Operating Systems

Lab Programs – 2019-20

Please Note:

- This document contains all OS lab programs along with sample output.
- The programs are taken from the official Lab Manual created by Mrs. Savitha Shetty, Assistant Professor, NMAMIT. Some modifications are done to these original programs.
- The lab programs in this document are not to be considered as final.
- The will give the proper output and these are only for reference.
- The document is not an official copy. Creator of this document is not responsible if you don't get the proper output for the programs.
- Please execute these and test by yourself. If any mistakes are found those will be modified and new copy will be uploaded to google drive under "V Semester Study Materials".

Prepared by:
Shawn Linton Miranda
4NM17CS164

Index

PART-A SHELL & PERL SCRIPTS

PART – B C PROGRAMS

PART - A

1. Largest and Smallest of three numbers

Write a shell program to find and display largest and smallest of three numbers.

Program:

```
#!/bin/sh      //Program 1
echo "Enter 3 numbers : "
read a
read b
read c
if [ $a -gt $b ]
then
        if [ $a -gt $c]
        then
                max=$a
        else
                max=$c
        fi
else
        if [ $b -gt $c ]
        then
                max=$b
        else
                max=$c
        fi
fi

if [ $a -lt $b ]
then
        if [ $a -lt $c ]
        then
                min=$a
        else
                min=$c
        fi
else
        if [ $b -lt $c ]
        then
                min=$b
        else
                min=$c
        fi
fi
echo "Largest number is $max"
echo "Smallest number is $min"
```

```
#!/bin/sh     //Program 2
echo "Please enter the three numbers : "
read x
read y
read z
if [ $x -ge $y ] && [ $x -ge $z ]
then
        echo "$x is the largest number."
elif [ $y -ge $x ] && [ $y -ge $z ]
then echo "$y is the largest number."
else
        echo "$z is the largest number."
fi

if [ $x -le $y ] && [ $x -le $z ]
then
        echo "$x is the smallest number."
elif [ $y -le $x ] && [ $y -le $z ]
then
        echo "$y is the smallest number."
else
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 1.sh
shawnlintonmiranda@HP:~/OS$ sh 1.sh
Please enter the three numbers :
4
8
2
8 is the largest number.
2 is smallest number.
```

2. Test for divisibility of n by m

Write a shell program to check the number n is divisible by m or not. Where m and n are read from command line or from the keyboard interactively.

Program:

```
#!/bin/sh    //Command line arguments
y=`expr $1 % $2`
if [ $y -eq 0 ]
then
  echo "$1 is divisible by $2."
else
  echo "$1 is not divisble by $2."
fi
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 2_1.sh
shawnlintonmiranda@HP:~/OS$ sh 2_1.sh 4 2
4 is divisible by 2.
```

```
shawnlintonmiranda@HP:~/OS$ vi 2_1.sh
shawnlintonmiranda@HP:~/OS$ sh 2_1.sh 7 3
7 is not divisble by 3.
```

```
#!/bin/sh   //Arguments from keyboard
echo "Enter the value of n : "
read n
echo "Enter the value of m : "
read m
y=`expr $n % $m`
if [ $y -eq 0 ]
then
  echo "$n is divisible by $m."
else
  echo "$n is not divisible by $m."
fi
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 2_2.sh
shawnlintonmiranda@HP:~/OS$ sh 2_2.sh
Enter the value of n :
6
Enter the value of m :
3
6 is divisible by 3.
```

```
shawnlintonmiranda@HP:~/OS$ vi 2_2.sh
shawnlintonmiranda@HP:~/OS$ sh 2_2.sh
Enter the value of n :
9
Enter the value of m :
2
9 is not divisible by 2.
```

3. Leap Year

Write a shell program to check whether a year is leap year or not. Display the appropriate messages.

Program:

```
#!/bin/sh
echo "Enter the year : "
read year
x=`expr $year % 400`
y=`expr $year % 100`
z=`expr $year % 4`
if [ $x -eq 0 ] || ([ $y -ne 0 ] && [ $z -eq 0 ])
then
      echo "$year is a leap year."
else
      echo "$year is not a leap year."
fi
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 3.sh
shawnlintonmiranda@HP:~/OS$ sh 3.sh
Enter the year :
2100
2100 is not a leap year.
```

```
shawnlintonmiranda@HP:~/OS$ vi 3.sh
shawnlintonmiranda@HP:~/OS$ sh 3.sh
Enter the year :
2016
2016 is a leap year.
```

```
shawnlintonmiranda@HP:~/OS$ vi 3.sh
shawnlintonmiranda@HP:~/OS$ sh 3.sh
Enter the year :
2001
2001 is not a leap year.
```

4. Comparing File Permissions

Write a shell program that takes two file names, checks whether the permissions for these files are identical. If they are identical display common permission, otherwise output each file followed by its permissions.

Program:

```
#!/bin/sh
echo "Enter two file names : "
read f1 f2
if [ -e $f1 -a -e $f2 ]
then
        p1=`ls -l $f1 | cut -c 2-10`
        p2=`ls -l $f2 | cut -c 2-10`
        if [ "$p1" = "$p2" ]
        then
                echo "$f1 and $f2 have same permission : $p1"
        else
                echo "$f1 has permission : $p1"
                echo "$f2 has permission : $p2"
        fi
else
        echo "Invalid file names!"
fi
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 4.sh
shawnlintonmiranda@HP:~/OS$ ls -l temp*
-rw-r--r-- 1 shawnlintonmiranda shawnlintonmiranda  2 Nov 11 08:15 temp1.txt
-rw-r--r-- 1 shawnlintonmiranda shawnlintonmiranda 15 Aug 20 15:36 temp2.txt
shawnlintonmiranda@HP:~/OS$ sh 4.sh
Enter two file names :
temp1.txt temp2.txt
temp1.txt and temp2.txt have same permission : rw-r--r--
```

```
shawnlintonmiranda@HP:~/OS$ chmod u+x temp1.txt
shawnlintonmiranda@HP:~/OS$ ls -l temp*
-rwxr--r-- 1 shawnlintonmiranda shawnlintonmiranda  2 Nov 11 08:15 temp1.txt
-rw-r--r-- 1 shawnlintonmiranda shawnlintonmiranda 15 Aug 20 15:36 temp2.txt
shawnlintonmiranda@HP:~/OS$ sh 4.sh
Enter two file names :
temp1.txt temp2.txt
temp1.txt has permission : rwxr--r--
temp2.txt has permission : rw-r--r--
```

5. Extraction of substring from string

Write a shell program to display length of the name and also display first and last three characters of the name in two different lines if the name contains at least 6 characters.

Program:

```
#!/bin/sh
echo "Enter a string : "
read string
if [ -z $string ]
then
        echo "Null string"
        z=0
else
        z=`expr "$string" : '.*'`
        echo "String length is $z."
fi
```

```
if [ $z -ge 6 ]
then
        echo "First 3 characters : "
        z=`expr "$string" : '\(...\).*'`
        echo "$z"
        echo "Last 3 characters : "
        z=`expr "$string" : '.*\(...\)'`
        echo "$z"
else
        echo "Length of the string is less than 6."
fi
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 5.sh
shawnlintonmiranda@HP:~/OS$ sh 5.sh
Enter a string :
abc
String length is 3.
Length of the string is less than 6.
```

```
shawnlintonmiranda@HP:~/OS$ vi 5.sh
shawnlintonmiranda@HP:~/OS$ sh 5.sh
Enter a string :

Null string
Length of the string is less than 6.
```

```
shawnlintonmiranda@HP:~/OS$ vi 5.sh
shawnlintonmiranda@HP:~/OS$ sh 5.sh
Enter a string :
abcdefghi
String length is 9.
First 3 characters :
abc
Last 3 characters :
ghi
```

6. Simple Calculator Operations

Write a shell program to implement simple calculator operations.

Program:

```
#!/bin/sh
echo "Choose your option : \n\t+ : Addition\n\t- : Subtraction\n\t* - Multiplication\n\t/ - Division\n"
echo "Enter your choice :"
read ch
echo "Enter the two numbers :"
read a
read b
case $ch in
    '+') y=`expr $a + $b`
        echo "Sum of $a and $b = $y";;
    '-') y=`expr $a - $b`
        echo "Difference of $a and $b = $y";;
    '*') y=`expr $a \* $b`
        echo "Product of $a and $b = $y";;
    '/') y=`expr $a / $b`
        echo "Quotient of $a by $b = $y";;
    *) echo "Invalid choice.";;
esac
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 6.sh
shawnlintonmiranda@HP:~/OS$ sh 6.sh
Choose your option :
        + : Addition
        - : Subtraction
        * - Multiplication
        / - Division

Enter your choice :
/
Enter the two numbers :
17
4
Quotient of 17 by 4 = 4
```

```
shawnlintonmiranda@HP:~/OS$ vi 6.sh
shawnlintonmiranda@HP:~/OS$ sh 6.sh
Choose your option :
        + : Addition
        - : Subtraction
        * - Multiplication
        / - Division

Enter your choice :
*
Enter the two numbers :
6
7
Product of 6 and 7 = 42
```

```
shawnlintonmiranda@HP:~/OS$ vi 6.sh
shawnlintonmiranda@HP:~/OS$ sh 6.sh
Choose your option :
        + : Addition
        - : Subtraction
        * - Multiplication
        / - Division

Enter your choice :
-
Enter the two numbers :
7
3
Difference of 7 and 3 = 4
```

```
shawnlintonmiranda@HP:~/OS$ vi 6.sh
shawnlintonmiranda@HP:~/OS$ sh 6.sh
Choose your option :
        + : Addition
        - : Subtraction
        * - Multiplication
        / - Division

Enter your choice :
+
Enter the two numbers :
4
6
Sum of 4 and 6 = 10
```

```
shawnlintonmiranda@HP:~/OS$ sh 6.sh
Choose your option :
        + : Addition
        - : Subtraction
        * - Multiplication
        / - Division

Enter your choice :
^
Enter the two numbers :
1
2
Invalid choice.
```

7. File name conversion to upper case

Write a shell script that accepts filenames as arguments and for every files, it should check whether it exists in the current directory and if exists convert its name to uppercase and only if there is no file exists with new name.

Program:

```
#!/bin/sh
for file in "$@"; do
        if [ -f $file ];
        then
                ufile=`expr $file | tr '[a-z]' '[A-Z]'`
                if [ -f $ufile ];
                then
                        echo "$ufile is already exists."
                else
                        mv $file $ufile
                        echo "File converted successfully."
                fi
        else
                echo "$file does not exists."
        fi
done
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 7.sh
shawnlintonmiranda@HP:~/OS$ ls
1a.sh  1.sh    2_2.sh  3.sh  5.sh  7.sh  temp1.txt  temp2.txt
1.pl   2_1.sh  2.pl    4.sh  6.sh  8.sh  TEMP1.TXT
shawnlintonmiranda@HP:~/OS$ sh 7.sh temp1.txt
TEMP1.TXT is already exists.
shawnlintonmiranda@HP:~/OS$ sh 7.sh temp2.txt
File converted successfully.
```

```
shawnlintonmiranda@HP:~/OS$ sh 7.sh temp3.txt
temp3.txt does not exists.
shawnlintonmiranda@HP:~/OS$ ls
1a.sh  1.sh    2_2.sh  3.sh  5.sh  7.sh  temp1.txt  TEMP2.TXT
1.pl   2_1.sh  2.pl    4.sh  6.sh  8.sh  TEMP1.TXT
```

8. String length and searching a character in a string

Write a shell script to determine length of the string, extract substring and locate a position of character.

Program:

```
#!/bin/sh
echo "Enter a string : "
read str
```

```
if [ -z "$str" ]
then
        echo "Invalid string"
else
        echo "Length of string : "
        z=`expr "$str" : '.*'`
        echo $z
        if [ $z -ge 3 ]
        then
                echo "Substring is "
                z=`expr "$str" : '.*\(...\)'`
                echo $z
        else
                echo "String length is less than 3"
        fi
        echo "Enter the character to be searched : "
        read ch
        res=`expr "$str" : '[^'$ch']*'$ch`
        if [ $res -ne 0 ]
        then
                echo "Position of character $ch in a string is $res"
        else
                echo "Character $ch does not exists in the string."
        fi
fi
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 8.sh
shawnlintonmiranda@HP:~/OS$ sh 8.sh
Enter a string :
NMAMIT Nitte
Length of string :
12
Substring is
tte
Enter the character to be searched :
A
Position of character A in a string is 3
```

```
shawnlintonmiranda@HP:~/OS$ sh 8.sh
Enter a string :
AB
Length of string :
2
String lenght is less than 3
Enter the character to be searched :
C
Character C does not exists in the string.
```

9. Repeating string multiple times

Write PERL program the prompts user to input a string and a number and prints the string that many times with each string on a separate line.

Program:

```
#!/usr/bin/perl
print "Enter a string : ";
$a=<STDIN>;
print "Number of time string need to displayed : ";
chop($b=<STDIN>);
$c=$a x $b;
print "Result is : \n$c";
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 1.pl
shawnlintonmiranda@HP:~/OS$ perl 1.pl
Enter a string : NMAMIT
Number of time string need to displayed : 5
Result is :
NMAMIT
NMAMIT
NMAMIT
NMAMIT
NMAMIT
```

10. Sum of digits of numbers

Write a perl program to find the sum of digits of unsigned numbers passed as command line argument. Display output for each argument separately

Program:

```
#!/urs/bin/perl
foreach $num (@ARGV)
{
        $sum=0;
        $original_no = $num;
        until($num==0)
        {
                $digit=$num%10;
                $sum=$sum+$digit;
                $num=int($num/10);
        }
        print ("Sum of digits of $original_no is $sum.\n");
}
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 2.pl
shawnlintonmiranda@HP:~/OS$ perl 2.pl 1234
Sum of digits of 1234 is 10.
```

```
shawnlintonmiranda@HP:~/OS$ vi 2.pl
shawnlintonmiranda@HP:~/OS$ perl 2.pl 1234 5674
Sum of digits of 1234 is 10.
Sum of digits of 5674 is 22.
```

PART - B

1. Creation of five child processes

Write a C program to create five child processes using system call fork( ) and display their process IDs.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
   int pid,i;
   printf("The parent process id=%d\n",getpid());
   for(i=1; i<=5; i++)
   {
        int pid=fork();
        if(pid<0)
        {
            printf("Fork failed!");
            exit(0);
        }
        else if(pid==0)
        {
            printf("Child process id=%d\n",getpid());
            exit(0);
        }
        wait(NULL);
   }
   return 0;
}
```

Note:

The first program can be executed only in linux environment. Because fork() is a linux system call. Windows can't able to recognize it. Other program can be executed in any environments as well as in codeblocks IDE

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 1.c
shawnlintonmiranda@HP:~/OS$ cc 1.c
shawnlintonmiranda@HP:~/OS$ ./a.out
The parent process di=8669
Child process id=8670
Child process id=8671
Child process id=8672
Child process id=8673
Child process id=8674
```

2. FCFS Scheduling algorithm

Write a C program to implement First Come First Serve Scheduling algorithm to determine total waiting time, average waiting time, total turn around time and average turn around time.

Program:

```
#include <stdio.h>
struct process
{
      int pid,bt,wt,tt;
};
```

```
int main()
{
	int i,n,totwt=0,tottt=0;
	float avgwt,avgtt;
	printf("Enter the number of processes : ");
	scanf("%d",&n);
	struct process p[n]; //Allocate space
	for(i=0; i<n; i++)
	{
		p[i].pid=i+1;
		printf("Enter burst time for process-%d : ", (i+1));
		scanf("%d",&p[i].bt);
	}
	p[0].wt=0;	//WT for 1st process is 0
	p[0].tt=p[0].bt+p[0].wt;	//BT for 1st process is wt+its BT
	for(i=1; i<n; i++)	//Remaining processes 2 to n
	{
		p[i].wt=p[i-1].bt+p[i-1].wt;	//OR p[i].wt=p[i-1].tt;
		p[i].tt=p[i].wt+p[i].bt;
	}
	printf("\nFCFS Scheduling\nProcess ID\tBT\tWT\tTAT\n");
	for(i=0; i<n; i++)
	{
		printf("\n\t%d\t%d\t%d\t%d",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
		totwt=totwt+p[i].wt;
		tottt=tottt+p[i].tt;
	}
	avgwt=(float)totwt/n;
	avgtt=(float)tottt/n;
	printf("\n\nTotal Waiting Time : %dms\nAverage Waiting Time : %.2fms\n",totwt,avgwt);
	printf("\nTotal turn-around time : %dms\nAverage turn-around time : %.2fms\n",tottt,avgtt);
	return 0;
}
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 2.c
shawnlintonmiranda@HP:~/OS$ cc 2.c
shawnlintonmiranda@HP:~/OS$ ./a.out
Enter the number of processes : 3
Enter burst time for process-1 : 24
Enter burst time for process-2 : 3
Enter burst time for process-3 : 3

FCFS Scheduling
Process ID	BT	WT	TAT

	1	24	0	24
	2	3	24	27
	3	3	27	30

Total Waiting Time : 51ms
Average Waiting Time : 17.00ms

Total turn-around time : 81ms
Average turn-around time : 27.00ms
```

3. SJF Scheduling algorithm

Write a C program to implement Shortest Job First Scheduling algorithm to determine total waiting time, average waiting time, total turn around time and average turn around time.

Program:

```
#include <stdio.h>
struct process
{
    int pid,bt,wt,tt;
};
int main()
{
    int i,j,n,totwt=0,tottt=0;
    float avgwt,avgtt;
    printf("Enter the number of processes : ");
    scanf("%d",&n);
    struct process p[n],temp;
    for(i=0; i<n; i++)
    {
        p[i].pid=i+1;
        printf("Enter burst time for process-%d : ",i+1);
        scanf("%d",&p[i].bt);
    }
    int min;
    for(i=0; i<n-1; i++)  //Selection sort
    {
        for(j=i+1; j<n; j++)
        {
          min=i;
          if(p[j].bt<p[min].bt)
                min=j;
        }
        temp.pid=p[i].pid;                  OR
        temp.bt=p[i].bt;                    temp=p[i];
        p[i].pid=p[min].pid;                p[i]=p[min];
        p[i].bt=p[min].bt;                  p[min]=temp;
        p[min].pid=temp.pid;
        p[min].bt=temp.bt;
    }
    p[0].wt=0;  //Proceed same as FCFS after sorting
    p[0].tt=p[0].bt+p[0].wt;
    for(i=1; i<n; i++)
    {
          p[i].wt=p[i-1].bt+p[i-1].wt;          //OR p[i].wt=p[i-1].tt;
          p[i].tt=p[i].wt+p[i].bt;
    }
    printf("\nSJF Scheduling\nProcess ID\tBT\tWT\tTAT\n");
    for(i=0; i<n; i++)
    {
        printf("\n\t%d\t%d\t%d\t%d",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
        totwt=totwt+p[i].wt;
        tottt=tottt+p[i].tt;
    }
```

```
    avgwt=(float)totwt/n;
    avgtt=(float)tottt/n;
    printf("\n\nTotal Waiting Time : %dms\nAverage Waiting Time : %.2fms\n",totwt,avgwt);
    printf("\nTotal turn-around time : %dms\nAverage turn-around time : %.2fms\n",tottt,avgtt);
    return 0;
}
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 3.c
shawnlintonmiranda@HP:~/OS$ cc 3.c
shawnlintonmiranda@HP:~/OS$ ./a.out
Enter the number of processes : 4
Enter burst time for process-1 : 6
Enter burst time for process-2 : 8
Enter burst time for process-3 : 7
Enter burst time for process-4 : 3

SJF Scheduling
Process ID      BT      WT      TAT

        4       3       0       3
        1       6       3       9
        3       7       9       16
        2       8       16      24

Total Waiting Time : 28ms
Average Waiting Time : 7.00ms

Total turn-around time : 52ms
Average turn-around time : 13.00ms
```

4. Round Robin Scheduling algorithm

Write a C program to implement Round Robin Scheduling algorithm to determine total waiting time, average waiting time, total turn around time and average turn around time.

Program:

```
#include <stdio.h>
struct process  {
        int pid,bt,wt,tt;
};
int main()
{
        int i,n,tq,time=0,totwt=0,tottt=0,flag=0;
        float avgwt,avgtt;
        printf("Enter the number of processes : ");
        scanf("%d",&n);
        struct process p[n];
        int remBT[n];
        for(i=0; i<n; i++)
        {
            p[i].pid=i+1;
            printf("Enter burst time for process-%d : ",i+1);
            scanf("%d",&p[i].bt);
            remBT[i]=p[i].bt;
        }
        printf("\nEnter the time quantum : ");
```

```
        scanf("%d",&tq);
        while(flag==0)
        {
            flag=1; //Assume all completed execution
            for(i=0; i<n; i++){
                if(remBT[i]>0)
                {
                    flag=0; // Some processes are remaining
                    if(remBT[i]>tq)  {
                        time=time+tq;
                        remBT[i]=remBT[i]-tq;
                    }
                    else  {
                        time=time+remBT[i];
                        remBT[i]=0;
                        p[i].tt=time;
                        p[i].wt=p[i].tt-p[i].bt;
                    }
                }
            }
        }
        printf("\nRound Robin Scheduling\nProcess ID\tBT\tWT\tTAT\n");
        for(i=0; i<n; i++)   {
            printf("\n\t%d\t%d\t%d\t%d",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
            totwt=totwt+p[i].wt;
            tottt=tottt+p[i].tt;
        }
        avgwt=(float)totwt/n;
        avgtt=(float)tottt/n;
        printf("\n\nTotal Waiting Time : %dms\nAverage Waiting Time : %.2fms\n",totwt,avgwt);
        printf("\nTotal turn-around time : %dms\nAverage turn-around time : %.2fms\n",tottt,avgtt);
        return 0;
}
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 4.c
shawnlintonmiranda@HP:~/OS$ cc 4.c
shawnlintonmiranda@HP:~/OS$ ./a.out
Enter the number of processes : 4
Enter burst time for process-1 : 3
Enter burst time for process-2 : 6
Enter burst time for process-3 : 4
Enter burst time for process-4 : 2

Enter the time quantum : 2

Round Robin Scheduling
Process ID      BT      WT      TAT

        1       3       6       9
        2       6       9       15
        3       4       9       13
        4       2       6       8

Total Waiting Time : 30ms
Average Waiting Time : 7.50ms

Total turn-around time : 45ms
Average turn-around time : 11.25ms
```

5. Producer Consumer problem

Write a C program to simulate producer – consumer problem using semaphores.

Program:

```
#include<stdio.h>
int main()
{
	int bufferSize,in=0,out=0,items=0,produce,consume,choice=0;
	printf("Enter the buffer size : ");
	scanf("%d",&bufferSize);
	int buffer[bufferSize];
	do  {
		printf("\n1.Proceduce\t2.Consume\t3.Exit\n");
		printf("Enter your choice : ");
		scanf("%d",&choice);
		switch(choice)
		{
			case 1:if(items==bufferSize)
				printf("\nBuffer is full!");
			else  {
				printf("Enter the item value : ");
				scanf("%d",&produce);
				buffer[in]=produce;
				in=(in+1)%bufferSize;
				items++;
			}
			break;
			case 2:if(items==0)
				printf("\nBuffer is empty!");
			else  {
				consume=buffer[out];
				printf("\nThe consumed item value is %d.",consume);
				out=(out+1)%bufferSize;
				items--;
			}
			break;
		}
	} while(choice!=3);
	return 0;
}
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 5.c
shawnlintonmiranda@HP:~/OS$ cc 5.c
shawnlintonmiranda@HP:~/OS$ ./a.out
Enter the buffer size : 2

1.Proceduce     2.Consume       3.Exit
Enter your choice : 1
Enter the item value : 10

1.Proceduce     2.Consume       3.Exit
Enter your choice : 1
Enter the item value : 20

1.Proceduce     2.Consume       3.Exit
Enter your choice : 1

Buffer is full!
```

```
1.Proceduce     2.Consume       3.Exit
Enter your choice : 2

The consumed item value is 10.
1.Proceduce     2.Consume       3.Exit
Enter your choice : 2

The consumed item value is 20.
1.Proceduce     2.Consume       3.Exit
Enter your choice : 2

Buffer is empty!
1.Proceduce     2.Consume       3.Exit
Enter your choice : 3
```

6. FIFO Page Replacement algorithm

Write a C program to demonstrate First In First Out page replacement algorithm to determine number of page faults.

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
	int n,f,i,j,found,victim=-1,pf=0;
	printf("Enter the number of pages : ");
	scanf("%d",&n);
	int ref[n];
	printf("Enter number of frames : ");
	scanf("%d",&f);
	int frame[f];
	for(j=0; j<f; j++)
		frame[j]=-1;
	printf("Enter the reference string : ");
	for(i=0; i<n; i++)
		scanf("%d",&ref[i]);

	printf("\nFIFO Page Replacement\nReference String is \n");
	for(i=0; i<n; i++)
		printf("%d  ",ref[i]);
	printf("\n\n");
	for(i=0; i<n; i++)
	{
		found=0;    //Assume page is not found
		for(j=0; j<f; j++)    //Search for page
			if(frame[j]==ref[i])
			{
				found=1;    //Page found
				break;
			}
		if(!found)    //Or use found==0
		{
			pf++;
			victim=(victim+1)%f;
			frame[victim]=ref[i];
		}
		printf("%d -> ",ref[i]);
		for(j=0; j<f; j++)
			printf(" %d  ",frame[j]);
		if(found==0) printf("\tPage miss\n");
		else printf("\tPage hit\n");
	}
	printf("\nTotal number of page faults : %d\n",pf);
	return 0;
}
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 6.c
shawnlintonmiranda@HP:~/OS$ cc 6.c
shawnlintonmiranda@HP:~/OS$ ./a.out
Enter the number of pages : 12
Enter number of frames : 4
Enter the reference string : 1 2 3 4 1 2 5 1 2 3 4 5

FIFO Page Replacement
Reference String is
1  2  3  4  1  2  5  1  2  3  4  5

1 ->  1  -1  -1  -1     Page miss
2 ->  1  2  -1  -1      Page miss
3 ->  1  2  3  -1       Page miss
4 ->  1  2  3  4        Page miss
1 ->  1  2  3  4        Page hit
2 ->  1  2  3  4        Page hit
5 ->  5  2  3  4        Page miss
1 ->  5  1  3  4        Page miss
2 ->  5  1  2  4        Page miss
3 ->  5  1  2  3        Page miss
4 ->  4  1  2  3        Page miss
5 ->  4  5  2  3        Page miss

Total number of page faults : 10
```

7. LRU Page Replacement algorithm

Write a C program to demonstrate Least Recently Used page replacement algorithm to determine number of page faults.

Program:

```
#include<stdio.h>
void main()  {
      int n,f,i,j,free,min,index,found,pos=0,time=0,pf=0;
      printf("Enter the number of pages : ");
      scanf("%d",&n);
      int ref[n];
      printf("Enter number of frames : ");
      scanf("%d",&f);
      free=f;
      int frame[f],count[f];
      for(j=0; j<f; j++)  {
            frame[j]=-1;
            count[j]=0;
      }
      printf("Enter the reference string : ");
      for(i=0; i<n; i++)
            scanf("%d",&ref[i]);
      for(i=0;i<n;i++)  {
                  found=0;     //Assume page not found
                  for(j=0;j<f;j++)   {
                              if(frame[j]==ref[i])
                              {
                                          count[j]=time;
                                          found=1;    //page found
                                          break;
                              }
                  }
```

```
            if(!found)   {
                    if(free)   {
                            frame[pos]=ref[i];
                            pf++;
                            free--;
                            count[pos]=time;
                            pos++;
                    }
                    else  {
                            min=count[0];
                            index=0;    //Assume 1st frame is LRU
                            for(j=1;j<f;j++)     //Find other min
                                    if(count[j]<min)   {
                                            min=count[j];
                                            index=j;
                                    }
                            frame[index]=ref[i];
                            count[index]=time;
                            pf++;
                    }
            }
            time++;
            printf("%d -> ",ref[i]);
            for(j=0; j<f; j++)
                     printf(" %d  ",frame[j]);
            if(found==0) printf("\tPage miss\n");
            else printf("\tPage hit\n");
        }
        printf("\nNo of page faults:%d\n",pf);
}
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 7.c
shawnlintonmiranda@HP:~/OS$ cc 7.c
shawnlintonmiranda@HP:~/OS$ ./a.out
Enter the number of pages : 20
Enter number of frames : 3
Enter the reference string : 1 2 3 5 2 3 5 7 2 1 2 3 8 6 4 3 2 2 3 6
1 ->  1   -1   -1       Page miss
2 ->  1   2   -1        Page miss
3 ->  1   2   3         Page miss
5 ->  5   2   3         Page miss
2 ->  5   2   3         Page hit
3 ->  5   2   3         Page hit
5 ->  5   2   3         Page hit
7 ->  5   7   3         Page miss
2 ->  5   7   2         Page miss
1 ->  1   7   2         Page miss
2 ->  1   7   2         Page hit
3 ->  1   3   2         Page miss
8 ->  8   3   2         Page miss
6 ->  8   3   6         Page miss
4 ->  8   4   6         Page miss
3 ->  3   4   6         Page miss
2 ->  3   4   2         Page miss
2 ->  3   4   2         Page hit
3 ->  3   4   2         Page hit
6 ->  3   6   2         Page miss

No of page faults:14
```

8. Optimal Page Replacement algorithm

Write a C program to demonstrate Optimal page replacement algorithm to determine number of page faults.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int n,f;
int optimal(int ref[],int frame[],int optindex[],int index)
{
        int pos,found,max,i,j;
        for(i=0;i<f;i++)
        {
                found=0;     //Assume page not found
                for(j=index+1;j<n;j++)
                {
                        if(frame[i]==ref[j])
                        {
                                found=1;    //Page found
                                optindex[i]=j;
                                break;
                        }
                }
                if(!found)
                        return i;
        }
        max=optindex[0];
        pos=0;
        for(i=1;i<f;i++)
        {
                if(max<optindex[i])
                {
                        max=optindex[i];
                        pos=i;
                }
        }
        return pos;
}
int main()
{
      int i,j,found,count=0,pf=0,victim=-1;
      printf("Enter the number of pages : ");
      scanf("%d",&n);
      int ref[n];
      printf("Enter number of frames : ");
      scanf("%d",&f);
      int frame[f],optindex[f];
      for(j=0; j<f; j++)   {
                frame[j]=-1;
                optindex[j]=-1;
      }
      printf("Enter the reference string : ");
      for(i=0; i<n; i++)
            scanf("%d",&ref[i]);
```

```
        for(i=0;i<n;i++)
        {
                found=0;  //Assume page not found.
                for(j=0;j<f;j++)
                {
                        if(ref[i]==frame[j])  {
                                found=1; //Page found
                                break;
                        }
                }
                if(!found)
                {
                        count++;
                        if(count<=f)
                                victim++;
                        else
                                victim=optimal(ref,frame,optindex,i);
                        pf++;
                        frame[victim]=ref[i];
                }
                printf("%d -> ",ref[i]);
           for(j=0; j<f; j++)
                 printf(" %d  ",frame[j]);
           if(found==0) printf("\tPage miss\n");
           else printf("\tPage hit\n");
        }
        printf("\nNumber of page fault:%d\n",pf);
}
```

Output:

```
Number of page fault:10shawnlintonmiranda@HP:~/OS$ cc 8.c
shawnlintonmiranda@HP:~/OS$ ./a.out
Enter the number of pages : 20
Enter number of frames : 3
Enter the reference string : 1 2 3 5 2 3 5 7 2 1 2 3 8 6 4 3 2 2 3 6
1 ->  1   -1   -1       Page miss
2 ->  1   2   -1        Page miss
3 ->  1   2   3         Page miss
5 ->  5   2   3         Page miss
2 ->  5   2   3         Page hit
3 ->  5   2   3         Page hit
5 ->  5   2   3         Page hit
7 ->  7   2   3         Page miss
2 ->  7   2   3         Page hit
1 ->  1   2   3         Page miss
2 ->  1   2   3         Page hit
3 ->  1   2   3         Page hit
8 ->  8   2   3         Page miss
6 ->  6   2   3         Page miss
4 ->  4   2   3         Page miss
3 ->  4   2   3         Page hit
2 ->  4   2   3         Page hit
2 ->  4   2   3         Page hit
3 ->  4   2   3         Page hit
6 ->  6   2   3         Page miss

Number of page fault:10
```

9. Bankers Algorithm for deadlock avoidance

Write a C program to demonstrate Bankers Deadlock avoidance algorithm.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int p,r,i,j,k,found=0,count=0;
    printf("Enter number of processes:\n");
    scanf("%d",&p);
    printf("Enter number of resources:\n");
    scanf("%d",&r);
    int alloc[p][r],max[p][r],avail[r],need[p][r],safeseq[p],finish[p];
    printf("\nEnter the allocation matrix:\n");
    for(i=0;i<p;i++)
        for(j=0;j<r;j++)
            scanf("%d",&alloc[i][j]);
    printf("\nEnter the max matrix:\n");
    for(i=0;i<p;i++)
        for(j=0;j<r;j++)
            scanf("%d",&max[i][j]);
    printf("\nEnter the availble matrix:\n");
    for(i=0;i<r;i++)
        scanf("%d",&avail[i]);
    for(i=0;i<p;i++)
        for(j=0;j<r;j++)
            need[i][j]=max[i][j]-alloc[i][j];
    printf("\nNeed Matrix :\n");
    for(i=0;i<p;i++)
    {
        for(j=0;j<r;j++)
            printf("%d\t",need[i][j]);
        printf("\n");
    }
    for(i=0;i<p;i++)
        finish[i]=0;
    while(count<p)
    {
        found=0; //Assume !(need<=available) for all processes
        for(i=0;i<p;i++)
        {
            if(finish[i]==0) //Didn't completed
            {
                for(j=0;j<r;j++)
                {
                    if(need[i][j]>avail[j])
                        break;
                }
                if(j==r) //need<available
                {
                    for(k=0;k<r;k++)
                        avail[k]=avail[k]+alloc[i][k];
                    safeseq[count++]=i;
                    found=1;  //need<available found
                    finish[i]=1; //Mark process as finished
                }
            }
```

```
            }
            if(found==0) //No process can be allocated.
            {
                printf("System is in unsafe state.\n");
                return 0;
            }
        }
        printf("System is in safe state.\n");
        printf("Safe Sequence is : ");
        for(k=0;k<p;k++)
            printf("P%d\t",safeseq[k]+1);
        printf("\n");
        return 0;
    }
```

Output:

```
shawnlintonmiranda@HP:~/OS$ vi 9.c
shawnlintonmiranda@HP:~/OS$ cc 9.c
shawnlintonmiranda@HP:~/OS$ ./a.out
Enter number of processes:
5
Enter number of resources:
4

Enter the allocation matrix:
0 0 1 2
1 0 0 0
1 3 5 4
0 6 3 2
0 0 1 4

Enter the max matrix:
0 0 1 2
1 7 5 0
2 3 5 6
0 6 5 2
0 6 5 6

Enter the availble matrix:
1 5 2 0

Need Matrix :
0       0       0       0
0       7       5       0
1       0       0       2
0       0       2       0
0       6       4       2
System is in safe state.
Safe Sequence is : P1   P3      P4      P5      P2
```

```
shawnlintonmiranda@HP:~/OS$ cc 9.c
shawnlintonmiranda@HP:~/OS$ ./a.out
Enter number of processes:
2
Enter number of resources:
2

Enter the allocation matrix:
0 1
1 0

Enter the max matrix:
3 4
4 5

Enter the availble matrix:
1 1

Need Matrix :
3       3
3       5
System is in unsafe state.
```

***