**Operating System Lab Syllabus**

**Assignment – 1: Introduction on Linux/Ubuntu OS and Shell.**

**Assignment – 2: Basic Linux Commands.**

**Assignment – 3: Advanced Linux Commands and Working with VI Editor.**

**Assignment – 4: Sample Shell Programs and If…Else Programs.**

**Assignment – 5: Shell Programs on Switch Case, while and for loop.**

**Assignment – 6: Programs on Process Creation and inter process communication.**

**Assignment – 7: Simulation of FCFS and RR CPU Scheduling Algorithm.**

**Assignment – 8: Simulation of SJF and Priority CPU Scheduling Algorithm.**

**Assignment – 9: Simulation Of Banker’s Algorithm for Dealock Avoidance.**

**Assignment – 10: Programs for FIFO and LRU Page replacement Algorithm.**

**Operating System Lab Assignment List**

**Concept explanation: Introduction on Linux/Ubuntu OS and Shell.UNIX SHELL PROGRAMMING:**

A shell is a command-line interpreter and typical operations performed by shell scripts include file manipulation, program execution, and printing text.

The following script uses the **read** command which takes the input from the keyboard and assigns it as the value of the variable PERSON and finally prints it on STDOUT.

# Script follows here:

echo "What is your name?"

read PERSON

echo "Hello, $PERSON"

Here is a sample run of the script −

$./test.sh

What is your name?

MURALI

Hello, MURALI

$

**Shell Prompt**

The prompt, **$**, which is called the **command prompt**, is issued by the shell. While the prompt is displayed, you can type a command.

## Shell Types

In Unix, there are two major types of shells −

1. **Bourne shell** − If you are using a Bourne-type shell, the **$** character is the default prompt.
2. **C shell** − If you are using a C-type shell, the % character is the default prompt.

The Bourne Shell has the following subcategories −

1. Bourne shell (sh)
2. Korn shell (ksh)
3. Bourne Again shell (bash)
4. POSIX shell (sh)

The different C-type shells follow −

1. C shell (csh)
2. TENEX/TOPS C shell (tcsh)

The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by **#**sign, describing the steps.

**Variable:**

 A variable is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data.

## Variable Names

The name of a variable can contain only letters (a to z or A to Z), numbers ( 0 to 9) or the underscore character ( \_).

By convention, Unix shell variables will have their names in UPPERCASE.

The following examples are valid variable names −

\_ALI

TOKEN\_A

VAR\_1

VAR\_2

Following are the examples of invalid variable names −

2\_VAR

-VARIABLE

VAR1-VAR2

VAR\_A!

## Defining Variables

Variables are defined as follows −

variable\_name=variable\_value

For example −

NAME="murali"

The above example defines the variable NAME and assigns the value "Zara Ali" to it. Variables of this type are called **scalar variables**. A scalar variable can hold only one value at a time.

Shell enables you to store any value you want in a variable. For example −

VAR1="murali"

VAR2=100

## Accessing Values

To access the value stored in a variable, prefix its name with the dollar sign (**$**) −

For example, the following script will access the value of defined variable NAME and print it on STDOUT −

!/bin/sh

NAME="murali"

echo $NAME

The above script will produce the following value −

murali

## Read-only Variables

Shell provides a way to mark variables as read-only by using the read-only command. After a variable is marked read-only, its value cannot be changed.

#!/bin/sh

NAME="murali"

readonly NAME

NAME="alok"

The above script will generate the following result −

/bin/sh: NAME: This variable is read only.

## Unsetting Variables

Unsetting or deleting a variable directs the shell to remove the variable from the list of variables that it tracks. Once you unset a variable, you cannot access the stored value in the variable.

example that demonstrates how the command works −

#!/bin/sh

NAME="murali"

unset NAME

echo $NAME

## Arithmetic Operators

The following arithmetic operators are supported by Bourne Shell.

Assume variable **a** holds 10 and variable **b** holds 20 then −

[Show Examples](https://www.tutorialspoint.com/unix/unix-arithmetic-operators.htm)

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| + (Addition) | Adds values on either side of the operator | `expr $a + $b` will give 30 |
| - (Subtraction) | Subtracts right hand operand from left hand operand | `expr $a - $b` will give -10 |
| \* (Multiplication) | Multiplies values on either side of the operator | `expr $a \\* $b` will give 200 |
| / (Division) | Divides left hand operand by right hand operand | `expr $b / $a` will give 2 |
| % (Modulus) | Divides left hand operand by right hand operand and returns remainder | `expr $b % $a` will give 0 |
| = (Assignment) | Assigns right operand in left operand | a = $b would assign value of b into a |
| == (Equality) | Compares two numbers, if both are same then returns true. | [ $a == $b ] would return false. |
| != (Not Equality) | Compares two numbers, if both are different then returns true. | [ $a != $b ] would return true. |

It is very important to understand that all the conditional expressions should be inside square braces with spaces around them, for example **[ $a == $b ]**is correct whereas, **[$a==$b]** is incorrect.

All the arithmetical calculations are done using long integers.

## Relational Operators

Bourne Shell supports the following relational operators that are specific to numeric values. These operators do not work for string values unless their value is numeric.

For example, following operators will work to check a relation between 10 and 20 as well as in between "10" and "20" but not in between "ten" and "twenty".

Assume variable **a** holds 10 and variable **b** holds 20 then −

[Show Examples](https://www.tutorialspoint.com/unix/unix-relational-operators.htm)

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| -eq | Checks if the value of two operands are equal or not; if yes, then the condition becomes true. | [ $a -eq $b ] is not true. |
| -ne | Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true. | [ $a -ne $b ] is true. |
| -gt | Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true. | [ $a -gt $b ] is not true. |
| -lt | Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true. | [ $a -lt $b ] is true. |
| -ge | Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true. | [ $a -ge $b ] is not true. |
| -le | Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true. | [ $a -le $b ] is true. |

It is very important to understand that all the conditional expressions should be placed inside square braces with spaces around them. For example, **[ $a <= $b ]** is correct whereas, **[$a <= $b]** is incorrect.

## Boolean Operators

The following Boolean operators are supported by the Bourne Shell.

Assume variable **a** holds 10 and variable **b** holds 20 then −

[Show Examples](https://www.tutorialspoint.com/unix/unix-boolean-operators.htm)

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| ! | This is logical negation. This inverts a true condition into false and vice versa. | [ ! false ] is true. |
| -o | This is logical OR. If one of the operands is true, then the condition becomes true. | [ $a -lt 20 -o $b -gt 100 ] is true. |
| -a | This is logical AND. If both the operands are true, then the condition becomes true otherwise false. | [ $a -lt 20 -a $b -gt 100 ] is false. |

## String Operators

The following string operators are supported by Bourne Shell.

Assume variable **a** holds "abc" and variable **b** holds "efg" then −

[Show Examples](https://www.tutorialspoint.com/unix/unix-string-operators.htm)

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| = | Checks if the value of two operands are equal or not; if yes, then the condition becomes true. | [ $a = $b ] is not true. |
| != | Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true. | [ $a != $b ] is true. |
| -z | Checks if the given string operand size is zero; if it is zero length, then it returns true. | [ -z $a ] is not true. |
| -n | Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true. | [ -n $a ] is not false. |
| str | Checks if str is not the empty string; if it is empty, then it returns false. | [ $a ] is not false. |

## The if...else statements

**Conditional Statements:** There are total 5 conditional statements which can be used in bash programming

1. if statement
2. if-else statement
3. if..elif..else..fi statement (Else If ladder)
4. if..then..else..if..then..fi..fi..(Nested if)
5. switch statement

Their description with syntax is as follows:

**if statement**  
This block will process if specified condition is true.  
***Syntax:***

if [ expression ]

then

statement

fi

**if-else statement**  
If specified condition is not true in if part then else part will be execute.  
***Syntax***

if [ expression ]

then

statement1

else

statement2

fi

**if..elif..else..fi statement (Else If ladder)**  
To use multiple conditions in one if-else block, then elif keyword is used in shell. If expression1 is true then it executes statement 1 and 2, and this process continues. If none of the condition is true then it processes else part.  
***Syntax***

if [ expression1 ]

then

statement1

statement2

.

elif [ expression2 ]

then

statement3

statement4

.

else

statement5

fi

**if..then..else..if..then..fi..fi..(Nested if)**  
Nested if-else block can be used when, one condition is satisfies then it again checks another condition. In the syntax, if expression1 is false then it processes else part, and again expression2 will be check.  
***Syntax:***

if [ expression1 ]

then

statement1

statement2

.

else

if [ expression2 ]

then

statement3

.

fi

fi

**switch statement**  
case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern  
When a match is found all of the associated statements until the double semicolon (;;) is executed.  
A case will be terminated when the last command is executed.  
If there is no match, the exit status of the case is zero.

***Syntax:***

case in

Pattern 1) Statement 1;;

Pattern n) Statement n;;

esac

Case syntax

case word in

pattern1)

# block of code for pattern1

;;

pattern2)

# block of code for pattern2

;;

\*)

# default block

;;

esac

Where, word is some value that is matched with the pattern1, pattern2 and so on.

If the word matches any pattern then the block of code belonging to that pattern is executed.

The ;; marks the end of the block and takes us out of the case.

The \*) represents the default pattern. If no match is found then the code in the default pattern is executed.

The default pattern \*) is optional and can be omitted.

The esac (reverse of case) marks the end of the case statement.

## Example #2: Write a Shell Script to display greetings

In the following example we will take user name and time of the day as input and display some greetings message.

#!/bin/sh

# take user name

echo "Enter your name:"

read name

# take time of the day

echo "Enter time of the day [Morning/Afternoon/Evening/Night]:"

read daytime

case $daytime in

"Morning")

echo "Good Morning $name"

;;

"Afternoon")

echo "Good Afternoon $name"

;;

"Evening")

echo "Good Evening $name"

;;

"Night")

echo "Good Night $name"

;;

esac

echo "End of script."

Output:

$ sh greetings.sh

Enter your name:

Yusuf Shakeel

Enter time of the day [Morning/Afternoon/Evening/Night]:

Morning

Good Morning Yusuf Shakeel

End of script.

$ sh greetings.sh

Enter your name:

Yusuf Shakeel

Enter time of the day [Morning/Afternoon/Evening/Night]:

End of script.

In the second run we don't get any greetings message because the time of the day was not provided and the default pattern \*) is not present in the case statement to handle such scenario.

So, we can modify the above code to handle such scenario by including the \*)default pattern.

#!/bin/sh

# take user name

echo "Enter your name:"

read name

# take time of the day

echo "Enter time of the day [Morning/Afternoon/Evening/Night]:"

read daytime

case $daytime in

"Morning")

echo "Good Morning $name"

;;

"Afternoon")

echo "Good Afternoon $name"

;;

"Evening")

echo "Good Evening $name"

;;

"Night")

echo "Good Night $name"

;;

\*)

echo "Time of the day missing!"

;;

esac

echo "End of script."

Output:

$ sh greetings-1.sh

Enter your name:

Yusuf Shakeel

Enter time of the day [Morning/Afternoon/Evening/Night]:

Time of the day missing!

End of script.

**Example Programs**

**Example 1:**  
Implementing if statement

filter\_none

brightness\_4

|  |
| --- |
| #Initializing two variables  a=10  b=20    #Check whether they are equal  if [ $a == $b ]  then      echo "a is equal to b"  fi    #Check whether they are not equal  if [ $a != $b ]  then      echo "a is not equal to b"  fi |

**Output**

$bash -f main.sh

a is not equal to b

***Example 2:***  
Implementing if.else statement

filter\_none

brightness\_4

|  |
| --- |
| #Initializing two variables  a=20  b=20    if [ $a == $b ]  then      #If they are equal then print this      echo "a is equal to b"  else      #else print this      echo "a is not equal to b"  fi |

**Output**

$bash -f main.sh

a is equal to b

**Example 3:**  
Implementing switch statement

filter\_none

brightness\_4

|  |
| --- |
| CARS="bmw"    #Pass the variable in string  case "$CARS" in      #case 1      "mercedes") echo "Headquarters - Affalterbach, Germany" ;;        #case 2      "audi") echo "Headquarters - Ingolstadt, Germany" ;;        #case 3      "bmw") echo "Headquarters - Chennai, Tamil Nadu, India" ;;  esac |

**Output**

$bash -f main.sh

Headquarters - Chennai, Tamil Nadu, India.

## Example #3: Write a Shell Script to check if a number is odd, even or zero

We will use the [modulus operator](https://www.dyclassroom.com/unix/shell-programming-arithmetic-operators) % to check if the number is odd or even.

If a number is divisible by 2 and gives no remainder then it is an even number.

#!/bin/sh

# take a numbers from the user

echo "Enter a number: "

read a

# check

if [ $a == 0 ]

then

echo "It's zero."

elif [ `expr $a % 2` == 0 ]

then

echo "It's even."

else

echo "It's odd."

fi

echo "End of script."

Output:

$ sh if-elif-else.sh

Enter a number:

0

It's zero.

End of script.

$ sh if-elif-else.sh

Enter a number:

10

It's even.

End of script.

$ sh if-elif-else.sh

Enter a number:

11

It's odd.

End of script

**About printf**

The printf command is used to print pre-formatted output. And it is similar to the printf() function of [C programming language](https://www.dyclassroom.com/c/c-input-output-operation-using-scanf-and-printf-functions).

We can say that printf is a successor of [echo](https://www.dyclassroom.com/unix/shell-programming-echo) command.

**printf syntax**

printf <format> <arguments>

Where, format is the format string used on the arguments.

format is optional and can be omitted.

In the following example we will take two integer numbers from the user and divide them and show 5 decimal place.

#!/bin/sh

# get the first number

printf "Enter first integer number: "

read a

# get the second number

printf "Enter second integer number: "

read b

# if b is 0

if [ $b == 0 ]

then

printf "Division by 0 not allowed.\n"

exit # exit the script

fi

# perform division

result=`expr "$a / $b" | bc -l`

printf "%d / %d = %.5f\n" "$a" "$b" "$result"

Output:

$ sh example03.sh

Enter first integer number: 5

Enter second integer number: 2

5 / 2 = 2.50000

$ sh example03.sh

Enter first integer number: 5

Enter second integer number: 0

Division by 0 not allowed.

**Example #04**

In the following example we will divide a by b and we will let the user decide the number of decimal places for the result.

#!/bin/sh

# get the first number

printf "Enter first integer number: "

read a

# get the second number

printf "Enter second integer number: "

read b

# get the decimal place

printf "Enter decimal place for the result: "

read d

# if b is 0

if [ $b == 0 ]

then

printf "Division by 0 not allowed.\n"

exit # exit the script

fi

# perform division

result=`expr "$a / $b" | bc -l`

printf "%d / %d = %.\*f\n" "$a" "$b" "$d" "$result"

Output:

$ sh example04.sh

Enter first integer number: 5

Enter second integer number: 2

Enter decimal place for the result: 1

5 / 2 = 2.5

$ sh example04.sh

Enter first integer number: 5

Enter second integer number: 2

Enter decimal place for the result: 3

5 / 2 = 2.500

$ sh example04.sh

Enter first integer number: 5

Enter second integer number: 2

Enter decimal place for the result: 0

5 / 2 = 2

In the above script we have %.\*f in the format string. The \* is replaced by the $dwhich tell us about the number of decimal place in the final result.

**LOOP CONTROLS:**

 the following types of loops available to shell programmers −

* [The while loop](https://www.tutorialspoint.com/unix/while-loop.htm)
* [The for loop](https://www.tutorialspoint.com/unix/for-loop.htm)
* [The until loop](https://www.tutorialspoint.com/unix/until-loop.htm)
* [The select loop](https://www.tutorialspoint.com/unix/select-loop.htm)

## for syntax

for variable in items

do

# code for each item

done

Where, variable holds an item from the list of multiple items.

**Example #1: Write a Shell Script to print from 1 to 5 using for loop**

In the following example we will use for loop to print from 1 to 5.

#!/bin/sh

for i in 1 2 3 4 5

do

echo $i

done

Output:

$ sh example01.sh

1

2

3

4

5

**Brace expansion**

We use the brace expansion {m..n} to generate string in shell script.

Example:

{1..5} will give 1 2 3 4 5

{a..f} will give a b c d e f

{Z..T} will give Z Y X W V U T

{-5..5} will give -5 -4 -3 -2 -1 0 1 2 3 4 5

{A,B,C,D} will give A B C D

{A,B,C{1..3},D} will give A B C1 C2 C3 D

**Example #2: Write a Shell Script to print from 1 to 10 using brace expansion and for loop**

#!/bin/sh

for i in {1..10}

do

echo $i

done

Where, {1..10} will expand to 1 2 3 4 5 6 7 8 9 10.

**Example #3: Write a Shell Script to print from A to Z using for loop**

#!/bin/sh

for ch in {A..Z}

do

echo $ch

done

**Example #4: Write a Shell Script to list all the files in the current directory**

For this example we will use the \* which is a special character and it helps to list all the files in the current directory.

#!/bin/sh

for f in \*

do

echo $f

done

Output:

$ sh example04.sh

example01.sh

example02.sh

example03.sh

example04.sh

**seq command**

We use the seq command to generate numeric sequence.

Example:

seq LAST

so, seq 5 will give

1

2

3

4

5

seq FIRST LAST

so, seq 7 10 will give

7

8

9

10

seq FIRST INCREMENT LAST

so, seq 1 2 10 will give

1

3

5

7

9

**Example #5: Write a Shell Script to print all odd numbers from 1 to 10**

For this we can use the seq command and set FIRST to 1, INCREMENT to 2 and LAST to 10.

#!/bin/sh

for i in $(seq 1 2 10)

do

echo $i

done

Output:

$ sh example05.sh

1

3

5

7

9

**For loop like C programming**

Following is the syntax to create for loop in shell script that looks simiarl to [for loop in C programming](https://www.dyclassroom.com/c/c-for-loop).

for (( var=val; var<=val2; var++ ))

do

# body of for loop

done

**Example #6: Write a Shell Script to print from 1 to 5 using C style for loop**

#!/bin/sh

for(( i = 1; i <= 5; i++ ))

do

echo $i

done

Output:

$ sh example06.sh

1

2

3

4

5

**Nested for loop**

We can nest one for loop inside another.

for var\_outer in list\_outer

do

# outer for loop body

for var\_inner in list\_inner

do

# inner for loop body

done

done

**Example #7: Write a Shell Script to print the following pattern**

1

1 2

1 2 3

1 2 3 4

To achive this we will use nested for loops. The first loop will help to manage the rows and the second for loop will help in printing the numbers per row.

#!/bin/sh

for r in {1..4}

do

for i in $(seq 1 $r)

do

printf "$i "

done

printf "\n"

done

In the above code we are using [printf](https://www.dyclassroom.com/unix/shell-programming-printf) which helps in printing the result in the terminal.

The while loop is similar to a [for loop](https://www.dyclassroom.com/unix/shell-programming-for-loop) and help in executing a block of code multiple times as long as some given condition is satisfied.

## while syntax

while [ condition ]

do

# body of while loop

done

Where condition is some condition which if satisfied results in the execution of the body of the loop.

To come out of the while loop we make the condition fail.

To quit the script we can use the exit command.

**Example #1: Write a Shell Script to print from 1 to 5 using while loop**

#!/bin/sh

# initialise i

i=1

while [ $i -le 5 ]

do

# echo i

echo $i

# update i

i=`expr $i + 1`

done

Output:

$ sh example01.sh

1

2

3

4

5

We can achieve the same result by write the following code.

#!/bin/sh

# initialise i

i=1

while [ $i -le 5 ]

do

# echo i

echo $i

# update i

i=$(( $i + 1 ))

done

**Nested while loop**

We can nest while loop by placing a while loop in the body of another while loop.

while [ condition\_outer ]

do

# body of the outer while loop

while [ condition\_inner ]

do

# body of the inner while loop

done

done

**Example #2: Write a Shell Script to print the following pattern**

1

1 3

1 3 5

1 3 5 7

For this we will use r variable to count the rows and c variable to count the columns. And we will use counter variable to print the number.

#!/bin/sh

# for the rows

r=1

while [ $r -le 4 ]

do

# for the output

count=1

# for the column

c=1

while [ $c -le $r ]

do

# print the value

printf "$count "

# update count

count=$(( $count + 2 ))

# update c

c=$(( $c + 1 ))

done

# go to new line

printf "\n"

# update r

r=$(( $r + 1 ))

done

Output:

$ sh example02.sh

1

1 3

1 3 5

1 3 5 7

**Assignment – 1: Basic Linux Commands.**

**man, date, cal, pwd, who, mkdir, cd, cat, cp, mv, rm, rmdir, wc, touch**

LINUX BASIC COMMANDS:

/ it represnted root directory

it contains all the sub directories like

home, usr, bin,mnt etc

linux prompt : it can be # or $

# is represented when we use admin

$ is represented when we use normal user.

1) pwd :it shows the present working directory and its path

ex: $pwd

2) ls :it shows the list of files and dierctories of present working directory.

ex1: $ls

ex2: $ ls -l - to show in long listing format

ex3: $ ls -la - to show all files and directories including hidden files in long listing format

3) cd : it is for changing directory

ex1: /home/master$cd ..

it is to move from present working directory to its parent directory.

ex2: /home$cd master

it is will allow to move into master directory.

syntax:

$cd </home/....path/present directory>

EX: $cd /home/student/desktop

cd / : to move out of all directories and return to root.

4) man : it will show manual of any command.

ex: $man ls

ex: $ man cd

5) date :it will show date and time of system.

ex: $date

6) cal :it will show calender of present month

ex: $ cal

7) who : it will show the user name logged in

ex: $ who

8) whoami : it will show user name and user id

ex: $ whoami

9) clear : it is used to clear the command prompt.

10) cat :

it is used to create a file.

ex: $cat>file1.txt

892348 8324 298347

jksdfkhjksdf hjksd

^D to save the content

cat is used to see the content of a file :

ex: $ cat file1.txt

892348 8324 298347

jksdfkhjksdf hjksd

cat is ued to append additional data to the end of an existing file:

ex: $ cat>>file1.txt

123 123 123

^ D

$cat file1.txt

892348 8324 298347

jksdfkhjksdf hjksd

123 123 123

11) mkdir: it is used to create a directory.

syntax:

$ mkdir <directory name>

ex:

$mkdir myplace

12) rmdir:it is used to remove an empty directory.

13) rm : it is used to delete a file.

ex: $rm file1.txt

14) touch : it is used to creat an empty file.

ex:

$touch a1.txt

$touch a1.txt a2.txt a3.txt

15) cp : It is used to copy the content of an existing file/directory into a new file/directory.

syntax:

$ cp </path/source filename> <path/new file name>

e.g:

$cp file1.txt f1.txt

16) mv : it is to move a file or directory.

it is also used to rename a file or directory.

ex1: to rename

$mv a1.txt abc1.txt

ex2: to move the file

mv abc1.txt /home/master/sample

17) rm : it is used to delete a file and also to delete a non empty directory.

ex1: to delete a file

$rm abc1.txt

ex2: to delete a non empty directory

$ rm -r sample

**Assignment – 2: Advanced Linux Commands and Working with VI Editor.**

**chmod, du, df, grep, head, tail, sort, echo, read, expr, vi editor**

18) head : it shows the 1st 10 lines of a text file

ex: $head f1.txt

19) tail : it shows the last 10 lines of a text file

ex: $tail f1.txt

20)ps : to displays the presently running processes.

$ps

21) top : it shows all the processes.

22) df : it shows the disk usage information

23) du : it shows the directory space usage

24) chmod : it is used to change mode of a file or directory.

ex1:$ chmod 777 s1.txt

here read represented by 4

write by 2 and execute by 1

when we write 7 means read, write and execute all permissions are required to allocate.

ex2: chmod 731 s1.txt

25) echo : it is used to show message.

ex1: $echo 'hello world'

ex2: x=25

echo $x

ex3: y=123.45

echo $y

ex4: z="it is sample text"

echo $z

26) read: it is used to input data

ex1:

$read x

$echo $x

ex2:

$read x y

$echo $x $y

27) expr : it is a keyword used to represent

calculative expression.

ex1:

$a=25

$b=35

$expr $a + $b

$expr $a - $b

$expr $a \\* $b

$expr $a \ $b

28) wc : it is used for word count in a text file.

ex1:

$ cat>file1.txt

this is a sample text entered by me here.

we are now in ground floor lab for

learning linux commands.

it is our scheduled lab hour maintained

in weekly basis.

master@lab1master:~$ wc file1.txt

5 30 159 file1.txt

29) sort : it is used to sort the lines of a text file in ascending order by default.

ex1:

$ sort names.txt

o/p it shows in ascending order.

by default 1st lower case words and then upper case word.

ex2:

$sort names.txt>nm1.txt

it redirects sotred data into another file.

ex3:

$sort -r names.txt>nm2.txt

it sorts in z to a manner and stored output into nm2.txt

ex4:

sort -n num1.txt

to sort the file numerically.

ex5:

sort -nr num1.txt

to sort the file numerically and in reverse order.

ex6:

sort -nr num1.txt>numrev.txt

ex7:

$sort -k emp.txt

to sort a tabular data as per columns.

master@lab1master:~$ cat>emp.txt

san 3000

manu 4500

mick 2400

alka 3300

tuna 5000

$ sort -k 1n emp.txt : to sort according to 1st col

$ sort -k 2n emp.txt

$sort -rk 1n emp.txt

$sort -rnk 1n emp.txt

ex8:

sort -M months.txt

to sort the content as per months.

30) grep : it is used to search for a pattern in the file. ( globally search for regular expression and print out)

ex1:

grep -i "IS" file1.txt

to search for the word "IS" in the file by case insensitive.

ex2:

grep -c "is" file1.txt

it prints the of no. of words or patterns.

ex3:

grep "akash" names.txt

to search for the string in the file names.txt

ex4:

grep "akash" \*

to search for the string in all files.

ex5:

$grep -n "akash" names.txt

to show the search string along with line number.

ex6:

$grep "^akash" names.txt

to display all the lines which are started with akash

ex7:

$grep "akash$" names.txt

to display all the lines which are ended with akash

31) pipe (|) : it is used to implement one command with another command.

ex1: $ls |wc

ex2: $ls | more

ex3: $ sort -n num1.txt |uniq

ex4: $cat sample.txt|head -7|tail -5

here it prints 1st 7 lines and last 5 lines.

32) text editors :

vi, vim, nano, gedit

gedit: it is a simple text editor to create/update a text file.

**Assignment – 3: Sample Shell Programs**

SHELL PROGRAMMING:

==================

the programming supported by operating system.

\* here we need to write a file and store with .sh as file name.

inside the file we can write one or more commands and can execute at a time.

ex1:

questions:

1. input two numbers and fine addition, subtraction, multiplication and division

solution:

echo "enter a no"

read a

echo "enter 2nd no"

read b

c=`expr $a + $b`

d=`expr $a - $b`

e=`expr $a \\* $b`

f=`expr $a / $b`

g=`expr $a % $b`

echo $c $d $e $f $g

1. input radius and find area, perimeter of a circle.

Solution:

echo "enter radius"

read r

pi=3.14

area=`echo $pi \\* $r \\* $r | bc`

echo "area=$area"

peri=`echo $pi \\* 2 \\* $r | bc`

echo "perimeter=$peri"

1. input two numbers and swap them using 3rd variable.

solution:

1. Swapping of two numbers using and without using third variable.

Solution:

echo "enter 2 nos"

read x

read y

x=`expr $x + $y`

y=`expr $x - $y`

x=`expr $x - $y`

echo "x= " $x

echo "y=" $y

1. Write a program to input P,T,R and calculate simple interest.

Solution:

echo "enter p,r,t"

read p

read r

read t

si=`expr $p \\* $r \\* $t / 100`

echo "SI=$si"

**Assignment – 4: If…Else Programs, case statements**

1. Check the given number is positive or negative using if..else
2. Check for given number is odd or even if..else
3. Find greatest among two unequal numbers using if else statement.  
   solution:

echo "enter 1st no"

read a

echo "enter 2nd no"

read b

if [ $a -eq $b ]

then

echo "both are equal"

else

echo "both are uneuqal"

fi

1. Find greatest among three unequal numbers using elif statement.

Solution:

echo "enter 1st no"

read a

echo "enter 2nd no"

read b

echo "enter 2nd no"

read c

if [ $a -gt $b -a $a -gt $c ]

then

printf "$a is greatest"

elif [ $b -gt $c `]

then

printf "$b is greater"

else

printf "$c is greater"

fi

1. Input an arithmetic operator and two operands. Calculate and find result using el..if

solution:

#find arithmetic value based on operator

echo "input two numbers"

read a b

echo "enter arithmetic operator(+,-,\*,/,%)"

read op

if [ $op = '+' ]

then

echo `expr $a + $b`

elif [ $op = '-' ]

then

echo `expr $a - $b`

elif [ $op = '\*' ]

then

echo `expr $a \\* $b`

elif [ $op = '/' ]

then

echo `expr $a / $b`

elif [ $op = '%' ]

then

echo `expr $a % $b`

else

echo "invalid arithmetic operator"

fi

1. Write a shell program to input a lower case character and check for it is vowel or consonant using case statement.
2. Write a program to input a digit within (0 to 6), display week day as per the given input. Ex: 0 – sunday, 1 – monday, 2-tuesday etc.

**Assignment – 5: Shell Programs on while and for loop.**

1. **Display the first 10 natural numbers.**

n=1

while [ $n -le 10 ]

do

echo $n

n=`expr $n + 1`

done

1. **Factorial of a number.**

echo "enter a number"

read num

fact=1

while [ $num -ge 1 ]

do

fact=`expr $fact\\* $num`

num=’expr $num – 1’

done

echo "factorial of $n is $fact"

1. **Print first 10 of fibonacci series.**

echo "Enter the number"

read n

x= -1

y=1

i=1

echo "Fibonacci Series up to $n terms :"

while [ $i -lt $n ]

do

i=`expr $i + 1 `

z=`expr $x + $y `

echo "$z"

x=$y

y=$z

done

1. **Print the prime numbers between 1 to 100.**

i=3

j=300

flag=0

tem=2

while [ $i -ne $j ]

do

temp=`echo $i`

while [ $temp -ne $tem ]

do

temp=`expr $temp - 1`

n=`expr $i % $temp`

if [ $n -eq 0 -a $flag -eq 0 ]

then

flag=1

fi

done

if [ $flag -eq 0 ]

then

echo $i

else

flag=0

fi

i=`expr $i + 1`

done

**Assignment – 6: Programs on Process Creation and inter process communication.**

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

int main() {

      // make two process which run same

    // program after this instruction

    fork();

      printf("Hello world!\n");

    return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include  <stdio.h>

#include  <sys/types.h>

int main()

{

     fork();

     fork();

     fork();

     printf("hello\n");

     return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(https://www.geeksforgeeks.org/fork-system-call/)

#include <stdio.h> /\* printf, stderr, fprintf \*/

#include <sys/types.h> /\* pid\_t \*/

#include <unistd.h> /\* \_exit, fork \*/

#include <stdlib.h> /\* exit \*/

#include <errno.h> /\* errno \*/

int main(void)

{

pid\_t pid;

/\* Output from both the child and the parent process

\* will be written to the standard output,

\* as they both run at the same time.

\*/

pid = fork();

if (pid == -1)

{

/\* Error:

\* When fork() returns -1, an error happened

\* (for example, number of processes reached the limit).

\*/

fprintf(stderr, "can't fork, error %d\n", errno);

exit(EXIT\_FAILURE);

}

else if (pid == 0)

{

/\* Child process:

\* When fork() returns 0, we are in

\* the child process.

\*/

int j;

for (j = 0; j < 10; j++)

{

printf("child: %d\n", j);

sleep(1);

}

\_exit(0); /\* Note that we do not use exit() \*/

}

else

{

/\* When fork() returns a positive number, we are in the parent process

\* (the fork return value is the PID of the newly created child process)

\* Again we count up to ten.

\*/

int i;

for (i = 0; i < 10; i++)

{

printf("parent: %d\n", i);

sleep(1);

}

exit(0);

}

return 0;

}

(https://stackoverflow.com/questions/15835045/child-process-creation-through-fork-in-c)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Assignment – 7: Simulation of FCFS and RR CPU Scheduling Algorithm.**

**FCFS**

#include<stdio.h>

main()

{

int n,a[10],b[10],t[10],w[10],g[10],i,m;

float att=0,awt=0;

for(i=0;i<10;i++)

{

a[i]=0; b[i]=0; w[i]=0; g[i]=0;

}

printf("enter the number of process");

scanf("%d",&n);

printf("enter the burst times");

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

scanf("%d",&b[i]);

printf("\nenter the arrival times");

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

scanf("%d",&a[i]);

g[0]=0;

for(i=0;i<10;i++)

g[i+1]=g[i]+b[i];

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

{

w[i]=g[i]-a[i];

t[i]=g[i+1]-a[i];

awt=awt+w[i];

att=att+t[i];

}

awt =awt/n;

att=att/n;

printf("\n\tprocess\twaiting time\tturn arround time\n");

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

{

printf("\tp%d\t\t%d\t\t%d\n",i,w[i],t[i]);

}

printf("the average waiting time is %f\n",awt);

printf("the average turn around time is %f\n",att);

}

**OUTPUT:**

enter the number of process 4

enter the burst times

4 9 8 3

enter the arrival times

0 2 4 3

process waiting time turn arround time

p0 0 4

p1 2 11

p2 9 17

p3 18 21

the average waiting time is 7.250000

the average turn around time is 13.250000

(http://meansofmine.blogspot.com/2011/04/c-program-to-implement-fcfsfirst-come.html)

**RR**

#include<stdio.h>

int main()

{

      int i, limit, total = 0, x, counter = 0, time\_quantum;

      int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

      float average\_wait\_time, average\_turnaround\_time;

      printf("\nEnter Total Number of Processes:\t");

      scanf("%d", &limit);

      x = limit;

      for(i = 0; i < limit; i++)

      {

            printf("\nEnter Details of Process[%d]\n", i + 1);

            printf("Arrival Time:\t");

            scanf("%d", &arrival\_time[i]);

            printf("Burst Time:\t");

            scanf("%d", &burst\_time[i]);

            temp[i] = burst\_time[i];

      }

      printf("\nEnter Time Quantum:\t");

      scanf("%d", &time\_quantum);

      printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

      for(total = 0, i = 0; x != 0;)

      {

            if(temp[i] <= time\_quantum && temp[i] > 0)

            {

                  total = total + temp[i];

                  temp[i] = 0;

                  counter = 1;

            }

            else if(temp[i] > 0)

            {

                  temp[i] = temp[i] - time\_quantum;

                  total = total + time\_quantum;

            }

            if(temp[i] == 0 && counter == 1)

            {

                  x--;

                  printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst\_time[i], total - arrival\_time[i],

total - arrival\_time[i] - burst\_time[i]);

                  wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

                  turnaround\_time = turnaround\_time + total - arrival\_time[i];

                  counter = 0;

            }

            if(i == limit - 1)

            {

                  i = 0;

            }

            else if(arrival\_time[i + 1] <= total)

            {

                  i++;

            }

            else

            {

                  i = 0;

            }

      }

      average\_wait\_time = wait\_time \* 1.0 / limit;

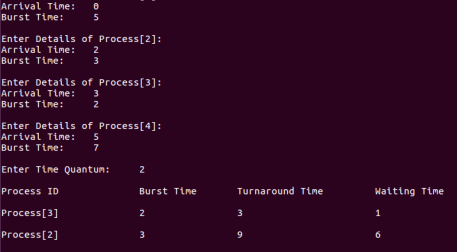
      average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

      printf("\n\nAverage Waiting Time:\t%f", average\_wait\_time);

      printf("\nAvg Turnaround Time:\t%f\n", average\_turnaround\_time);

      return 0;

}



**(http://www.codingalpha.com/round-robin-scheduling-algorithm-c-program/)**

**Assignment – 8: Simulation of SJF and Priority CPU Scheduling Algorithm. (SJF)**

#include<stdio.h>

 void main()

{

    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

    float avg\_wt,avg\_tat;

    printf("Enter number of process:");

    scanf("%d",&n);

    printf("\nEnter Burst Time:\n");

    for(i=0;i<n;i++)

    {

        printf("p%d:",i+1);

        scanf("%d",&bt[i]);

        p[i]=i+1;           //contains process number

    }

    //sorting burst time in ascending order using selection sort

    for(i=0;i<n;i++)

    {

        pos=i;

        for(j=i+1;j<n;j++)

        {

            if(bt[j]<bt[pos])

                pos=j;

        }

        temp=bt[i];

        bt[i]=bt[pos];

        bt[pos]=temp;

        temp=p[i];

        p[i]=p[pos];

        p[pos]=temp;

    }

    wt[0]=0;            //waiting time for first process will be zero

    //calculate waiting time

    for(i=1;i<n;i++)

    {

        wt[i]=0;

        for(j=0;j<i;j++)

            wt[i]+=bt[j];

        total+=wt[i];

    }

    avg\_wt=(float)total/n;      //average waiting time

    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");

    for(i=0;i<n;i++)

    {

        tat[i]=bt[i]+wt[i];     //calculate turnaround time

        total+=tat[i];

        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

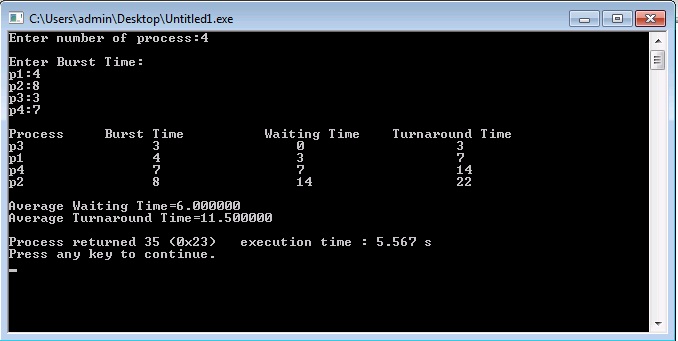
    }

    avg\_tat=(float)total/n;     //average turnaround time

    printf("\n\nAverage Waiting Time=%f",avg\_wt);

    printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}



**(https://www.thecrazyprogrammer.com/2014/08/c-program-for-shortest-job-first-sjf.html)**

**Priority**

#include<stdio.h>

int main()

{

      int burst\_time[20], process[20], waiting\_time[20], turnaround\_time[20], priority[20];

      int i, j, limit, sum = 0, position, temp;

      float average\_wait\_time, average\_turnaround\_time;

      printf("Enter Total Number of Processes:\t");

      scanf("%d", &limit);

      printf("\nEnter Burst Time and Priority For %d Processes\n", limit);

      for(i = 0; i < limit; i++)

      {

            printf("\nProcess[%d]\n", i + 1);

            printf("Process Burst Time:\t");

            scanf("%d", &burst\_time[i]);

            printf("Process Priority:\t");

            scanf("%d", &priority[i]);

            process[i] = i + 1;

      }

      for(i = 0; i < limit; i++)

      {

            position = i;

            for(j = i + 1; j < limit; j++)

            {

                  if(priority[j] < priority[position])

                  {

                        position = j;

                  }

            }

            temp = priority[i];

            priority[i] = priority[position];

            priority[position] = temp;

            temp = burst\_time[i];

            burst\_time[i] = burst\_time[position];

            burst\_time[position] = temp;

            temp = process[i];

            process[i] = process[position];

            process[position] = temp;

      }

      waiting\_time[0] = 0;

      for(i = 1; i < limit; i++)

      {

            waiting\_time[i] = 0;

            for(j = 0; j < i; j++)

            {

                  waiting\_time[i] = waiting\_time[i] + burst\_time[j];

            }

            sum = sum + waiting\_time[i];

      }

      average\_wait\_time = sum / limit;

      sum = 0;

      printf("\nProcess ID\t\tBurst Time\t Waiting Time\t Turnaround Time\n");

      for(i = 0; i < limit; i++)

      {

            turnaround\_time[i] = burst\_time[i] + waiting\_time[i];

            sum = sum + turnaround\_time[i];

            printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\n", process[i], burst\_time[i], waiting\_time[i], turnaround\_time[i]);

      }

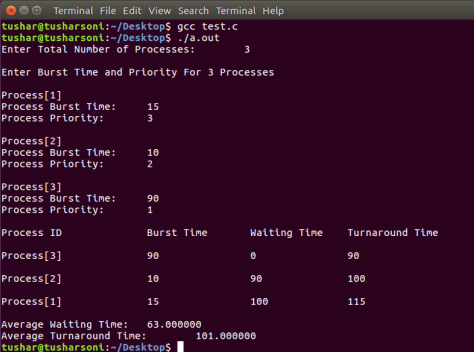
      average\_turnaround\_time = sum / limit;

      printf("\nAverage Waiting Time:\t%f", average\_wait\_time);

      printf("\nAverage Turnaround Time:\t%f\n", average\_turnaround\_time);

      return 0;

}



**(http://www.codingalpha.com/priority-scheduling-algorithm-c-program/)**

**Assignment – 9: Simulation Of Banker’s Algorithm for Dealock Avoidance.**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];

int p, r, i, j, process, count;

count = 0;

printf("Enter the no of processes : ");

scanf("%d", &p);

for(i = 0; i< p; i++)

completed[i] = 0;

printf("\n\nEnter the no of resources : ");

scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");

for(i = 0; i < p; i++)

{

printf("\nFor process %d : ", i + 1);

for(j = 0; j < r; j++)

scanf("%d", &Max[i][j]);

}

printf("\n\nEnter the allocation for each process : ");

for(i = 0; i < p; i++)

{

printf("\nFor process %d : ",i + 1);

for(j = 0; j < r; j++)

scanf("%d", &alloc[i][j]);

}

printf("\n\nEnter the Available Resources : ");

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];

do

{

printf("\n Max matrix:\tAllocation matrix:\n");

for(i = 0; i < p; i++)

{

for( j = 0; j < r; j++)

printf("%d ", Max[i][j]);

printf("\t\t");

for( j = 0; j < r; j++)

printf("%d ", alloc[i][j]);

printf("\n");

}

process = -1;

for(i = 0; i < p; i++)

{

if(completed[i] == 0)//if not completed

{

process = i ;

for(j = 0; j < r; j++)

{

if(avail[j] < need[i][j])

{

process = -1;

break;

}

}

}

if(process != -1)

break;

}

if(process != -1)

{

printf("\nProcess %d runs to completion!", process + 1);

safeSequence[count] = process + 1;

count++;

for(j = 0; j < r; j++)

{

avail[j] += alloc[process][j];

alloc[process][j] = 0;

Max[process][j] = 0;

completed[process] = 1;

}

}

}

while(count != p && process != -1);

if(count == p)

{

printf("\nThe system is in a safe state!!\n");

printf("Safe Sequence : < ");

for( i = 0; i < p; i++)

printf("%d ", safeSequence[i]);

printf(">\n");

}

else

printf("\nThe system is in an unsafe state!!");

}

Enter the no of processes : 5

Enter the no of resources : 3

Enter the Max Matrix for each process :

For process 1 : 7

5

3

For process 2 : 3

2

2

For process 3 : 7

0

2

For process 4 : 2

2

2

For process 5 : 4

3

3

Enter the allocation for each process :

For process 1 : 0

1

0

For process 2 : 2

0

0

For process 3 : 3

0

2

For process 4 : 2

1

1

For process 5 : 0

0

2

Enter the Available Resources : 3

3

2

Max matrix: Allocation matrix:

7 5 3 0 1 0

3 2 2 2 0 0

7 0 2 3 0 2

2 2 2 2 1 1

4 3 3 0 0 2

Process 2 runs to completion!

Max matrix: Allocation matrix:

7 5 3 0 1 0

0 0 0 0 0 0

7 0 2 3 0 2

2 2 2 2 1 1

4 3 3 0 0 2

Process 3 runs to completion!

Max matrix: Allocation matrix:

7 5 3 0 1 0

0 0 0 0 0 0

0 0 0 0 0 0

2 2 2 2 1 1

4 3 3 0 0 2

Process 4 runs to completion!

Max matrix: Allocation matrix:

7 5 3 0 1 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

4 3 3 0 0 2

Process 1 runs to completion!

Max matrix: Allocation matrix:

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

4 3 3 0 0 2

Process 5 runs to completion!

The system is in a safe state!!

Safe Sequence : < 2 3 4 1 5 >

**(https://stackoverflow.com/questions/15501861/bankers-algorithm-for-deadlock-avoidance-in-c)**

**Assignment – 10: Programs for FIFO and LRU Page replacement Algorithm.**

**FIFO REPLACEMENT:**

#include<stdio.h>  
int main()  
{  
int i,j,n,a[50],frame[10],no,k,avail,count=0;  
printf(“\n ENTER THE NUMBER OF PAGES:\n”);  
scanf(“%d”,&n);  
printf(“\n ENTER THE PAGE NUMBER :\n”);  
for(i=1;i<=n;i++)  
scanf(“%d”,&a[i]);  
printf(“\n ENTER THE NUMBER OF FRAMES :”);  
scanf(“%d”,&no);  
for(i=0;i<no;i++)  
frame[i]= -1;  
j=0;  
printf(“\tref string\t page frames\n”);  
for(i=1;i<=n;i++)  
{  
printf(“%d\t\t”,a[i]);  
avail=0;  
for(k=0;k<no;k++)  
if(frame[k]==a[i])  
avail=1;  
if (avail==0)  
{  
frame[j]=a[i];  
j=(j+1)%no;  
count++;  
for(k=0;k<no;k++)  
printf(“%d\t”,frame[k]);  
}  
printf(“\n”);  
}  
printf(“Page Fault Is %d”,count);  
return 0;  
}

###### /\*

###### OUTPUT:

ENTER THE NUMBER OF PAGES: 20  
ENTER THE PAGE NUMBER : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1  
ENTER THE NUMBER OF FRAMES :3  
ref string page frames  
7 7 -1 -1  
0 7 0 -1  
1 7 0 1  
2 2 0 1  
0  
3 2 3 1  
0 2 3 0  
4 4 3 0  
2 4 2 0  
3 4 2 3  
0 0 2 3  
3  
2  
1 0 1 3  
2 0 1 2  
0  
1  
7 7 1 2  
0 7 0 2  
1 7 0 1  
Page Fault Is 15

\*/

**LRU REPLACEMENT:**

**ALGORITHM :**

1. Start the process

2. Declare the size

3. Get the number of pages to be inserted

4. Get the value

5. Declare counter and stack

6. Select the least recently used page by counter value

7. Stack them according the selection.

8. Display the values

9. Stop the process

**PROGRAM:**

#include<stdio.h>

main()

{

int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];

printf("Enter no of pages:");

scanf("%d",&n);

printf("Enter the reference string:");

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

scanf("%d",&p[i]);

printf("Enter no of frames:");

scanf("%d",&f);

q[k]=p[k];

printf("\n\t%d\n",q[k]);

c++;

k++;

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

{

c1=0;

for(j=0;j<f;j++)

{

if(p[i]!=q[j])

c1++;

}

if(c1==f)

{

c++;

if(k<f)

{

q[k]=p[i];

k++;

for(j=0;j<k;j++)

printf("\t%d",q[j]);

printf("\n");

}

else

{

for(r=0;r<f;r++)

{

c2[r]=0;

for(j=i-1;j<n;j--)

{

if(q[r]!=p[j])

c2[r]++;

else

break;

}

}

for(r=0;r<f;r++)

b[r]=c2[r];

for(r=0;r<f;r++)

{

for(j=r;j<f;j++)

{

if(b[r]<b[j])

{

t=b[r];

b[r]=b[j];

b[j]=t;

}

}

}

for(r=0;r<f;r++)

{

if(c2[r]==b[0])

q[r]=p[i];

printf("\t%d",q[r]);

}

printf("\n");

}

}

}

printf("\nThe no of page faults is %d",c);

}

**OUTPUT:**

Enter no of pages:10

Enter the reference string:7 5 9 4 3 7 9 6 2 1

Enter no of frames:3

7

7 5

7 5 9

4 5 9

4 3 9

4 3 7

9 3 7

9 6 7

9 6 2

1 6 2

The no of page faults is 10