d", &frames);

    int temp[frames];
    for (m = 0; m < frames; m++) {
        temp[m] = -1; // Initialize frames to -1 (indicating empty)
    }
    for (m = 0; m < pages; m++) {
        s = 0; // Reset flag for page found
        for (n = 0; n < frames; n++) {
if (reference_string[m] == temp[n]) {
                s = 1; // Page hit
                break;
            }
        }
        // If page is not found, it is a page fault
        if (s == 0) {
            temp[page_faults % frames] = reference_string[m]; // Replace using
FIFO
            page_faults++;
        }

        // Display current frame state
        printf("\nCurrent Frame State: ");
        for (n = 0; n < frames; n++) {
            printf("%d\t", temp[n]);
        }
    }

    printf("\nTotal Page Faults: %d\n", page_faults);
    return 0;
}
```

OUTPUT:

```
Enter Total Number of Pages: 5
Enter values of Reference String:
Value No. [1]: 3
Value No. [2]: 5
Value No. [3]: 2
Value No. [4]: 3
Value No. [5]: 4
Enter Total Number of Frames: 3

Current Frame State: 3      -1      -1
Current Frame State: 3      5       -1
Current Frame State: 3      5       2
Current Frame State: 3      5       2
Current Frame State: 4      5       2
Total Page Faults: 4
```

RESULT:

Thus the program to implement FIFO page replacement ALGORITHM was executed successfully.

EX.NO: 9 (C) **Least Recently Used (LRU)**

AIM:

To write a C program to implement LRU page replacement ALGORITHM.

ALGORITHM:

```
Step1: Read the size of the frame, no. of elements and elements one by one.
Step2: Initialize the frames with value -1.
Step3: Insert each element into frame, if it’s already not present.
Step4: If the frame is full and new element is not already present then replace the least
recently used element by the new element.
Step5: Increment no. of page faults by one while inserting each element into the
frames.
Step6: Display the contents of frames during processing and the total no. of page
faults.
```

PROGRAM:

```
#include <stdio.h>

int main() {
    int frames[10], pages[10], temp[10];
    int total_pages, total_frames;
    int page_faults = 0;

    printf("Enter Total Number of Frames: ");
    scanf("%d", &total_frames);
    for (int m = 0; m < total_frames; m++) {
        frames[m] = -1; // Initialize frames to -1 (empty)
    }
    printf("Enter Total Number of Pages: ");
    scanf("%d", &total_pages);

    printf("Enter Values for Reference String:\n");
    for (int m = 0; m < total_pages; m++) {
        printf("Value No.[%d]: ", m + 1);
        scanf("%d", &pages[m]);
    }

    for (int n = 0; n < total_pages; n++) {
        int page_found = 0; // Flag to check if page is in frames
        // Check if the page is already in one of the frames
        for (int m = 0; m < total_frames; m++) {
            if (frames[m] == pages[n]) {
                page_found = 1;
                break;
            }
        }
        // If the page is not found, we have a page fault
        if (!page_found) {
            int empty_frame = -1;
// Check for an empty frame
            for (int m = 0; m < total_frames; m++) {
                if (frames[m] == -1) {
                    empty_frame = m;
                    break;
                }
            }

            // If there is an empty frame, use it
            if (empty_frame != -1) {
                frames[empty_frame] = pages[n];
            } else {
                // If no empty frame, find the LRU page to replace
                for (int m = 0; m < total_frames; m++) {
                    temp[m] = 0; // Reset temp array
                }

                for (int k = n - 1, l = 1; l <= total_frames - 1; l++, k--) {
                    for (int m = 0; m < total_frames; m++) {
                        if (frames[m] == pages[k]) {
                            temp[m] = 1; // Mark page as used
                        }
                    }
                }

                // Find the first unused page to replace
                for (int m = 0; m < total_frames; m++) {
                    if (temp[m] == 0) {
                        frames[m] = pages[n]; // Replace the LRU page
                        break;
                    }
                }
            }
            page_faults++; // Increment page fault count
        }
        // Display current frame state
        printf("\nCurrent Frame State: ");
        for (int m = 0; m < total_frames; m++) {
            printf("%d\t", frames[m]);
        }
    }
    printf("\nTotal Number of Page Faults: %d\n", page_faults);
    return 0;
}
}
```

OUTPUT:

```
Enter Total Number of Frames: 3
Enter Total Number of Pages: 7
Enter Values for Reference String:
Value No.[1]: 5
Value No.[2]: 6
Value No.[3]: 3
Value No.[4]: 2
Value No.[5]: 5
Value No.[6]: 1
Value No.[7]: 8

Current Frame State: 5      -1      -1
Current Frame State: 5      6       -1
Current Frame State: 5      6       3
Current Frame State: 2      6       3
Current Frame State: 2      6       5
Current Frame State: 2      1       5
Current Frame State: 8      1       5
Total Number of Page Faults: 4
```

RESULT

Thus the program to implement LRU page replacement ALGORITHM was executed successfully.

EX.NO: 10 OPTIMAL (LFU) PAGE REPLACEMENT ALGORITHMS

Data: 07.10.2024

AIM:

To implement Optimal (The page which is not used for longest time) page replacement ALGORITHMS.

ALGORITHM:

```
Step1: Read the size of the frame, no. of elements and elements one by one.
Step2: Initialize the frames with value -1.
Step3: Insert each element into frame, if it's already not present.
Step4: If the frame is full and new element is not already present then replace the least
        frequently used element by the new element.
Step5: Increment no. of page faults by one while inserting each element into the
        frames.
Step6: Display the contents of frames during processing and the total no. of page
        faults.
```

PROGRAM:

```
#include <stdio.h>

int main() {
   int reference_string[25], frames[25], interval[25];
   int pages, total_frames, page_faults = 0;

   printf("Enter Total Number of Pages: ");
   scanf("%d", &pages);

   printf("Enter Values of Reference String:\n");
   for (int m = 0; m < pages; m++) {
      printf("Value No.[%d]: ", m + 1);
      scanf("%d", &reference_string[m]);
   }
   printf("Enter Total Number of Frames: ");
   scanf("%d", &total_frames);

   // Initialize frames
   for (int m = 0; m < total_frames; m++) {
      frames[m] = -1;
   }
```

```
for (int m = 0; m < pages; m++) {
 int flag = 0;
// Check if the page is already in one of the frames
 for (int n = 0; n < total_frames; n++) {
     if (frames[n] == reference_string[m]) {
         flag = 1; // Page is found
         printf("\t"); // Indicate page hit
         break;
     }
 }
// Page fault occurred
 if (flag == 0) {
     int position = -1, maximum_interval = -1;
     // Find the position to replace
     for (int n = 0; n < total_frames; n++) {
         interval[n] = 0; // Reset interval
         for (int temp = m + 1; temp < pages; temp++) {
             if (frames[n] == reference_string[temp]) {
                 interval[n] = temp - m; // Calculate the interval
                 break;
             }
         }

         // If the page is never referenced again
         if (interval[n] == 0) {
             position = n; // Choose this frame for replacement
             break;
         }

         // Find the frame with the maximum interval
         if (interval[n] > maximum_interval) {
             maximum_interval = interval[n];
             position = n;
         }
     }

     // Replace the page
     frames[position] = reference_string[m];
     printf("FAULT\t");
     page_faults++;
 }

 // Print current frame state
 for (int n = 0; n < total_frames; n++) {
     if (frames[n] != -1) {
         printf("%d\t", frames[n]);
     }
 }
```

```
        printf("\n");
    }

    printf("\nTotal Number of Page Faults: %d\n", page_faults);
    return 0;
}
```

OUTPUT:

```
Enter Total Number of Pages: 7
Enter Values of Reference String:
Value No.[1]: 5
Value No.[2]: 6
Value No.[3]: 3
Value No.[4]: 2
Value No.[5]: 5
Value No.[6]: 1
Value No.[7]: 8
Enter Total Number of Frames: 3

FAULT        5
FAULT        6        5
FAULT        3        6        5
FAULT        2        3        5
FAULT        5        2        3
FAULT        1        2        5
FAULT        8        1        5

Total Number of Page Faults: 6
```

RESULT:

Thus the program to implement optimal page replacement algorithm waswritten and executed successfully.