**Experiment No-1**

**Title :** To explore basic commands for handling file system under Unix/Linux using shell script ( creating groups , chown , chmod , directory name, tty , diff, umask )

**References :**

1. “Operating Systems Concepts”, Silbershatz,Peterson, Galvin,

Addison Wesley 2nd Edition

2. “Modern Operating Systems”,Tannenbaum,

Eastern Economy edition , 2nd Edition Year 1995

**Pre-requisite:**

Knowledge of shell script

**Description:**

**groupadd - Create a new group**

**groupadd** [**-g***gid*[**-o**]] [**-r**] [**-f**] *group*

## DESCRIPTION

The groupadd command creates a new group account using the values specified on the command line and the default values from the system. The new group will be entered into the system files as needed. The options which apply to the groupadd command are

-g gid

The numerical value of the group's ID. This value must be unique, unless the -o option is used. The value must be non-negative. The default is to use the smallest ID value greater than 500 and greater than every other group. Values between 0 and 499 are typically reserved for system accounts

-r

This flag instructs groupadd to add a system account. The first available gid lower than 499 will be automatically selected unless the -g option is also given on the command line. 

-f

This is the force flag. This will cause groupadd to exit with an error when the group about to be added already exists on the system. If that is the case, the group won't be altered (or added again).

**chown command**

The chown command changes the [owner](http://www.computerhope.com/jargon/o/owner.htm) and owning group of files.

**Syntax**

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

**chown [*OPTION*]... --reference=*RFILE* *FILE*...**

**chown** owner-user **file**

**Description**

chown changes the user and/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](http://www.computerhope.com/unix/uchgrp.htm). If only a colon is given, or if the entire operand is empty, neither the owner nor the group is changed.

**chown** owner-user:owner-group **file**

**chown** owner-user:owner-group directory

**chown** options owner-user:owner-group **file**

In this example change file ownership to vivek user and list the permissions, run:  
# chown vivek demo.txt  
# ls -l demo.txt

**Chmod**

Changes the [permissions](http://www.computerhope.com/jargon/p/permissi.htm) of a file.

## Syntax

chmod [*OPTION*]... *MODE*[,*MODE*]... *FILE*...

chmod [*OPTION*]... *OCTAL-MODE* *FILE*...

chmod [*OPTION*]... --reference=*RFILE FILE*...

Description

chmod changes the file mode [bits](http://www.computerhope.com/jargon/b/bit.htm) of each given file according to *mode*, which can be either a symbolic representation of changes to make, or an [octal](http://www.computerhope.com/jargon/o/octal.htm) number representing the bit pattern for the new mode bits.

The format of a symbolic mode is

[ugoa...][[+-=][*perms*...]...]

where *perms* is either zero or more letters from the set rwxXst, or a single letter from the set ugo. Multiple symbolic modes can be given, separated by commas.

A combination of the letters ugoa controls which users' access to the file will be changed: the user who owns it (u), other users in the file's group (g), other users not in the file's group (o), or all users (a). If none of these are given, the effect is as if a were given, but bits that are set in the [umask](http://www.computerhope.com/unix/uumask.htm) are not affected.

The operator + causes the selected file mode bits to be added to the existing file mode bits of each file; - causes them to be removed; and = causes them to be added and causes unmentioned bits to be removed except that a directory's unmentioned set user and group ID bits are not affected.

**Directory and file commands**

|  |  |
| --- | --- |
| cd /home | enter to directory '/ home'   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cd .. | go back one level   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cd ../.. | go back two levels   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cd | go to home directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cd ~user1 | go to home directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cd - | go to previous directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cd)] |
| # cp file1 file2 | copying a file   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cp)] |
| # cp dir/\* . | copy all files of a directory within the current work directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cp)] |
| # cp -a /tmp/dir1 . | copy a directory within the current work directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cp)] |
| # cp -a dir1 dir2 | copy a directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=cp)] |
| # cp file file1 | outputs the mime type of the file as text   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=file)] |
| # iconv –l | lists known encodings   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=iconv)] |
| # iconv -f fromEncoding -t toEncoding inputFile > outputFile | converting the coding of characters from one format to another   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=iconv)] |
| # find . -maxdepth 1 -name \*.jpg -print -exec convert | batch resize files in the current directory and send them to a thumbnails directory (requires convert from Imagemagick)   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=iconv)] |
| # ln -s file1 lnk1 | create a symbolic link to file or directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ln)] |
| # ln file1 lnk1 | create a physical link to file or directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ln)] |
| # ls | view files of directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ls)] |
| # ls –F | view files of directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ls)] |
| # ls –l | show details of files and directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ls)] |
| # ls –a | show hidden files   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ls)] |
| # ls \*[0-9]\* | show files and directory containing numbers   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=ls)] |
| # lstree | show files and directories in a tree starting from root(2)   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=lstree)] |
| # mkdir dir1 | create a directory called 'dir1'   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=mkdir)] |
| # mkdir dir1 dir2 | create two directories simultaneously   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=mkdir)] |
| # mkdir -p /tmp/dir1/dir2 | create a directory tree   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=mkdir)] |
| # mv dir1 new\_dir | rename / move a file or directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=mv)] |
| # pwd | show the path of work directory   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=pwd)] |
| # rm -f file1 | delete file called 'file1'   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=rm)] |
| # rm -rf dir1 | remove a directory called 'dir1' and contents recursively   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=rm)] |
| # rm -rf dir1 dir2 | remove two directories and their contents recursively   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=rm)] |
| # rmdir dir1 | delete directory called 'dir1'   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=rmdir)] |
| # touch -t 0712250000 file1 | modify timestamp of a file or directory - (YYMMDDhhmm)   [[man](http://www.linuxguide.it/command_line/linux-manpage/do.php?file=touch)] |
| # tree | show files and direct |

**Tty Command**

Print the file name of the terminal connected to standard input

**SYNOPSIS**

**tty** [*OPTION*]...

**DESCRIPTION**

Print the file name of the terminal connected to standard input.

**-s, --silent, --quiet**

print nothing, only return an exit status

**--help**

display this help and exit

**--version**

output version information and exit

**Diff Command**

Displays two files and prints the lines that are different.

**Syntax**

diff [*OPTION*]... *FILES*

**Examples**

diff help dir2

Compares the directory named help with the directory named dir2. Below is an example of the output when running this command:

Only in help: tab2.gif

Only in help: tab3.gif

Only in help: tab4.gif

Only in help: tape.htm

Only in help: tbernoul.htm

Only in help: tconner.htm

Only in help: tempbus.psd

**Umask command**

Return, or set, the value of the system's file mode creation mask.

**Syntax**

umask [-S] [*mask definition*]

**Description**

UMASK (User Mask or User file creation MASK) is the default permission or base permissions given when a new file (even folder too, as Linux treats everything as files) is created on a Linux machine. Most of the Linux distros give 022 (0022) as default UMASK. In other words, it is a system default permissions for newly created files/folders in the machine.

Though umask value is the same for files and folders, but calculation of File base permissions and Directory base permissions are different.

The minimum and maximum UMASK value for a folder is 000 and 777

The minimum and maximum UMASK value for a file is 000 and 666

**Example-**

umask

umask -S

**Experiment No-2**

**Title :** Using Pattern matching utilities like awk, grep , nroff , troff , sort etc.

**References :**

1. “Operating Systems Concepts”, Silbershatz,Peterson, Galvin,

Addison Wesley 2nd Edition

2. “Modern Operating Systems”,Tannenbaum,

Eastern Economy edition , 2nd Edition Year 1995

**Pre-requisite :**

Knowledge of basic linux commands

**Description:**

$ cat file

Medicine,200

Grocery,500

Rent,900

Grocery,800

Medicine,600

**1**. To **print only the records containing** Rent:

$ awk '$0 ~ /Rent/{print}' file

Rent,900

     ~ is the symbol used for pattern matching.  The / / symbols are used to specify the pattern. The above line indicates: If the line($0) contains(~) the pattern *Rent*, print the line. 'print' statement by default prints the entire line. This is actually the simulation of grep command using awk.  
  
**2**. awk, while doing pattern matching, by default does on the entire line, and hence $0 can be left off as shown below:

$ awk '/Rent/{print}' file

Rent,900

**3**. Since awk prints the line by default on a true condition, print statement can also be left off.

$ awk '/Rent/' file

Rent,900

**4**. **To match a pattern only in the first column**($1),

$ awk -F, '$1 ~ /Rent/' file

Rent,900

      The -F option in awk is used to specify the delimiter. It is needed here since we are going to work on the specific columns which can be retrieved only when the delimiter is known.  
  
**5.** **To match exactly for the word** "Rent" in the first column:

$ awk -F, '$1=="Rent"' file

Rent,900

**6**. To **print only the 2nd column** for all "Medicine" records:

$ awk -F, '$1 == "Medicine"{print $2}' file

200

600

**7.** To **match for patterns "Rent" or "Medicine"** in the file:

$ awk '/Rent|Medicine/' file

Medicine,200

Rent,900

Medicine,600

**8**. Similarly, to **match** for this above pattern **only in the first colum**n:

$ awk -F, '$1 ~ /Rent|Medicine/' file

Medicine,200

Rent,900

Medicine,600

**9.** What if the the first column contains the word "Medicine**s**". The above example will match it as well. In order **to exactly match only for *Rent* or *Medicine***,

$ awk -F, '$1 ~ /^Rent$|^Medicine$/' file

Medicine,200

Rent,900

Medicine,600

 The ^ symbol indicates beginning of the line, $ indicates the end of the line. ^Rent$ matches exactly for the word *Rent* in the first column, and the same is for the word *Medicine* as well.  
**10**. To **print the lines which does not contain the pattern** *Medicine*:

$ awk '!/Medicine/' file

Grocery,500

Rent,900

Grocery,800

    The ! is used to negate the pattern search.  
  
**11**. To **negate the pattern** only on the first column alone:

$ awk -F, '$1 !~ /Medicine/' file

Grocery,500

Rent,900

Grocery,800

**12.** To **print all records whose amount is greater than** *500*:

$ awk -F, '$2>500' file

Rent,900

Grocery,800

Medicine,600

**13**. To print the *Medicine* record only if it is the 1st record:

$ awk 'NR==1 && /Medicine/' file

Medicine,200

This is how the logical AND(&&) condition is used in awk.  The records needed to be retrieved is only if it is the first record(NR==1) and the record is a medicine record.  
  
**14.** To print all those *Medicine* records whose amount is greater than *500*:

$ awk -F, '/Medicine/ && $2>500' file

Medicine,600

**15**. To print all the *Medicine* records and also those records whose amount is greater than *600*:

$ awk -F, '/Medicine/ || $2>600' file

Medicine,200

Rent,900

Grocery,800

Medicine,600

GREP

First create the following demo\_file that will be used in the examples below to demonstrate grep command.

**$ cat demo\_file**

**THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.**

**this line is the 1st lower case line in this file.**

**This Line Has All Its First Character Of The Word With Upper Case.**

**Two lines above this line is empty.**

**And this is the last line.**

**1. Search for the given string in a single file**

The basic usage of grep command is to search for a specific string in the specified file as shown below.

Syntax:

grep "literal\_string" filename

$ grep "this" demo\_file

this line is the 1st lower case line in this file.

Two lines above this line is empty.

And this is the last line.

**2. Checking for the given string in multiple files.**

Syntax:

grep "string" FILE\_PATTERN

$ cp demo\_file demo\_file1

$ grep "this" demo\_\*

demo\_file:this line is the 1st lower case line in this file.

demo\_file:Two lines above this line is empty.

demo\_file:And this is the last line.

demo\_file1:this line is the 1st lower case line in this file.

demo\_file1:Two lines above this line is empty.

demo\_file1:And this is the last line.

### 3. Case insensitive search using grep -i

Syntax:

grep -i "string" FILE

$ grep -i "the" demo\_file

THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.

this line is the 1st lower case line in this file.

This Line Has All Its First Character Of The Word With Upper Case.

And this is the last line.

### 4. Match regular expression in files

Syntax:

grep "REGEX" filename

$ grep "lines.\*empty" demo\_file

Two lines above this line is empty.

### 5. Checking for full words, not for sub-strings using grep -w

If you want to search for a word, and to avoid it to match the substrings use -w option. Just doing out a normal search will show out all the lines.  
  
$ grep -i "is" demo\_file

THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.

this line is the 1st lower case line in this file.

This Line Has All Its First Character Of The Word With Upper Case.

Two lines above this line is empty.

And this is the last line.

$ grep -iw "is" demo\_file

THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.

this line is the 1st lower case line in this file.

Two lines above this line is empty.

And this is the last line.

### 6. Displaying lines before/after/around the match using grep -A, -B and -C

$ cat demo\_text

4. Vim Word Navigation

You may want to do several navigation in relation to the words, such as:

\* e - go to the end of the current word.

\* E - go to the end of the current WORD.

\* b - go to the previous (before) word.

\* B - go to the previous (before) WORD.

\* w - go to the next word.

\* W - go to the next WORD.

WORD - WORD consists of a sequence of non-blank characters, separated with white space.

word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

\* 192.168.1.1 - seven words.

#### 6.1 Display N lines after match

The following example prints the matched line, along with the 3 lines after it.

$ grep -A 3 -i "example" demo\_text

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

\* 192.168.1.1 - seven words.

#### 6.2 Display N lines before match

$ grep -B 2 "single WORD" demo\_text

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

#### 6.3 Display N lines around match

$ grep -C 2 "Example" demo\_text

word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

### 7. Invert match using grep -v

$ grep -v "go" demo\_text

4. Vim Word Navigation

You may want to do several navigation in relation to the words, such as:

WORD - WORD consists of a sequence of non-blank characters, separated with white space.

word - word consists of a sequence of letters, digits and underscores.

Example to show the difference between WORD and word

\* 192.168.1.1 - single WORD

\* 192.168.1.1 - seven words.

### 8. display the lines which does not matches all the given pattern.

Syntax:

grep -v -e "pattern" -e "pattern"

$ cat test-file.txt

a

b

c

$ grep -v -e "a" -e "b" test-file.txt

c

### 9. Counting the number of matches using grep -c

Syntax:

grep -c "pattern" filename

$ grep -c "go" demo\_text

6

When you want do find out how many lines matches the pattern

$ grep -c this demo\_file

3

When you want do find out how many lines that does not match the pattern

$ grep -v -c this demo\_file

4

### 10. Display only the file names which matches the given pattern using grep -l

$ grep -l this demo\_\*

demo\_file

demo\_file1

### 11. Show only the matched string

$ grep -o "is.\*line" demo\_file

is line is the 1st lower case line

is line

is is the last line

### 12. Show line number while displaying the output using grep -n

To show the line number of file with the line matched. It does 1-based line numbering for each file. Use -n option to utilize this feature.

$ grep -n "go" demo\_text

5: \* e - go to the end of the current word.

6: \* E - go to the end of the current WORD.

7: \* b - go to the previous (before) word.

8: \* B - go to the previous (before) WORD.

9: \* w - go to the next word.

10: \* W - go to the next WORD.

## About nroff

Formats documents for display or line-printer.

## Syntax

nroff [-e] [-h] [-i] [-q] [-m*name*] [-n*N*] [-o*pagelist*] [-r*aN*] [-s*N*] [-T*name*]

## Examples

nroff -s4 -me users.guide

Formats **users.guide** using the **-me** [macro](http://www.computerhope.com/jargon/m/macro.htm) package.

## About troff

**troff** performs typesetting functions and formats documents. It is the major component of the document processing system developed by [AT&T](http://www.computerhope.com/comp/att.htm) for [Unix](http://www.computerhope.com/jargon/u/unix.htm).

## Syntax

troff [-abcivzCERU] [-d *cs*] [-f *fam*] [-F *dir*] [-I *dir*] [-m *name*] [-M *dir*]

[-n *num*] [-o *list*] [-r *cn*] [-T *name*] [-w *name*] [-W *name*] [*file* ...]

## Examples

troff myfile

Process file **myfile**.

**sort** command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ascii. Using options in sort command, it can also be used to sort numerically. Let us discuss it with some examples:  
  
**File with Ascii data:**  
 Let us consider a file with the following contents:

$ cat file

Unix

Linux

Solaris

AIX

Linux

HPUX

**1. sort simply sorts the file in alphabetical order**:

$ sort file

AIX

HPUX

Linux

Linux

Solaris

Unix

**2. sort removes the duplicates** using the -u option:

$ sort -u file

AIX

HPUX

Linux

Solaris

Unix

**File with numbers**:  
 Let us consider a file with numbers:

$ cat file

20

19

5

49

200

**3.** The default sort 'might' give incorrect result on a file containing numbers:

$ sort file

19

20

200

49

5

In the above result, 200 got placed immediately below 20, not at the end which is incorrect. This is because the sort did  ASCII sort. If the file had not contained '200', the default sort would have given proper result. However, it is incorrect to sort a numerical file in this way since the sorting logic is incorrect.  
  
**4. To sort a file numericallly**:

$ sort -n file

5

19

20

49

200

   -n option can sort the decimal numbers as well.  
  
**5. sort file numerically in reverse order**:

$ sort -nr file

200

49

20

19

5

'r' option does a reverse sort.  
  
**Multiple Files**:  
  
**6. sort can sort multiple files** as well.

$ sort -n file1 file2

5

5

18

19

20

25

48

49

200

200

The result of sort with multiple files will be a sorted and merged output of the multiple files.  
  
**7. Sort, merge and remove duplicates**:

$ sort -nu file1 file2

5

18

19

20

25

48

49

200

-u option becomes more handy in case of multiple files. With this, the output is now sorted, merged and without duplicate records.  
  
**Files with multiple fields and delimiter**:  
 Let us consider a file with multiple fields:

$ cat file

Linux,20

Unix,30

AIX,25

Linux,25

Solaris,10

HPUX,100

**8. sorting a file containing multiple fields**:

$ sort file

AIX,25

HPUX,100

Linux,20

Linux,25

Solaris,10

Unix,30

As shown above, the file got sorted on the 1st field, by default.  
  
**9. sort file on the basis of 1st field**:

$ sort -t"," -k1,1 file

AIX,25

HPUX,100

Linux,20

Linux,25

Solaris,10

Unix,30

**10. sorting file on the basis of the 2nd field**:

$ sort -t"," -k2,2 file

Solaris,10

HPUX,100

Linux,20

AIX,25

Linux,25

Unix,30

**11. sorting file on the basis of 2nd field , numerically**:

$ sort -t"," -k2n,2 file

Solaris,10

Linux,20

AIX,25

Linux,25

Unix,30

HPUX,100

**12. Remove duplicates from the file based on 1st field**:

$ sort -t"," -k1,1 -u file

AIX,25

HPUX,100

Linux,20

Solaris,10

Unix,30

**13. Sort the file numerically on the 2nd field in reverse order**:

$ sort -t"," -k2nr,2 file

HPUX,100

Unix,30

AIX,25

Linux,25

Linux,20

Solaris,10

**14. sort the file alphabetically on the 1st field, numerically on the 2nd field**:

$ sort -t"," -k1,1 -k2n,2 file

AIX,25

HPUX,100

Linux,20

Linux,25

Solaris,10

Unix,30

**Experiment No-3**

3. Exploring the boot process of Unix/Linux and implementing practical on it (for ex.

MBR, passing different parameter to kernel, do different activity while booting and

poweroff).

**Experiment No-4**

**Title :** To write a program to implement the following process scheduling algorithms

1)First Come First Serve

2)Shortest Job First

3)Round Robin

4) Priority Scheduling

5) Multilevel Queue

**References :**

1. “Operating Systems Concepts”, Silbershatz,Peterson, Galvin,

Addison Wesley 2nd Edition

2. “Modern Operating Systems”,Tannenbaum,

Eastern Economy edition , 2nd Edition Year 1995

**Pre-requisite :**

Knowledge of Scheduling Policies.

**Description:**

Scheduling algorithms are used when more than one process is executable and the OS has to decide which one to run first.

Terms used

1)Submit time :The process at which the process is given to CPU

2)Burst time : The amount of time each process takes for execution

3)Response time :The difference between the time when the process

starts execution and the submit time.

4)Turnaround time :The difference between the time when the

process completes execution and the submit time.

First Come First Serve(FCFS)

The processes are executed in the order in which they have been submitted.

Shortest Job First(SJF)

The processes are checked at each arrival time and the process which have the shortest remaining burst time at that moment gets executed first. This is non-preemptive algorithm.

Shortest Remaining Time Next(SRTN)

This is preemptive version of SJF. In this scheduling algorithm, the [process](http://en.wikipedia.org/wiki/Process_(computing)) with the smallest amount of time remaining until completion is selected to execute.

Round Robin

Each process is assigned a time interval called its quantum(time slice)

If the process is still running at the end of the quantum the CPU is preempted and given to another process, and this continues in circular fashion, till all the processes are completely executed.

Priority Scheduling

Each process is assigned a priority and executable process with highest priority is allowed to run

Multilevel Queue Scheduling

There will be 'n' number of queues, where 'n' is the number of groups the processes are classified into. Each queue will be assigned a priority and will have its own scheduling algorithm. For the process in a queue to execuete, all the queues of priority higher than it should be empty,

**Conclusion :** Various process scheduling algorithms have been studied successfully.

**Post Lab Assignment:** Comparative Assessment of various Scheduling Policies.

**Experiment No-5A**

**Title :** write a program to implement producer consumer problem

( Using POSIX semaphores)

**Objective :** Solving producer – consumer problem using POSIX semaphore.

**References :**

1. Unix Network Programming By Richard Steven
2. Modern operating system by Tenenbaum

**Pre-requisite :** Producer consumer problem

**Description :**

The producer-consumer problem (Also called the bounded-buffer problem.) illustrates the need for synchronization in systems where many processes share a resource. In the problem, two processes share a fixed-size buffer. One process(producer) produces information and puts it in the buffer, while the other process (consumer) consumes information from the buffer. These processes do not take turns accessing the buffer, they both work concurrently. Herein lies the problem. What happens if the producer tries to put an item into a full buffer? What happens if the consumer tries to take an item from an empty buffer?

In order to synchronize these processes, we will block the producer when the buffer is full, and we will block the consumer when the buffer is empty. So the two processes, Producer and Consumer, should work as follows:

**Algorithm :**

Assuming there are total N number of slots.

Initialization: semaphores: mutex = 1; Full = 0 , empty = N;

integers : in = 0; out = 0;

producer :

Repeat for ever

produce (item);

wait(empty);

wait(mutex);

enter\_item(item);

signal( mutex );

signal( full );

Consumer :

Repeat forever

wait(full );

wait( mutex );

remove\_tem(item);

signal( mutex );

signal(empty);

POSIX : POSIX stands for Portable Operating System Interface

**sem\_init()**

Initializes a semaphore .

#include <semaphore.h>

int sem\_init(sem\_t \*sem, int pshared, unsigned int value);

* First argument is the pointer to semaphore , that you want to initialize. sem\_init initializes the semaphore object pointed to by sem.
* The pshared argument indicates whether the semaphore is local to the current process ( pshared is zero) or is to be shared between several processes ( pshared is not zero). LinuxThreads currently does not support process-shared semaphores, thus sem\_init always returns with error ENOSYS if pshared is not zero.
* Third argument is the value of the semaphore. The count associated with the semaphore is set initially to value.

**pthread\_create()**

#include <pthread.h>

int pthread\_create(pthread\_t \* thread, pthread\_attr\_t \* attr, void \*(\*start\_routine)(void \*), void \* arg);

pthread\_create creates a new thread of control that executes concurrently with the calling thread. The new thread applies the function start\_routine passing it arg as first argument. The new thread terminates either explicitly, by calling pthread\_exit(3), or implicitly, by returning from the start\_routine function. The latter case is equivalent to calling pthread\_exit(3) with the result returned by start\_routine as exit code.

The attr argument specifies thread attributes to be applied to the new thread.

**pthread\_join()**

#include <pthread.h>

int pthread\_join(pthread\_t th, void \*\*thread\_return);

pthread\_join suspends the execution of the calling thread until the thread identified by **th** terminates, either by calling pthread\_exit(3) or by being cancelled. If thread\_return is not NULL, the return value of th is stored in the location pointed to by thread\_return. The return value of th is either the argument it gave to pthread\_exit(3), or PTHREAD\_CANCELED if th was cancelled.

The joined thread th must be in the joinable state.

**Conclusion :** Producer consumer problem is solved successfully using POSIX threads and semaphore.

**Post Lab Assignment:**

1. What is race condition ?
2. What is multi-threading ? write advantages of multi-threading

**Experiment No-5B**

**Title :** To study Dinning Philosophers Problem.

**Objective:** To understand How to avoid Starvation

**References :** 1. “Modern Operating Systems”,Tannenbaum,

2. Eastern Economy edition , 2nd Edition Year 1995

**Pre-requisite:** Concept of semaphore,System calls related to semaphore

**Description :**Five philosophers are seated around a circular table. Each philosopher has a place of spaghetti and he needs two forks to eat. Between each plate there is a fork. The life of a philosopher consists of alternate period of eating & thinking. When a philosopher gets hungry, he tries to acquire his left fork, if he gets it, it tries to acquire right fork.

In this solution, we check after picking the left fork whether the right fork is available or not. If not, then philosopher puts down the left fork & continues to think. Even this can fail if all the philosophers pick the left fork simultaneously & no right forks available & putting the left fork down again. This repeats & leads to starvation.

Now, we can modify the problem by making the philosopher wait for a random amount of time instead of same time after failing to acquire right hand fork. This will reduce the problem of starvation.

Solution to dinning philosophers problem.

**Algorithm :**

philosopher()

{

int i;10

while(true)

{

think();

take\_fork(i);

eat();

putfork(i);

}

}

putfork(i)

{

int i;

down(mutex);

state[i] = THINKING;

test(left);

test(right);

up(mutex);

test(i);

if( state[i] == HUNGRY && state[i] != EATING && state[right] != EATING)

{

state[i] = EATING;

up(s[i]);

}

}

take\_frok(i)

{

int i;

down(mutex);

state[i] = HUNGRY;

test[i];

up(mutex);

down(s[i]);

}10

**Conclusion:** Dining philosopher problem has been solved.

**Post Lab Assignment :**

What is starvation? How is it different from deadlock? How do you avoid starvation?

**Experiment No-6**

**Title :** To implement Banker’s algorithm.

**Objective:** To understand How to avoid deadlock.

**References :** 1. “Operating System Principles”, Wiely Publication

**Pre-requisite:** Concept of semaphore,System calls related to semaphore

**Description :** Banker’s Algorithm ( Banker’s behavior (example of one resource type with many instances):

* Clients are asking for loans up-to an agreed limit.
* The banker knows that not all clients need their limit simultaneously.
* All clients must achieve their limits at some point of time but not necessarily simultaneously.
* After fulfilling their needs, the clients will pay-back their loans.
* Always keep so many resources that satisfy the needs of at least one client.
* Multiple instances.
* Each process must a priori claim maximum use.
* When a process requests a resource it may have to wait.
* When a process gets all its resources it must return them in a finite amount of time.

**Algorithm :**

**Safety Algorithm**

* + 1. Let Work and Finish be vectors of length m and n, respectively.

Initialize: Work = Available Finish [i] = false for i = 1,2, …, n.

2. Find and i such that both: (a) Finish [i] = false (b) Needi ≤ Work If no such i exists, go to step 4.

3. Work = Work + Allocationi Finish[i] = true go to step 2.

4. If Finish [i] == true for all i, then the system is in a safe state.

**Resource-Request Algorithm** **for Process Pi**

Requesti = request vector for process Pi .

Requesti [ j ] == k means that process Pi wants k instances of resource type Rj.

* + - 1. If Requesti ≤ Needi go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim 2.
      2. If Requesti ≤ Available, go to step 3. Otherwise Pi must wait, since resources are not available.
      3. Test to allocate requested resources to Pi by modifying the state as follows:

Available = Available – Request(i) ;

Allocation(i) = Allocation(i) + Request(i);

Need(i) = Need(i) – Request(i) ;

* If safe ⇒ the resources are allocated to Pi.
* If unsafe ⇒ Pi must wait, and the old resource-allocation state is restored

**Conclusion:** Bankers algorithm is implemented.

**Post Lab Assignment:**

How Banker’s algorithm will be applicable for several instances of various resource types.

**Experiment No-7**

**Title:** To study page replacement policies like

1) OPTIMAL

2) LEAST RECENTLY USED(LRU)

3) FIRST-IN-FIRST-OUT

**Objective :** To study various system calls

**References:**

1. “Operating Systems”, William Stallings

2. “Operating Systems Concepts”, Silbershatz,Peterson, Galvin,

Addison Wesley 2nd Edition

3. “Modern Operating Systems”,Tannenbaum,

Eastern Economy edition , 2nd Edition Year 1995

**Pre-requisite:** Knowledge of Page Replacement Policies.

**Description:**

In multiprogramming system using dynamic partitioning there will come a time when all of the processes in the main memory are in a blocked state and there is insufficient memory. To avoid wasting processor time waiting for an active process to become unblocked. The OS will swap one of the process out of the main memory to make room for a new process or for a process in Ready-Suspend state.

Therefore, the OS must choose which process to replace.

Thus, when a page fault occurs, the OS has to change a page to remove from memory to make room for the page that must be brought in. If the page to be removed has been modified while in memory it must be written to disk to bring the disk copy up to date.

Replacement algorithms can affect the system's performance. Following are the three basic page replacement algorithms:

Optimal Page Replacement Policy

The idea is to replace the page that will not be referenced for the longest period of time.

Least Recently Used Algorithm

This paging algorithm selects a page for replacement that has been unused for the longest time.

First-In-First\_Out

Replace the page that has been in memory longest, is the policy applied by FIFO. Pages from memory are removed in round-robin fashion. Its advantage is it's simplicity.

**Conclusion :**

Various page replacement algorithms have been studied successfully.

**Post Lab Assignments:**

Comparative Assessment of various page replacement policies.

**Experiment No-8**

**Title :** To study Disk Scheduling algorithms like

1) FCFS

2) SSTF

3) SCAN

4) CSCAN

5) LOOK

**Objective :** To study disk schedulling

**References:**

1. “Operating Systems”, William Stallings

2. “Operating Systems Concepts”, Silbershatz,Peterson, Galvin,

Addison Wesley 2nd Edition

3. “Modern Operating Systems”,Tannenbaum,

Eastern Economy edition , 2nd Edition Year 1995

**Pre-requisite:** Knowledge of disk scheduling basics

**Description:**

1) FCFS

All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next.

2) SSTF

In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of.

3) SCAN

This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up

4) CSCAN

Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction.

5) LOOK

This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request.

**Conclusion:**

Various disk scheduling algorithms have been studied successfully.

**Post Lab Assignments:**

Comparative Assessment of various disk scheduling algorithms.

**Experiment No-9**

**Title :** To simulate paging and segmentation

**Objective :** To simulate the working of paging and segmentation

**References:**

1. “Operating Systems”, William Stallings

2. “Operating Systems Concepts”, Silbershatz,Peterson, Galvin,

Addison Wesley 2nd Edition

3. “Modern Operating Systems”,Tannenbaum,

Eastern Economy edition , 2nd Edition Year 1995

**Pre-requisite:** knowledge about working of paging and segmenttion

**Description:**

**Paging**

**paging** is one of the [memory-management](http://en.wikipedia.org/wiki/Memory_management) schemes by which a computer can store and retrieve data from [secondary storage](http://en.wikipedia.org/wiki/Computer_data_storage#Secondary_storage) for use in [main memory](http://en.wikipedia.org/wiki/Computer_data_storage#Primary_storage).[[1]](http://en.wikipedia.org/wiki/Paging#cite_note-ostep-1-1) In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size [blocks](http://en.wikipedia.org/wiki/Block_(data_storage)) called *pages*. The main advantage of paging over [memory segmentation](http://en.wikipedia.org/wiki/Memory_segmentation) is that it allows the physical address space of a process to be [noncontiguous](http://en.wikipedia.org/wiki/Contiguous#Computer_science)

**Segmentation**

Segmentation is a Memory Management technique in which memory is divided into variable sized chunks which can be allocated to processes. Each chunk is called a segment. A table stores the information about all such segments and is called Global Descriptor Table (GDT). A GDT entry is called Global Descriptor.

**Conclusion:**

The working of paging and segmentation is observed using a simulator.

**Post Lab Assignments:**

Comparative Assessment of paging and segmentation

**Experiment No-10**

**Title :** To implement system calls for file management like printing a file

**Objective :** To gain knowledge about how to make a system call

**References:**

1. “Operating Systems”, William Stallings

2. “Operating Systems Concepts”, Silbershatz,Peterson, Galvin,

Addison Wesley 2nd Edition

3. “Modern Operating Systems”,Tannenbaum,

Eastern Economy edition , 2nd Edition Year 1995

**Pre-requisite: basic knowledge of linux commands**

**Description**

**System call**

A system call is used by application (user) programs to request service from the operating system An operating system can access a system's hardware directly, but a user program is not given direct access to the hardware. This is done so that the kernel can keep the system safe and secure from malicious user programs. But often, a user program requires some information from the hardware (e.g., from a web camera to show you the picture), but it cannot get the information directly. So, it requests the operating system to supply it the information. This request is made by using an appropriate system call.

A system call executes in the kernel mode. Every system call has a number associated with it. This number is passed to the kernel and that's how the kernel knows which system call was made. When a user program issues a system call, it is actually calling a library routine. The library routine issues a trap to the Linux operating system by executing INT 0x80 assembly instruction. It also passes the system call number to the kernel using the EAX register. The arguments of the system call are also passed to the kernel using other registers (EBX, ECX, etc.). The kernel executes the system call and returns the result to the user program using a register. If the system call needs to supply the user program with large amounts of data, it will use another mechanism (e.g., copy\_to\_user call).

**List of files to be modified/created**

Assume that your linux source base directory is /usr/src/linux.

Kernel files to be modified are listed below:

1. /usr/src/linux/arch/i386/kernel/syscall\_table.S

2. /usr/src/linux/include/asm-i386/unistd.h

3. /usr/src/linux/include/linux/syscalls.h

4. /usr/src/linux/Makefile

New kernel files/directories to be created are listed below:

1. /usr/src/linux/mycall - Directory that will contain the source file, header file and the Makefile for our system call (You can also implement your system call in an existing file).

2. /usr/src/linux/mycall/mycall.c - Source file containing our system call code.

3. /usr/src/linux/mycall/Makefile - Makefile

New user space files, to be created, to test our system call are listed below:

1. testmycall.c â€“ Source file that will call our system call.

2. testmycall.h - Header file.

**4. Kernel files to be modified**

**syscall\_table.S**

Full path of the file - /usr/src/linux/arch/i386/kernel/syscall\_table.S

This file contains system call names.

1. Add a line to the end of this file (Let's assume that the name of our system call is mycall).
2. Add ".long sys\_mycall" at the end of the list.

**unistd.h**

Full path of the file - /usr/src/linux/include/asm-i386/unistd.h

This file contains the system call number that is passed to the kernel through the register (EAX) when a system call is invoked.

1. Add "#define \_\_NR\_mycall <Last\_System\_Call\_Num + 1>" at the end of the list.

If the last system call defined here is:

"#define \_\_NR\_vmsplice316", then add:

"#define \_\_NR\_mycall317" at the end of the list.

1. Increment the "NR\_syscalls" by 1. So, if NR\_syscalls is defined as:

"#define NR\_syscalls 317", then change it to:

"#define NR\_syscalls 318"

**syscalls.h**

Full path of the file - /usr/src/linux/include/linux/syscalls.h

This file contain the declarations for system calls.

1. Add the following line at the end of the file:

"asmlinkage long sys\_mycall(int i);"

**Makefile**

Full path of the file - /usr/src/linux/Makefile

1. Add mycall/ to core-y (Search for regex: core-y.\*+=). You will be creating this directory. This directory will contain the source file, header file and the Makefile for our system call.

**New kernel files/directories to be created**

**mycall**

Full path of the file - /usr/src/linux/mycall

1. Create a new directory in /usr/src/linux and name it "mycall".

**mycall.c**

Full path of the file - /usr/src/linux/mycall/mycall.c

1. Create a source file named "mycall.c" in dir "mycall". mycall.c will have the code for our system call. The definition of the system call in the source file would be asmlinkage long sys\_mycall(...){...} . It should include the file linux/linkage.h So, the file "mycall.c" will look like:

/\*---Start of mycall.c----\*/

#include<linux/linkage.h>

asmlinkage long sys\_mycall(int i)

{

return i+10;

}

/\*---End of mycall.c------\*/

What is asmlinkage?

Asmlinkage is used to look for the arguments on the kernel stack.

**12. Makefile**

Full path of the file - /usr/src/linux/mycall/Makefile

The Makefile in dir "mycall" will have only one line:

#####Makefile Start#####

obj-y := mycall.o

#####Makefile End#######

**New user space files, to be created, to test our system call**

**testmycall.h (new user space header file to be created)**

testmycall.h

1. Create a header file called testmycall.h. This header file should be included by any program calling our system call.
2. Add three lines to it
3. Line 1: This is needed because we need the definition of \_syscall1.

#include<linux/unistd.h>

1. Line 2: This is needed because we need the number of our system call.

#define \_\_NR\_mycall 317

1. Line 3: This is needed for system calls with 1 argument. It is explained in detail below.

\_syscall1(long, mycall, int, i)

So, our user header file looks like:

/\*---Start of header file------\*/

#include<linux/unistd.h>

#define \_\_NR\_mycall 317

\_syscall1(long, mycall, int, i)

/\*---End of header file--------\*/

**testmycall.c (new user space source file to be created)**

testmycall.c

1. Create a C file called testmycall.c in the same directory as testmycall.h. The C file will look like:

/\*---Start of C file------\*/

#include<stdio.h>

#include "testmycall.h"

int main(void)

{

printf("%d\n", mycall(15));

}

/\*---End of C file------\*/

**\_syscallN macro**

1. \_syscall0(int,mycall) indicates that:
2. The name of the system call is mycall.
3. It takes zero arguments.
4. It returns an int.
5. \_syscall1(int,mycall,int,number) indicates that:
6. The name of the system call is mycall.
7. It takes one argument.
8. The argument is an int named number.
9. It returns an int.

When you expand \_syscall1(long,mycall,int,i), you get the following code:

long mycall(int i)

{

return syscall(\_\_NR\_mycall, i);

}

But the definition of \_syscallN macros are different in the kernel. You can look at /usr/src/linux/include/asm-i386/unistd.h for the definition.

**Testing our new system call**

1. Step 1: Recompile and install the new kernel so that our system call becomes available to the operating system.
2. Step 2: Compile and execute the user space C file (testmycall.c) that we created above.
3. RESULT: You should see the output as 25. This has been tested on kernel 2.6.17.13.

**Conclusion:**

The system call for file management is implemented