ASSIGNMENT NO: 01

TITLE:

Study of Unix/Linux general purpose utility command list obtained from (man, who,cat, cd, cp, ps, ls, mv, srm, mkdir, rmdir, echo, more, date, time, kill, history, chmod,chown, finger, pwd, cal, logout, shutdown) commands.

AIM :

To Study basic linux commands.

THEORY :

1. **man :**

   The man command itself is extremely easy to use. Its basic syntax is

   man [option(s)] keyword(s)

man is most commonly used without any options and with only one keyword. The keyword is the exact name of the command or other item for which information is desired. For example, the following provides information about the *ls* command (which is used to list the contents of any specified directory):

2. **who**

   The **who** command prints information about all users who are currently logged in.

   who Syntax

   who [OPTION]... [FILE] [am i]

3. **cat :**

   Sends file contents to standard output. This is a way to list the contents of short files to the screen. It works well with piping.
   cat .bashrc : Sends the contents of the ".bashrc" file to the screen

4. **cd :**

5. Type **cd** followed by the name of a directory to access that directory. Keep in mind that you are always in a directory and allowed access to any directories hierarchically above or below. You may also benefit from reviewing my directory commandspage

   .Ex:
   cd games
   If the directory games is not located hierarchically below the current directory, then

the complete path must be written out. Ex:
cd /usr/games
To move up one directory, use the shortcut command. Ex:
cd ..

6. **cp**
Creates a copy of the file in the working directory named origfile. The copy will be named newfile, and will be located in the working directory.

cp origfile newfile

cp -i oldfile newfile

7. **ps**
Display all processes
The following command will give a full list of processes
$ ps ax
$ ps –ef

8. **ls :**
List the files
Type **ls** to see a list of the files and directories located in the current directory. If you're in the directory named *games* and you type **ls**, a list will appear that contains files in the games directory and sub-directories in the games directory. Examples lsMail
ls/usr/bin
Type ls -alt to see a list of all files (including .rc files) and all directories located in the current directory. The listing will include detailed, often useful information. Examples:
ls-alt
ls-alt/usr/bin
If the screen flies by and you miss seeing a number of files, try using the |more at the end like:
ls -alt |more

9. **mv :**
mv -i myfile yourfile :Move the file from "myfile" to "yourfile". This effectively changes the name of "myfile" to "yourfile

10. **rm**
rm fillename :Remove file from directory.

11. **Mkdir :**
To create directory
Mkdir dir name

12. **Rmdir :**
To remove directory from system
rmdir directory name

13. **Echo :**

It takes input
Example –
Echo Enter name

14. More :

more is a filter for paging through text one screen at a time. It does not provide as many options or enhancements as less, but is nevertheless quite useful and simple to use.

more [-dlfpcsu] [-num lines] [+/pattern] [+linenum] [file ...]

15. Date :

Type **date** followed by the two digit month, the two digit date, the two digit time, and two digit minutes. The syntax is easy enough and resembles this: MMDDhhmm This command is helpful but must be used when superuser or logged in as root.

You can get more help with administrator commands by clicking this link. Note that if you don't use root, you will get an "Operation not permitted" reply.As root user you can use the command such as:
date 11081300
The above command will set the server date and time to the 11th month (November), the 8th day, at 1:00pm.

16. Time

It will give current system time
Date
It will give today's date

17. Kill

kill command is used on Linux and other Unix-like operating systems to terminate processes without having to log out or reboot (i.e., restart) the computer. Thus, it is particularly important to the stability of such systems.

18. History

The history command

The *history* command can be used to list Bash's log of the commands you have typed:

This log is called the "history". To access it type:

history n

This will only list the last *n* commands. Type "history" (without options) to see the entire history list

19. Chmod

This command will do the trick:
chmod u=rwx,g=rx,o=r myfile

This example uses symbolic permissions notation. The letters u, g, and o stand for "user", "group", and "other". The equals sign ("=") means "set the permissions exactly like this," and the letters "r", "w", and "x" stand for "read", "write", and "execute", respectively. The commas separate the different classes of permissions, and there are no spaces in between them.

20. Chown

chown changes the user or group ownership of each given file. If only an owner (a user name or numeric user ID) is given, that user is made the owner of each given file, and the files' group is not changed. If the owner is followed by a colon and a group name (or numeric group ID), with no spaces between them, the group ownership of the files is changed as well. If a colon but no group name follows the user name, that user is made the owner of the files and the group of the files is changed to that user's login group. If the colon and group are given, but the owner is omitted, only the group of the files is changed; in this case, chown performs the same function as chgrp. If only a colon is given, or if the entire operand is empty, neither the owner nor the group is changed.

chown syntax

chown [OPTION]... [OWNER][:[GROUP]] FILE...

21. Finger :

Typing **finger** allows you to see who else is on the system or get detailed information about a person who has access to the system.
Type finger followed by the name of a user's account to get information about that user. Or, type finger and press enter to see who's on the system and what they are doing. Ex: finger johndoe

22. Pwd :

Type **passwd** and press enter. You'll see the message *Changing password* for *yourname*.
At the Old password: prompt, type in your old password .
Then, at the Enter new password: prompt, type in your new password .
The system double checks your new password. Beside the Verify: prompt, type the new password and press again.

Create a secure password that combines parts of words and numbers. For instance, your dog's name may be Rufus. He may have been born in 1980. Create a password that uses **parts** of both the name and date of birth, such as 80rufuS. Note the use of at least one capital letter. This is a fairly secure password and easy to remember.

23. Cal :

Gives calender

24. Logout:

Type logout at the prompt to disconnect from your Linux machine or to logout a particular user session from the system. Keep in mind that although rudimentary, leaving your critical account logged on may be a security concern. We always recommend promptly using logout when you are finished using your root account! Ex: logout

25. Shutdown :

Shuts the system down.
shutdown -h now : Shuts the system down to halt immediately
shutdown -r now : Shuts the system down immediately and the system reboots

ASSIGNMENT NO: 02

TITLE:

Write a program in shell for printing table of any number.

Algorithm:

1. Input the number for which the multiplication table is to be generated.
2. Input the end value until which the table has to be generated.
3. Repeat from i = 1 to end.
4. Display the table values in the given output format.(num * i = num*i)

Recursive approach to print multiplication table of a number

Approach:

Get the number for which multiplication table is to print.

Recursively iterate from value 1 to 10:

Base case: If the value called recursively is greater than 10, exit from the function.

```
if(i > N)
  return ;
```

Recursive call: If the base case is not met, then print its multiplication table for that value and then call the function for next iteration.

```
print("N*i = ", N*i)
recursive_function(N, i+1);
```

Return statement: At each recursive call(except the base case), return the recursive function for next iteration.

```
return recursive_function(N, i+1);
```

Below is the implementation of the above approach:

Program:

```
echo "Enter a Number"
read n
i=1

while [ $i -le 10 ]
do
        echo " $n x $i = $(( n * i ))"
        i=$(( i + 1 ))
done
```

Output:

Enter a Number

3

3

6

9

12

15

18

21

24

27

30

ASSIGNMENT NO: 03

TITLE:

Implement program in shell script:

a) To find factorial of given number
b) To find greatest of three number

Theory:

Here we are going to see to calculate the factorial of a number. Factorial of a non-negative integer is the multiplication of all integers smaller than or equal to n.

For example factorial of 5 is 5*4*3*2*1 which is 120.

Method 1: Using Recursive

Factorial can be calculated using the following recursive formula.

n!=n*(n-1)!

n!=1 if n=0 or n=1

Below is implementation of factorial:

```
#!/bin/bash

# Recursive factorial function


factorial()

{

   product=$1


   # Defining a function to calculate factorial using recursion

   if((product <= 2)); then

      echo $product

   else
```

```
    f=$((product -1))

# Recursive call

f=$(factorial $f)
f=$((f*product))
echo $f
fi
}

# main program
# reading the input from user
echo "Enter the number:"
read num

# defining a special case for 0! = 1
if((num == 0)); then
echo 1
else
#calling the function
factorial $num
fi
```

Output:

```
Enter the number
5
120
```

Enter the number

3

24

Enter the number

6

720

Method 2: Using for loop

Approach:

Get a number

Use for loop to compute the factorial by using the below formula

fact(n) = n * n-1 * n-2 * …

Display the result.

Below is the Implementation using for loop:

```
# shell script for factorial of a number
# factorial using for loop

echo "Enter a number"

# Read the number
read num

fact=1
```

```
for((i=2;i<=num;i++))
{
  fact=$((fact * i))
}

echo $fact
```

Output:

```
Enter a number
5
120

Enter a number
7
5040

Enter a number
4
24
```

Method 3: using do-while loop

Get a number

Use do-while loop to compute the factorial by using the below formula

fact(n) = n * n-1 * n-2 * .. 1

Display the result.

Below is the Implementation using a while loop.

```
# shell script for factorial of a number
# factorial using while loop

echo "Enter a number"
# Read the number
read num
fact=1
# -gt is used for '>' Greater than sign
while [ $num -gt 1 ]
do
  fact=$((fact * num))
  num=$((num - 1))
done
# Printing the value of the factorial
echo $fact
```

Output:

```
Enter a number
10
3628800
Enter a number
2
2
Enter a number
9
362880
```

To find greatest of three number :

ALGORITHM:

Below algorithm shows steps of how to find greatest of three numbers in linux:

STEP 1: START THE PROBLEM

STEP 2: TAKE THREE INPUTS FROM THE USER

STEP 3: IN IF-ELSE CONDITION, CHECK WHICH IS THE GREATEST

STEP 4: ALSO CHECK WITH THE THIRD NUMBER

STEP 5: FIND THE RESULT

STEP 6: PRNT THE RESULT

STEP 7: STOP THE PROGRAM

Program :

```
#!/bin/sh
echo "Enter value of 'a':"
read a
echo "Enter value of 'b':"
read b
echo "Enter value of 'c':"
read c
if [ $a -gt $b ]
then
   if [ $a -gt $c ]
   then
    echo "a is greatest"
   else
```

```
    echo "c is greatest"
  fi
else
  if [ $b -gt $c ]
  then
  echo "b is greatest"
  else
  echo "c is greatest"
  fi
fi
```

Output 1

```
$ ./biggest-of-three-nested-if.sh
Enter value of 'a':
6
Enter value of 'b':
3
Enter value of 'c':
2
a is greatest
```

Output 2

```
$ ./biggest-of-three-nested-if.sh
Enter value of 'a':
5
Enter value of 'b':
6
Enter value of 'c':
```

3

b is greatest

Output 3

$ ./biggest-of-three-nested-if.sh

Enter value of 'a':

3

Enter value of 'b':

5

Enter value of 'c':

9

c is greatest

ASSIGNMENT NO: 04

TITLE:

Implement program in shell script :
a) To print given number in reverse order
b) To find even and odd numbers from given array

Program:

To print given number in reverse order

```
read -p "Enter a number: " number

temp=$number

while [ $temp -ne 0 ]

do

   reverse=$reverse$((temp%10))

   temp=$((temp/10))

done

echo "Reverse of $number is $reverse"
```

Output

Enter a number: 123

Reverse of 123 is 321

To find even and odd numbers from given array:

Program:

```
clear
echo "---- EVEN OR ODD IN SHELL SCRIPT -----"
echo -n "Enter a number:"
read n
echo -n "RESULT: "
if [ `expr $n % 2` == 0 ]
then
```

```
        echo "$n is even"
else
        echo "$n is Odd"
fi
```

Output

Enter a number: 12

RESULT: 12 is even

ASSIGNMENT NO: 05

TITLE:

Write a menu driven shell script program to develop a scientific calculator

AIM :

To write a shell script program to develop a scientific calculator

THEORY :

Here two numbers are taken as input

Following operations are performed on it:

1.addition

2.substraction

3.division

4.multiplication

Let n2,n2 are the numbers all is stored in sum variable

Sum=n1+n2

Subtraction=n1-n2

Division=n1/n2

Multiplication=n1*n2

The basic syntax of the **case...esac** statement is to give an expression to evaluate and to execute several different statements based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

```
case word in
  pattern1)
    Statement(s) to be executed if pattern1 matches
    ;;
```

```
  pattern2)
    Statement(s) to be executed if pattern2 matches
    ;;
  pattern3)
    Statement(s) to be executed if pattern3 matches
    ;;
esac
```

Here the string word is compared against every pattern until a match is found. The statement(s) following the matching pattern executes. If no matches are found, the case statement exits without performing any action.

There is no maximum number of patterns, but the minimum is one.

When statement(s) part executes, the command ;; indicates that the program flow should jump to the end of the entire case statement. This is similar to break in the C programming language.

Example

```
#!/bin/sh
FRUIT="kiwi"
case "$FRUIT" in
  "apple") echo "Apple pie is quite tasty."
  ;;
  "banana") echo "I like banana nut bread."
  ;;
  "kiwi") echo "New Zealand is famous for kiwi."
  ;;
esac
```

PROGRAM:

```
clear
sum=0
i="y"
```

```
echo " Enter one no."
read n1
echo "Enter second no."
read n2
while [ $i = "y" ]
do
echo "1.Addition"
echo "2.Subtraction"
echo "3.Multiplication"
echo "4.Division"
echo "Enter your choice"
read ch
case $ch in
    1)sum=`expr $n1 + $n2`
     echo "Sum ="$sum;;
        2)sum=`expr $n1 - $n2`
     echo "Sub = "$sum;;
    3)sum=`expr $n1 \* $n2`
     echo "Mul = "$sum;;
    4)sum=`expr $n1 / $n2`
     echo "Div = "$sum;;
    *)echo "Invalid choice";;
esac
echo "Do u want to continue ?"
read i
if [ $i != "y" ]
then
```

```
   exit
fi
done
```

SAMPLE OUTPUT:

```
04mca58@LINTEL 04mca58]$ sh calculator.sh
Enter any no.
121
 Enter one no.
 21
 Enter second no.
 58
 1.Addition
 2.Subtraction
 3.Multiplication
 4.Division
 Enter your choice
 1
 Sum =79
 Do u want to continue ?
 y
 1.Addition
 2.Subtraction
 3.Multiplication
 4.Division
 Enter your choice
 2
```

Sub = -37

Do u want to continue ?

y

1.Addition

2.Subtraction

3.Multiplication

4.Division

Enter your choice

3

Mul = 1218

Do u want to continue ?

y

1.Addition

2.Subtraction

3.Multiplication

4.Division

Enter your choice

4

Div = 0

Do u want to continue ?

n

ASSIGNMENT NO: 06

TITLE:

Write a menu driven program for a) Find factorial of a no. b) Find greatest of three numbers c) print number in reverse order

AIM :

To write a menu driven program for a) Find factorial of a no. b) Find greatest of three numbers c) print number in reverse order

THEORY :

As in other programming languages we can't live without variables. In shell programming all variables have the data type string and you do not need to declare them. To assign a value to a variable you write:

varname=value
To get the value back you just put a dollar sign in front of the variable:

```
#!/bin/sh
# assign a value:
a="hello\ world"
# now print the content of "a":
echo "A is:"
echo $a
```

Type this lines into your text editor and save it e.g. as first. Then make the script executable by typing chmod +x first in the shell and then start it by typing ./first
The script will just print:

```
A is:
hello world
```

Sometimes it is possible to confuse variable names with the rest of the text:

```
num=2
echo "this is the $numnd"
```

This will not print "this is the 2nd" but "this is the " because the shell searches for a variable called numnd which has no value. To tell the shell that we mean the variable num we have to use curly braces:

```
num=2
echo "this is the ${num}nd"
```

This prints what you want: this is the 2nd

There are a number of variables that are always automatically set. We will discuss them further down when we use them the first time.

If you need to handle mathematical expressions then you need to use programs such as expr (see table below).
Besides the normal shell variables that are only valid within the shell program there are also environment variables. A variable preceded by the keyword export is an environment variable. We will not talk about them here any further since they are normally only used in login scripts.

1. Shell commands and control structures

There are three categories of commands which can be used in shell scripts:

Unix commands:

Although a shell script can make use of any unix commands here are a number of commands which are more often used than others. These commands can generally be described as commands for file and text manipulation.

Command syntax	Purpose
echo "some text"	write some text on your screen

PROGRAM AND OUTPUT :

```
if [ "$#" -eq  "0" ]
then
   echo ""
        echo "Enter atleast 5 argument"
        echo "Ex,    sh 2.sh vicky TechiCraze 1 2 3"
   echo ""
   exit
fi

echo ""
echo "Argument passed "
echo "$1 $2 $3 $4 $5"
echo ""
```

```
echo "Argument passed in reversed order"
echo "$5 $4 $3 $2 $1"
echo ""
```

OUTPUT:

```
# techicraze@TechiCraze:~/shell script$ sh 2.sh

# Enter atleast 5 argument
# Ex,    sh 3.sh vicky TechiCraze 1 2 3

# techicraze@TechiCraze:~/shell script$ sh 2.sh vicky TechiCraze 1  2 3

# Argument passed
# vicky TechiCraze 1 2 3

# Argument passed in reversed order
# 3 2 1 TechiCraze vicky

techicraze@TechiCraze:~/shell script$
```

Program 2

```
/* Shell Programming Write a menu driven program for option given below.
1. Factorial 2. Gratest of 3 number 3. Reverse of a number 4. Exit*/
```

```
ch=0
while [ $ch -ne 4 ]
do
        echo "menu"
        echo "1.factorial"
        echo "2.greatest of 3 NoS"
        echo "3.reverse of a Number"
        echo "4.exit"
        echo "Enter your choice : "
        read ch
        case $ch in
                1)echo "Enter the No : "
                read num
                i=1
                fact=1
                while [ $i -le $num ]
                do
                        fact=`expr $fact \* $i`
                        i=`expr $i + 1`
                done
                echo "Factorial of $num is : $fact"
                ;;
              2)echo "Enter 3 NoS : "
                read n1
                read n2
                read n3
                if [ $n1 -gt $n2 -a $n1 -gt $n3 ]
```

```
        then
            echo "$n1 is the greatest No"
        else if [ $n2 -gt $n3 -a $n2 -gt $n1 ]
        then
            echo "$n2 is the greatest No"
        else
            echo "$n3 is the greatest No"
        fi
        fi;;
        3)echo "enter number"
            read num
            n=$num
            rev=0
            while [ $num -gt 0 ]
            do
                s=` expr $num % 10 `
                rev=` expr $rev \* 10 + $s `
                num=` expr $num / 10 `
            done
            echo "rev = $rev"
            4)exit 0;;
    esac
done
```

OUTPUT:

```
[Nhg@NHG ~]$ sh 1b.sh
menu
1.factorial
```

2. greatest of 3 NoS

3. reverse of a Number

4. exit

Enter your choice :

1

Enter the No :

6

Factorial of 6 is : 720

ASSIGNMENT NO: 07

TITLE:

Write a program for creating child process by fork () command

AIM :

To write a program to create a process in UNIX

THEORY :

STEP 1: Start the program.

STEP 2: Declare pid as integer.

STEP 3: Create the process using Fork command.

STEP 4: Check pid is less than 0 then print error else if pid is equal to 0 then execute command

Else parent process waits for child process.

STEP 5: Stop the program.

FORK:

fork() creates a child process that differs from the parent process only in its PID and PPID, and in the fact that resource utilizations are set to 0. File locks and pending signals are not inherited.

Under Linux, **fork**() is implemented using copy-on-write pages, so the only penalty that it incurs is the time and memory required to duplicate the parent's page tables, and to create a unique task structure for the child.

RETURN VALUE

On success, the PID of the child process is returned in the parent's thread of execution, and a 0 is returned in the child's thread of execution. On failure, a -1 will be returned in the parent's context, no child process will be created, and *errno* will be set appropriately.

ERRORS

Error Code	Description
EAGAIN	**fork**() cannot allocate sufficient memory to copy the parent's page tables and allocate a task structure for the child.

EAGAIN	It was not possible to create a new process because the caller's **RLIMIT_NPROC** resource limit was encountered. To exceed this limit, the process must have either the **CAP_SYS_ADMIN** or the**CAP_SYS_RESOURCE** capability.
ENOMEM	**fork**() failed to allocate the necessary kernel structures because memory is tight.

PROGRAM:

```
Void main()
{
   int pid;
   pid=fork();
   if(pid<0)
   {
      printf("\n Error ");
      exit(1);
   }
   else if(pid==0)
   {
      printf("\n Hello I am the child process ");
      printf("\n My pid is %d ",getpid());
      exit(0);
   }
   else
   {
      printf("\n Hello I am the parent process ");
      printf("\n My actual pid is %d \n ",getpid());
      exit(1);
```

```
    }
}
```

SAMPLE OUTPUT:

```
$cc pc.c
$/a.out
Hello I am the parent process
My actual pid is 4287
$ps
 PID      CLS    PRI     TTY        TIME      COMD
4287    TS        70      pts0      22 0:00    ksh
```

ASSIGNMENT NO: 08

TITLE:

To implement BANKER'S algorithm for deadlock avoidance.

AIM :

To implement BANKER'S algorithm.

THEORY :

BANKER'S Algorithm :

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Available :
It is a 1-d array of size 'm' indicating the number of available resources of each type.

Available[j] = k means there are 'k' instances of resource type Rj

Max :
It is a 2-d array of size 'n*m' that defines the maximum demand of each process in a system.

Max[i, j] = k means process Pi may request at most 'k' instances of resource type Rj.

Allocation :
It is a 2-d array of size 'n*m' that defines the number of resources of each type currently allocated to each process.

Allocation[i, j] = k means process Pi is currently allocated 'k' instances of resource type Rj

Need :
It is a 2-d array of size 'n*m' that indicates the remaining resource need of each process.

Need [i, j] = k means process Pi currently need 'k' instances of resource type Rj

Need [i, j] = Max [i, j] – Allocation [i, j]

Allocation specifies the resources currently allocated to process Pi and Needi specifies the additional resources that process Pi may still request to complete its task. Banker's algorithm consists of Safety algorithm and Resource request algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.
   Initialize: Work = Available
   Finish[i] = false; for i=1, 2, 3, 4….n
   2) Find an i such that both
   a) Finish[i] = false
   b) Needi <= Work
   if no such i exists goto step (4)
   3) Work = Work + Allocation[i]
   Finish[i] = true
   goto step (2)
   4) if Finish [i] = true for all i
   then the system is in a safe state

ALGORITHM:

Step-1: Start the program.

Step-2: Declare the memory for the process.

Step-3: Read the number of process, resources, allocation matrix and available matrix.

Step-4: Compare each and every process using the banker‟s algorithm.

Step-5: If the process is in safe state then it is a not a deadlock process otherwise it is a deadlock process

Step-6: produce the result of state of process

Step-7: Stop the program

Program:

```
// Banker's Algorithm
#include <stdio.h>
int main()
{
   // P0, P1, P2, P3, P4 are the Process names here

   int n, m, i, j, k;
   n = 5; // Number of processes
   m = 3; // Number of resources
   int alloc[5][3] = { { 0, 1, 0 }, // P0    // Allocation Matrix
```

```
                { 2, 0, 0 }, // P1
                { 3, 0, 2 }, // P2
                { 2, 1, 1 }, // P3
                { 0, 0, 2 } }; // P4

int max[5][3] = { { 7, 5, 3 }, // P0    // MAX Matrix
                { 3, 2, 2 }, // P1
                { 9, 0, 2 }, // P2
                { 2, 2, 2 }, // P3
                { 4, 3, 3 } }; // P4

int avail[3] = { 3, 3, 2 }; // Available Resources

int f[n], ans[n], ind = 0;
for (k = 0; k < n; k++) {
   f[k] = 0;
}
int need[n][m];
for (i = 0; i < n; i++) {
   for (j = 0; j < m; j++)
      need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
for (k = 0; k < 5; k++) {
   for (i = 0; i < n; i++) {
      if (f[i] == 0) {

         int flag = 0;
         for (j = 0; j < m; j++) {
            if (need[i][j] > avail[j]){
               flag = 1;
                break;
            }
         }

         if (flag == 0) {
            ans[ind++] = i;
            for (y = 0; y < m; y++)
               avail[y] += alloc[i][y];
            f[i] = 1;
         }
      }
   }
}

  int flag = 1;
```

```
    for(int i=0;i<n;i++)
   {
    if(f[i]==0)
    {
      flag=0;
       printf("The following system is not safe");
      break;
    }
   }

    if(flag==1)
   {
    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
      printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);
   }


   return (0);

}
```

Output:

Following is the SAFE Sequence

P1 -> P3 -> P4 -> P0 -> P2

ASSIGNMENT NO: 09

TITLE:

Implement of following Non pre-emptive CPU scheduling algorithms: First Come First Serve, Shortest Job First, Priority.

AIM :

Implementation of First Come First Serve, Shortest Job First, Priority scheduling algorithms.

Theory:

CPU Scheduling

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Dispatcher

Another component involved in the CPU scheduling function is the **dispatcher**. The dispatcher is the module that gives control of the CPU to the process selected by the **short-term scheduler**. This function involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency**.

CPU scheduling decisions may take place under the following four circumstances:

- When a process switches from the **running** state to the **waiting** state(for I/O request or invocation of wait for the termination of one of the child processes).

- When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
- When a process switches from the **waiting** state to the **ready** state(for example, completion of I/O).
- When a process **terminates**.

Scheduling Algorithms

major scheduling algorithms here which are following :

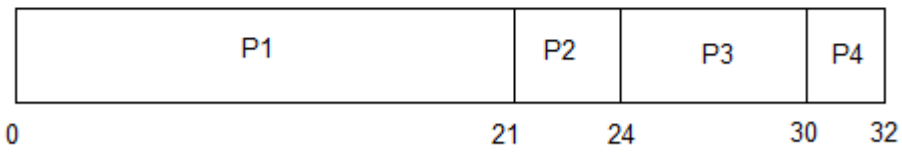
- First Come First Serve(FCFS) Scheduling
- Shortest-Job-First(SJF) Scheduling
- Priority Scheduling

First Come First Serve(FCFS) Scheduling

- Jobs are executed on first come, first serve basis.
- Easy to understand and implement.
- Poor in performance as average wait time is high.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



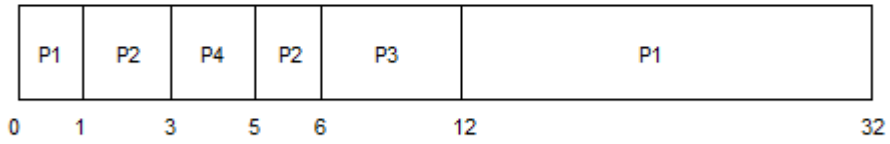
This is the GANTT chart for the above processes

Shortest-Job-First(SJF) Scheduling

- Best approach to minimize waiting time.
- Actual time taken by the process is already known to processor.
- Impossible to implement.
  - In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with short burst time arrives, the existing process is preempted.

PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



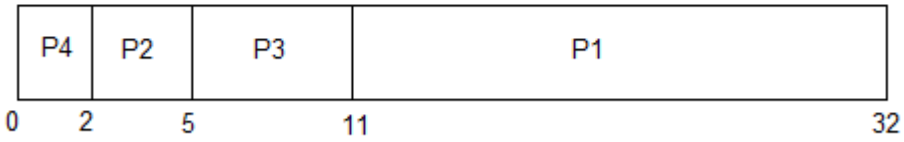
The average waiting time will be, ((5-3) + (6-2) + (12-1))/4 = 4.25 ms

The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

Priority Scheduling

- Priority is assigned for each process.
- Process with highest priority is executed first and so on.
- Processes with same priority are executed in FCFS manner.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,

The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

Program :

```
*/
#include<stdio.h>
#include<stdlib.h>
void main()
{  int i,j,wt[10],tt[10],bt[10],p[10],pr[10],n,ch,temp,w=0;
int st[10],tq,count,sq=0,tat[10];
int ttt=0,twt=0;
float att,awt;
clrscr();
printf("\nCPU SCHEDULING\n");
printf("\nEnter the no of processes: ");
scanf("%d",&n);
do
```

```
{
printf("\nMENU:");
printf("\n1>FCFS\n2>SJF\n3>Priority\n4>Round Robin\n5>Exit\n");
printf("\nEnter your choice : ");
scanf("%d",&ch);
switch(ch)
{
case 1:
        w=0;
        for(i=1;i<=n;i++)
        {
        p[i]=i;
        printf("\nEnter the Burst time for the process p[%d]: ",i);
        scanf("%d",&bt[i]);
        }
        //Calculation of Waiting, Turn around time
  for(i=1;i<=n;i++)
        {
        wt[i]=w;
        tt[i]=wt[i]+bt[i];
        twt=twt+wt[i];
        ttt=ttt+tt[i];
        w=w+bt[i];
        }
        break;
case 2: w=0;
        for(i=1;i<=n;i++)
        {
        p[i]=i;
```

```
printf("\nEnter the Burst time for the process p[%d]: ",i);
scanf("%d",&bt[i]);
}
for(i=1;i<=n;i++)
{
for(j=i+1;j<=n;j++)
{
if(bt[i]>bt[j])
{
temp=bt[i];
bt[i]=bt[j];
bt[j]=temp;
temp=p[i];
p[i]=p[j];
p[j]=temp;
}
}
}
//Calculation of Waiting, Turn around time
for(i=1;i<=n;i++)
{
wt[i]=w;
tt[i]=wt[i]+bt[i];
twt=twt+wt[i];
ttt=ttt+tt[i];
w=w+bt[i];
}
        break;
case 3:
```

```
 w=0;
 for(i=1;i<=n;i++)
   {
   p[i]=i;
         printf("\nEnter the Burst time and priority for the process p[%d]: ",i);
         scanf("%d%d",&bt[i],&pr[i]);
   }
         for(i=1;i<=n;i++)
         for(j=i+1;j<=n;j++)
         {
         if(pr[i]>pr[j])
         {
         temp=bt[i];
         bt[i]=bt[j];
         bt[j]=temp;
temp=pr[i];
pr[i]=pr[j];
pr[j]=temp;
temp=p[i];
p[i]=p[j];
p[j]=temp;
         }
         }
//Calculation of Waiting, Turn around time
for(i=1;i<=n;i++)
{
wt[i]=w;
tt[i]=wt[i]+bt[i];
twt=twt+wt[i];
```

```
ttt=ttt+tt[i];
w=w+bt[i];
}

        break;
case 4:
 for(i=1;i<=n;i++)
{
p[i]=i;
printf("\nEnter the Burst time for the process p[%d]: ",i);
scanf("%d",&bt[i]);
st[i]=bt[i];
}
 printf("\nEnter time quantum:");
 scanf("%d",&tq);
 while(1)
 {
        for(i=1,count=0;i<=n;i++)
         {
        temp=tq;
        if(st[i]==0)
                {
          count++;
          continue;
          }
        if(st[i]>tq)
        st[i]=st[i]-tq;
        else
        if(st[i]>=0)
```

```
                    {
                    temp=st[i];
                    st[i]=0;
                    }
                    sq=sq+temp;
                    tat[i]=sq;
                    }
                   if(n==count)
                   break;
                    }
                   //Calculation of Waiting, Turn around time
          for(i=1;i<=n;i++)
          {          wt[i]=tat[i]-bt[i];
           tt[i]=tat[i];
           twt=twt+wt[i];
          ttt=ttt+tt[i];
          w=w+bt[i];
          }
                   break;
case 5: exit(0);
                   break;
}
  awt=twt/n;
  att=ttt/n;
printf("Process ID\tBT\t\tWT\t\tTT");
for(i=1;i<=n;i++)
{
printf("\np[%d]\t\t%d\t\t%d\t\t%d",p[i],bt[i],wt[i],tt[i]);
}
```

```
printf("\nAvg. Waiting time = %2f",awt);
printf("\nAvg. turnaround time = %2f",att);
}while(ch!=5);
getch();
}
/*OUTPUT
CPU SCHEDULING
Enter the no of processes: 3
MENU:
1>FCFS
2>SJF
3>Priority
4>Round Robin
5>Exit
Enter your choice : 1
Enter the Burst time for the process p[1]: 24
Enter the Burst time for the process p[2]: 3
Enter the Burst time for the process p[3]: 3
Process ID      BT          WT          TT
p[1]        24          0           24
p[2]        3           24          27
p[3]        3           27          30
Avg. Waiting time = 17.000000
Avg. turnaround time = 27.000000
MENU:
1>FCFS
2>SJF
3>Priority
4>Round Robin
```

5>Exit

Enter your choice :2

Enter the Burst time for the process p[1]: 3

Enter the Burst time for the process p[2]: 1

Enter the Burst time for the process p[3]: 4

Process ID	BT	WT	TT
p[2]	1	0	1
p[1]	3	1	4
p[3]	4	4	8

Avg. Waiting time = 18.000000

Avg. turnaround time = 31.000000

MENU:

1>FCFS

2>SJF

3>Priority

4>Round Robin

5>Exit

Enter your choice :3

Enter the Burst time and priority for the process p[1]: 4 1

Enter the Burst time and priority for the process p[2]: 5 2

Enter the Burst time and priority for the process p[3]: 6 3

Process ID	BT	WT	TT
p[1]	4	0	4
p[2]	5	4	9
p[3]	6	9	15

Avg. Waiting time = 23.000000

Avg. turnaround time = 40.000000

MENU:

1>FCFS

2>SJF

3>Priority

4>Round Robin

5>Exit

Enter your choice :4

Enter your choice : 4

Enter the Burst time for the process p[1]: 3

Enter the Burst time for the process p[2]: 6

Enter the Burst time for the process p[3]: 3

Enter time quantum:3

Process ID	BT	WT	TT
p[1]	3	0	3
p[2]	6	6	12
p[3]	3	6	9

Avg. Waiting time = 27.000000

Avg. turnaround time = 48.000000

MENU:

1>FCFS

2>SJF

3>Priority

4>Round Robin

5>Exit

Enter your choice :5

*/

ASSIGNMENT NO: 10

TITLE:

Implement of following Pre-emptive CPU scheduling algorithms: Shortest Job First, Priority, and Round Robin.

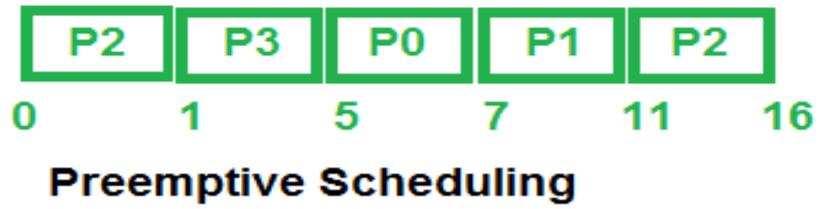
Theory :

Preemptive Scheduling:

Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for a limited amount of time and then taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in the ready queue till it gets its next chance to execute.

Algorithms based on preemptive scheduling are: Round Robin (RR),Shortest Remaining Time First (SRTF), Priority (preemptive version), etc.

Process	Arrival Time	CPU Burst Time (in millisec.)
P0	3	2
P1	2	4
P2	0	6
P3	1	4



Preemptive Scheduling

CPU Scheduling

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Dispatcher

Another component involved in the CPU scheduling function is the **dispatcher**. The dispatcher is the module that gives control of the CPU to the process selected by the **short-term scheduler**. This function involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency**.

CPU scheduling decisions may take place under the following four circumstances:

- When a process switches from the **running** state to the **waiting** state(for I/O request or invocation of wait for the termination of one of the child processes).
- When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
- When a process switches from the **waiting** state to the **ready** state(for example, completion of I/O).
- When a process **terminates**.

Scheduling Algorithms

major scheduling algorithms here which are following :

- Shortest-Job-First(SJF) Scheduling
- Priority Scheduling
- Round Robin

Shortest-Job-First(SJF) Scheduling

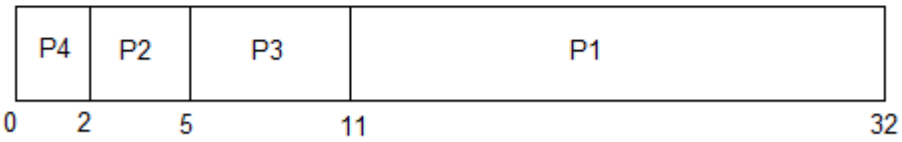
- Best approach to minimize waiting time.
- Actual time taken by the process is already known to processor.
- Impossible to implement.

- In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with short burst time arrives, the existing process is preempted.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

Priority Scheduling

- Priority is assigned for each process.
- Process with highest priority is executed first and so on.
- Processes with same priority are executed in FCFS manner.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,

The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

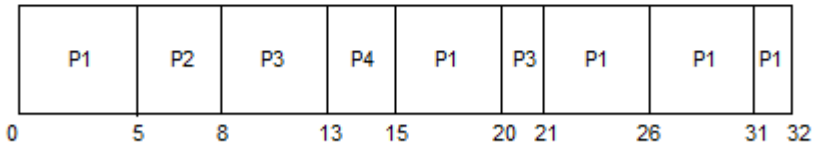
Round Robin(RR) Scheduling

- A fixed time is allotted to each process, called **quantum**, for execution.
- Once a process is executed for given time period that process is preemptied and other process executes for given time period.

- Context switching is used to save states of preemptied processes.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,



The average waiting time will be, 11 ms.

Program :

```
*/
#include<stdio.h>
#include<stdlib.h>
void main()
{  int i,j,wt[10],tt[10],bt[10],p[10],pr[10],n,ch,temp,w=0;
int st[10],tq,count,sq=0,tat[10];
int ttt=0,twt=0;
float att,awt;
clrscr();
printf("\nCPU SCHEDULING\n");
printf("\nEnter the no of processes: ");
scanf("%d",&n);
do
{
```

```
printf("\nMENU:");
printf("\n1>FCFS\n2>SJF\n3>Priority\n4>Round Robin\n5>Exit\n");
printf("\nEnter your choice : ");
scanf("%d",&ch);
switch(ch)
{
case 1:
        w=0;
        for(i=1;i<=n;i++)
        {
        p[i]=i;
        printf("\nEnter the Burst time for the process p[%d]: ",i);
        scanf("%d",&bt[i]);
        }
        //Calculation of Waiting, Turn around time
  for(i=1;i<=n;i++)
        {
        wt[i]=w;
        tt[i]=wt[i]+bt[i];
        twt=twt+wt[i];
        ttt=ttt+tt[i];
        w=w+bt[i];
        }
        break;
case 2: w=0;
        for(i=1;i<=n;i++)
        {
        p[i]=i;
        printf("\nEnter the Burst time for the process p[%d]: ",i);
```

```
scanf("%d",&bt[i]);
}
for(i=1;i<=n;i++)
{
for(j=i+1;j<=n;j++)
{
if(bt[i]>bt[j])
{
temp=bt[i];
bt[i]=bt[j];
bt[j]=temp;
temp=p[i];
p[i]=p[j];
p[j]=temp;
}
}
}
//Calculation of Waiting, Turn around time
for(i=1;i<=n;i++)
{
wt[i]=w;
tt[i]=wt[i]+bt[i];
twt=twt+wt[i];
ttt=ttt+tt[i];
w=w+bt[i];
}
        break;
case 3:
 w=0;
```

```
 for(i=1;i<=n;i++)
   {
   p[i]=i;
         printf("\nEnter the Burst time and priority for the process p[%d]: ",i);
         scanf("%d%d",&bt[i],&pr[i]);
   }
         for(i=1;i<=n;i++)
         for(j=i+1;j<=n;j++)
         {
         if(pr[i]>pr[j])
         {
         temp=bt[i];
         bt[i]=bt[j];
         bt[j]=temp;
temp=pr[i];
pr[i]=pr[j];
pr[j]=temp;
temp=p[i];
p[i]=p[j];
p[j]=temp;
         }
         }
//Calculation of Waiting, Turn around time
for(i=1;i<=n;i++)
{
wt[i]=w;
tt[i]=wt[i]+bt[i];
twt=twt+wt[i];
ttt=ttt+tt[i];
```

```
w=w+bt[i];
}

        break;
case 4:
 for(i=1;i<=n;i++)
{
p[i]=i;
printf("\nEnter the Burst time for the process p[%d]: ",i);
scanf("%d",&bt[i]);
st[i]=bt[i];
}
 printf("\nEnter time quantum:");
 scanf("%d",&tq);
 while(1)
 {
        for(i=1,count=0;i<=n;i++)
         {
        temp=tq;
        if(st[i]==0)
                {
          count++;
          continue;
          }
        if(st[i]>tq)
        st[i]=st[i]-tq;
        else
        if(st[i]>=0)
         {
```

```
                    temp=st[i];
                    st[i]=0;
                     }
                    sq=sq+temp;
                    tat[i]=sq;
                    }
                   if(n==count)
                   break;
                    }
                   //Calculation of Waiting, Turn around time
          for(i=1;i<=n;i++)
          {          wt[i]=tat[i]-bt[i];
           tt[i]=tat[i];
           twt=twt+wt[i];
          ttt=ttt+tt[i];
          w=w+bt[i];
          }
                   break;
case 5: exit(0);
                   break;
}
  awt=twt/n;
  att=ttt/n;
printf("Process ID\tBT\t\tWT\t\tTT");
for(i=1;i<=n;i++)
{
printf("\np[%d]\t\t%d\t\t%d\t\t%d",p[i],bt[i],wt[i],tt[i]);
}
printf("\nAvg. Waiting time = %2f",awt);
```

```
printf("\nAvg. turnaround time = %2f",att);
}while(ch!=5);
getch();
}
/*OUTPUT
CPU SCHEDULING
Enter the no of processes: 3
MENU:
1>FCFS
2>SJF
3>Priority
4>Round Robin
5>Exit
Enter your choice : 1
Enter the Burst time for the process p[1]: 24
Enter the Burst time for the process p[2]: 3
Enter the Burst time for the process p[3]: 3
Process ID      BT          WT          TT
p[1]            24          0           24
p[2]            3           24          27
p[3]            3           27          30
Avg. Waiting time = 17.000000
Avg. turnaround time = 27.000000
MENU:
1>FCFS
2>SJF
3>Priority
4>Round Robin
5>Exit
```

Enter your choice :2

Enter the Burst time for the process p[1]: 3

Enter the Burst time for the process p[2]: 1

Enter the Burst time for the process p[3]: 4

Process ID	BT	WT	TT
p[2]	1	0	1
p[1]	3	1	4
p[3]	4	4	8

Avg. Waiting time = 18.000000

Avg. turnaround time = 31.000000

MENU:

1>FCFS

2>SJF

3>Priority

4>Round Robin

5>Exit

Enter your choice :3

Enter the Burst time and priority for the process p[1]: 4 1

Enter the Burst time and priority for the process p[2]: 5 2

Enter the Burst time and priority for the process p[3]: 6 3

Process ID	BT	WT	TT
p[1]	4	0	4
p[2]	5	4	9
p[3]	6	9	15

Avg. Waiting time = 23.000000

Avg. turnaround time = 40.000000

MENU:

1>FCFS

2>SJF

```
3>Priority
4>Round Robin
5>Exit
Enter your choice :4
Enter your choice : 4
Enter the Burst time for the process p[1]: 3
Enter the Burst time for the process p[2]: 6
Enter the Burst time for the process p[3]: 3
Enter time quantum:3
Process ID      BT          WT          TT
p[1]            3           0           3
p[2]            6           6           12
p[3]            3           6           9
Avg. Waiting time = 27.000000
Avg. turnaround time = 48.000000
MENU:
1>FCFS
2>SJF
3>Priority
4>Round Robin
5>Exit
Enter your choice :5
*/
```

ASSIGNMENT NO: 11

TITLE:

Implementation of Page replacement algorithms
a) First In First Out b) List Recently Used c) Optimal Page replacement algorithm

Theory:

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

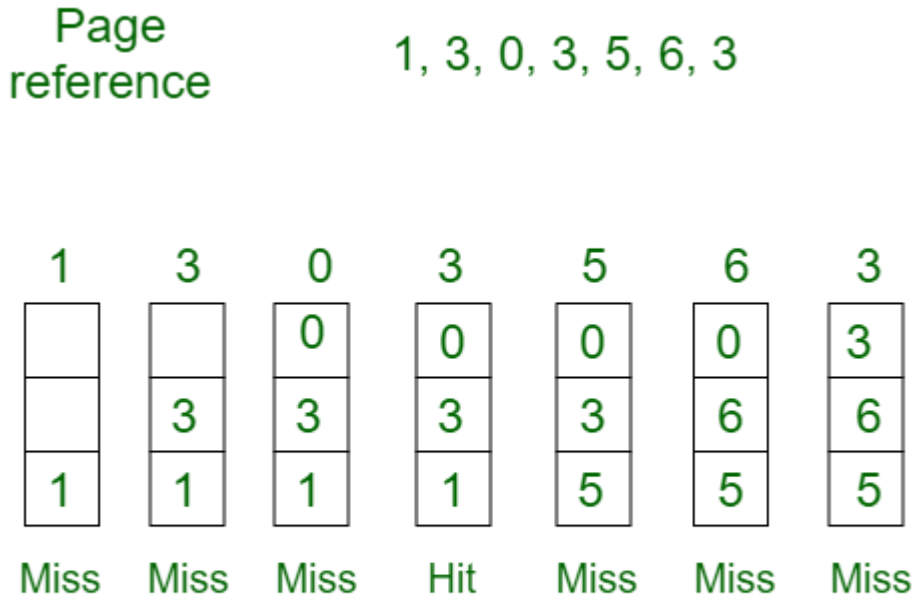
Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

Page Replacement Algorithms :

1. First In First Out (FIFO) –
This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example-1Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames.Find number of

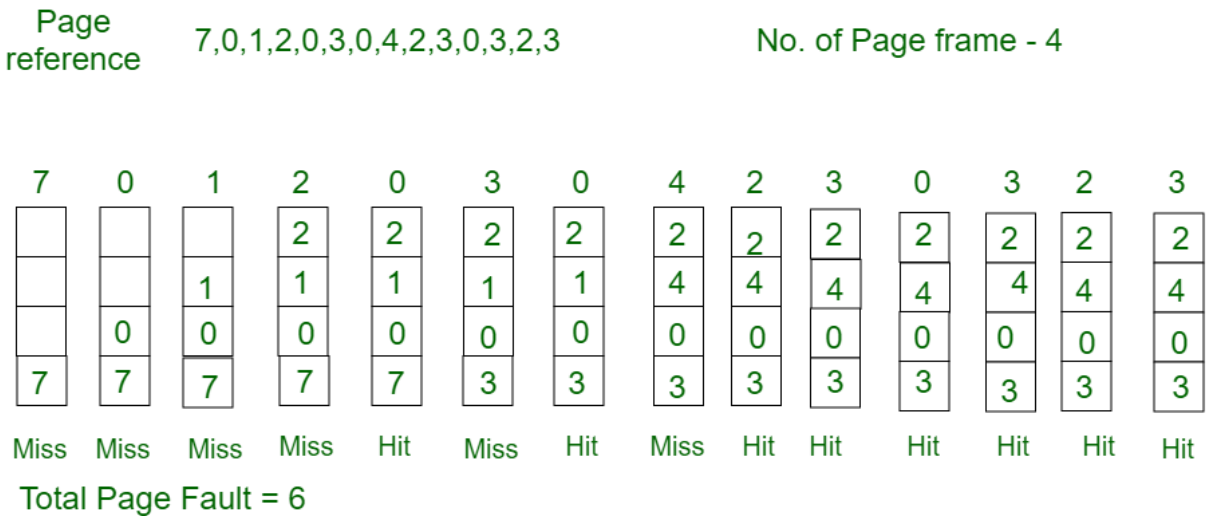


page faults. Total Page Fault = 6

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> 3 Page Faults.
when 3 comes, it is already in memory so —> 0 Page Faults.
Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>1 Page Fault.
6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>1 Page Fault.
Finally when 3 come it is not available so it replaces 0 1 page fault

Belady's anomaly – Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

2. Optimal Page replacement –
In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.
Example-2:Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.



Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> 4 Page faults
0 is already there so —> 0 Page fault.
when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>1 Page fault.
0 is already there so —> 0 Page fault..
4 will takes place of 1 —> 1 Page Fault.

Now for the further page reference string —> 0 Page fault because they are already available in the memory.
Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

3. Least Recently Used –
In this algorithm page will be replaced which is least recently used.
Example-3Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames.Find number of page faults.

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3 No. of Page frame - 4

Here LRU has same number of page fault as optimal but it may differ according to question.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> 4 Page faults
0 is already their so —> 0 Page fault.
when 3 came it will take the place of 7 because it is least recently used —>1 Page fault
0 is already in memory so —> 0 Page fault.
4 will takes place of 1 —> 1 Page Fault
Now for the further page reference string —> 0 Page fault because they are already available in the memory.

Program:

```
#include<stdio.h>

#include<conio.h>

void lru();

void fifo();

void optimal();

void getdata();

int cnt=0,count=1,m,arr[20],arr1[5],flag=0,store[20];

int n,s,f,h,i,j,k,pos[10],min=200,pos1=0,max=0;

main()
```

```
{ int choice;
 clrscr();
 do
 {
 printf("\n1:First in First Out\n2:Least Recently Used\n3:Optimal Paged\n4:Exit");
 printf("\n\nEnter ur choice:");
 scanf("%d",&choice);
 switch(choice)
 { case 1:
  fifo();
  break;
 case 2:
  lru();
  break;
 case 3:
 optimal();
  break;
 case 4:
  exit(0);
 }
  }while(choice!=3);
 getch();
 return(0);
}
void getdata()
{  printf("Enter the no of elements in sequence:");
 scanf("%d",&s);
```

```
 printf("Enter the elements in sequence:");
 for(i=1;i<=s;i++)
 scanf("%d",&arr[i]);
 printf("Enter the no of frames:");
 scanf("%d",&f);
 for(i=1;i<=f;i++)
 arr1[i]=0;
}
void lru()
{   cnt=0;count=1;flag=0;min=200,pos1=0;
  getdata();
  m=1;
  while(cnt<3)
  {  for(i=1;i<=f;i++)
 {   if(arr[m]==arr1[i])
  {    store[count]=arr[m];
   flag=1;
   count++;
   break;
  }
  else
   flag=0;
 }
   if(flag==0)
   {
  cnt++;
  arr1[cnt]=arr[m];
```

```
 store[count]=arr[m];
 count++;
  }
  m++;
 }
 printf("\nThe frames are:\n");
 for(k=1;k<=f;k++)
 printf("%d ",arr1[k]);
//-----------------------------------------------------
 for(j=m;j<=s;j++)
{  for(i=1;i<=f;i++)
 {     if(arr[j]==arr1[i])
   {    flag=1;
   store[count]=arr[j];
   count++;
   break;
   }
   else
   flag=0;
  }
 if(flag==0)
 {    for(i=1;i<=f;i++)
   {     for(h=1;h<count;h++)
    {      if(arr1[i]==store[h])
     {      pos[i]=h;
      }
    }
```

```
  }
  min=200;
  for(i=1;i<=f;i++)
  {     if(pos[i]<min)
   {
    min=pos[i];
    pos1=i;
   }
  }
  cnt++;
  arr1[pos1]=arr[j];
  store[count]=arr[j];
  count++;
  printf("\n");
  for(k=1;k<=f;k++)
  printf("%d ",arr1[k]);
 }
 }
  printf("\nThe no of page faults:%d",cnt);
}
void fifo()
{   cnt=0;count=1;flag=0;
  getdata();
  m=1;
  while(cnt<3)
  {
  for(i=1;i<=f;i++)
```

```
{
 if(arr[m]==arr1[i])
 {
  flag=1;
  break;
 }
 else
 flag=0;
}
if(flag==0)
{
 cnt++;
 arr1[cnt]=arr[m];
 store[cnt]=arr[m];
}
m++;
 }
n=1;
printf("\nThe frames are:\n");
for(k=1;k<=f;k++)
printf("%d ",arr1[k]);
for(j=m;j<=s;j++)
{
 for(i=1;i<=f;i++)
 {
  if(arr[j]==arr1[i])
  {
```

```
    flag=1;
    break;
   }
   else
   flag=0;
  }
 if(flag==0)
 {
  for(i=1;i<=f;i++)
  {
   if(arr1[i]==store[n])
   {
    cnt++;
    arr1[i]=arr[j];
    store[cnt]=arr[j];
    }
  }
  n++;
  printf("\n");
  for(k=1;k<=f;k++)
   printf("%d ",arr1[k]);
 }
}
 printf("\nThe no of page faults:%d",cnt);
}
void optimal()
{     cnt=0;count=1;flag=0;max=0;pos1=0;
```

```
 getdata();
 m=1;
 for(i=1;i<=s;i++)
 {     store[i]=arr[i];
 }
 while(cnt<3)
 {  for(i=1;i<=f;i++)
{   if(arr[m]==arr1[i])
 {    flag=1;
  count++;
  break;
 }
 else
  flag=0;
}
  if(flag==0)
  {
 cnt++;
 arr1[cnt]=arr[m];
 count++;
  }
  m++;
}
printf("\nThe frames are:\n");
for(k=1;k<=f;k++)
printf("%d ",arr1[k]);
//-------------------------------------------------
```

```
for(j=m;j<=s;j++)
{  for(i=1;i<=f;i++)
{    if(arr[j]==arr1[i])
   {    flag=1;
  count++;
  break;
  }
  else
  flag=0;
 }
if(flag==0)
{
  for(i=1;i<=f;i++)
  {
   for(h=count+1;h<=s;h++)
   {
    if(arr1[i]==store[h])
    {
     pos[i]=h;
     break;
    }
    else
    pos[i]=200;
   }
 }
  max=0;
  for(i=1;i<=f;i++)
```

```
  {
   if(pos[i]>max)
   {
    max=pos[i];
    pos1=i;
   }
  }
  cnt++;
  arr1[pos1]=arr[j];
  count++;
  printf("\n");
  for(k=1;k<=f;k++)
  printf("%d ",arr1[k]);
 }
 }
  printf("\nThe no of page faults:%d",cnt);
}
/*Output:
1:First in First Out
2:Least Recently Used
3:Optimal Paged
4:Exit
Enter ur choice:2
Enter the no of elements in sequence:6
Enter the elements in sequence:
5
7
```

5

6

7

3

Enter the no of frames:3

The frames are:

5 7 6

3 7 6

The no of page faults:4

1:First in First Out

2:Least Recently Used

3:Optimal Paged

4:Exit

Enter ur choice:4

*/

ASSIGNMENT NO: 12

Title:

Write an IPC program using pipe. Process A accepts a character string and Process B

Inverses the string. Pipe is used to establish communication between A and B processes

using Python or C++.

Theory:

Inter-process communication (IPC) is a mechanism that allows the exchange of data between processes. By providing a user with a set of programming interfaces, IPC helps a programmer organize the activities among different processes. IPC allows one application to control another application, thereby enabling data sharing without interference.

IPC enables data communication by allowing processes to use segments, semaphores, and other methods to share memory and information. IPC facilitates efficient message transfer between processes.

In computing, inter-process communication (IPC) is a set of methods for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network. IPC methods are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC).

Pipe:

Description:

The method pipe() creates a pipe and returns a pair of file descriptors

(r, w) usable for reading and writing, respectively

Syntax:

Following is the syntax for pipe() method:

os.pipe()

Parameters: NA

Return Value:

This method returns a pair of file descriptors.

Design Analysis / Implementation Logic:

The import statements are similar to the #include statements in C: they allow you to use functions elsewhere.

For example, import os, import sys etc.

1.sys:

This module provides a number of functions and variables that can be used to manipulate different parts of the Python runtime environment.

2. os:

The OS module in Python provides a way of using operating system dependent functionality.

3. OS functions

a. getpid:

Returns the real process ID of the current process.

You can always access the PID of the current process using os.getpid(), no matter if you're in the parent or child process.

Syntax : os.getpid();

b. Fork:

Returns 0 inside the child process and (this is standard for Unix systems) and the child PID inside the parent.

Syntax :os.fork();

c. close:

The method close () closes the associated with file descriptor fd.

Syntax :os.close(fd);

fd -- This is the file descriptor of the file.

Return Value: This method does not return any value.

d. Exit : It means a clean exit without any errors / problems.

Syntax : exit(0);

Program:

Write an IPC program using pipe. Process A accepts a character string and Process B Inverses the string. Pipe is used to establish communication between A and B processes using Python or C++.

import os

```
import time
pid=os.fork()
if pid:
    # parent
   # while True:
            wfPath = "./p1"
            rfPath = "./p2"
            msg= raw_input("Enter a message to send to child via pipe : ")
            try:
               os.mkfifo(wfPath)
               os.mkfifo(rfPath)
            except OSError:
               pass
            wp = open(wfPath, 'w')
            wp.write(msg)
            wp.close()
            rp = open(rfPath, 'r')
            response = rp.read()
            print "Reverse String from process B: %s" % response
else:
    # child
    #while True:
        rfPath = "./p1"
            wfPath = "./p2"
            try:  os.mkfifo(wfPath)
               os.mkfifo(rfPath)
            except OSError:
```

```
    pass
rp = open(rfPath, 'r')
response = rp.read()
print "String received from process A: %s" % response
s=response
s=s[::-1]
rp.close()
wp = open(wfPath, 'w')
wp.write(s)
wp.close()
rp.close()
```

Output:

Enter a message to send to child via pipe :raisoni

String received from process A : raisoni

Reverse String from process B :inosiar

ASSIGNMENT NO: 13

Title:

Simulation of Memory allocation algorithms (First Fit, Best Fit , Next Fit)

Theory:

Memory Allocation:

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free

First fit:

In the first fit, partition is allocated which is first sufficient from the top of Main Memory.

Example :

Input : blockSize[] = {100, 500, 200, 300, 600};

processSize[] = {212, 417, 112, 426};

Output:

Process No.	Process Size	Block no.
1	212	2
2	417	5
3	112	2
4	426	Not Allocated

Its advantage is that it is the fastest search as it searches only the first block i.e. enough to assign a process.

It may problems of not allowing processes to take space even if it was possible to allocate. Consider the above example, process number 4 (of size 426) does not get memory. However it was possible to allocate memory if we had allocated using best fit allocation [block number 4

(of size 300) to process 1, block number 2 to process 2, block number 3 to process 3 and block number 5 to process 4.

Implementation:

1- Input memory blocks with size and processes with size.

2- Initialize all memory blocks as free.

3- Start by picking each process and check if it can be assigned to current block.

4- If size-of-process <= size-of-block if yes then assign and check for next process.

5- If not then keep checking the further blocks.

Best Fit

Best fit allocates the process to a partition which is the smallest sufficient partition among the free available partitions.

Example:

Input : blockSize[] = {100, 500, 200, 300, 600};

processSize[] = {212, 417, 112, 426};

Output:

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

Implementation:

1- Input memory blocks and processes with sizes.

2- Initialize all memory blocks as free.

3- Start by picking each process and find the minimum block size that can be assigned to current process i.e., find min(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current], if found then assign it to the current process.

5- If not then leave that process and keep checking the further processes.

Next Fit

Next fit is a modified version of 'first fit'. It begins as first fit to find a free partition but when called next time it starts searching from where it left off, not from the beginning. This policy makes use of a roving pointer. The pointer roves along the memory chain to search for a next fit. This helps in, to avoid the usage of memory always from the head (beginning) of the free block chain.

Advantage over first fit:

First fit is a straight and fast algorithm, but tends to cut large portion of free parts into small pieces due to which, processes that needs large portion of memory block would not get anything even if the sum of all small pieces is greater than it required which is so called internal fragmentation problem.

Another problem of first fit is that it tends to allocate memory parts at the beginning of the memory, which may leads to more internal fragements at the beginning. Next fit tries to address this problem by starting search for the free portion of parts not from the start of the memory, but from where it ends last time.

Next fit is a very fast searching algorithm and is also comparatively faster than First Fit and Best Fit Memory Management Algorithms.

Example:

Input : blockSize[] = {5, 10, 20};

processSize[] = {10, 20, 30};

Output:

Process No.	Process Size	Block no.
1	10	2
2	20	3
3	30	Not Allocated

Recommended: Please try your approach on {IDE} first, before moving on to the solution.

Algorithm:

Input the number of memory blocks and their sizes and initializes all the blocks as free.

Input the number of processes and their sizes.

Start by picking each process and check if it can be assigned to current block, if yes, allocate it the required memory and check for next process but from the block where we left not from starting.

If current block size is smaller then keep checking the further blocks.

Program: Simulation of Memory Allocation Algorithm(First Fit, Best Fit, Next Fit)

```
#include<stdio.h>
#define MAX 20
typedef struct mem
{
        int flag,size,index;
}M;
typedef struct pro
{
        int id,size;
}P;

void first_fit(M m[],P a[],int n,int no)
{
        int i,j,flag;
        for(j=0;j<n;j++)
        {
                flag=0;
                for(i=0;i<no;i++)
                {
                        if(m[i].flag==0 && m[i].size >= a[j].size)
```

```
                        {
                                m[i].flag=1;
                                m[i].index=j;
                                flag=1;
                                break;
                        }
                }
                if(flag!=1)
                        printf("\nNo slot for proces with id = %d and size
=%d",a[j].id,a[j].size);
        }
}
void accept(P p[],int n)
{
        int i;
        for(i=0;i<n;i++)
        {
                printf("\nEnter the pid and size of process:");
                scanf("%d%d",&p[i].id,&p[i].size);
        }
}
void memconfig(M m[],int no)
{
        int i;
        for(i=0;i<no;i++)
        {
                printf("Enter the partition size:");
                scanf("%d",&m[i].size);
                printf("\nEnter the status 0 for free and 1 for occupied:");
                scanf("%d",&m[i].flag);
                m[i].index=-1;
        }
}
void next_fit(M m[],P a[],int n,int no)
{
        int i,j,flag,k=0,l;
        for(j=0;j<n;j++)
        {
                flag=0,l=0;
                for(i=k;l<no;i=(i+1)%no,l++)
                {
                        if(m[i].flag==0 && m[i].size >= a[j].size)
                        {
                                m[i].flag=1;
                                m[i].index=j;
                                flag=1;
                                k=i+1;
                                break;
                        }
```

```
                }
                if(flag!=1)
                        printf("\nNo slot for proces with id = %d and size
=%d",a[j].id,a[j].size);
        }
}
void display(M m[],int no)
{
        int i;
        printf("SIZE\tSTATUS");
        for(i=0;i<no;i++)
        {
                printf("\n%d",m[i].size);
                if(m[i].flag==0)
                        printf("\tEMPTY");
                else
                        printf("\tOCCUPIED");
        }

}
void best_fit(M m[],P a[],int n,int no)
{
        int i,j,flag,k;
        M temp;
        temp.flag=0;
        for(j=0;j<n;j++)
        {
                flag=0;
                temp.size=999;
                for(i=0;i<no;i++)
                {
                        if(m[i].flag==0 && m[i].size >= a[j].size)
                        {
                                if(a[j].size == m[i].size)
                                {
                                        flag=1;
                                        k=i;
                                        break;
                                }
                                else if (temp.size > m[i].size)
                                {
                                        temp=m[i];
                                        k=i;
                                        flag=1;
                                }
                        }
                }
                if(flag!=1)
                {
```

```
                        printf("\nNo slot for proces with id = %d and size
=%d",a[j].id,a[j].size);
                }
                else
                {
                        m[k].index=j;
                        m[k].flag=1;
                }
        }
}
void worst_fit(M m[],P a[],int n,int no)
{
        int i,j,flag,k;
        M temp;
        temp.flag=0;
        for(j=0;j<n;j++)
        {
                flag=0;
                temp.size=0;
                for(i=0;i<no;i++)
                {
                        if(m[i].flag==0 && m[i].size >= a[j].size)
                        {

                                if (temp.size < m[i].size)
                                {
                                        temp=m[i];
                                        k=i;
                                        flag=1;
                                }
                        }
                }
                if(flag!=1)
                {
                        printf("\nNo slot for proces with id = %d and size
=%d",a[j].id,a[j].size);
                }
                else
                {
                        m[k].index=j;
                        m[k].flag=1;
                }
        }
}

void disp(M m[],P p[],int n,int no)
{
        int i;
        printf("\nSIZE\tPID\tPSIZE\tSTATUS");
        for(i=0;i<no;i++)
```

```
	{

		if(m[i].flag==0)
		{
			printf("\n%d\t--\t--",m[i].size);
			printf("\tEMPTY");
		}
		else
		{
			if(m[i].index==-1)
				printf("\n%d\t--\t--",m[i].size);
			else

	printf("\n%d\t%d\t%d",m[i].size,p[m[i].index].id,p[m[i].index].size);
			printf("\tOCCUPIED");
		}
	}

}
void main()
{
	M m[MAX];
	P p[MAX];
	int n,ch,no;
	do
	{
		printf("\n1.FIRST FIT");
		printf("\n2.NEXT FIT");
		printf("\n3.BEST FIT");
		printf("\n4.WORST FIT");
		printf("\n5.EXIT");
		printf("\nEnter your chocie:");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:
				printf("\nEnter the no of memory slots:");

				scanf("%d",&no);
				memconfig(m,no);
				printf("\nEnter the no of process:");
				scanf("%d",&n);
				accept(p,n);
				display(m,no);
				first_fit(m,p,n,no);
				disp(m,p,n,no);
				break;
			case 2:
				printf("\nEnter the no of memory slots:");
```

```
                    scanf("%d",&no);
                    memconfig(m,no);
                    printf("\nEnter the no of process:");
                    scanf("%d",&n);
                    accept(p,n);
                    display(m,no);
                    next_fit(m,p,n,no);
                    disp(m,p,n,no);
                    break;
            case 3:
                    printf("\nEnter the no of memory slots:");

                    scanf("%d",&no);
                    memconfig(m,no);
                    printf("\nEnter the no of process:");
                    scanf("%d",&n);
                    accept(p,n);
                    display(m,no);
                    best_fit(m,p,n,no);
                    disp(m,p,n,no);
                    break;
            case 4:
                    printf("\nEnter the no of memory slots:");

                    scanf("%d",&no);
                    memconfig(m,no);
                    printf("\nEnter the no of process:");
                    scanf("%d",&n);
                    accept(p,n);
                    display(m,no);
                    worst_fit(m,p,n,no);
                    disp(m,p,n,no);

            }
    }
    while(ch!=5);
}

/*
```

OUTPUT:-

```
**************************

1.FIRST FIT
2.NEXT FIT
3.BEST FIT
4.WORST FIT
EXIT
Enter your chocie:1

Enter the no of memory slots:4
```

```
Enter the partition size:3

Enter the status 0 for free and 1 for occupied:0
Enter the partition size:4

Enter the status 0 for free and 1 for occupied:0
Enter the partition size:2

Enter the status 0 for free and 1 for occupied:1
Enter the partition size:5

Enter the status 0 for free and 1 for occupied:0

Enter the no of process:4

Enter the pid and size of process:2 2

Enter the pid and size of process:1 6

Enter the pid and size of process:3 3

Enter the pid and size of process:4 4
SIZE   STATUS
3      EMPTY
4      EMPTY
2      OCCUPIED
5      EMPTY
No slot for proces with id = 1 and size =6
SIZE   PID    PSIZE STATUS
3      2      2     OCCUPIED
4      3      3     OCCUPIED
2      --     --    OCCUPIED
5      4      4     OCCUPIED
1.FIRST FIT
2.NEXT FIT
3.BEST FIT
4.WORST FIT
EXIT
Enter your chocie:2

Enter the no of memory slots:3
Enter the partition size:3

Enter the status 0 for free and 1 for occupied:0
Enter the partition size:2

Enter the status 0 for free and 1 for occupied:0
Enter the partition size:5

Enter the status 0 for free and 1 for occupied:0
```

```
Enter the no of process:4

Enter the pid and size of process:3 3

Enter the pid and size of process:5 5

Enter the pid and size of process:2 2

Enter the pid and size of process:1 1
SIZE   STATUS
3      EMPTY
2      EMPTY
5      EMPTY
No slot for proces with id = 1 and size =1
SIZE   PID    PSIZE STATUS
3      3      3     OCCUPIED
2      2      2     OCCUPIED
5      5      5     OCCUPIED
1.FIRST FIT
2.NEXT FIT
3.BEST FIT
4.WORST FIT
EXIT
Enter your chocie:3

Enter the no of memory slots:4
Enter the partition size:5

Enter the status 0 for free and 1 for occupied:0
Enter the partition size:2

Enter the status 0 for free and 1 for occupied:0
Enter the partition size:3

Enter the status 0 for free and 1 for occupied:1
Enter the partition size:4

Enter the status 0 for free and 1 for occupied:0

Enter the no of process:3

Enter the pid and size of process:2 2

Enter the pid and size of process:4 4

Enter the pid and size of process:1 1
SIZE   STATUS
5      EMPTY
2      EMPTY
```

```
3       OCCUPIED
4       EMPTY
SIZE    PID     PSIZE STATUS
5       1       1       OCCUPIED
2       2       2       OCCUPIED
3       --      --      OCCUPIED
4       4       4       OCCUPIED
1.FIRST FIT
2.NEXT FIT
3.BEST FIT
4.WORST FIT
5.EXIT
Enter your chocie:4

Enter the no of memory slots:3
Enter the partition size:2

Enter the status 0 for free and 1 for occupied:0
Enter the partition size:3

Enter the status 0 for free and 1 for occupied:0
Enter the partition size:4

Enter the status 0 for free and 1 for occupied:0

Enter the no of process:3

Enter the pid and size of process:2 2

Enter the pid and size of process:3 3

Enter the pid and size of process:4 4
SIZE    STATUS
2       EMPTY
3       EMPTY
4       EMPTY
No slot for proces with id = 4 and size =4
SIZE    PID     PSIZE STATUS
2       --      --      EMPTY
3       3       3       OCCUPIED
4       2       2       OCCUPIED

*/
```

****