

Ex. No. 4

LINUX COMMANDS

Date:

Aim:

a) Basics Commands

1. echo SRM → to display the string SRM
2. clear → to clear the screen
3. date → to display the current date and time
4. cal 2003 → to display the calendar for the year 2003
cal 6 2003 → to display the calendar for the June-2003
5. passwd → to change password
6. free -m → to view the size of RAM in MB
free -g → to view the size of RAM in GB
7. df -h → to view the disk space available and used.
8. uptime → to view the system up time
9. bc → to open a basic calculator
10. ps → to view the current terminal running processes
11. history → to get the history of all the past commands
12. whoami → to know which user i am

b) Working with Files

1. ls → list files in the present working directory
ls -l → list files with detailed information (long list)
ls -a → list all files including the hidden files
ls -r root → list the directory recursively
ls -lh → list the current location content in human redable format
ls -lt → to list the files based on modification time
ls -li → to view the inode number of files and directories
lscpu → to view the system specifications
2. cat > f1 → to create a file (Press ^d to finish typing)
3. cat f1 → display the content of the file f1
4. wc f1 → list no. of characters, words & lines of a file f1
wc -c f1 → list only no. of characters of file f1
wc -w f1 → list only no. of words of file f1

- wc -l f1 → list only no. of lines of file f1
- 5. cp f1 f2 → copy file f1 into f2
- 6. mv f1 f2 → rename file f1 as f2
- 7. rm f1 → remove the file f1
- 8. head -5 f1 → list first 5 lines of the file f1
tail -5 f1 → list last 5 lines of the file f1

c) Working with Directories

- 1. mkdir elias → to create the directory elias
- 2. cd elias → to change the directory as elias
- 3. rmdir elias → to remove the directory elias
- 4. pwd → to display the path of the present working directory
- 5. cd → to go to the home directory
cd .. → to go to the parent directory
cd - → to go to the previous working directory
cd / → to go to the root directory

d) File name substitution

- 1. ls f? → list files start with 'f' and followed by any one character
- 2. ls *.c → list files with extension 'c'
- 3. ls [gpy]et → list files whose first letter is any one of the character g, p or y and followed by the word et
- 4. ls [a-d,l-m]ring → list files whose first letter is any one of the character from a to d and l to m and followed by the word ring.

e) I/O Redirection

- 1. Input redirection
wc -l < ex1 → To find the number of lines of the file 'ex1'
- 2. Output redirection
who > f2 → the output of 'who' will be redirected to file f2
- 3. cat >> f1 → to append more into the file f1

f) Piping

Syntax : Command1 | command2

Output of the command1 is transferred to the command2 as input. Finally output of the command2 will be displayed on the monitor.

ex. cat f1 | more → list the contents of file f1 screen by screen

head -6 f1 |tail -2 → prints the 5th & 6th lines of the file f1.

g) Environment variables

1. echo \$HOME → display the path of the home directory
2. echo \$PS1 → display the prompt string \$
3. echo \$PS2 → display the second prompt string (> symbol by default)
4. echo \$LOGNAME → login name
5. echo \$PATH → list of pathname where the OS searches
for an executable file

h) File Permission

-- chmod command is used to change the access permission of a file.

Method-1

Syntax : chmod [ugo] [+/-] [rwx a] filename

u : user, g : group, o : others
+ : Add permission - : Remove the
permission r : read, w : write, x : execute, a :
all permissions

ex. chmod ug+rw fl
adding 'read & write' permissions of file fl to both user and group
members.

Method-2

Syntax : chmod octnum file1

The 3 digit octal number represents as follows

- first digit -- file permissions for the user
- second digit -- file permissions for the group
- third digit -- file permissions for others

Each digit is specified as the sum of following

4 – read permission, 2 – write permission, 1 – execute

permission ex. chmod 754 fl

it change the file permission for the file as follows

- read, write & execute permissions for the user ie; $4+2+1 = 7$
- read, & execute permissions for the group members ie; $4+0+1 = 5$
- only read permission for others ie; $4+0+0 = 4$

QUESTIONS FOR PRACTICE:

Q1. Write a command to cut 5 to 8 characters of the file *f1*.

\$

Q2. Write a command to display user-id of all the users in your system.

\$

Q3. Write a command to paste all the lines of the file *f1* into single line

\$

Q4. Write a command to cut the first field of file *f1* and second field of file *f2* and paste into the file *f3*.

\$

Q5. Write a command to change all small case letters to capitals of file *f2*.

\$

Q6. Write a command to replace all *tab* character in the file *f2* by :

\$

Q7. Write a command to check whether the user *judith* is available in your system or not.(use *grep*)

\$

Q8. Write a command to display the lines of the file *f1* starts with SRM.

\$

Q9. Write a command to display the name of the files in the directory */etc/init.d* that contains the pattern *grep*.

\$

Q10. Write a command to display the names of nologin users. (Hint: the command *nologin* is specified in the last field of the file */etc/passwd* for nologin users)

\$

Q11. Write a command to sort the file */etc/passwd* in descending order

\$

Q12. Write a command to sort the file `/etc/passwd` by user-id numerically. (Hint : user-id is in 3rd field)

\$

Q13. Write a command to sort the file `f2` and write the output into the file `f22`. Also eliminate duplicate lines.

\$

Q14. Write a command to display the unique lines of the sorted file `f21`. Also display the number of occurrences of each line.

\$

Q15. Write a command to display the lines that are common to the files `f1` and `f2`.

\$

Result:

Ex. No. 5**SYSTEM ADMIN COMMANDS****Date:****Aim:****INSTALLING SOFTWARE****Procedure:**

- Open the Ubuntu software Center.
- To install any package, open the terminal (Ctrl + Alt + T) and type `sudo apt-get install <package name>`.
- For instance, to get Chrome type `sudo apt-get install chrome-browser`.
- Likewise user can work package update, remove and reinstall the package using the following commands.

To update the package repositories

```
sudo apt-get update
```

To update installed software

```
sudo apt-get upgrade
```

To install a package/software

```
sudo apt-get install <package-name>
```

To remove a package from the system

```
sudo apt-get remove <package-name>
```

To reinstall a package

```
sudo apt-get install <package-name> --reinstall
```

To completely remove a software and it's dependent packages run the apt-get purge

```
sudo apt-get purge <package-name>
```

To remove all Debian (.deb) files those are no longer installed

---files in /var/cache/apt/archives

```
sudo apt-get autoclean
```

To empty whole cache files – to reduce the space consumption

```
sudo apt-get clean
```

To remove old dependent files and footprints installed by previous applications

```
sudo apt-get autoremove
```

To configure installed package

```
sudo dpkg --configure -a
```

To download but not install package

```
sudo apt-get download <package-name>
```

MANAGING USERS

- Managing users is a critical aspect of server management.
- In Ubuntu, the root user is disabled for safety.
- Root access can be completed by using the sudo command by a user who is in the “admin” group.
- When you create a user during installation, that user is added automatically to the admin group.

To add a user:

```
sudo adduser username
```

To disable a user:

```
sudo passwd -l username
```

To enable a user:

```
sudo passwd -u username
```

To delete a user:

```
sudo userdel -r username
```

To create a group:

```
sudo addgroup groupname
```

To delete a group:

```
sudo delgroup groupname
```

To create a user with group:

```
sudo adduser username groupname
```

To see the password expiry value for a user,

```
sudo chage -l username
```

To make changes:

```
sudo chage username
```

GUI TOOL FOR USER MANAGEMENT

GUI Tool allow the admin to run the commands in terminal to manage users and groups.

To install a GUI add-on

```
sudo apt install gnome-system-tools
```

Once done, type

```
users-admin
```

MANAGING THE FILE SYSTEM

A filesystem is a permanent storage for containing data. Any non-volatile storage device like hard disk, usb etc has a filesystem in place, on top of which data is stored. While installing Linux, you may opt for either EXT4 or EXT3 file system.

Ext3 : A journaling filesystem: logs changes in a journal to increase reliability in case of power failure or system crash.

EXT4: It is an advanced file syste. This file system supports 64-bit storage limits, columns up to 1 exabytes and you may store files up to 16 terabytes

Disk Partitions can be viewed by the command `sudo fdisk -l`

File system information are available in the file `/etc/fstab`

MANGING THE NETWORK CONFIGURATION

Most networking is configured by editing two files:

- `/etc/network/interfaces`
 - Ethernet, TCP/IP, bridging
- `/etc/resolv.conf`
 - DNS

Other networking files:

- `/etc/hosts`
- `/etc/dhcp3/dhcpd.conf`

To test any host's connectivity

```
ping <ip-address>
```

To start/stop/restart/reload networking services

```
sudo /etc/init.d/mnetworking <function>
```


Note : <function> can be any one of stop or start or reload or restart

To list of all active network interface cards, including wireless and the loopback interface

```
sudo ifconfig
```

To display host Fully Qualified Domain Name

```
sudo hostname
```

To display arp table (ip to mac resolution)

```
sudo arp -a
```

To remove entry from arp table

```
sudo arp -d <user name>
```

To display or change network card settings, use ethtool

```
sudo ethtool eth0
```

To displays extensive status information when queried with the service iptables status command

```
sudo service iptables status
```

To start/stop services

```
sudo service iptables start/stop
```

INSTALLING INTERNET SERVICES

Installing Apache server

```
sudo apt-get install apache2
```

Configuration file for Apache server

```
apache2.conf
```

Restart apache services after any configuration changes made

```
sudo /etc/init.d/mnetworking restart
```

Similarly all services can be installed, configured and restarted

MANAGING BACKGROUND JOBS

To display jobs running in background

```
sudo jobs
```

To check the process id of background processes

```
sudo jobs -p
```

To bring a background job to the foreground

```
sudo fg
```

To start the Jobs suspended in background

```
sudo bg
```

QUESTIONS FOR PRACTICE:

Q1. Update the package repositories

Q2. Install the package “simplescreenrecorder”

Q3. Remove the package “simplescreenrecorder”

Q4. Create a user ‘elias’. Login to the newly created user and exit.

Q5. Disable the user ‘elias’, try to login and enable again.

Q6. Create a group ‘cse’ and add the user ‘elias’ in that group

Q7. List the account expiry information of the user ‘elias’

Q8. Change the ‘Number of days warning before password expires’ as 5 for the user ‘elias’

Q9. Delete the user ‘elias’ and then delete the group ‘cse’

Q10. List the partitions available in your system

Q11. What are the file systems used in your system

Q12. Stop the networking service and then start the service

Q13. Check the connectivity of the host with IP address 127.0.0.1

Q14. Find the IP address of the localhost

Q15. Find the IP address of the DNS Server (name server)

Q16. Install mysql server

Q17. Restart mysql server

Q18. Check the configuration file for mysql server

Q19. Log on as root into mysql server

Q20. Create a new database for mysql server

Result:

Ex. No. 6**SIMPLE TASK AUTOMATION****Date:****Aim:****Crontab**

Linux Cron utility is an effective way to schedule a routine background job at a specific time and/or day on an on-going basis. User can use this to schedule activities, either as one-time events or as recurring tasks.

Scheduling of Tasks (For Ubuntu)

Step 1 : Open terminal and type the command `crontab -e`

Step 2 : Choose the editor. Better to select `nano` editor

Step 3 : Edit the file based on the syntax given above

Step 4 : Save and Exit the file

Step 5 : Start cron daemon using the following command

```
systemctl start cron
```

Linux Crontab Format

MIN HOUR DOM MON DOW CMD

Table: Crontab Fields and Allowed Ranges (Linux Crontab Syntax)

Field	Description	Allowed Value
MIN	Minute field	0 to 59
HOUR	Hour field	0 to 23
DOM	Day of Month	1-31
MON	Month field	1-12
DOW	Day Of Week	0-6
CMD	Command	Any command to be executed

Create a new crontab file, or edit an existing file

```
# crontab -e [username]
```

where *username* specifies the name of the user's account for which you want to create or edit a crontab file.

Verify your crontab file changes

```
# crontab -l [username]
```

Install crontab

```
crontab -a filename
```

Edit the crontab

```
# crontab -e
```

Display crontab

```
crontab -l
```

Display the last edit the crontab file

```
crontab -v
```

Remove crontab

```
crontab -r
```

Following are the syntax for cron

```
minute(s) hour(s) day(s) month(s) weekday(s) command(s) "Argument1" "Argument2"
```

```
1 * 3 4 5 /path/to/command arg1 arg2
```

If you don't have parameter put star(*)

Commands:

- 1) **-l** - List or manage the task with crontab command
- 2) **-e** - edit crontab entry.
- 3) **-u** - To list scheduled jobs of a particular user called **tecmin** using.
- 4) **-r** - parameter will remove complete scheduled jobs without confirmation from crontab.
- 5) **-i** - prompt you confirmation from user before deleting user's crontab.

Allowed special character (*, -, /, ?, #)

1. **Asterik(*)** – Match all values in the field or any possible value.

2. **Hyphen(-)** – To define range.
3. **Slash (/)** – 1st field /10 meaning every ten minute or increment of range.
4. **Comma (,)** – To separate items.

System Wide Cron Schedule

System administrator can use predefine cron directory as shown below.

1. /etc/cron.d
2. /etc/cron.daily
3. /etc/cron.hourly
4. /etc/cron.monthly
5. /etc/cron.weekly

To Schedule a Job for Specific Time

The below jobs delete empty files and directory from **/tmp** at **12:30** am daily.

User need to mention user name to perform crontab command.

In below example **root** user is performing cron job.

```
# crontab -e
```

```
30 0 * * * root find /tmp -type f -empty -delete
```

Special Strings for Common Schedule

Strings	Meanings
@reboot	Command will run when the system reboot.
@daily	Once per day or may use @midnight.
@weekly	Once per week.
@yearly	Once per year.user can use @annually keyword also.

Multiple Commands with Double ampersand (&&)

To run the command1 and command2 daily

```
# crontab -e
```

```
@daily <command1> && <command2>
```

Disable Email Notification.

By default cron send mail to user account executing cronjob. If user want to disable using **>/dev/null 2>&1** option at the end of the file will redirect all the output of the cron results under **/dev/null**.

```
[root@tecmint ~]# crontab -e
```

```
* * * * * >/dev/null 2>&1
```

Scheduling a Job for a Specific Time

The basic usage of cron is to execute a job in a specific time as shown below. This will execute the full backup shell script (full-backup) on **10th June 08:30 AM**.

The below time field uses 24 hours format. So, for 8 AM use 8, and for 8 PM use 20.

```
30 08 10 06 * /home/username/full-backup
```

- **30** – 30th Minute
- **08** – 08 AM
- **10** – 10th Day
- **06** – 6th Month (June)
- ***** – Every day of the week

Schedule a Job for More Than One Instance (e.g. Twice a Day)

The following script takes a incremental backup twice a day every day. This example executes the specified incremental backup shell script (incremental-backup) at 11:00 and 16:00 on every day. The comma separated value in a field specifies that the command needs to be executed in all the mentioned time.

```
00 11,16 * * * /home/username/bin/incremental-backup
```

- **00** – 0th Minute (Top of the hour)
- **11,16** – 11 AM and 4 PM
- ***** – Every day
- ***** – Every month
- ***** – Every day of the week

Schedule a Job for Specific Range of Time (e.g. Only on Weekdays)

- To schedule the job for every hour with in a specific range of time then use the following.

Cron Job everyday during working hours

This example checks the status of the database everyday (including weekends) during the working hours 9 a.m – 6 p.m

```
00 09-18 * * * /home/username/bin/check-db-status
```

- **00** – 0th Minute (Top of the hour)
- **09-18** – 9 am, 10 am, 11 am, 12 am, 1 pm, 2 pm, 3 pm, 4 pm, 5 pm, 6 pm
- * – Every day
- * – Every month
- * – Every day of the week

Schedule a Job for Every Minute Using Cron.

Ideally user may not have a requirement to schedule a job every minute. But understanding this example will help user understand the other examples mentioned below in this article.

```
* * * * * CMD
```

The * means all the possible unit — i.e every minute of every hour throughout the year. More than using this * directly, user will find it very useful in the following cases.

- When user specify */5 in minute field means every 5 minutes.
- When user specify 0-10/2 in minute field mean every 2 minutes in the first 10 minute.
- Thus the above convention can be used for all the other 4 fields.

Schedule a Background Cron Job For Every 10 Minutes.

Use the following, to check the disk space every 10 minutes.

```
*/10 * * * * /home/username/check-disk-space
```

It executes the specified command check-disk-space every 10 minutes throughout the year.

There are special cases in which instead of the above 5 fields you can use @ followed by a keyword — such as reboot, midnight, yearly, hourly.

Table: Cron special keywords and its meaning

Keyword	Equivalent
@yearly	0 0 1 1 *
@daily	0 0 * * *
@hourly	0 * * * *
@reboot	Run at startup.

Schedule a Job for First Minute of Every Year using @yearly

User can specify a job to be executed on the first minute of every year, then user can use the **@yearly** cron keyword as shown below.

This will execute the system annual maintenance using annual-maintenance shell script at 00:00 on Jan 1st for every year.

```
@yearly /home/username/red-hat/bin/annual-maintenance
```

Schedule a Cron Job Beginning of Every Month using @monthly

Executes the command monthly once using **@monthly** cron keyword.

This will execute the shell script tape-backup at 00:00 on 1st of every month.

```
@monthly /home/username/suse/bin/tape-backup
```

Schedule a Background Job Every Day using @daily

Using the **@daily** cron keyword, this will do a daily log file cleanup using cleanup-logs shell script at 00:00 on every day.

```
@daily /home/username/arch-linux/bin/cleanup-logs "day started"
```

To Execute a Linux Command After Every Reboot using @reboot

Using the **@reboot** cron keyword, this will execute the specified command once after the machine got booted every time.

```
@reboot CMD
```

To Disable/Redirect the Crontab Mail Output using MAIL keyword

By default crontab sends the job output to the user who scheduled the job. To redirect the output to a specific user, add or update the MAIL variable in the crontab as shown below.

```
username@dev-db$ crontab -l
MAIL="username"
@yearly /home/username/annual-maintenance
*/10 * * * * /home/username/check-disk-space
[Note: Crontab of the current logged in user with MAIL variable]
```

To stop the crontab output to be emailed, add or update the MAIL variable in the crontab as shown below.

```
MAIL=""
```

Specify PATH Variable in the Crontab

To set absolute path of the Linux command or the shell-script :

Instead of specifying /home/username/tape-backup, user can specify tape-backup, then add the path /home/username to the PATH variable in the crontab as shown below.

```
username@dev-db$ crontab -l
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/home/username
@yearly annual-maintenance
*/10 * * * * check-disk-space
[Note: Crontab of the current logged in user with PATH variable]
```

To Install Crontab from a Cron File

Instead of directly editing the crontab file, user can also add all the entries to a cron-file first.

Once user have all those entries in the file, user can upload or install them to the cron as shown below.

```
username@dev-db$ crontab -l
no crontab for username
$ cat cron-file.txt
@yearly /home/username/annual-maintenance
*/10 * * * * /home/username/check-disk-space
username@dev-db$ crontab cron-file.txt
username@dev-db$ crontab -l
@yearly /home/username/annual-maintenance
*/10 * * * * /home/username/check-disk-space
```

To View Crontab Entries:

View Current Logged-In User's Crontab entries

To view crontab entries type

```
crontab -l
```

```
Username@dev-db$ crontab -l
```

```
@yearly /home/username/annual-maintenance
```

```
*/10 * * * * /home/username/check-disk-space
```

[Note: This displays crontab of the current logged in user]

To View Root Crontab entries

Login as root user (su – root) and do crontab -l as shown below.

```
root@dev-db# crontab -l
```

```
no crontab for root
```

To View Other Linux User's Crontabs entries:

To view crontab entries of other Linux users,

login to root and use **-u {username} -l**

```
root@dev-db# crontab -u username -l
```

```
@monthly /home/username/monthly-backup
```

```
00 09-18 * * * /home/username/check-db-status
```

To Edit Crontab Entries:

Edit Current Logged-In User's Crontab entries

To edit a crontab entries,

```
use crontab -e
```

By default this will edit the current logged-in users crontab.

```
username@dev-db$ crontab -e
```

```
@yearly /home/username/centos/bin/annual-maintenance
```

```
*/10 * * * * /home/username/debian/bin/check-disk-space
```

```
"/tmp/crontab.XXXXyjWkHw" 2L, 83C
```

[Note: This will open the crontab file in Vim editor for editing.]

Please note cron created a temporary /tmp/crontab.XX...]

When user save the above temporary file with :wq, it will save the crontab and display the following message indicating the crontab is successfully modified.

QUESTIONS FOR PRACTICE:

Q1. Schedule a task to display the following message on the monitor for every 2 minutes.

Q2. Schedule a task to take backup of your important file (say file f1) for every 30 minutes

Q3. Schedule a task to take backup of login information everyday 9:30am

Result:

Ex. No. 7**SHELL PROGRAMS****Date:****Aim:****How to run a Shell Script**

- Edit and save your program using editor
- Add execute permission by *chmod* command
- Run your program using the name of your program
./program-name

Important Hints

- No space before and after the assignment operator Ex. sum=0
- *Single quote* ignores all special characters. Dollar sign, Back quote and Back slash are not ignored inside *Double quote*. *Back quote* is used as command substitution. *Back slash* is used to remove the special meaning of a character.
- Arithmetic expression can be written as follows : i=\$((i+1)) or i=\$((expr \$i + 1))
- Command line arguments are referred inside the program as \$1, \$2, ..and so on
- \$* represents all arguments, \$# specifies the number of arguments
- read statement is used to get input from input device. Ex. read a b

Syntax for if statement

```

if [ condition ]
then
    ...
elif [ condition ]
then
    ...
else
    ...
fi

```

Syntax for case structure

```

case value in
pat1) ...
    statement;;
pat2) ...
    Statement;;
*) ...
    Statement;;
esac

```

Syntax for for-loop

```

for var in list-of-values
do
    ...
done

```

Syntax for printf statement

```
printf "string and format" arg1 arg2 ... ..
```

- Break and continue statements functions similar to C programming
- Relational operators are -lt, -le, -gt, -ge, -eq, -ne
- Ex. (i>= 10) is written as [\$i -ge 10]
- Logical operators (and, or, not) are -o, -a, !
- Ex. (a>b) && (a>c) is written as [\$a -gt \$b -a \$a -gt \$c]
- Two strings can be compared using = operator

Q1. Given the following values

```
num=10, x=*, y=`date` a="Hello, 'he said'"
```

Execute and write the output of the following commands

Command	Output
<i>echo num</i>	
<i>echo \$num</i>	
<i>echo \$x</i>	
<i>echo '\$x'</i>	
<i>echo "\$x"</i>	
<i>echo \$y</i>	
<i>echo \$(date)</i>	
<i>echo \$a</i>	
<i>echo \ \$num</i>	
<i>echo \ \$ \$num</i>	

Q2. Find the output of the following shell scripts

```
$ vi ex51
echo Enter value for n
read n
sum=0
i=1
while [ $i -le $n ]
do
    sum=$((sum+i))
    i=$((i+2))
done

echo Sum is $sum
```

Output :

Q3. Write a program to check whether the file has execute permission or not. If not, add the permission.

\$ vi ex52

Q4. Write a shell script to print a greeting as specified below.

If hour is greater than or equal to 0 (midnight) and less than or equal to 11 (up to 11:59:59), "Good morning" is displayed.

If hour is greater than or equal to 12 (noon) and less than or equal to 17 (up to 5:59:59 p.m.), "Good afternoon" is displayed.

If neither of the preceding two conditions is satisfied, "Good evening" is displayed.

\$ vi ex53

```
hour=$(date | cut -c12-13)
```

```
if [ "$hour" -ge 0 -a "$hour" -le 11 ]
```

then

← complete the program

Q5. Write a shell script to list only the name of sub directories in the present working directory

```
$ vi ex54
```

Q6. Write a program to check all the files in the present working directory for a pattern (passed through command line) and display the name of the file followed by a message stating that the pattern is available or not available.

```
$ vi ex55
```

Result:

Ex. No. 8**PROCESS CREATION****Date:****Aim:****Syntax for process creation**`int fork();`

Returns 0 in child process and child process ID in parent process.

Other Related Functions`int getpid()` → returns the current process ID`int getppid()` → returns the parent process ID`wait()` → makes a process wait for other process to complete**Virtual fork**

`vfork()` function is similar to `fork` but both processes shares the same address space.

Q1. Find the output of the following program

```
#include <stdio.h>
#include<unistd.h>

int main()
{
    int a=5,b=10,pid;

    printf("Before fork a=%d b=%d \n",a,b);
    pid=fork();

    if(pid==0)
    {
        a=a+1;
        b=b+1;
        printf("In child a=%d b=%d \n",a,b);
    }
    else
    {
        sleep(1);
        a=a-1;
        b=b-1;
        printf("In Parent a=%d b=%d \n",a,b);
    }
    return 0;
}
```

Output :-

Q2. Rewrite the program in Q1 using vfork() and write the output

Q3. Calculate the number of times the text “SRMIST” is printed.

```
#include <stdio.h>
#include<unistd.h>

int main()
{
    fork();
    fork();
    fork();
    printf("SRMIST\n");
    return 0;
}
```

Output :

Q4. Complete the following program as described below :

The child process calculates the sum of odd numbers and the parent process calculate the sum of even numbers up to the number 'n'. Ensure the Parent process waits for the child process to finish.

```
#include <stdio.h>
#include<unistd.h>

int main()
{
    int pid,n,oddsum=0,evensum=0;

    printf("Enter the value of n : ",a);
    scanf("%d",&n);
    pid=fork();
    // Complete the program


    return 0;
}
```

Sample Output :

Enter the value of n	10
Sum of odd numbers	25
Sum of even numbers	30

Q5. How many child processes are created for the following code?

Hint : Check with small values of 'n'.

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

Output :

Q6. Write a program to print the Child process ID and Parent process ID in both Child and Parent processes

```
#include <stdio.h>
#include<unistd.h>
int main()
{

return 0;
}
```

Sample Output:

```
In Child Process
Parent Process ID      :      18
Child Process ID       :      20

In Parent Process
Parent Process ID      :      18
Child Process ID       :      20
```

Q7. How many child processes are created for the following code?

```
#include <stdio.h>
#include<unistd.h>

int main()
{
    fork();
    fork()&&fork()||fork();fork();
    printf("Yes ");
    return 0;
}
```

Output :**Result:**

Ex. No. 9 & 10**OVERLAY CONCEPTS****Date:****Aim:****Overlay**

Overlay is the concept which enables the user to run another new process from the currently running process address space.

System call used :**Exec() System Call**

The exec() system call replaces (overwrites) the current process with the new process image. The PID of the new process remains the same however code, data, heap and stack of the process are replaced by the new program. There are 6 system calls in the family of exec(). All of these functions mentioned below are layered on top of execve(), and they differ from one another and from execve() only in the way in which the program name, argument list, and environment of the new program are specified

Syntax:

```
int execl(const char* path, const char* arg, ...)
```

```
int execlp(const char* file, const char* arg, ...)
```

```
int execlp(const char* path, const char* arg, ..., char* const envp[])
```

```
int execv(const char* path, const char* argv[])
```

```
int execvp(const char* file, const char* argv[])
```

```
int execvpe(const char* file, const char* argv[], char *const envp[])
```

- The names of the first five of above functions are of the form execXY.
- X is either l or v depending upon whether arguments are given in the list format (arg0, arg1, ..., NULL) or arguments are passed in an array (vector).

- Y is either absent or is either a p or an e. In case Y is p, the PATH environment variable is used to search for the program. If Y is e, then the environment passed in envp array is used.
- In case of `execvpe`, X is v and Y is e. The `execvpe` function is a GNU extension. It is named so as to differentiate it from the `execve` system call.

Simple overlay concept

Procedure:

- Step1: Create two different c programs. Name it as `example.c` file and `hello.c` file
- Step2: Make `example.c` is the current running process,
- Step3: call the function `execv()` which takes the `hello.c` as an argument.
- Step4: Print process id of both the processes(`hello.c` and `example.c` processes).
- Step5: trace the system control by having simple print statement in both programs.

Expected Output:

\$ process id of `example.c`=4733

We are in `hello.c`

Process id of `hello.c`=4733

COMBINING FORK() AND EXEC() SYSTEM CALL

Procedure:

- Step1: Create two different c programs. Name it as `example.c` file and `hello.c` file
- Step2: Make `example.c` is the current running process and use `fork()` system call to create child process,
- Step3: call the function `execv()` in child process which takes the `hello.c` as an argument.
- Step4: Print Process id of both parent, child and overlay processes (`hello.c` and `example.c`

processes).

- Step5: trace the system control by having simple print statement in both programs.

Expected Output:

\$ process id of example.c=4790

The control is in parent process

The control is in child process

Process id of child = 4791

Calling hello.c from child

We are in hello.c

Process id of hello.c=4791

PRACTICE QUESTIONS:

1. Execute the Following Program and write the output

\$vi ex1.c.

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
int main()
```

```
printf("Transfer to execlp function \n");
```

```
execlp("head", "head", "", "-2", "fl", NULL); // Assume fl is any text file
```

```
printf("This line will not execute \n");
```

```
return 0;
```

Output :

Why second printf statement is not executing?

2. Rewrite question 1 with execl() function. Pass the 3rd and 4th argument of the function

execl() through command line arguments.

\$vi ex2.c

Input : /a.out -3 fi

Output:

Result:

Ex. No. 11

INTER PROCESS COMMUNICATION – PIPE()

Date:

Aim:

Inter process communication (IPC):

A process can be of two types:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co operating process can be affected by other executing processes.

Inter process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

- Shared Memory
- Message passing

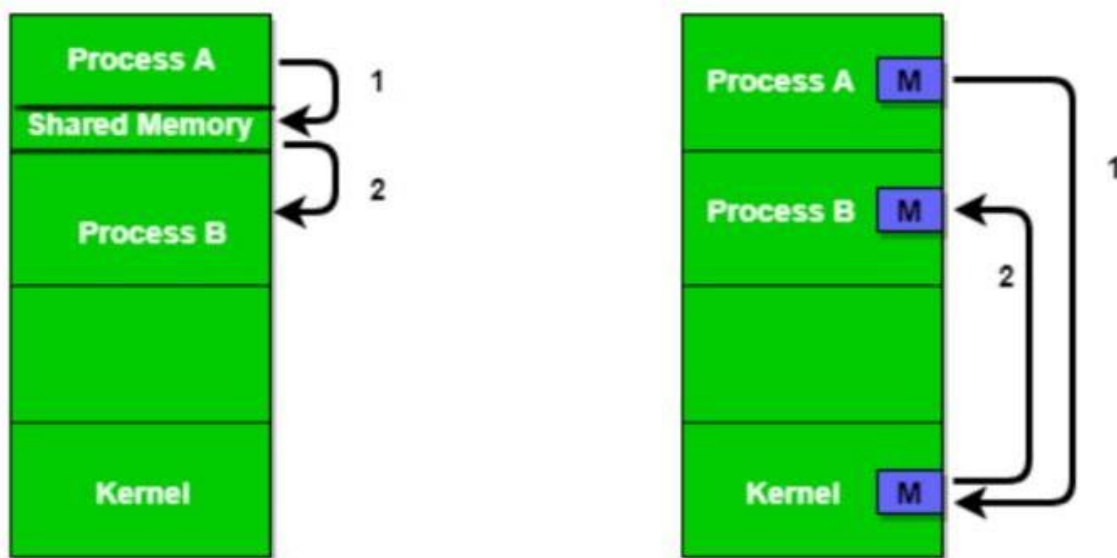


Figure 1 - Shared Memory and Message Passing

Inter process communication (IPC) is used for exchanging data between multiple threads in one or more processes or programs. The Processes may be running on single or multiple computers connected by

a network. The full form of IPC is Inter-process communication.

It is a set of programming interface which allow a programmer to coordinate activities among various program processes which can run concurrently in an operating system. This allows a specific program to handle many user requests at the same time.

Since every single user request may result in multiple processes running in the operating system, the process may require to communicate with each other. Each IPC protocol approach has its own advantage and limitation, so it is not unusual for a single program to use all of the IPC methods.

Important methods for inter process communication:

Pipes

Pipe is widely used for communication between two related processes. This is a half-duplex method, so the first process communicates with the second process. However, in order to achieve a full-duplex, another pipe is needed.

Message Passing:

It is a mechanism for a process to communicate and synchronize. Using message passing, the process communicates with each other without resorting to shared variables.

IPC mechanism provides two operations:

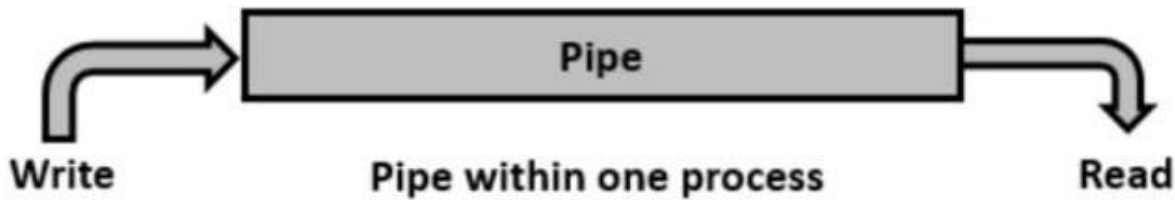
- Send (message)- message size fixed or variable
- Received (message)

Message Queues:

A message queue is a linked list of messages stored within the kernel. It is identified by a message queue identifier. This method offers communication between single or multiple processes with full-duplex capacity.

PIPES

8MPipe is a communication medium between two or more related or interrelated processes. It can be either within one process or a communication between the child and the parent processes. Communication can also be multi-level such as communication between the parent, the child and the grand-child, etc. Communication is achieved by one process writing into the pipe and other reading from the pipe. To achieve the pipe system call, create two files, one to write into the file and another to read from the file.



```
#include<unistd.h>

int pipe(int pipedes[2]);
```

This system call would create a pipe for one-way communication i.e., it creates two descriptors, first one is connected to read from the pipe and other one is connected to write into the pipe.

Descriptor `pipedes[0]` is for reading and `pipedes[1]` is for writing. Whatever is written into `pipedes[1]` can be read from `pipedes[0]`.

This call would return zero on success and -1 in case of failure. To know the cause of failure, check with `errno` variable or `perror()` function.

```
#include <sys/types.h> #include <sys/stat.h>

#include <fcntl.h>

int open(const char *pathname, int flags);

int open(const char *pathname, int flags, mode_t mode);
```

Even though the basic operations for file are read and write, it is essential to open the file before performing the operations and closing the file after completion of the required operations. Usually, by default, 3 descriptors opened for every process, which are used for input (standard input - `stdin`), output (standard output - `stdout`) and error (standard error `stderr`) having file descriptors 0, 1 and 2 respectively.

Uni directional pipe:

Algorithm

- **Step 1** - Create a pipe using `pipe()` system call.
- **Step 2** – Send a message to the one end of the pipe.

- **Step 3** – Retrieve the message from the other end of the pipe and write it to the standard output.

Expected Output:

hello, world #1

hello, world #2

hello, world #3

Implementing command line pipe using exec() family of function:

Follow the steps to transfer the output of a process to pipe:

- (i) Close the standard output descriptor
- (ii) Use the following system calls, to take duplicate of output file descriptor of the pipe `int dup(int fd); int dup2(int oldfd, int newfd);`
- (iii) Close the input file descriptor of the pipe
- (iv) Now execute the process.

Follow the steps to get the input from the pipe for a process:

- (i) Close the standard input descriptor
- (ii) Take the duplicate of input file descriptor of the pipe using `dup()` system call
- (iii) Close the input file descriptor of the pipe
- (iv) Now execute the process

Named pipe

Named pipe (also known as FIFO) is one of the inter process communication tool. The system for FIFO is as follows

```
int mkfifo(const char *pathname, mode_t mode);
```

`mkfifo()` makes a FIFO special file with name `pathname`. Here mode specifies the FIFO's permissions. The permission can be like : `O_CREAT|0644` Open FIFO in read-mode (`O_RDONLY`) to read and write-mode (`O_WRONLY`) to write.

Write and read two messages using pipe.

Algorithm

Step 1 - Create a pipe.

Step 2 – Send a message to the pipe.

Step 3 – Retrieve the message from the pipe and write it to the standard output.

Step 4 - Send another message to the pipe.

Step 5 - Retrieve the message from the pipe and write it to the standard output.

Note – Retrieving messages can also be done after sending all messages.

Expected Output:

Writing to pipe - Message 1 is Hi

Reading from pipe - Message 1 is Hi

Writing to pipe - Message 2 is Hello

Reading from pipe - Message 2 is Hello

2. Program to write and read two messages through the pipe using the parent and the child processes.

Algorithm

Step 1 - Create a pipe.

Step 2 - Create a child process.

Step 3 – Parent process writes to the pipe.

Step 4 - Child process retrieves the message from the pipe and writes it to the standard output.

Step 5 – Repeat step 3 and step 4 once again.

Expected Output:

Parent Process - Writing to pipe - Message 1 is Hi

Parent Process - Writing to pipe - Message 2 is Hello

Child Process - Reading from pipe - Message 1 is Hi

Child Process - Reading from pipe – Message 2 is Hello

Two-way Communication Using Pipes

Following are the steps to achieve two-way communication -

Step 1 - Create two pipes. First one is for the parent to write and child to read, say as pipe 1.

Second one is for the child to write and parent to read, say as pipe2.

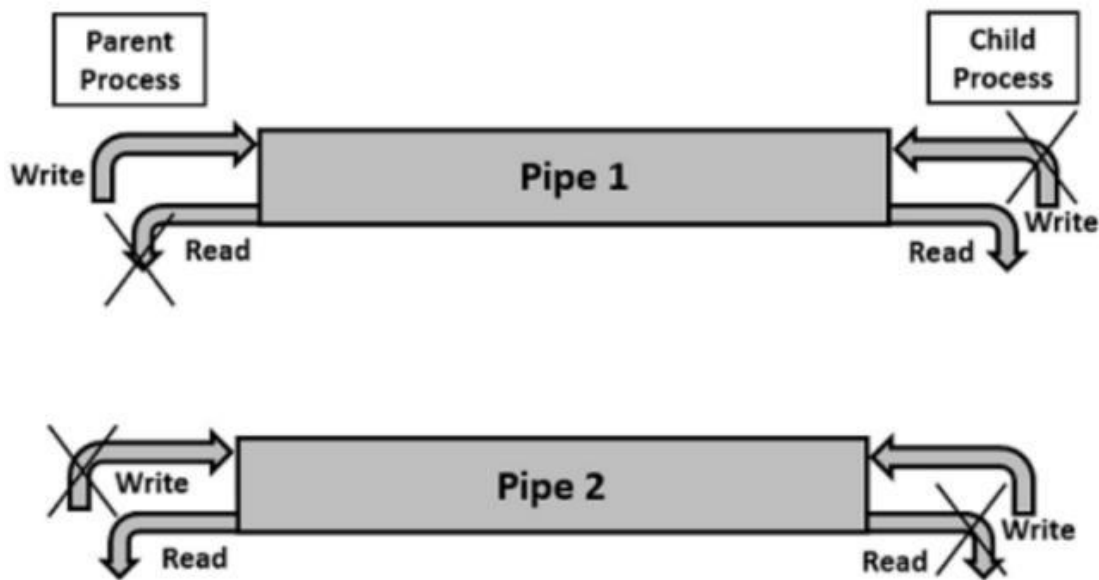
Step 2 - Create a child process.

Step 3 – Close unwanted ends as only one end is needed for each communication.

Step 4 - Close unwanted ends in the parent process, read end of pipe1 and write end of pipe2.

Step 5 – Close the unwanted ends in the child process, write end of pipe1 and read end of pipe2.

Step 6 - Perform the communication as required.



3. Achieving two-way communication using pipes

Algorithm

Step 1 - Create pipe1 for the parent process to write and the child process to read.

Step 2 - Create pipe2 for the child process to write and the parent process to read.

Step 3 – Close the unwanted ends of the pipe from the parent and child side.

Step 4 - Parent process to write a message and child process to read and display on the screen. **Step 5** –

Child process to write a message and parent process to read and display on the screen.

Expected Output:

In Parent: Writing to pipe 1 - Message is Hi

In Child: Reading from pipe 1 - Message is Hi

In Child: Writing to pipe 2 – Message is Hello

In Parent: Reading from pipe 2 – Message is Hello

Practice:**Q. Write the output of the following program**

```
#include<stdio.h>
#include<unistd.h>
#include <sys/wait.h>
int main()int p[2];
char buff[25];
if(fork()==0)
printf("Child : Writing to pipe n");
write(p[1],"Welcome",8);
printf("Child Exiting\n");
else
wait(NULL);
printf("Parent : Reading from pipe \n");
read(p[0],buff,8);
printf("Pipe content is : %s
\n",buff);
return 0;
```

Output:**Result:**

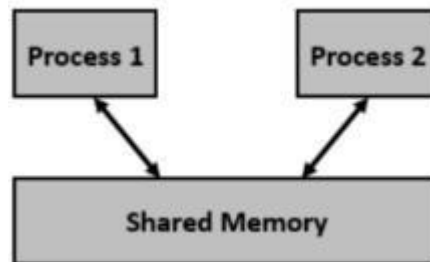
Ex. No. 12(a)**INTER PROCESS COMMUNICATION – SHARED MEMORY****Date:****Aim:****Inter Process Communication:**

The popular IPC techniques are Shared Memory and Message Queues.

Shared memory:

We know that to communicate between two or more processes, we use shared memory but before using the shared memory what needs to be done with the system calls,

- Create the shared memory segment or use an already created shared memory segment (`shmget()`)
- Detach the process from the already attached shared memory segment (`shmdt()`)
- Control operations on the shared memory segment (`shmctl()`)



Inter Process Communication through shared memory is a concept where two or more process can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by another process.

The problem with pipes, fifo and message queue – is that for two process to exchange information. The information has to go through the kernel.

- Server reads from the input file.
- The server writes this data in a message using either a pipe, fifo or message queue.
- The client reads the data from the IPC channel, again requiring the data to be copied from kernel's IPC buffer to the client's buffer.
- Finally the data is copied from the client's buffer.

A total of four copies of data are required (2 read and 2 write). So, shared memory provides a way by letting two or more processes share a memory segment. With Shared Memory the data is only copied twice – from input file into shared memory and from shared memory to the output file.

SYSTEM CALLS USED ARE:

- **ftok()**: is use to generate a unique key.
- **shmget()**: `int shmget(key_t, size_tsize, intshmflg);` upon successful completion, `shmget()` returns an identifier for the shared memory segment.
- **shmat()**: Before you can use a shared memory segment, you have to attach yourself to it using `shmat()`. `void *shmat(int shmid , void *shmaddr , int shmflg);`
- `shmid` is shared memory id. `shmaddr` specifies specific address to use but we should set it to zero and OS will automatically choose the address.
- **shmdt()**: When you're done with the shared memory segment, your program should detach itself from it using `shmdt()`. `int shmdt(void *shmaddr);`
- **shmctl()**: when you detach from shared memory, it is not destroyed. So, to destroy `shmctl()` is used. `shmctl(int shmid,IPC_RMID,NULL);` The second argument, `cmd`, is the command to perform the required control operation on the shared memory segment.

Valid values for `cmd` are -

- **IPC_STAT** - Copies the information of the current values of each member of `struct shmid_ds` to the passed structure pointed by `buf`. This command requires read permission to the sharedmemory segment.
- **IPC_SET** - Sets the user ID, group ID of the owner, permissions, etc. pointed to by structure `buf`.
- **IPC_RMID** – Marks the segment to be destroyed. The segment is destroyed only after the last process has detached it. **IPC_INFO** - Returns the information about the shared memory limits and parameters in the structure pointed by `buf`.
- **SHM_INFO** – Returns a `shm_info` structure containing information about the consumed system resources by the shared memory.

Program:1- Shared memory implementation using readers writers problem.

Writer process:

Algorithm:

Reader process:

Algorithm:

Output:

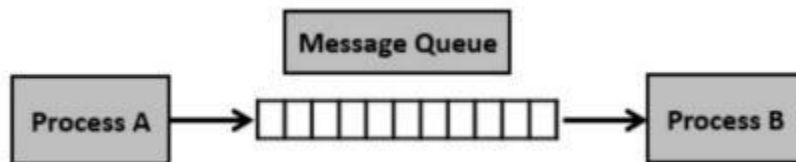
Result:

Ex. No. 12(b)**INTER PROCESS COMMUNICATION – MESSAGE QUEUE****Date:****Aim:****Inter Process Communication-Message Queue**

A message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. A new queue is created or an existing queue opened **by msgget()**.

New messages are added to the end of a queue by **msgsnd()**. Every message has a positive long integer type field, a non-negative length, and the actual data bytes (corresponding to the length), all of which are specified to **msgsnd()** when the message is added to a queue. Messages are fetched from a queue by **msgrcv()**. We don't have to fetch the messages in a first-in, first-out order. Instead, we can fetch messages based on their type field.

All processes can exchange information through access to a common system message queue. The sending process places a message (via some (OS) message-passing module) onto a queue which can be read by another process. Each message is given an identification or type so that processes can select the appropriate message. Process must share a common key in order to gain access to the queue in the first place.



System calls used for message queues:

- **ftok():** is use to generate a unique key.
- **msgget():** either returns the message queue identifier for a newly created message queue or returns the identifiers for a queue which exists with the same key value.
- **msgsnd():** Data is placed on to a message queue by calling **msgsnd()**.
- **msgrcv():** messages are retrieved from a queue.
- **msgctl():** It performs various operations on a queue. Generally it is use to destroy message queue.

Program :To perform communication using message queues, following are the steps -

Writer Process:

Reader Process:

Output:

Result:

Ex. No. 13**PROCESS SYNCHRONIZATION****Date:****Aim:****Semaphore**

Semaphore is used to implement process synchronization. This is to protect critical regions shared among multiple processes.

Include the following header files for shared memory

```
<sys/ipc.h>, <sys/sem.h>, <sys/types.h>
```

System V Semaphore System Calls

To create a semaphore array,

```
int semget(key_t key, int nsems, int semflg)
```

key -> semaphore id

nsems -> no. of semaphores in the semaphore array

semflg -> IPC_CREATE|0664 : to create a new semaphore

IPC_EXCL|IPC_CREAT|0664 : to create new semaphore and the call fails if the semaphore already exists

To perform operations on the semaphore sets viz., allocating resources, waiting for the resources or freeing the resources,

```
int semop(int semid, struct sembuf *semops, size_t nsemops)
```

semidsa-> semaphore id returned by semget()

nsemops -> the number of operations in that array

semops -> The pointer to an array of operations to be performed on the semaphore set.

The structure is as follows

```
struct sembuf {
    unsigned short sem_num; /* Semaphore set num */
    short sem_op; /* Semaphore operation */
    short sem_flg; /* Operation flags, IPC_NOWAIT, SEM_UNDO */
};
```

Element, sem_op, in the above structure, indicates the operation that needs to be performed –

- If sem_op is -ve, allocate or obtain resources. Blocks the calling process until enough resources have been freed by other processes, so that this process can allocate.
- If sem_op is zero, the calling process waits or sleeps until semaphore value reaches 0.
- If sem_op is +ve, release resources.

To perform control operation on semaphore,

```
int semctl(int semid, int semnum, int cmd,...);
```

semid -> identifier of the semaphore returned by semget()

semnum -> semaphore number

cmd -> the command to perform on the semaphore. Ex. GETVAL, SETVAL

semun -> value depends on the cmd. For few cases, this is not applicable.

Q1. Execute and write the output of the following program for *mutual exclusion*.

```

#include<sys/ipc.h>
#include<sys/sem.h>
int main()
{
    int pid,semid,val;
    struct sembuf sop;

    semid=semget((key_t)6,1,IPC_CREAT|0666);

    pid=fork();

    sop.sem_num=0;
    sop.sem_op=0;
    sop.sem_flg=0;

    if (pid!=0)
    {
        sleep(1);
        printf("The Parent waits for WAIT signal\n");
        semop(semid,&sop,1);
        printf("The Parent WAKED UP & doing her job\n");
        sleep(10);
        printf("Parent Over\n");
    }
    else
    {
        printf("The Child sets WAIT signal & doing her job\n");
        semctl(semid,0,SETVAL,1);
        sleep(10);
        printf("The Child sets WAKE signal & finished her job\n");
        semctl(semid,0,SETVAL,0);
        printf("Child Over\n");
    }
    return 0;
}

```

Output :

Q2. Write a program to perform process synchronization in producer-consumer problem

Result:

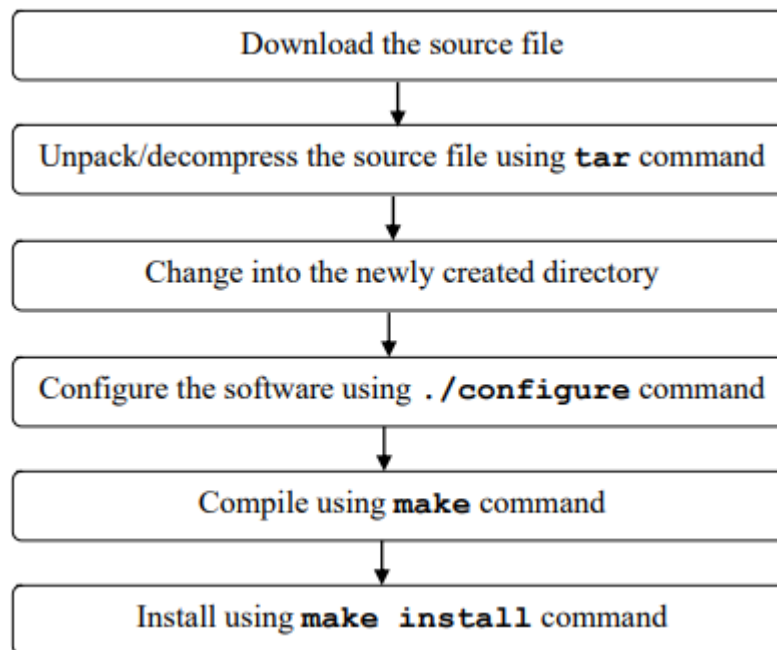
Ex. No. 14 &15

STUDY AND UNDERSTANDING OF OS161

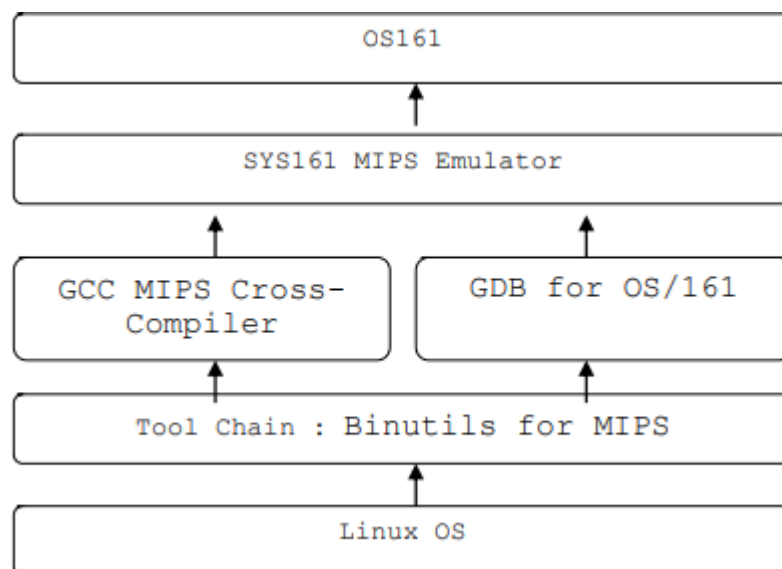
Date:

Aim:

STEPS TO BUILD SOFTWARE FROM SOURCE FILE



BUILD SOFTWARE FRAMEWORK FOR OS/161



OS/161 INSTALLATION

- Linux desktop with UBUNTU Version 12.04 or later.
- Internet connections to download and install packages

Pre Installation Steps

1. Install the packages gettext (to translate native language statements into English) ,texinfo (to translate source code into other formats) and libncurses5-dev (allows users to write text-based GUI)

```
sudo apt-get install gettext
sudo apt-get install texinfo
sudo apt-get install libncurses5-dev
```

2. Include the paths \$HOME/sys161/bin and \$HOME/sys161/tools/bin into the PATH environment variable. Add the following line at the end of the file **.bashrc**

```
export PATH=$HOME/sys161/bin:$HOME/sys161/tools/bin:$PATH
```

Now logout and login to get the PATH updated. You can check the current setting of the PATH environment variable using the command

```
printenv PATH
```

Installation Steps

STEP 1: Download the following source codes one by one

- **Binutils for MIPS**
http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161-binutils.tar.gz
- **GCC MIPS Cross-Compiler**
http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161-gcc.tar.gz
- **GDB for Use with OS/161**
http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161-gdb.tar.gz
- **bmake for use with OS/161**
http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161-bmake.tar.gz
- **mk for use with OS/161**
http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161-mk.tar.gz
- **sys/161**
http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/sys161.tar.gz
- **OS/161**
http://www.student.cs.uwaterloo.ca/~cs350/os161_repository/os161.tar.gz

Note : bmake and mk utilities are BSD make utilities used for OS161

STEP 2: Build and Install the Binary Utilities (Binutils)

Unpack the binutils archive:

```
tar -xzf os161-binutils.tar.gz
```

Move into the newly-created directory:

```
cd binutils-2.17+os161-2.0.1
```

Configure binutils:

```
./configure --nfp --disable-werror --target=mips  
harvard-os161 --prefix=$HOME/sys161/tools
```

Make binutils:

```
make
```

Finally, once **make** has succeeded, install the binutils into their final location:

```
make install
```

This will create the directory **\$HOME/sys161/tools/** and populate it.

Step 3: Install the GCC MIPS Cross-Compiler

Unpack the gcc archive:

```
tar -xzf os161-gcc.tar.gz
```

Move into the newly-created directory:

```
cd gcc-4.1.2+os161-2.0
```

Configure gcc

```
./configure -nfp --disable-shared --disable-threads --  
disable-libmudflap --disable-libssp --target=mips  
harvard-os161 --prefix=$HOME/sys161/tools
```

Make it and install it:

```
make
```

```
make install
```

Step 4: Install GDB

Unpack the gdb archive:

```
tar -xzf os161-gdb.tar.gz
```

Move into the newly-created directory:

```
cd gdb-6.6+os161-2.0
```

Configure gdb

```
./configure --target=mips-harvard-os161 --  
prefix=$HOME/sys161/tools --disable-werror
```

Make it and install it:

```
make
```

```
make install
```

Step 5: Install bmake

Unpack the bmake archive:

```
tar -xzf os161-bmake.tar.gz
```

Move into the newly-created directory:

```
cd bmake
```

Unpack mk within the bmake directory:

```
tar -xzf ../os161-mk.tar.gz
```

Run the bmake bootstrap script

```
./boot-strap --prefix=$HOME/sys161/tools
```

As the **boot-strap** script finishes, it should print a list of commands that you can run to install bmake under **\$HOME/sys161/tools**. The list should look something like this:

```
mkdir -p /home/kmsalem/sys161/tools/bin
cp /home/kmsalem/bmake/Linux/bmake
/home/kmsalem/sys161/tools/bin/bmake-20101215
rm -f /home/kmsalem/sys161/tools/bin/bmake
ln -s bmake-20101215 /home/kmsalem/sys161/tools/bin/bmake
mkdir -p /home/kmsalem/sys161/tools/share/man/cat1
cp /home/kmsalem/bmake/bmake.cat1
/home/kmsalem/sys161/tools/share/man/cat1/bmake.1
sh /home/kmsalem/bmake/mk/install-mk
/home/kmsalem/sys161/tools/share/mk
```

Run the commands printed by boot-strap in the order in which they are listed in your terminal screen.

Step 6: Set Up Links for Toolchain Binaries

```
mkdir $HOME/sys161/bin
```

```
cd $HOME/sys161/tools/bin
```

```
sh -c 'for i in mips-*; do ln -s $HOME/sys161/tools/bin/$i
$HOME/sys161/bin/cs350-`echo $i | cut -d- -f4-`; done'
```

```
ln -s $HOME/sys161/tools/bin/bmake $HOME/sys161/bin/bmake
```

When you are finished with these steps, a listing of the directory

\$HOME/sys161/bin should look similar to this:

```
bmake@ cs350-gcc@ cs350-ld@ cs350-run@ cs350-addr2line@ cs350-gcc-4.1.2@
cs350-nm@ cs350-size@ cs350-ar@ cs350-gccbug@ cs350-objcopy@ cs350-
strings@ cs350-as@ cs350-gcov@ cs350-objdump@ cs350-strip@ cs350-c++filt@
cs350-gdb@ cs350-ranlib@
cs350-cpp@ cs350-gdbtui@ cs350-readelf@
```

Step 7: Build and Install the sys161 Simulator

Unpack the sys161 archive:

```
tar -xzf sys161.tar.gz
```

Move into the newly-created directory:

```
cd sys161-1.99.06
```

Next, configure sys161:

```
./configure --prefix=$HOME/sys161 mipseb
```

Build sys161 and install it:

```
make  
make install
```

Finally, set up a link to a sample sys161 configuration file

```
cd $HOME/sys161  
ln -s share/examples/sys161/sys161.conf.sample  
sys161.conf
```

Step 8: Install OS/161

First, create a directory to hold the OS/161 source code, your compiled OS/161 kernels, and related test programs.

```
cd $HOME  
mkdir cs350-os161
```

Next, move the OS/161 archive into your new directory and unpack it: **mv**

```
os161.tar.gz cs350-os161  
cd cs350-os161  
tar -xzf os161.tar.gz
```

This will create a directory called **os161-1.99** (under **cs350-os161**) containing the OS/161 source code. You should now be able build, install, and run an OS/161 kernel and related application and test programs by following steps.

Step 9: Configure OS/161 and Build the OS/161 Kernel

The next step is to configure OS/161 and compile the kernel. From the **cs350-os161** directory, do the following:

```
cd os161-1.99  
./configure --ostree=$HOME/cs350-os161/root --  
toolprefix=cs350-  
cd kern/conf  
./config ASST0  
cd ../compile/ASST0  
bmake depend  
bmake  
bmake install
```

Step 10: Build the OS/161 User-level

Next, build the OS/161 user level utilities and test programs:

```
cd $HOME/cs350-os161/os161-1.99  
bmake  
bmake install
```

Step 11: Try Running OS/161

You should now be able to use the SYS/161 simulator to run the OS/161 kernel that you built and installed. The SYS/161 simulator requires a configuration file in order to run. To obtain one, do this:

```
cd $HOME/cs350-os161/root
cp $HOME/sys161/sys161.conf sys161.conf
sys161 kernel-ASST0
```

You should see some output that looks something like this:

```
sys161: System/161 release 1.99.06, compiled Aug 23 2013 10:23:34
```

```
OS/161 base system version 1.99.05
Copyright (c) 2000, 2001, 2002, 2003, 2004, 2005, 2008, 2009
President and Fellows of Harvard College. All rights reserved.
```

```
Put-your-group-name-here's system version 0 (ASST0 #1)
```

```
316k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
ltrace0 at lamebus0
ltimer0 at lamebus0
beep0 at ltimer0
rtclock0 at ltimer0
lrandom0 at lamebus0
random0 at lrandom0
lhd0 at lamebus0
lhd1 at lamebus0
lser0 at lamebus0
con0 at lser0
```

```
cpu0: MIPS r3000
OS/161 kernel [? for menu]:
```

The last line is a command prompt from the OS/161 kernel. For now, just enter the command **q** to shut down the simulation and return to your shell.

Result: