Name: Meet Brijwani

Roll no: 14

Batch: S11

EXPERIMENT NO : 01

Aim: Explore usage of basic linux commands and system calls for files, directory and process management.

Theory:

1. who : it is used to find out the current user who is logged into the system.
2. pwd : present working directory, lets you know the current directory you are in.
3. cal : show the calendar of the complete month.
4. date : It shows you the current date, along with the time, along with the day, along with the year.
5. mkdir: to create a new directory under any directory
6. chdir/cd : to change the current working directory
7. cat : to create the file and display the contents of the file
8. chmod: to change the mode of the file. There are three modes read(r), write(w) and execute(e)
9. ls : to list all directories and subdirectories

a. ls-l : to show the long listing information about the directory
b. ls-lh : human readable format.
c. ls-ld : shows the details of the directory content.
d. ls-d* : to show the sub directories in a directory
e. ls-a : to show hidden files
f. ls-lhs : show files in the descending order in which you have used your files.

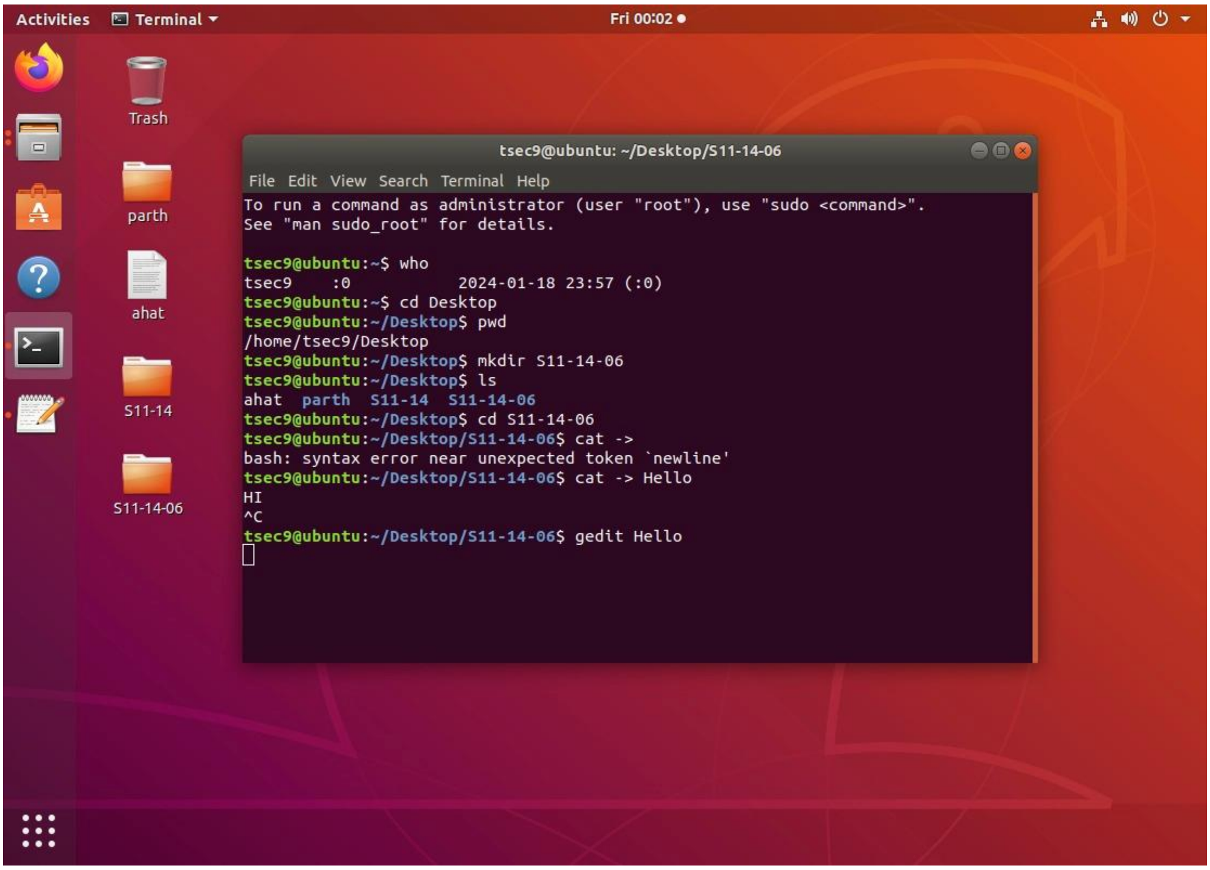
10. sort-r file name.txt : sorts the list in reverse order
11. sort-n file name.txt : its sorts the numerical list in ascending order
12. sort nr file name.txt : its sorts the numerical list in reverse order
13. sort u file name.txt : to remove the duplicates
14. sort m file name.txt : Sorts the months in ascending order
15. awk : it is used for the user that defines text patterns that are to be searched for each line of the file.

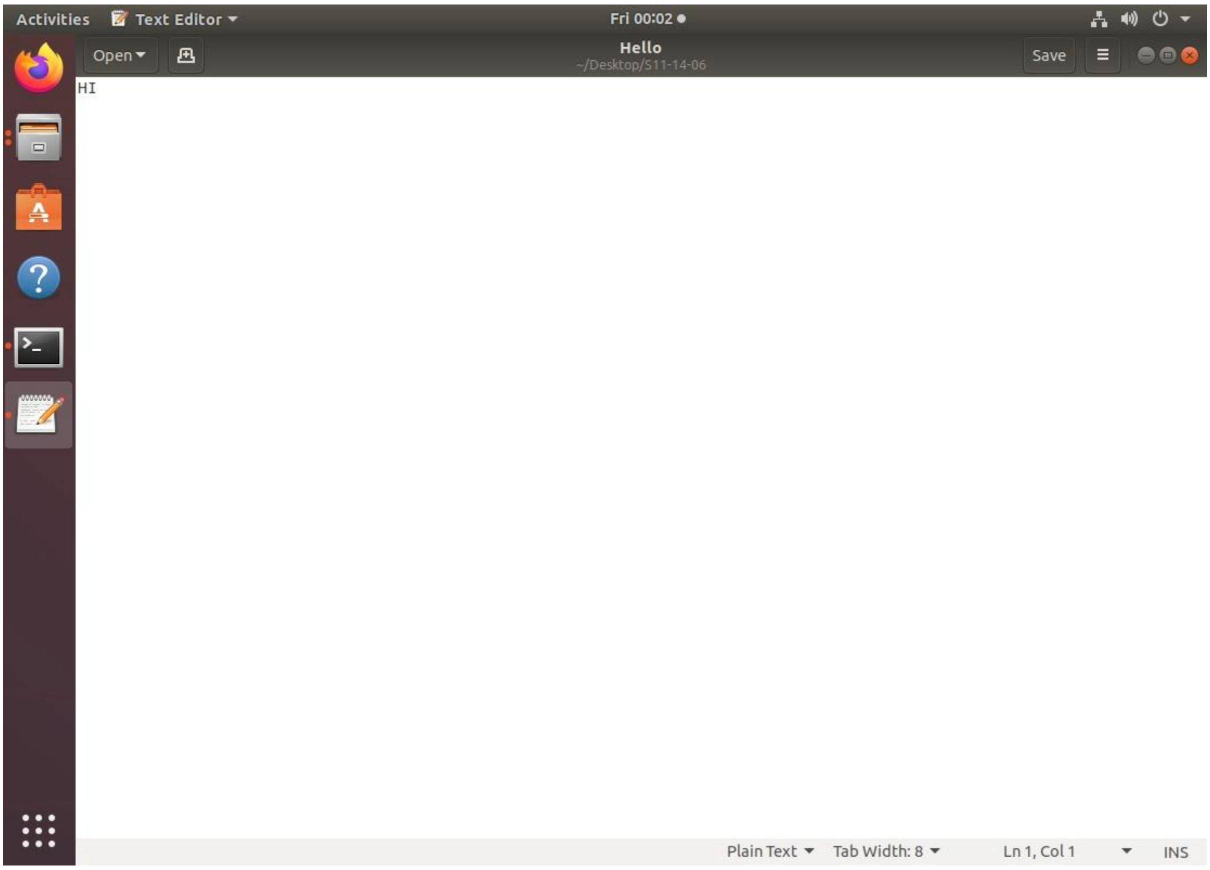
    Syntax , awk '{print}' file name.txt

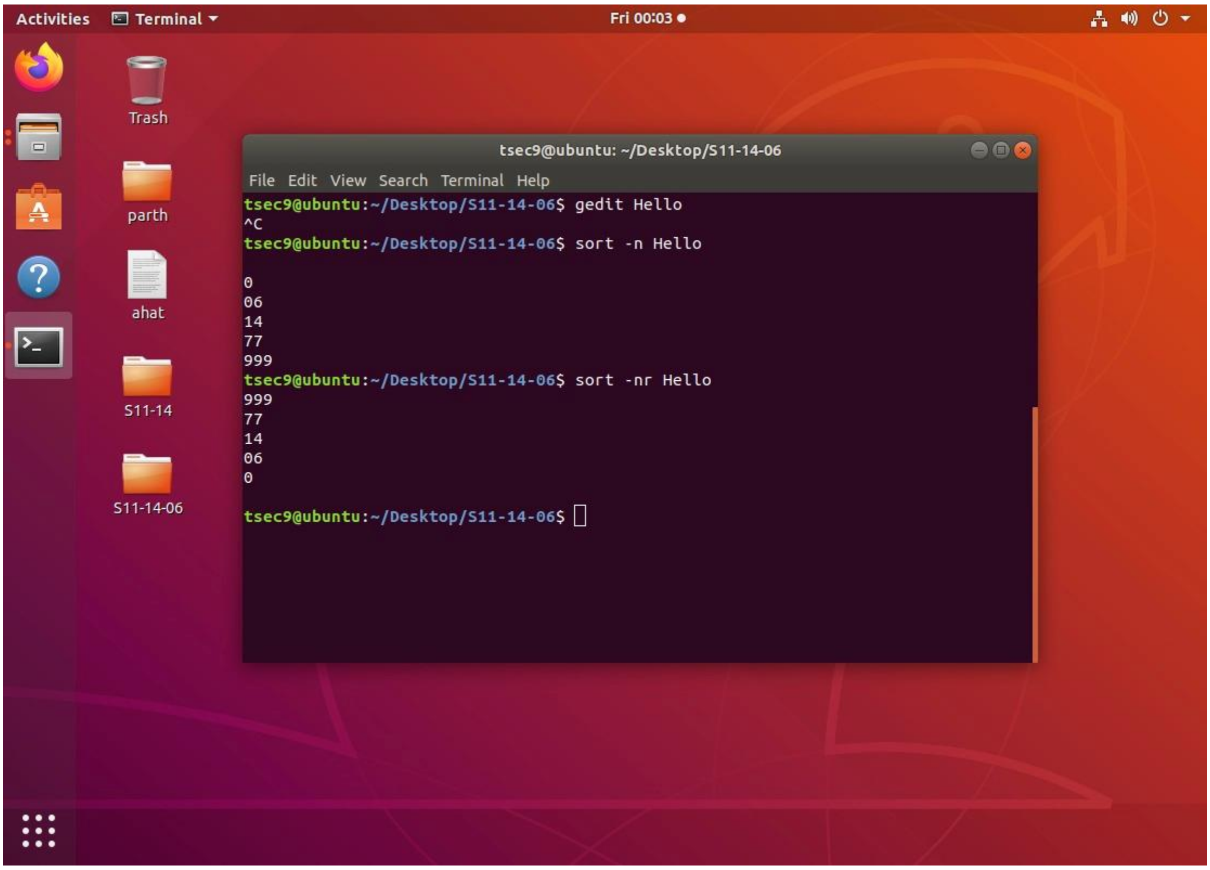
        awk '/faculty/{print}' file name.txt :

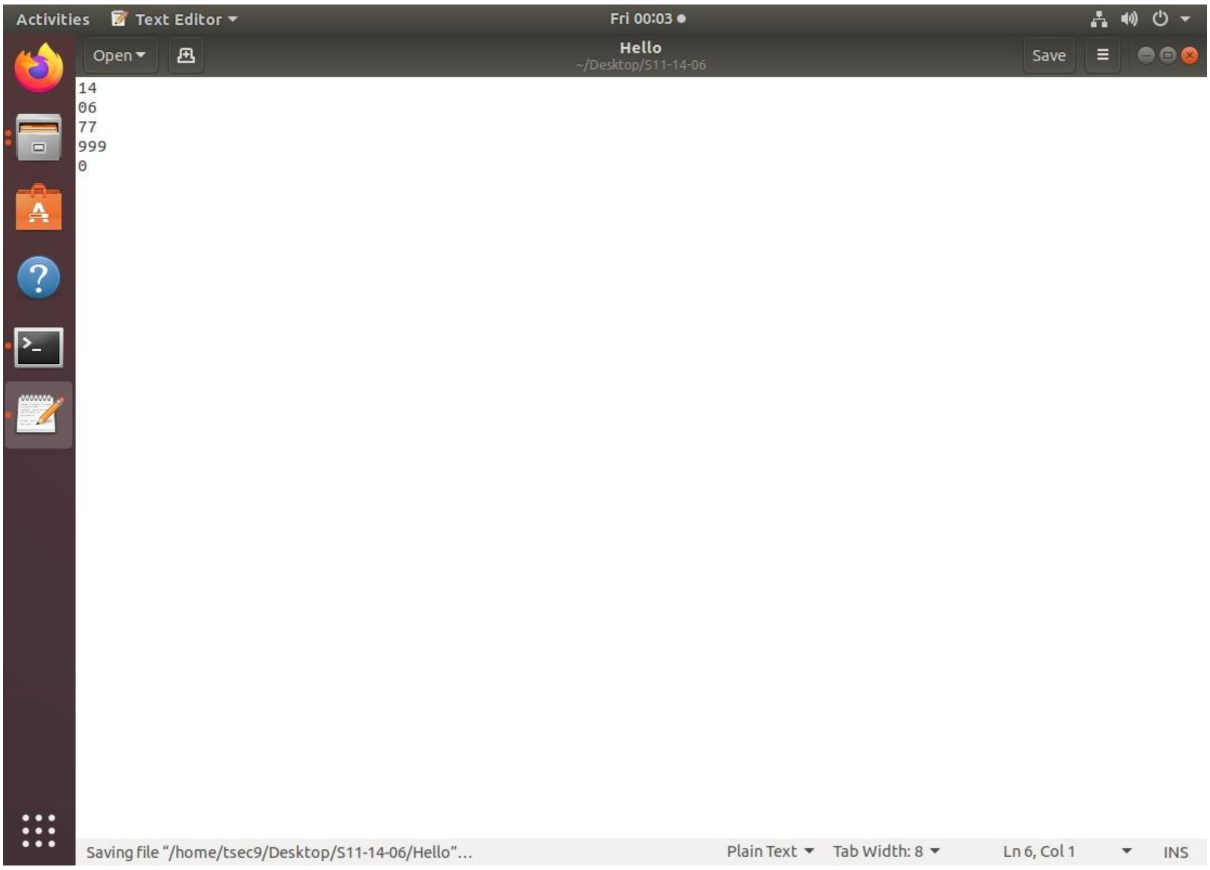
awk '{print}NR, $0}' file name.txt :
NR - specifies the number of lines.

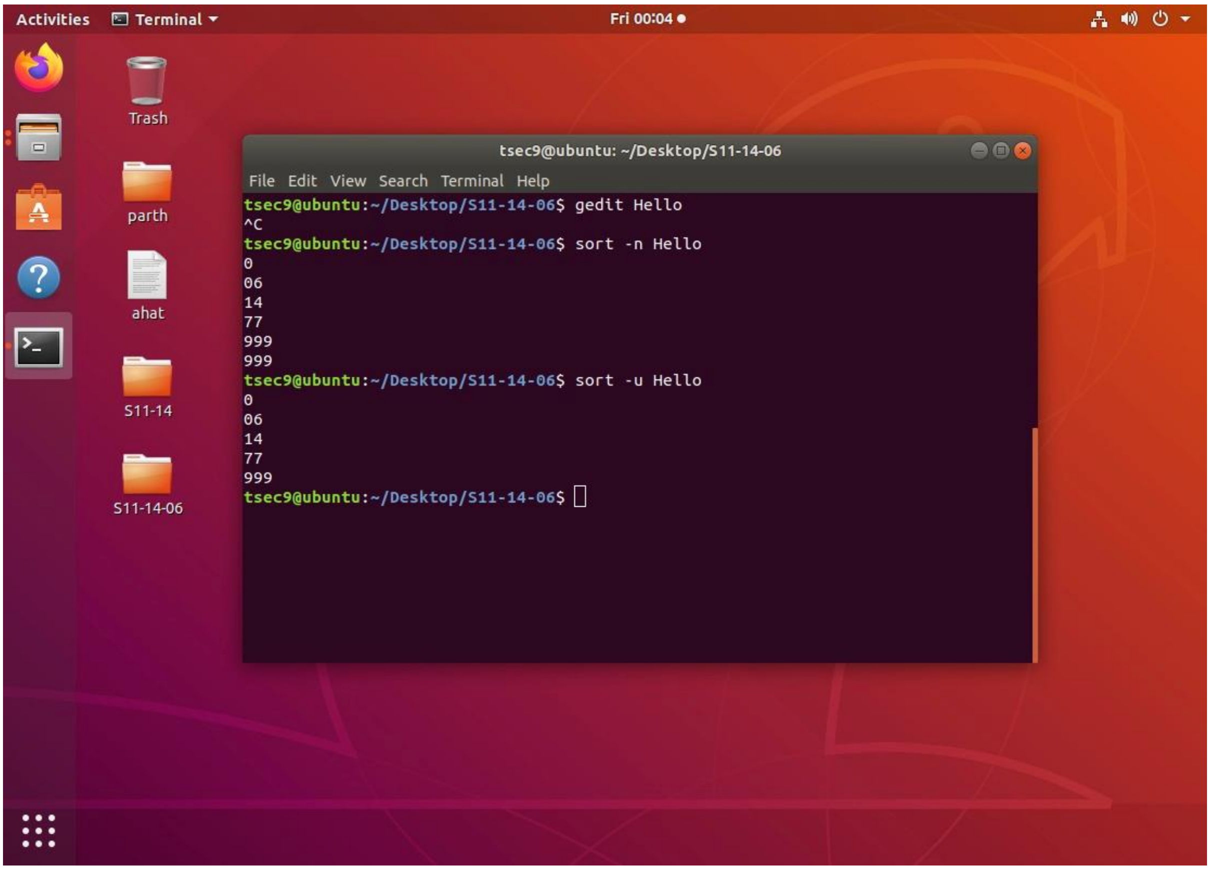
Output :

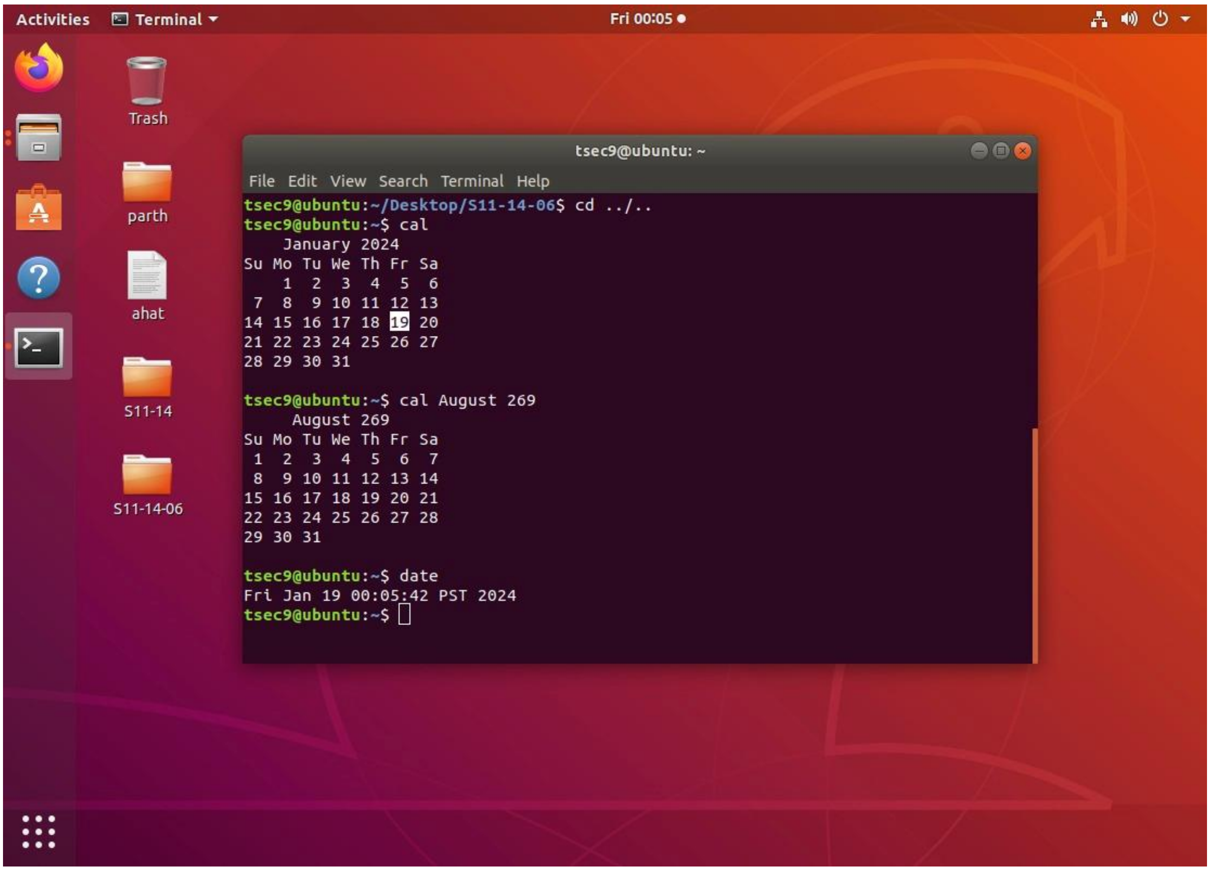


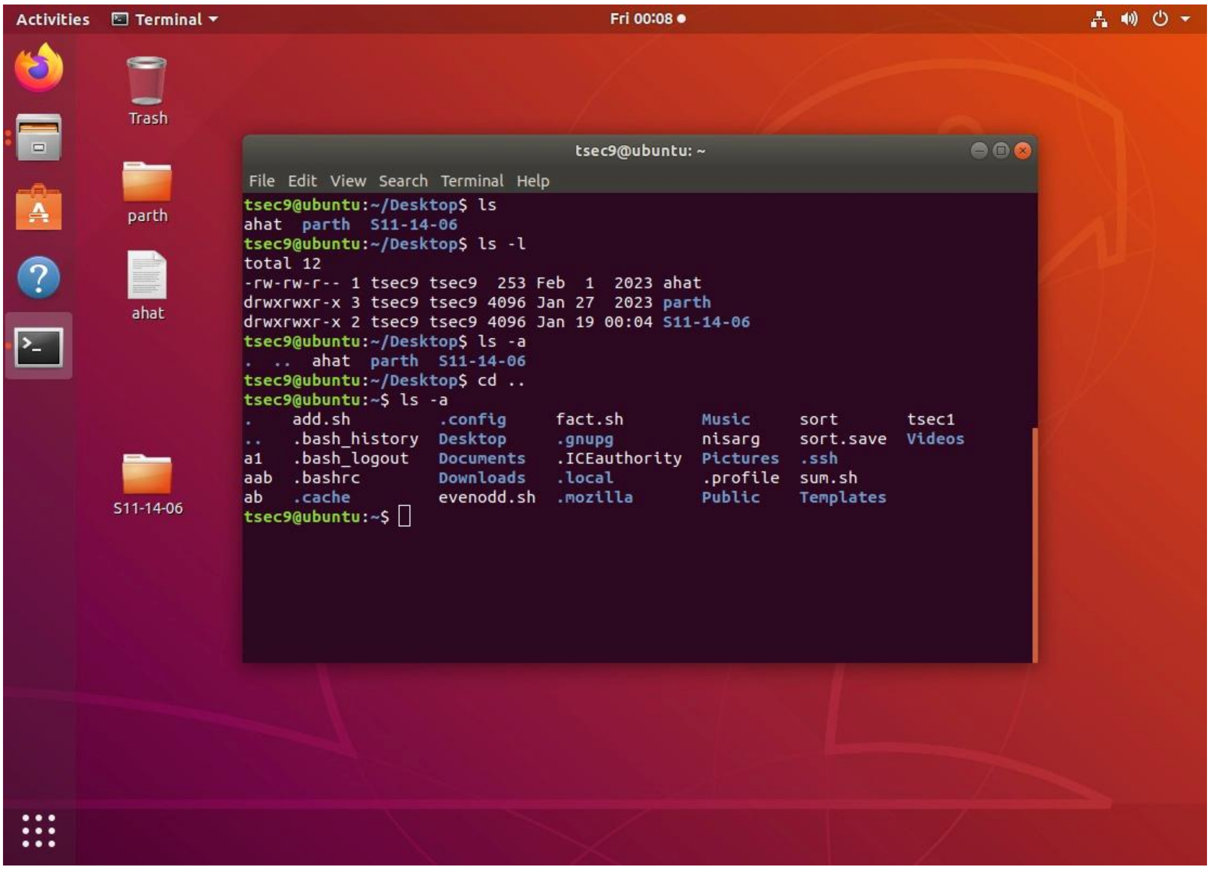

Activities
Text Editor
Fri 00:02
Open
Hello
~/Desktop/S11-14-06
Save
HI
Plain Text
Tab Width: 8
Ln 1, Col 1
INS


Activities
Terminal
Fri 00:03
Trash
parth
ahat
S11-14
S11-14-06
tsec9@ubuntu: ~/Desktop/S11-14-06
File Edit View Search Terminal Help
tsec9@ubuntu:~/Desktop/S11-14-06$ gedit Hello
^C
tsec9@ubuntu:~/Desktop/S11-14-06$ sort -n Hello
0
06
14
77
999
tsec9@ubuntu:~/Desktop/S11-14-06$ sort -nr Hello
999
77
14
06
0
tsec9@ubuntu:~/Desktop/S11-14-06$


Activities
Text Editor
Fri 00:03
Open
Hello
~/Desktop/S11-14-06
Save
14
06
77
999
0
Saving file "/home/tsec9/Desktop/S11-14-06/Hello"…
Plain Text
Tab Width: 8
Ln 6, Col 1
INS


Activities
Terminal
Fri 00:04
Trash
parth
ahat
S11-14
S11-14-06
tsec9@ubuntu: ~/Desktop/S11-14-06
File Edit View Search Terminal Help
tsec9@ubuntu:~/Desktop/S11-14-06$ gedit Hello
^C
tsec9@ubuntu:~/Desktop/S11-14-06$ sort -n Hello
0
06
14
77
999
999
tsec9@ubuntu:~/Desktop/S11-14-06$ sort -u Hello
0
06
14
77
999
tsec9@ubuntu:~/Desktop/S11-14-06$


Activities
Terminal
Fri 00:05
Trash
parth
ahat
S11-14
S11-14-06
tsec9@ubuntu: ~
File Edit View Search Terminal Help
tsec9@ubuntu:~/Desktop/S11-14-06$ cd ../..
tsec9@ubuntu:~$ cal
January 2024
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
tsec9@ubuntu:~$ cal August 269
August 269
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
tsec9@ubuntu:~$ date
Fri Jan 19 00:05:42 PST 2024
tsec9@ubuntu:~$


Activities
Terminal
Fri 00:08
Trash
parth
ahat
S11-14-06
tsec9@ubuntu: ~
File Edit View Search Terminal Help
tsec9@ubuntu:~/Desktop$ ls
ahat parth S11-14-06
tsec9@ubuntu:~/Desktop$ ls -l
total 12
-rw-rw-r-- 1 tsec9 tsec9 253 Feb 1 2023 ahat
drwxrwxr-x 3 tsec9 tsec9 4096 Jan 27 2023 parth
drwxrwxr-x 2 tsec9 tsec9 4096 Jan 19 00:04 S11-14-06
tsec9@ubuntu:~/Desktop$ ls -a
. .. ahat parth S11-14-06
tsec9@ubuntu:~/Desktop$ cd ..
tsec9@ubuntu:~$ ls -a
. add.sh .config fact.sh Music sort tsec1
.. .bash_history Desktop .gnupg nisarg sort.save Videos
a1 .bash_logout Documents .ICEauthority Pictures .ssh
aab .bashrc Downloads .local .profile sum.sh
ab .cache evenodd.sh .mozilla Public Templates
tsec9@ubuntu:~$

Name: Meet Brijwani

Roll no: 14

Batch: S11

Experiment 2: Linux shell script

Aim:

Write shell scripts to do the following:

a. Display OS version, release number, kernel version

b. Display top 10 processes in descending order

c. Display processes with highest memory usage.

d. Display current logged in user and log name.

Display current shell, home directory, operating system type, current path setting, current working directory.

Theory:

A]

i) uname - r{username}
To show the OS version, release no. OS release version
ii) uname - v
Shows Kernel version (last when launched)
iii) uname - a
Shows all the details including OS used, PC and other details.
us - c + u - r → us - a

B]

i) ps-aux (i) sort : nk +41 tail
Sorts 10 processes in descending order. It shows logged in users, modes of memory & storage.
OR
ii) ps-aux | tail
Same as above
OR

iii) ps-aux | sort -nl+4 | tail - n15 (shows top 15 processes)
same with 15

C]

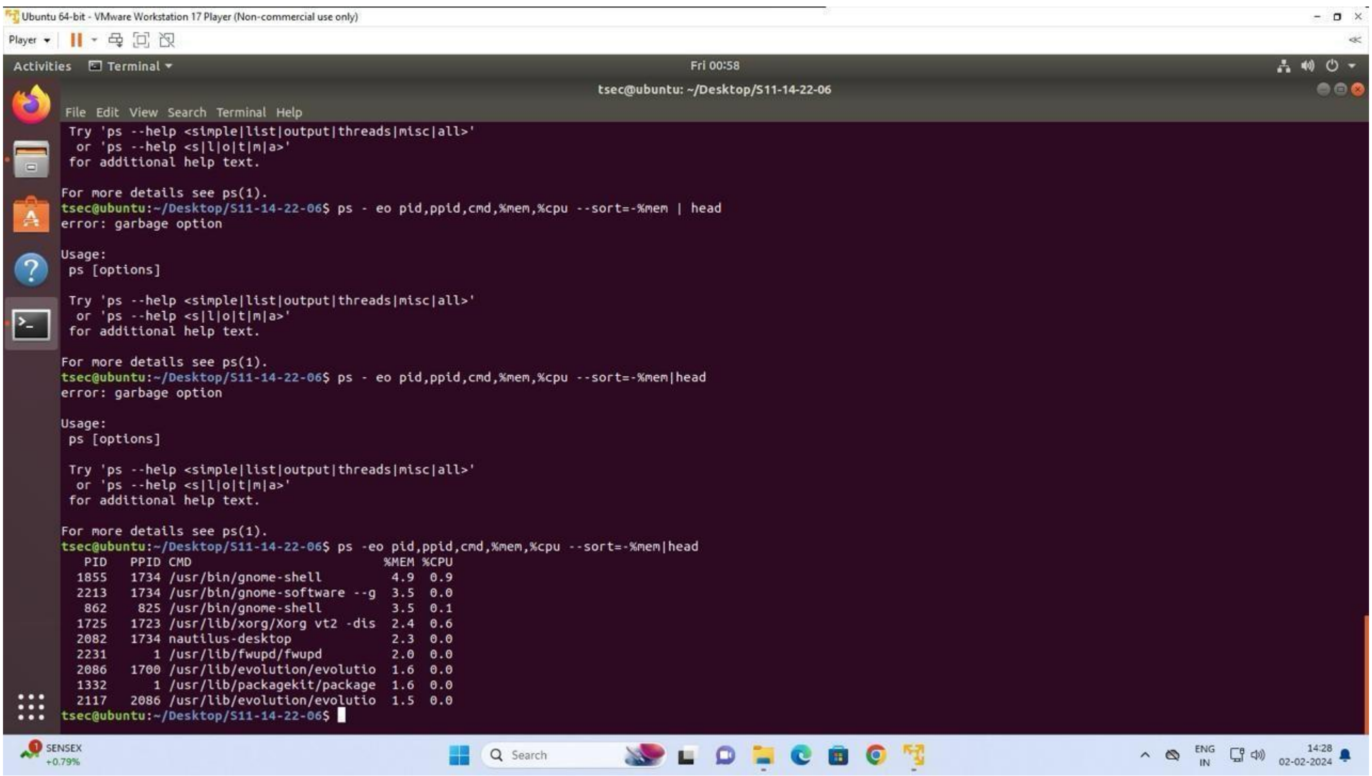
i) sudo apt install htop
shows date of processes with highest memory
ii) ps - eo pid, ppid, cmd, % mem, % cpu –sort =-% mem | head
same as above same as above

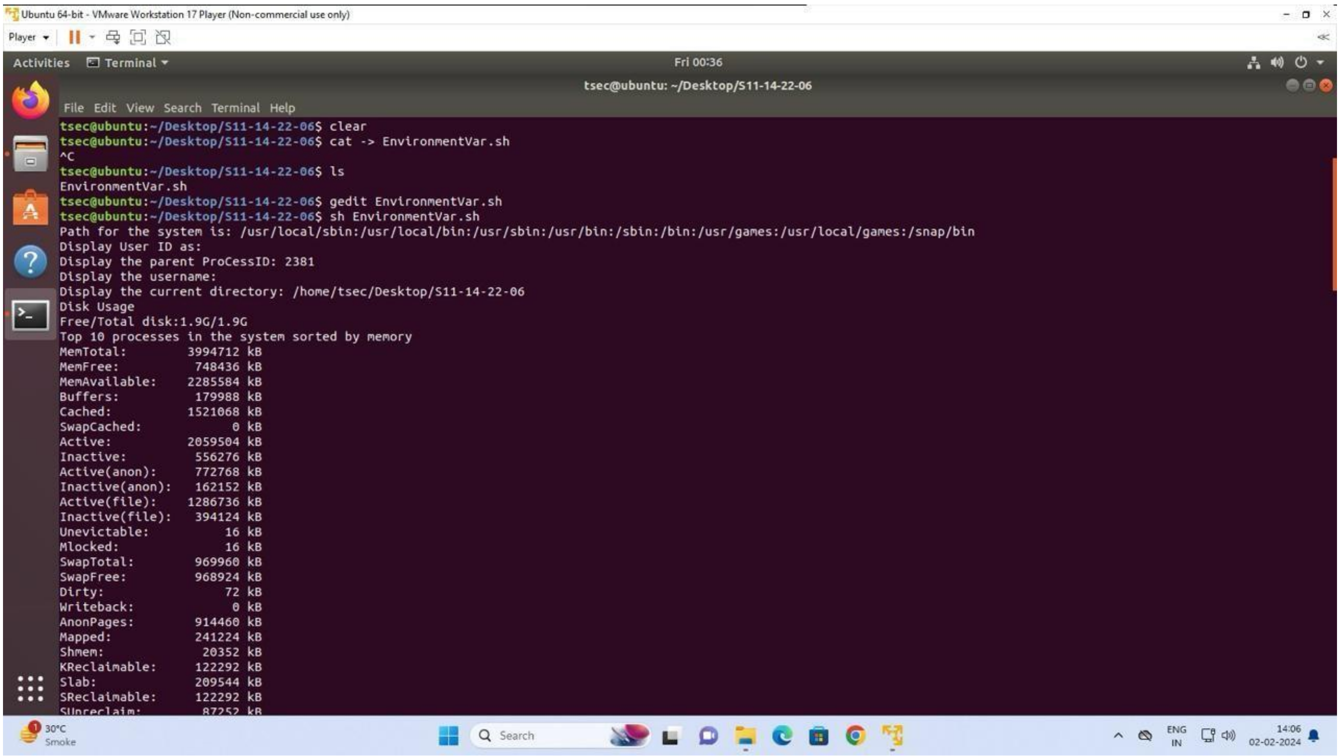
D]

i) ps-p$$
Shows process ID and current shell

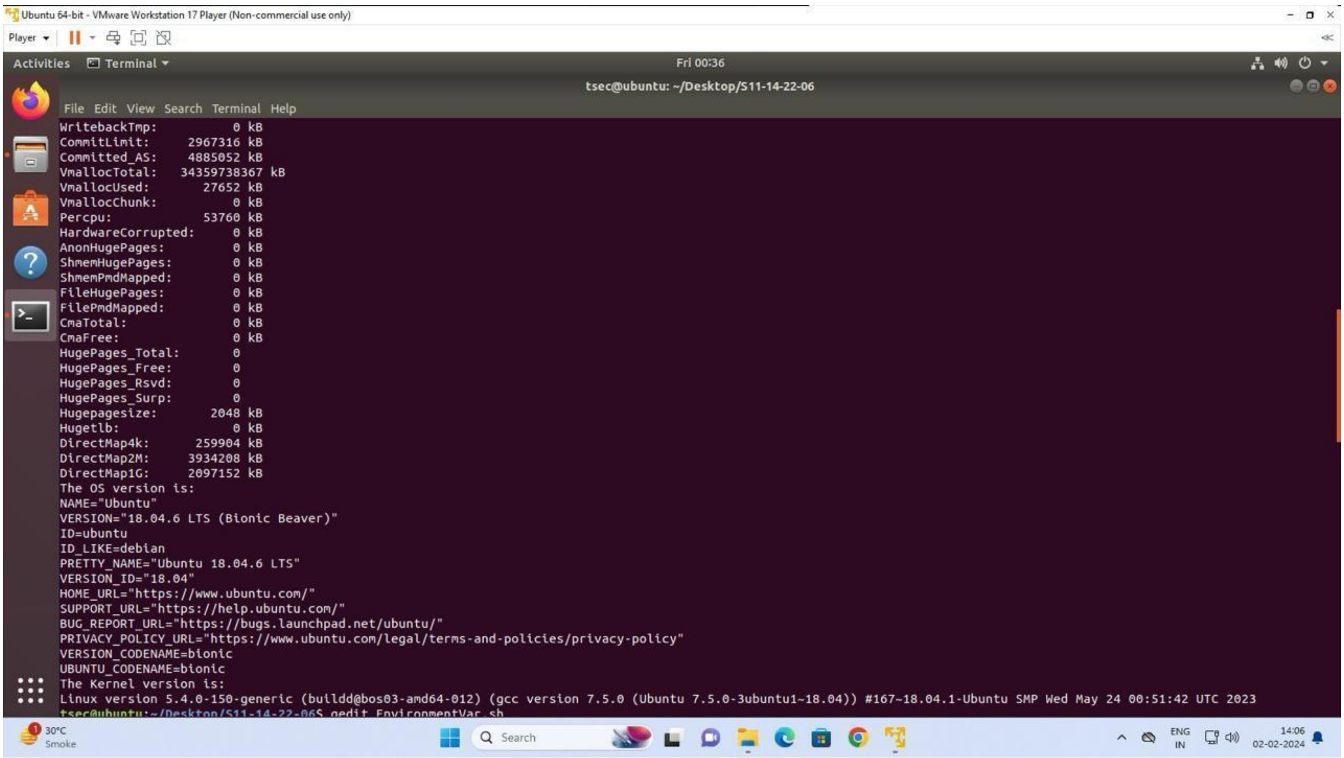
ii) echo display home directory
echo $ home
To display home directory

Output:

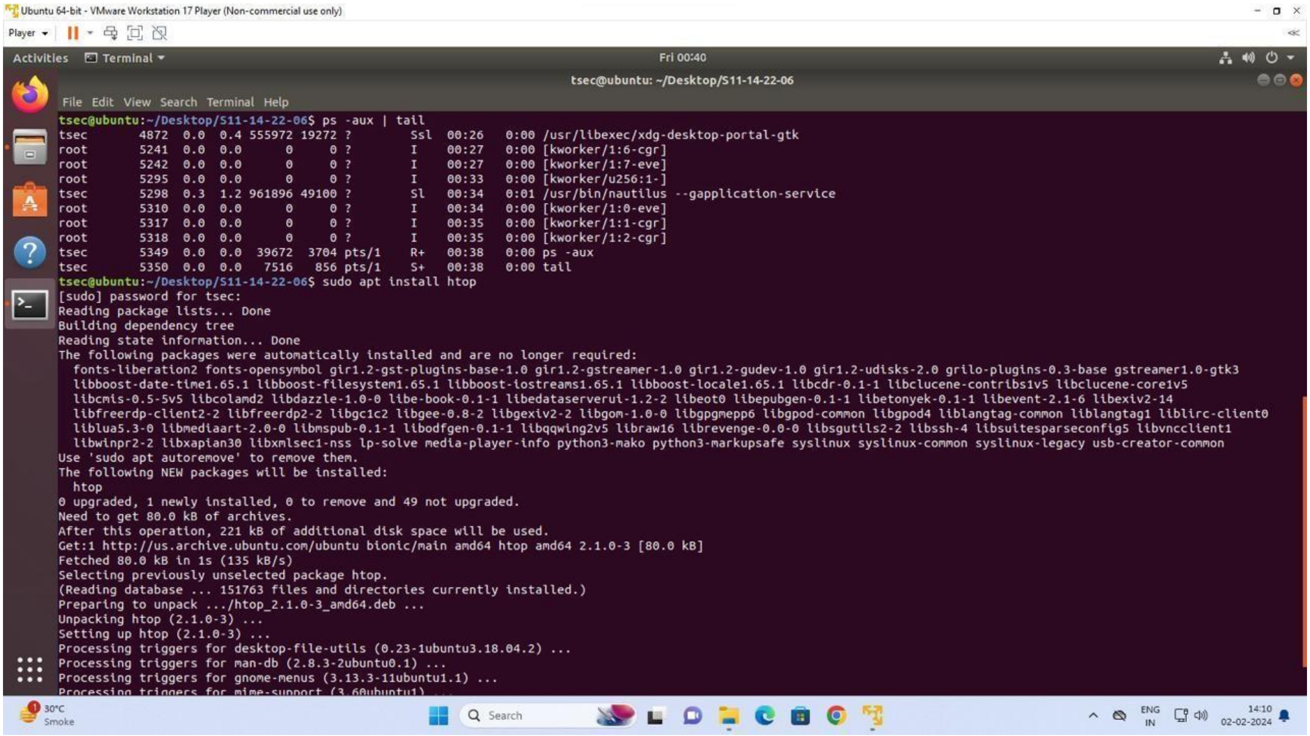


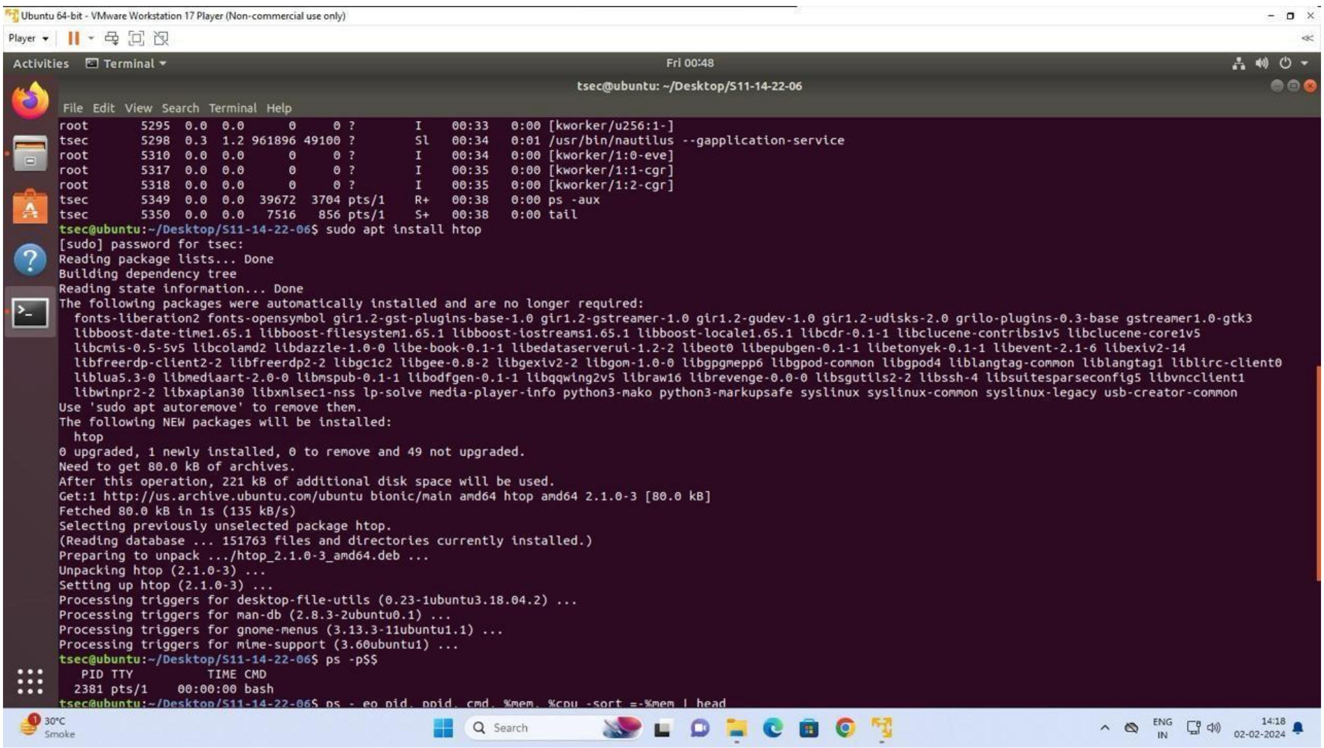


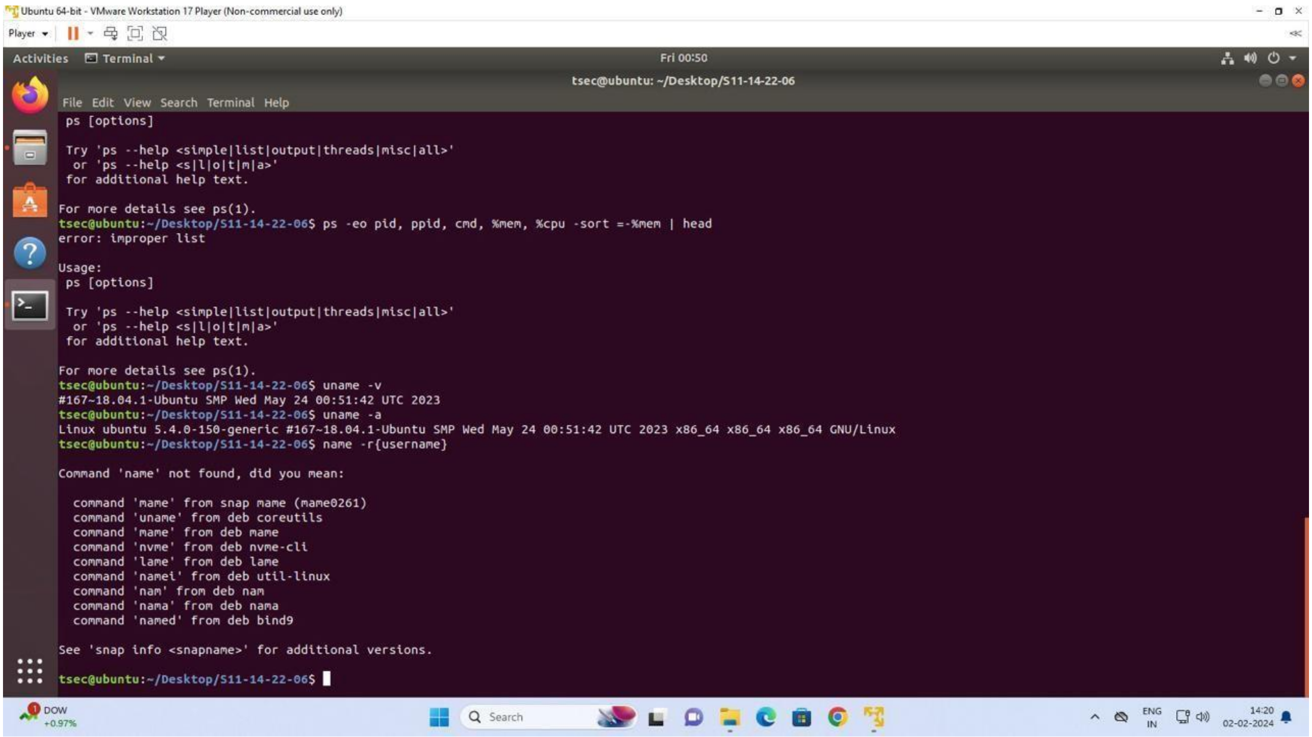
```
tsec@ubuntu:~/Desktop/S11-14-22-06$ clear
tsec@ubuntu:~/Desktop/S11-14-22-06$ cat -> EnvironmentVar.sh
^C
tsec@ubuntu:~/Desktop/S11-14-22-06$ ls
EnvironmentVar.sh
tsec@ubuntu:~/Desktop/S11-14-22-06$ gedit EnvironmentVar.sh
tsec@ubuntu:~/Desktop/S11-14-22-06$ sh EnvironmentVar.sh
Path for the system is: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
Display User ID as:
Display the parent ProcessID: 2381
Display the username:
Display the current directory: /home/tsec/Desktop/S11-14-22-06
Disk Usage
Free/Total disk:1.9G/1.9G
Top 10 processes in the system sorted by memory
MemTotal:        3994712 kB
MemFree:          748436 kB
MemAvailable:    2285584 kB
Buffers:          179988 kB
Cached:          1521068 kB
SwapCached:            0 kB
Active:          2059504 kB
Inactive:         556276 kB
Active(anon):     772768 kB
Inactive(anon):   162152 kB
Active(file):    1286736 kB
Inactive(file):   394124 kB
Unevictable:          16 kB
Mlocked:              16 kB
SwapTotal:        969960 kB
SwapFree:         968924 kB
Dirty:                72 kB
Writeback:             0 kB
AnonPages:        914460 kB
Mapped:           241224 kB
Shmem:             20352 kB
KReclaimable:     122292 kB
Slab:             209544 kB
SReclaimable:     122292 kB
SUnreclaim:        87252 kB
```

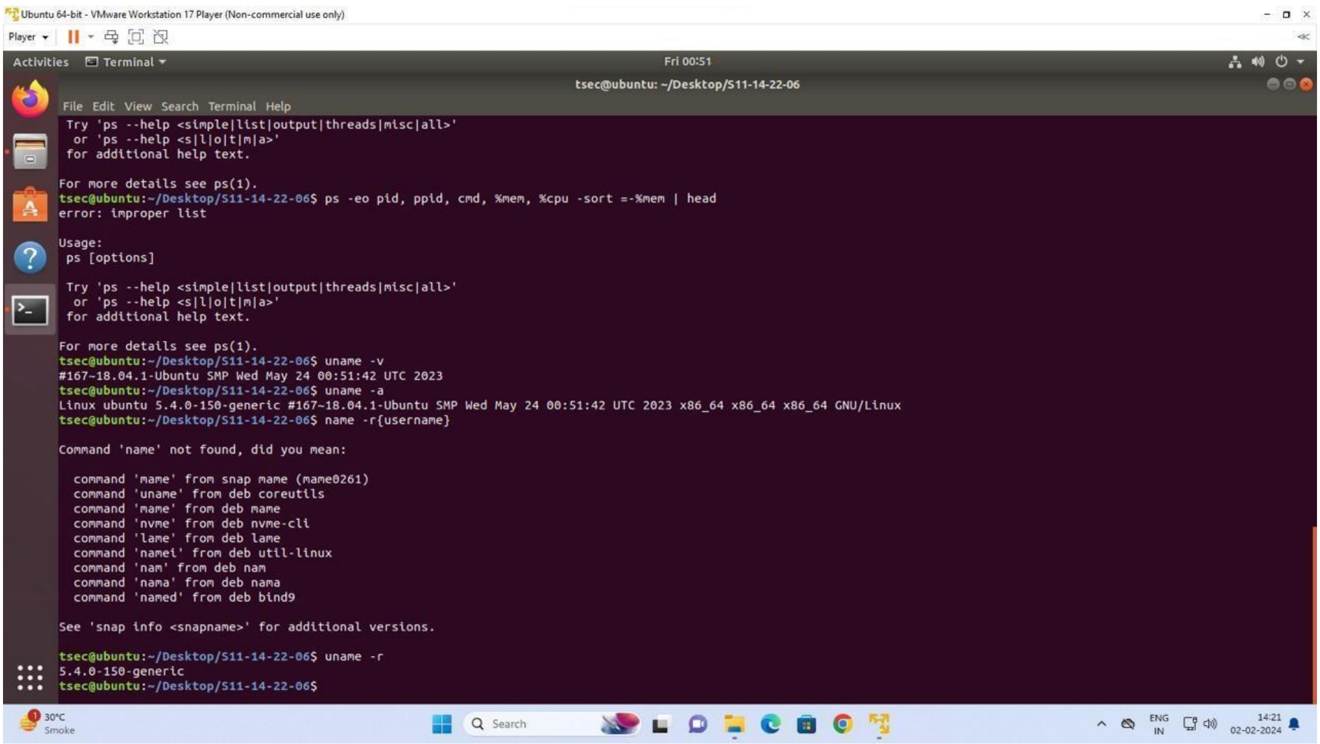


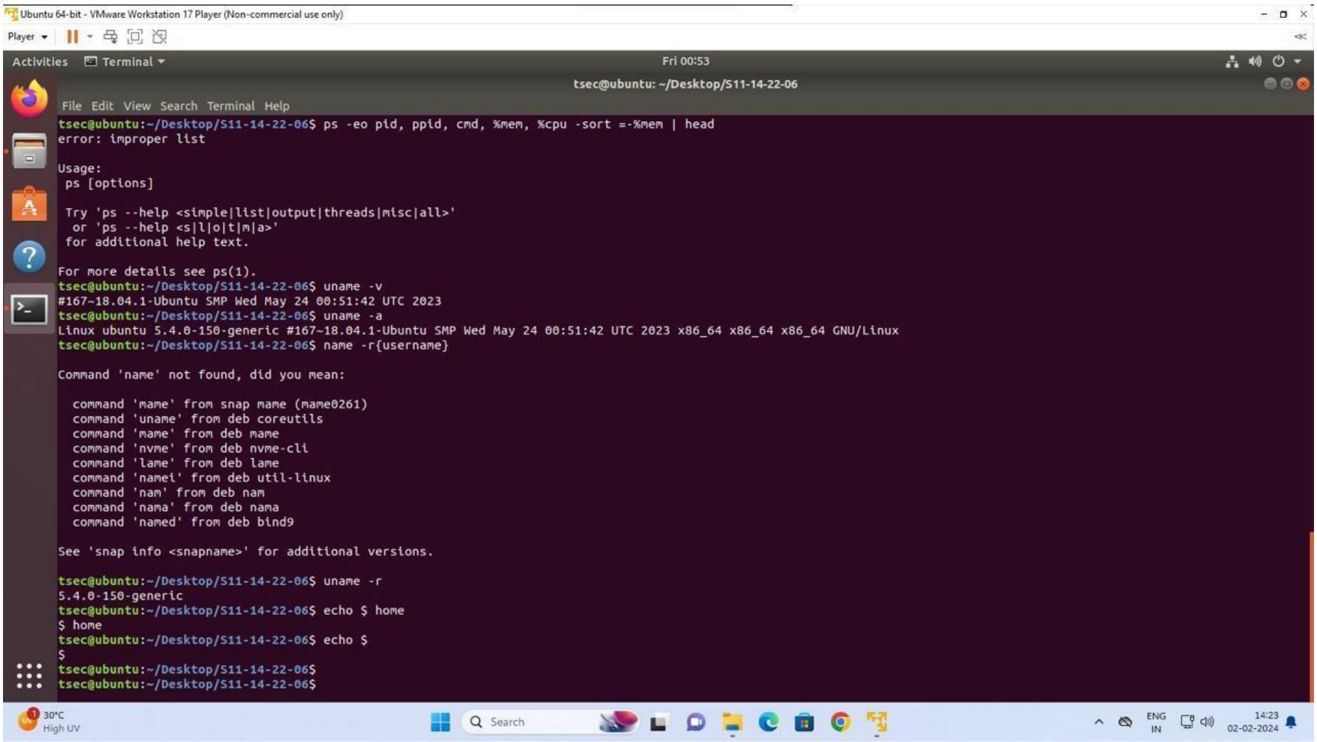
```
WritebackTmp:          0 kB
CommitLimit:     2967316 kB
Committed_AS:    4885052 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       27652 kB
VmallocChunk:          0 kB
Percpu:            53760 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
FileHugePages:         0 kB
FilePmdMapped:         0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
DirectMap4k:      259904 kB
DirectMap2M:     3934208 kB
DirectMap1G:     2097152 kB
The OS version is:
NAME="Ubuntu"
VERSION="18.04.6 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.6 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
The Kernel version is:
Linux version 5.4.0-150-generic (buildd@bos03-amd64-012) (gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023
tsec@ubuntu:~/Desktop/S11-14-22-06$ gedit EnvironmentVar.sh
```

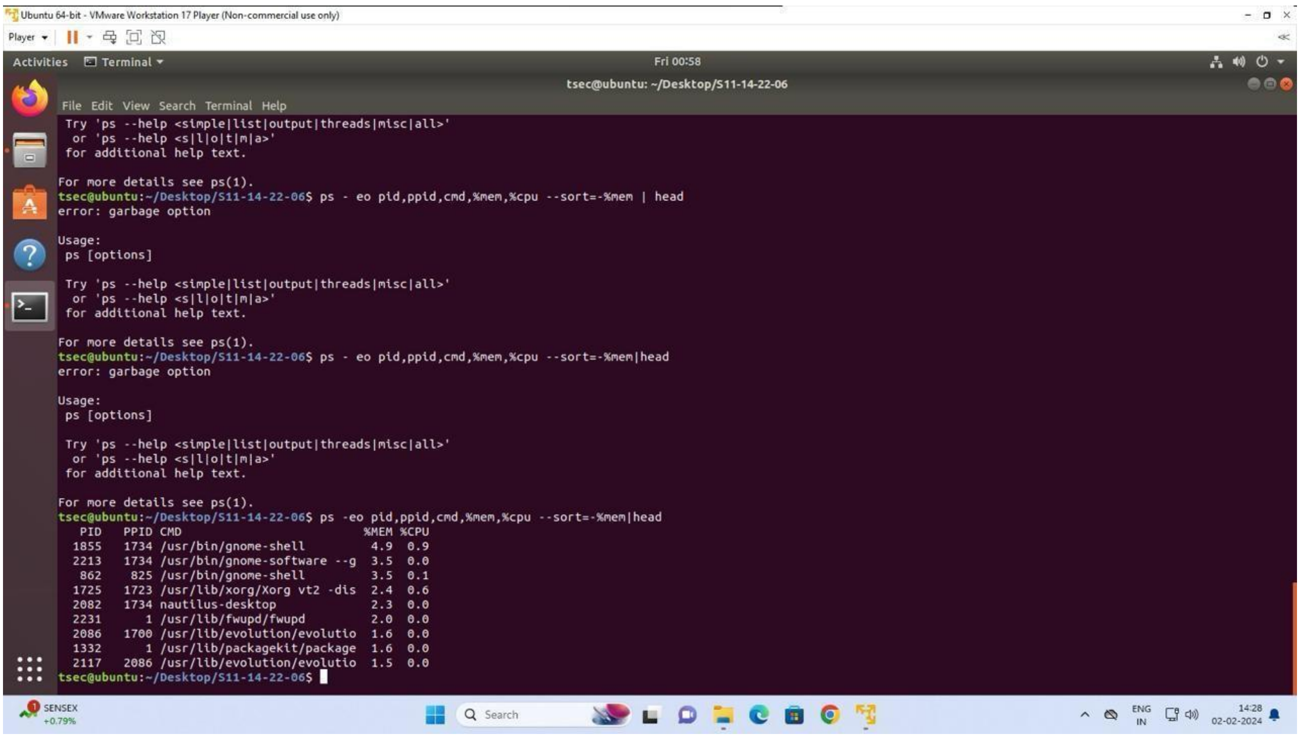

Ubuntu 64-bit - VMware Workstation 17 Player (Non-commercial use only)
Player
Activities
Terminal
Fri 00:40
tsec@ubuntu: ~/Desktop/S11-14-22-06
File Edit View Search Terminal Help
tsec@ubuntu:~/Desktop/S11-14-22-06$ ps -aux | tail
tsec 4872 0.0 0.4 555972 19272 ? Ssl 00:26 0:00 /usr/libexec/xdg-desktop-portal-gtk
root 5241 0.0 0.0 0 0 ? I 00:27 0:00 [kworker/1:6-cgr]
root 5242 0.0 0.0 0 0 ? I 00:27 0:00 [kworker/1:7-eve]
root 5295 0.0 0.0 0 0 ? I 00:33 0:00 [kworker/u256:1-]
tsec 5298 0.3 1.2 961896 49100 ? Sl 00:34 0:01 /usr/bin/nautilus --gapplication-service
root 5310 0.0 0.0 0 0 ? I 00:34 0:00 [kworker/1:0-eve]
root 5317 0.0 0.0 0 0 ? I 00:35 0:00 [kworker/1:1-cgr]
root 5318 0.0 0.0 0 0 ? I 00:35 0:00 [kworker/1:2-cgr]
tsec 5349 0.0 0.0 39672 3704 pts/1 R+ 00:38 0:00 ps -aux
tsec 5350 0.0 0.0 7516 856 pts/1 S+ 00:38 0:00 tail
tsec@ubuntu:~/Desktop/S11-14-22-06$ sudo apt install htop
[sudo] password for tsec:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
fonts-liberation2 fonts-opensymbol gir1.2-gst-plugins-base-1.0 gir1.2-gstreamer-1.0 gir1.2-gudev-1.0 gir1.2-udisks-2.0 grilo-plugins-0.3-base gstreamer1.0-gtk3
libboost-date-time1.65.1 libboost-filesystem1.65.1 libboost-iostreams1.65.1 libboost-locale1.65.1 libcdr-0.1-1 libclucene-contribs1v5 libclucene-core1v5
libcmis-0.5-5v5 libcolamd2 libdazzle-1.0-0 libe-book-0.1-1 libedataserverui-1.2-2 libeot0 libepubgen-0.1-1 libetonyek-0.1-1 libevent-2.1-6 libexiv2-14
libfreerdp-client2-2 libfreerdp2-2 libgc1c2 libgee-0.8-2 libgexiv2-2 libgom-1.0-0 libgpgmepp6 libgpod-common libgpod4 liblangtag-common liblangtag1 liblirc-client0
liblua5.3-0 libmediaart-2.0-0 libmspub-0.1-1 libodfgen-0.1-1 libqqwing2v5 libraw16 librevenge-0.0-0 libsgutils2-2 libssh-4 libsuitesparseconfig5 libvncclient1
libwinpr2-2 libxapian30 libxmlsec1-nss lp-solve media-player-info python3-mako python3-markupsafe syslinux syslinux-common syslinux-legacy usb-creator-common
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
htop
0 upgraded, 1 newly installed, 0 to remove and 49 not upgraded.
Need to get 80.0 kB of archives.
After this operation, 221 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu bionic/main amd64 htop amd64 2.1.0-3 [80.0 kB]
Fetched 80.0 kB in 1s (135 kB/s)
Selecting previously unselected package htop.
(Reading database ... 151763 files and directories currently installed.)
Preparing to unpack .../htop_2.1.0-3_amd64.deb ...
Unpacking htop (2.1.0-3) ...
Setting up htop (2.1.0-3) ...
Processing triggers for desktop-file-utils (0.23-1ubuntu3.18.04.2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for gnome-menus (3.13.3-11ubuntu1.1) ...
30°C
Smoke
Search
ENG
IN
14:10
02-02-2024


Ubuntu 64-bit - VMware Workstation 17 Player (Non-commercial use only)
Player
Activities
Terminal
Fri 00:48
tsec@ubuntu: ~/Desktop/S11-14-22-06
File Edit View Search Terminal Help
root 5295 0.0 0.0 0 0 ? I 00:33 0:00 [kworker/u256:1-]
tsec 5298 0.3 1.2 961896 49100 ? Sl 00:34 0:01 /usr/bin/nautilus --gapplication-service
root 5310 0.0 0.0 0 0 ? I 00:34 0:00 [kworker/1:0-eve]
root 5317 0.0 0.0 0 0 ? I 00:35 0:00 [kworker/1:1-cgr]
root 5318 0.0 0.0 0 0 ? I 00:35 0:00 [kworker/1:2-cgr]
tsec 5349 0.0 0.0 39672 3704 pts/1 R+ 00:38 0:00 ps -aux
tsec 5350 0.0 0.0 7516 856 pts/1 S+ 00:38 0:00 tail
tsec@ubuntu:~/Desktop/S11-14-22-06$ sudo apt install htop
[sudo] password for tsec:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
fonts-liberation2 fonts-opensymbol gir1.2-gst-plugins-base-1.0 gir1.2-gstreamer-1.0 gir1.2-gudev-1.0 gir1.2-udisks-2.0 grilo-plugins-0.3-base gstreamer1.0-gtk3
libboost-date-time1.65.1 libboost-filesystem1.65.1 libboost-iostreams1.65.1 libboost-locale1.65.1 libcdr-0.1-1 libclucene-contribs1v5 libclucene-core1v5
libcmis-0.5-5v5 libcolamd2 libdazzle-1.0-0 libe-book-0.1-1 libedataserverui-1.2-2 libeot0 libepubgen-0.1-1 libetonyek-0.1-1 libevent-2.1-6 libexiv2-14
libfreerdp-client2-2 libfreerdp2-2 libgc1c2 libgee-0.8-2 libgexiv2-2 libgom-1.0-0 libgpgmepp6 libgpod-common libgpod4 liblangtag-common liblangtag1 liblirc-client0
liblua5.3-0 libmediaart-2.0-0 libmspub-0.1-1 libodfgen-0.1-1 libqqwing2v5 libraw16 librevenge-0.0-0 libsgutils2-2 libssh-4 libsuitesparseconfig5 libvncclient1
libwinpr2-2 libxapian30 libxmlsec1-nss lp-solve media-player-info python3-mako python3-markupsafe syslinux syslinux-common syslinux-legacy usb-creator-common
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
htop
0 upgraded, 1 newly installed, 0 to remove and 49 not upgraded.
Need to get 80.0 kB of archives.
After this operation, 221 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu bionic/main amd64 htop amd64 2.1.0-3 [80.0 kB]
Fetched 80.0 kB in 1s (135 kB/s)
Selecting previously unselected package htop.
(Reading database ... 151763 files and directories currently installed.)
Preparing to unpack .../htop_2.1.0-3_amd64.deb ...
Unpacking htop (2.1.0-3) ...
Setting up htop (2.1.0-3) ...
Processing triggers for desktop-file-utils (0.23-1ubuntu3.18.04.2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for gnome-menus (3.13.3-11ubuntu1.1) ...
Processing triggers for mime-support (3.60ubuntu1) ...
tsec@ubuntu:~/Desktop/S11-14-22-06$ ps -p$$
PID TTY TIME CMD
2381 pts/1 00:00:00 bash
30°C
Smoke
Search
ENG
IN
14:18
02-02-2024


Ubuntu 64-bit - VMware Workstation 17 Player (Non-commercial use only)
Player
Activities
Terminal
Fri 00:50
tsec@ubuntu: ~/Desktop/S11-14-22-06
File Edit View Search Terminal Help
ps [options]
Try 'ps --help <simple|list|output|threads|misc|all>'
or 'ps --help <s|l|o|t|m|a>'
for additional help text.
For more details see ps(1).
tsec@ubuntu:~/Desktop/S11-14-22-06$ ps -eo pid, ppid, cmd, %mem, %cpu -sort =-%mem | head
error: improper list
Usage:
ps [options]
Try 'ps --help <simple|list|output|threads|misc|all>'
or 'ps --help <s|l|o|t|m|a>'
for additional help text.
For more details see ps(1).
tsec@ubuntu:~/Desktop/S11-14-22-06$ uname -v
#167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023
tsec@ubuntu:~/Desktop/S11-14-22-06$ uname -a
Linux ubuntu 5.4.0-150-generic #167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
tsec@ubuntu:~/Desktop/S11-14-22-06$ name -r{username}
Command 'name' not found, did you mean:
command 'mame' from snap mame (mame0261)
command 'uname' from deb coreutils
command 'mame' from deb mame
command 'nvme' from deb nvme-cli
command 'lame' from deb lame
command 'namei' from deb util-linux
command 'nam' from deb nam
command 'nama' from deb nama
command 'named' from deb bind9
See 'snap info <snapname>' for additional versions.
tsec@ubuntu:~/Desktop/S11-14-22-06$
DOW
+0.97%
Search
ENG
IN
14:20
02-02-2024


Ubuntu 64-bit - VMware Workstation 17 Player (Non-commercial use only)
Player
Activities
Terminal
Fri 00:51
tsec@ubuntu: ~/Desktop/S11-14-22-06
File Edit View Search Terminal Help
Try 'ps --help <simple|list|output|threads|misc|all>'
or 'ps --help <s|l|o|t|m|a>'
for additional help text.
For more details see ps(1).
tsec@ubuntu:~/Desktop/S11-14-22-06$ ps -eo pid, ppid, cmd, %mem, %cpu -sort =-%mem | head
error: improper list
Usage:
ps [options]
Try 'ps --help <simple|list|output|threads|misc|all>'
or 'ps --help <s|l|o|t|m|a>'
for additional help text.
For more details see ps(1).
tsec@ubuntu:~/Desktop/S11-14-22-06$ uname -v
#167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023
tsec@ubuntu:~/Desktop/S11-14-22-06$ uname -a
Linux ubuntu 5.4.0-150-generic #167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
tsec@ubuntu:~/Desktop/S11-14-22-06$ name -r{username}
Command 'name' not found, did you mean:
command 'mame' from snap mame (mame0261)
command 'uname' from deb coreutils
command 'mame' from deb mame
command 'nvme' from deb nvme-cli
command 'lame' from deb lame
command 'namei' from deb util-linux
command 'nam' from deb nam
command 'nama' from deb nama
command 'named' from deb bind9
See 'snap info <snapname>' for additional versions.
tsec@ubuntu:~/Desktop/S11-14-22-06$ uname -r
5.4.0-150-generic
tsec@ubuntu:~/Desktop/S11-14-22-06$
30°C
Smoke
Search
ENG
IN
14:21
02-02-2024


Ubuntu 64-bit - VMware Workstation 17 Player (Non-commercial use only)
Activities Terminal Fri 00:53
tsec@ubuntu: ~/Desktop/S11-14-22-06
File Edit View Search Terminal Help
tsec@ubuntu:~/Desktop/S11-14-22-06$ ps -eo pid, ppid, cmd, %mem, %cpu -sort =-%mem | head
error: improper list
Usage:
ps [options]
Try 'ps --help <simple|list|output|threads|misc|all>'
or 'ps --help <s|l|o|t|m|a>'
for additional help text.
For more details see ps(1).
tsec@ubuntu:~/Desktop/S11-14-22-06$ uname -v
#167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023
tsec@ubuntu:~/Desktop/S11-14-22-06$ uname -a
Linux ubuntu 5.4.0-150-generic #167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
tsec@ubuntu:~/Desktop/S11-14-22-06$ name -r{username}
Command 'name' not found, did you mean:
command 'mame' from snap mame (mame0261)
command 'uname' from deb coreutils
command 'mame' from deb mame
command 'nvme' from deb nvme-cli
command 'lame' from deb lame
command 'namei' from deb util-linux
command 'nam' from deb nam
command 'nama' from deb nama
command 'named' from deb bind9
See 'snap info <snapname>' for additional versions.
tsec@ubuntu:~/Desktop/S11-14-22-06$ uname -r
5.4.0-150-generic
tsec@ubuntu:~/Desktop/S11-14-22-06$ echo $ home
$ home
tsec@ubuntu:~/Desktop/S11-14-22-06$ echo $
$
tsec@ubuntu:~/Desktop/S11-14-22-06$
tsec@ubuntu:~/Desktop/S11-14-22-06$
30°C High UV
Search
ENG IN 14:23 02-02-2024


Ubuntu 64-bit - VMware Workstation 17 Player (Non-commercial use only)
Activities Terminal Fri 00:58
tsec@ubuntu: ~/Desktop/S11-14-22-06
File Edit View Search Terminal Help
Try 'ps --help <simple|list|output|threads|misc|all>'
or 'ps --help <s|l|o|t|m|a>'
for additional help text.
For more details see ps(1).
tsec@ubuntu:~/Desktop/S11-14-22-06$ ps - eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head
error: garbage option
Usage:
ps [options]
Try 'ps --help <simple|list|output|threads|misc|all>'
or 'ps --help <s|l|o|t|m|a>'
for additional help text.
For more details see ps(1).
tsec@ubuntu:~/Desktop/S11-14-22-06$ ps - eo pid,ppid,cmd,%mem,%cpu --sort=-%mem|head
error: garbage option
Usage:
ps [options]
Try 'ps --help <simple|list|output|threads|misc|all>'
or 'ps --help <s|l|o|t|m|a>'
for additional help text.
For more details see ps(1).
tsec@ubuntu:~/Desktop/S11-14-22-06$ ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem|head
PID PPID CMD %MEM %CPU
1855 1734 /usr/bin/gnome-shell 4.9 0.9
2213 1734 /usr/bin/gnome-software --g 3.5 0.0
862 825 /usr/bin/gnome-shell 3.5 0.1
1725 1723 /usr/lib/xorg/Xorg vt2 -dis 2.4 0.6
2082 1734 nautilus-desktop 2.3 0.0
2231 1 /usr/lib/fwupd/fwupd 2.0 0.0
2086 1700 /usr/lib/evolution/evolutio 1.6 0.0
1332 1 /usr/lib/packagekit/package 1.6 0.0
2117 2086 /usr/lib/evolution/evolutio 1.5 0.0
tsec@ubuntu:~/Desktop/S11-14-22-06$
SENSEX +0.79%
Search
ENG IN 14:28 02-02-2024

Conclusion: Thus, we have successfully studied and implemented shell scripting languages

Name: Meet Brijwani
Roll No: 14
Batch: S11

<u>Experiment No 3</u> : Linux- API

Aim: Implement Basic Commands of Linux like ls, cp, mv using Kernel APIs.

Theory:

1) cp → cp - r → copies the file cp - backups → It copies the backups of the destination file in the source
folder.

cp - i → It asks for confirmation of the user whether yes or no and gives warning to the user before overriding the destination file.

cp-ie.txt b.txt

cp -f → Unable to open destination file for writing because the user has
not allowed the writing operation .By writing the command, the destination file is deleted and the content is copied from source to destination.

cp - v → it is used for which command is running in backup.

cp - p → It preserves the attributes of files such as last date modification
time, time of last access owners and the file permissions.

Syntax : cp-p source file destination file.

cp - l → To create a hand link file

2) mv → (move or rename). mv.i → Ask the user for confirmation before moving the file.

mv - f → It provides the protection and overrides destination file forcefully and deletes the source file.

mv - f a.txt b.txt mv - n → It takes/prevents the existing file from

being overridden.

mv - n a.txt b.txt mv - b → To take the backup of

an existing file

Output:

```
File  Edit  View  Search  Terminal  Help
tsec10@ubuntu:~$ uname -v
#167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023
tsec10@ubuntu:~$ uname -r
5.4.0-150-generic
tsec10@ubuntu:~$ uname -a
Linux ubuntu 5.4.0-150-generic #167~18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux
tsec10@ubuntu:~$ ps -aux|tail
tsec10     2214  1.0  3.6 1267372 144828 ?      SLl  19:32   0:02 /usr/bin/gnome-software --gapplication-service
root       2223  0.1  2.0 623192 80568 ?        Ssl  19:32   0:00 /usr/lib/fwupd/fwupd
tsec10     2264  0.3  1.2 825444 51668 ?        Sl   19:32   0:00 /usr/bin/nautilus --gapplication-service
root       2282  0.0  0.0      0     0 ?        I    19:32   0:00 [kworker/0:4-cgr]
root       2283  0.0  0.0      0     0 ?        I    19:32   0:00 [kworker/0:5]
tsec10     2293  0.2  0.9 791688 36292 ?        Ssl  19:33   0:00 /usr/lib/gnome-terminal/gnome-terminal-server
tsec10     2347  0.0  0.1  22452  4800 pts/0    Ss   19:33   0:00 bash
tsec10     2359  0.0  0.6 586872 25208 tty2     Sl+  19:33   0:00 update-notifier
tsec10     2420  0.0  0.0  39672  3712 pts/0    R+   19:35   0:00 ps -aux
tsec10     2421  0.0  0.0   7516   824 pts/0    S+   19:35   0:00 tail
tsec10@ubuntu:~$ ps -aux|tail -n15
tsec10     2160  0.0  0.1 187908  5128 ?        Sl   19:32   0:00 /usr/lib/dconf/dconf-service
tsec10     2168  0.1  1.5 1129008 62600 ?       Sl   19:32   0:00 /usr/lib/evolution/evolution-calendar-factory-subprocess --factory all --bus-name org.gnome.evolution.dataserver.Subprocess.Backend.Calend
arx2137x2 --own-path /org/gnome/evolution/dataserver/Subprocess/Backend/Calendar/2137/2
tsec10     2178  0.0  0.6 725716 24588 ?        Ssl  19:32   0:00 /usr/lib/evolution/evolution-addressbook-factory
tsec10     2193  0.0  0.6 1075640 26204 ?       Sl   19:32   0:00 /usr/lib/evolution/evolution-addressbook-factory-subprocess --factory all --bus-name org.gnome.evolution.dataserver.Subprocess.Backend.Add
ressBookx2178x2 --own-path /org/gnome/evolution/dataserver/Subprocess/Backend/AddressBook/2178/2
tsec10     2208  0.0  0.1 197368  5628 ?        Ssl  19:32   0:00 /usr/lib/gvfs/gvfsd-metadata
tsec10     2214  0.9  3.6 1267372 144828 ?      SLl  19:32   0:02 /usr/bin/gnome-software --gapplication-service
root       2223  0.1  2.0 623192 80568 ?        Ssl  19:32   0:00 /usr/lib/fwupd/fwupd
tsec10     2264  0.3  1.2 825444 51668 ?        Sl   19:32   0:00 /usr/bin/nautilus --gapplication-service
root       2282  0.0  0.0      0     0 ?        I    19:32   0:00 [kworker/0:4-cgr]
root       2283  0.0  0.0      0     0 ?        I    19:32   0:00 [kworker/0:5]
tsec10     2293  0.2  0.9 791688 36344 ?        Ssl  19:33   0:00 /usr/lib/gnome-terminal/gnome-terminal-server
tsec10     2347  0.0  0.1  22452  4804 pts/0    Ss   19:33   0:00 bash
tsec10     2359  0.0  0.6 586872 25208 tty2     Sl+  19:33   0:00 update-notifier
tsec10     2422  0.0  0.0  39672  3700 pts/0    R+   19:36   0:00 ps -aux
tsec10     2423  0.0  0.0   7516   828 pts/0    S+   19:36   0:00 tail -n15
tsec10@ubuntu:~$ ps -p$$
   PID TTY          TIME CMD
  2347 pts/0    00:00:00 bash
tsec10@ubuntu:~$  sudo apt install htop
[sudo] password for tsec10:
1
Sorry, try again.
[sudo] password for tsec10:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  fonts-liberation2 fonts-opensymbol gir1.2-gst-plugins-base-1.0 gir1.2-gstreamer-1.0 gir1.2-gudev-1.0 gir1.2-udisks-2.0
  grilo-plugins-0.3-base gstreamer1.0-gtk3 libboost-date-time1.65.1 libboost-filesystem1.65.1 libboost-iostreams1.65.1
  libboost-locale1.65.1 libcdr-0.1-1 libclucene-contribs1v5 libclucene-core1v5 libcmis-0.5-5v5 libcolamd2 libdazzle-1.0-0
```

```
tsec10@ubuntu:~$  sudo apt install htop
[sudo] password for tsec10:
1
Sorry, try again.
[sudo] password for tsec10:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  fonts-liberation2 fonts-opensymbol gir1.2-gst-plugins-base-1.0 gir1.2-gstreamer-1.0 gir1.2-gudev-1.0 gir1.2-udisks-2.0
  grilo-plugins-0.3-base gstreamer1.0-gtk3 libboost-date-time1.65.1 libboost-filesystem1.65.1 libboost-iostreams1.65.1
  libboost-locale1.65.1 libcdr-0.1-1 libclucene-contribs1v5 libclucene-core1v5 libcmis-0.5-5v5 libcolamd2 libdazzle-1.0-0
  libe-book-0.1-1 libedataserverui-1.2-2 libeot0 libepubgen-0.1-1 libetonyek-0.1-1 libevent-2.1-6 libexiv2-14
  libfreerdp-client2-2 libfreerdp2-2 libgc1c2 libgee-0.8-2 libgexiv2-2 libgom-1.0-0 libgpgmepp6 libgpod-common libgpod4
  liblangtag-common liblangtag1 liblirc-client0 liblua5.3-0 libmediaart-2.0-0 libmspub-0.1-1 libodfgen-0.1-1 libqqwing2v5
  libraw16 librevenge-0.0-0 libsgutils2-2 libssh-4 libsuitesparseconfig5 libvncclient1 libwinpr2-2 libxapian30 libxmlsec1-nss
  linux-hwe-5.4-headers-5.4.0-137 linux-hwe-5.4-headers-5.4.0-84 lp-solve media-player-info python3-mako python3-markupsafe
  syslinux syslinux-common syslinux-legacy usb-creator-common
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  htop
0 upgraded, 1 newly installed, 0 to remove and 53 not upgraded.
Need to get 80.0 kB of archives.
After this operation, 221 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu bionic/main amd64 htop amd64 2.1.0-3 [80.0 kB]
Fetched 80.0 kB in 1s (84.8 kB/s)
Selecting previously unselected package htop.
(Reading database ... 186880 files and directories currently installed.)
Preparing to unpack .../htop_2.1.0-3_amd64.deb ...
Unpacking htop (2.1.0-3) ...
Setting up htop (2.1.0-3) ...
Processing triggers for desktop-file-utils (0.23-1ubuntu3.18.04.2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for gnome-menus (3.13.3-11ubuntu1.1) ...
Processing triggers for mime-support (3.60ubuntu1) ...
tsec10@ubuntu:~$ ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem|head
   PID  PPID CMD                         %MEM %CPU
  1901  1778 /usr/bin/gnome-shell         4.6  1.5
  2214  1749 /usr/bin/gnome-software --g  3.6  0.3
  1021  1015 /usr/bin/gnome-shell         3.5  0.2
  2106  1778 nautilus-desktop             2.7  0.1
  2223     1 /usr/lib/fwupd/fwupd         2.0  0.0
  1769  1767 /usr/lib/xorg/Xorg vt2 -dis  1.9  0.6
  2137  1749 /usr/lib/evolution/evolutio  1.6  0.0
  1281     1 /usr/lib/packagekit/package  1.6  0.5
  2168  2137 /usr/lib/evolution/evolutio  1.5  0.0
tsec10@ubuntu:~$
tsec10@ubuntu:~$
```

copy.c

```c
#include<stdlib.h>
int main()
{
FILE *fptr1, *fptr2;
char filename [100], c;
printf("Enter the filename to open for reading \n");
scanf("%s", filename); // Open one file for reading

fptr1 = fopen(filename, "r");

if(fptr1 ==NULL)
{printf("Cannot open file %s in", filename); exit(0);}
printf("Enter the filename to open for writing \n");
scanf("%s", filename);
// Open another file for writing

fptr2= fopen(filename, "w"); if (fptr2 == NULL)
{printf("Cannot open file %s\n", filename); exit(0);} // Read contents from file
c = fgetc(fptr1); while (c != EOF) {fputc(c, fptr2);
c = fgetc(fptr1);}printf("\nContents copied to %s", filename);
fclose(fptr1); fclose(fptr2); return 0;
}
```

```
tsec@ubuntu:~$ gcc -o copy.out copy.c
tsec@ubuntu:~$ ./copy.out
Enter the filename to open for reading
example.txt
Enter the filename to open for writing
example1.txt
Contents copied to example1.txt
```

```
tsec@ubuntu:~$ cat example1.txt
Hello!!!
```

Conclusion: Thus we have successfully studied and implemented various basic commands of Linux like ls, cp and mv using kernel APIs

Name: Meet Brijwani
Roll no: 14
Batch: S11

Experiment 4: Linux - Process

Aim:

a. To create a child process in Linux using the fork system call.From the child process obtain the process ID of both child and parent by using getpid and getpid system call.

b. Explore wait and waitpid before termination of process.

Theory:

1] System call

When a program is in user mode and requires access to Ram or hardware resources, it must ask the kernel to provide access to that resource. This is done via something called a system call. When a program makes a system call, the mode is switched from user mode to Kernel mode. This is called context switch. The kernel provides the resources which the program requested. System calls are made by user level programmes in following cases:

Creating, opening, closing and delete files in the system
Creating and managing new processes
Creating a connection in the network, sending and receiving packets
Requesting access to a hardware device like a mouse or a printer

2] Fork ( )

The fork system call is used to create processes. When a process makes a fork().call, an exact copy of the process is created.

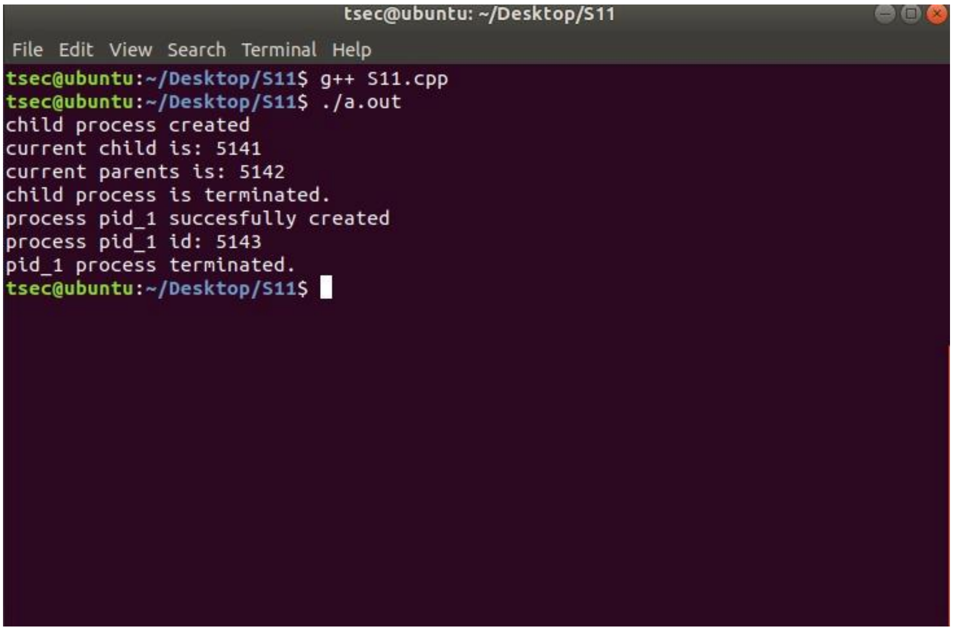
There are now two processes, one being the parent process and the other being the child process. The process which called fork()call is the parent process and the process which is created newly is called child process. The child process will be exactly the same as the parent. The process state of the parent i.e the address space, variables, open files etc is copied into the child process. The change of values in the parent process doesn't affect the child and vice versa.

Code:

```
#include <iostream>
#include<sys/wait.h>
#include<unistd.h>

using namespace std;
int wait_func()
{
  int pid_1 = fork();
if(pid_1==0)
  {     cout<<"process pid_1 succesfully
created"<<"\n";     cout<<"process pid_1 id: "<<
getpid()<<"\n";     exit(0);   }
  waitpid(pid_1, NULL,0);
  cout << "pid_1 process terminated."<<"\n";
return 0;
}
int main()
{
  int pid=fork();
if(pid==0)
  {
   cout<<"child process created"<<"\n";
cout<<"current child is: "<<getppid()<<"\n";
cout<<"current parents is: "<<getpid()<<"\n";
exit(0);   }   wait(NULL);   cout<<"child process
is terminated."<<"\n";
  wait_func();
  return 0;
}
```

Output:



Conclusion:

Thus, we have successfully studied and implemented how to create child process in Linux using fork() system calls.

Name: Meet Brijwani

Roll no: 14

Batch: S-11

Aim : : a) To study and implement non preemptive scheduling algorithm FCFS.

b) To study and implement preemptive scheduling algorithm SRTF.

Theory :

Non preemptive algorithm :

FCFS, SJF 1] FCFS : First Come

First Serve

It is a non-preemptive scheduling algorithm and the criteria for this is the arrival time of the

process CPU is allotted to the process that requires it first. Jobs arriving later are placed at the end of the queue.

FCFS (Example)

Process	Duration	Oder	Arrival Time
P1	24	1	0
P2	3	2	0
P3	4	3	0

Gantt Chart :

P1 waiting time : 0
P2 waiting time : 24
P3 waiting time : 27

The Average waiting time :

(0+24+27)/3 = 17

2] SJF : shortest job first

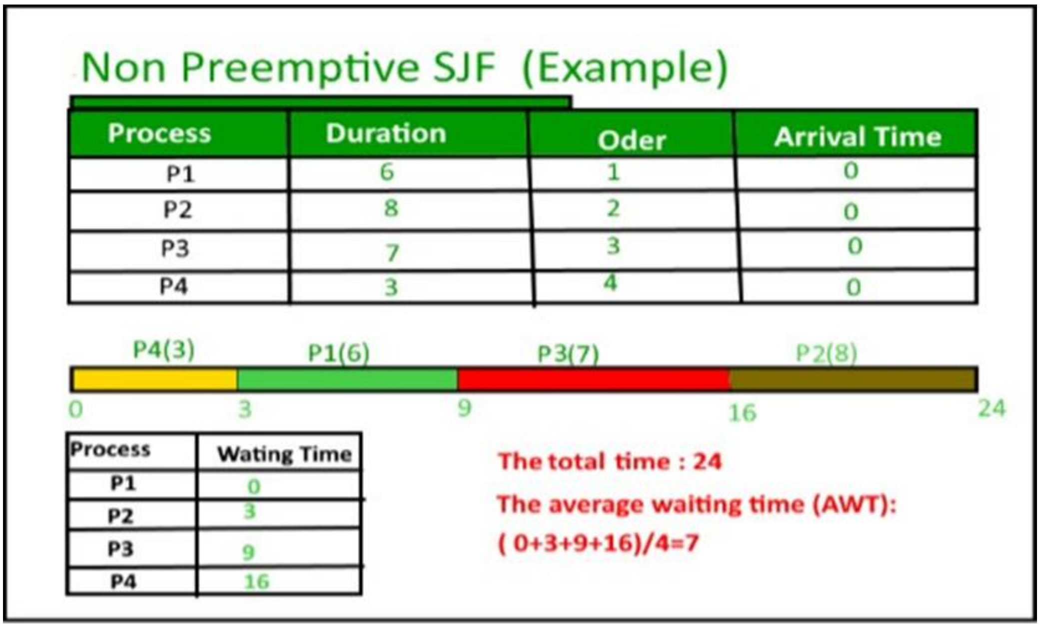
This is a non preemptive scheduling algorithm, which associates with each process the length of

the processes next CPU

first. Criteria : Burst Time.

This algorithm assigns processes according to BT if BT of two processes is the same we use

FCFS scheduling criteria.



3] Priority Scheduling

This is a non preemptive scheduling algorithm. Each process here has a priority that is either

assigned already or externally done.

Process	Burst Time	Priority
P1	10	2
P2	5	0
P3	8	1

P1	P3	P2

0 10 18 23

Preemptive Scheduling Algorithm : SRTF/STRN

Shortest Remaining Time First / Shortest Remaining Time Next scheduling.

It is a preemptive SJF Algorithm. The choice arrives when a new process arrives as the ready

queue. While a previous process is still executing. The next CPU burst if the newly arrived

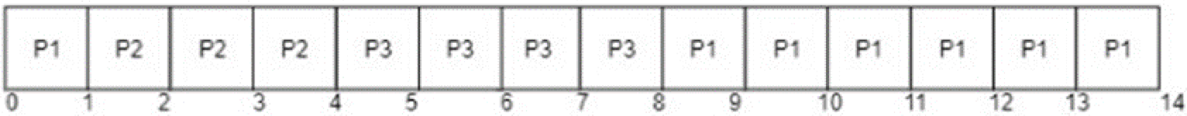
process may be shorter than what is left of the currently executing process. A preemptive SJF

will preempt the current executing process and not allow the currently running process to finish its CPU burst.

Round Robin is also one preemptive algorithm.

Process	Burst Time	Arrival Time
P1	7	0
P2	3	1
P3	4	3

The Gantt Chart for SRTF will be:



Implementation with code FCFS:

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

int main() {
    int n;
    printf("Enter no. of processes : ");
    scanf("%d", &n);

    int *at = (int *)malloc(n * sizeof(int));
    int *bt = (int *)malloc(n * sizeof(int));
    int *ct = (int *)malloc(n * sizeof(int));
    int *wt = (int *)malloc(n * sizeof(int));
    int *tat = (int *)malloc(n * sizeof(int));

    printf("Enter arrival and burst times of the processes:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d%d", &at[i], &bt[i]);
    }
```

```
    int total_tat = 0;
    int t = at[0];
    for (int i = 0; i < n; i++) {
        t += bt[i];
        ct[i] = t;
        tat[i] = ct[i] - at[i];
        total_tat += tat[i];
        wt[i] = ct[i] - bt[i];
    }

    printf("    AT    BT    CT    TAT    WT\n");
    for (int i = 0; i < n; i++) {
        printf("%6d%6d%6d%7d%6d\n", at[i], bt[i], ct[i], tat[i], wt[i]);
    }

    printf("Average TAT = %.2f\n", (float)total_tat / n);

    free(at);
    free(bt);
    free(ct);
    free(wt);
    free(tat);

    return 0;
}
```

Output :

```
Enter no. of processes : 3
Enter arrival and burst times of the processes:
0 5
1 3
2 8
    AT    BT    CT    TAT    WT
     0     5     5      5     0
     1     3     8      7     5
     2     8    16     14     8
Average TAT = 8.67
```

Implementation with code SRTF:

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

struct Process {
    int pid; // Process ID
    int bt; // Burst Time
    int art; // Arrival Time
};

void findWaitingTime(struct Process proc[], int n, int wt[]) {
    int rt[n];
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;

    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    int check = 0;
```

```
    while (complete != n) {
        for (int j = 0; j < n; j++) {
            if ((proc[j].art <= t) && (rt[j] < minm) && rt[j] > 0) {
                minm = rt[j];
                shortest = j;
                check = 1;
            }
        }

        if (check == 0) {
            t++;
            continue;
        }

        rt[shortest]--;
        minm = rt[shortest];
        if (minm == 0)
            minm = INT_MAX;

        if (rt[shortest] == 0) {
            complete++;
            check = 0;
            finish_time = t + 1;
            wt[shortest] = finish_time - proc[shortest].bt - proc[shortest].art;
            if (wt[shortest] < 0)
                wt[shortest] = 0;
        }
        t++;
    }
}

void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}

void findavgTime(struct Process proc[], int n) {
```

```
    int *wt = (int *)malloc(n * sizeof(int));
    int *tat = (int *)malloc(n * sizeof(int));
    int total_wt = 0, total_tat = 0;

    findWaitingTime(proc, n, wt);
    findTurnAroundTime(proc, n, wt, tat);

    printf(" P\t\tBT\t\tWT\t\tTAT\t\t\n");
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
        printf(" %d\t\t%d\t\t%d\t\t%d\n", proc[i].pid, proc[i].bt, wt[i], tat[i]);
    }

    printf("\nAverage waiting time = %.2f\n", (float)total_wt / (float)n);
    printf("Average turn around time = %.2f\n", (float)total_tat / (float)n);

    free(wt);
    free(tat);
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process *proc = (struct Process *)malloc(n * sizeof(struct
Process));
    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d %d", &proc[i].art, &proc[i].bt);
        proc[i].pid = i + 1;
    }

    findavgTime(proc, n);
```

```
    free(proc);

    return 0;
}
```

Output:

```
Enter the number of processes: 5
Enter arrival time and burst time for each process:
Process 1: 2 6
Process 2: 5 2
Process 3: 1 8
Process 4: 0 3
Process 5: 4 4
 P              BT              WT              TAT
 1              6               7               13
 2              2               0               2
 3              8               14              22
 4              3               0               3
 5              4               2               6

Average waiting time = 4.60
Average turn around time = 9.20
```

Conclusion : Thus we have successfully implemented preemptive and non preemptive scheduling algorithms.

Name: Meet Brijwani
Roll no: 14
Batch: S11

EXP 6:

Aim : Write a C program to implement solution of Producer consumer problem through Semaphore.

Theory :

In concurrent programming, concurrency represents a pivotal concept necessary to comprehend fully how such systems operate. Among the various challenges encountered by practitioners working with these systems stands out the producer-consumer problem - one of the most renowned synchronization issues. In this text, our objective consists of analyzing this topic and highlighting its significance for concurrent computing while also examining possible solutions rooted within C.

Introduction

In concurrent systems, multiple threads or processes may access shared resources simultaneously. The producer-consumer problem involves two entities: producers that generate data or tasks, and consumers that process or consume the generated data. The challenge lies in ensuring that producers and consumers synchronize their activities to avoid issues like race conditions or resource conflicts.

Understanding the Producer-Consumer Problem

Problem Statement

One possible definition of the producer-consumer problem involves two main groups: producers of data who store their work in a communal space called the buffer, and processors (consumers) of that content saved in said space. These people use their expertise around gathered items within this temporary holding scenario to analyze it comprehensively before delivering insightful results.

Synchronization Requirements

Achieving resolution of the producer-consumer conundrum will necessarily involve implementing techniques for synchronized collaboration among all stakeholders involved. The integration of optimal synchronization protocols is fundamental in avoiding scenarios where device buffers are either overloaded by producing units or depleted by consuming ones.

Implementing the Producer-Consumer Problem in C

Shared Buffer

In C, a shared buffer can be implemented using an array or a queue data structure. The buffer should have a fixed size and support operations like adding data (producer) and retrieving data (consumer).

Synchronization Techniques

Several synchronization techniques can be used to solve the producer-consumer problem in C, including −

- Mutex and Condition Variables − Mutexes provide mutual exclusion to protect critical sections of code, while condition variables allow threads to wait for specific conditions to be met before proceeding.
- Semaphores − Semaphores can be used to control access to the shared buffer by tracking the number of empty and full slots.
- Monitors − Monitors provide a higher-level abstraction for synchronization and encapsulate shared data and the operations that can be performed on it.

Solutions to the Producer-Consumer Problem in C

Bounded Buffer Solution

One common solution to the producer-consumer problem is the bounded buffer solution. It involves using a fixed-size buffer with synchronization mechanisms to ensure that producers and consumers cooperate correctly. The capacity for item production is limited by the buffer size making it crucial to factor in this specification so as not to exceed the available space in the buffer.

Producer and Consumer Threads

In C, the producer and consumer activities can be implemented as separate threads. Each producer thread generates data and adds it to the shared buffer, while each consumer thread retrieves data from the buffer and processes it. Synchronization mechanisms are used to coordinate the activities of the threads.

Handling Edge Cases

In real-world scenarios, additional considerations may be necessary. For example, if the producers generate data at a faster rate than the consumers can process, buffering mechanisms like blocking or dropping data may be required to prevent data loss or deadlock situations.

Code :

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


// Initialize a mutex to 1 and Number of full slots as 0 and Number of
empty slots as size of buffer

int mutex = 1;

int full = 0;

int empty = 10, x = 0;



// Function to produce an item and add it to the buffer
void producer()


{

    // Decrease mutex value by 1 and Increase the number of full slots
```

```
by 1 and Decrease the number of empty slots by 1

    --mutex;

    ++full;

    --empty;
```

```
// Item produced

    x++;

    printf("\nProducer produces item %d",x);

// Increase mutex value by 1

    ++mutex;

}


// Function to consume an item and remove it from buffer

void consumer()

{

     // Decrease mutex value by 1 and Decrease the number of full slots
by 1

    --mutex;

    --full;

 // Increase the number of empty slots by 1

    ++empty;

    printf("\nConsumer consumes item %d",x);

    x--;

// Increase mutex value by 1

    ++mutex;

}


int main()
```

```
{

    int n, i; printf("\n1. Press 1 for Producer \n2. Press 2 for
    Consumer \n3.

Press 3 for Exit");



    for (i = 1; i > 0; i++) {
```

```
        printf("\nEnter your choice:");

        scanf("%d", &n);

        switch (n) {

        case 1:


// If mutex is 1 and empty is non-zero, then it is possible to produce

            if ((mutex == 1)

                && (empty != 0)) {

                producer();

            }

            else { printf("Buffer is
                full!");


            }

            break;


        case 2:

 // If mutex is 1 and full is non-zero, then it is possible to consume

            if ((mutex == 1) && (full != 0)) {

                consumer();

            }
```

```
            else {
                printf("Buffer is empty!");
            }
            break;
        case 3:
            exit(0);
            break;
        }
    }
}
```

Output :

Output

Clear

```
/tmp/QgcMuoqYbY.o

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:2
Buffer is empty!
Enter your choice:1

Producer produces item 1
Enter your choice:1

Producer produces item 2
Enter your choice:1

Producer produces item 3
Enter your choice:1

Producer produces item 4
Enter your choice:1

Producer produces item 5
Enter your choice:1

Producer produces item 6
Enter your choice:1
```

```
Producer produces item 7
Enter your choice:1

Producer produces item 8
Enter your choice:1

Producer produces item 9
Enter your choice:1

Producer produces item 10
Enter your choice:1
Buffer is full!
Enter your choice:2

Consumer consumes item 10
Enter your choice:2

Consumer consumes item 9
Enter your choice:2

Consumer consumes item 8
Enter your choice:2

Consumer consumes item 7
Enter your choice:2
```

```
Consumer consumes item 6
Enter your choice:2

Consumer consumes item 5
Enter your choice:2

Consumer consumes item 4
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!
Enter your choice:3
```

Conclusion : Thus we have successfully implemented Producer Consumer problem using Semaphore.

Name: Meet Brijwani

Roll no: 14

Batch: S11

EXP 7:

Aim : Process Management: Deadlock

a. Write a program to demonstrate the concept of deadlock avoidance through Banker’s Algorithm

b. Write a program demonstrate the concept of Dining Philospher’s Problem

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

Available

It is a 1-d array of size ‘m’ indicating the number of available resources of each type.

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

Max

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

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

Allocation

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

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

Need

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

Need [i, j] = k means process Pi currently needs ‘k’ instances of resource type Rj Need [i,
j] = Max [i, j] – Allocation [i, j]

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

Banker’s Algorithm

1. Active:= Running U Blocked;

for k=1...r

New_ request[k]:= Requested_ resources[requesting_ process, k];

2. Simulated_ allocation:= Allocated_ resources; for k=1.....r

//Compute projected allocation state

Simulated_ allocation [requesting _process, k]:= Simulated_ allocation [requesting _process, k] + New_ request[k]; 3. feasible:= true; for

k=1....r // Check whether projected allocation state is feasible if Total_

resources[k]< Simulated_ total_ alloc [k] then feasible:= false;

4. if feasible= true
then // Check whether projected allocation state is a safe allocation state

while set Active contains a process P1 such that

For all k, Total _resources[k] – Simulated_ total_ alloc[k]>= Max_ need [l

,k]-Simulated_
allocation[l, k]

Delete Pl from Active;

for k=1.....r
Simulated_ total_ alloc[k]:= Simulated_ total_ alloc[k]- Simulated_ allocation[l, k];

5. If set Active is empty then // Projected allocation state is a safe

   allocation state for k=1....r // Delete the request from pending

   requests Requested_ resources[requesting_ process, k]:=0; for

   k=1....r // Grant the request

Allocated_ resources[requesting_ process, k]:= Allocated_ resources[requesting_ process, k] + New_ request[k];

Total_ alloc[k]:= Total_ alloc[k] + New_ request[k];

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

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

Resource-Request Algorithm

Let Requesti be the request array for process Pi. Requesti [j] = k means process Pi wants k

instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:

1) If Requesti <= Needi

Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If Requesti <= Available

Goto step (3); otherwise, Pi must wait, since the resources are not available. 3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

Available = Available – Requesti

Allocationi = Allocationi + Requesti

Needi = Needi– Requesti

Code :

```
// Banker's Algorithm
#include <iostream>

using namespace std;
```

```
int main()

{

int n, m, i, j, k;

n = 5;

m = 3;
```

```
int alloc[5][3] = { { 0, 1, 0 },

{ 2, 0, 0 },

{ 3, 0, 2 },

{ 2, 1, 1 },

{ 0, 0, 2 } };

int max[5][3] = { { 7, 5, 3 },

{ 3, 2, 2 },

{ 9, 0, 2 },

{ 2, 2, 2 },

{ 4, 3, 3 } };

int avail[3] = { 3, 3, 2 };

int f[n], ans[n], ind = 0;

for (k = 0; k < n; k++) {
f[k] = 0;
```

```
}

int need[n][m];

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

```
for (j = 0; j < m; j++)

need[i][j] = max[i][j] - alloc[i][j];

}

int y = 0;

for (k = 0; k < 5; k++) {

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

if (f[i] == 0) {


int flag = 0;

for (j = 0; j < m; j++) {

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}


if (flag == 0) {
ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;
```

```
}

}

}

}



int flag = 1;



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

{



if(f[i]==0)
{


flag = 0;

cout << "The given sequence is not safe";

break;

}

}





if(flag==1)
{

cout << "Following is the SAFE Sequence" << endl;

for (i = 0; i < n - 1; i++)

cout << " P" << ans[i] << " ->";

cout << " P" << ans[n - 1] <<endl;

} return
(0);


}
```

Output :

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

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2


=== Code Execution Successful ===
```

Code :

```c
#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define NUM_PHILOSOPHERS 5

#define NUM_CHOPSTICKS 5



void dine(int n);

pthread_t philosopher[NUM_PHILOSOPHERS];

pthread_mutex_t chopstick[NUM_CHOPSTICKS];


int main()
```

```
{

int i, status_message;

void *msg;

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

{

status_message = pthread_mutex_init(&chopstick[i], NULL);

if (status_message == -1)

{

printf("\n Mutex initialization failed");
exit(1);

}

}

for (i = 1; i <= NUM_PHILOSOPHERS; i++)
```

```
{

status_message = pthread_create(&philosopher[i], NULL, (void *)dine,
(int *)i);
```

```c
if (status_message != 0)

{

printf("\n Thread creation error \n");

exit(1);

}

}



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

{

status_message = pthread_join(philosopher[i], &msg);



if (status_message != 0)

{

printf("\n Thread join failed \n");

exit(1);
}


}



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

{

status_message = pthread_mutex_destroy(&chopstick[i]);

if (status_message != 0)

{

printf("\n Mutex Destroyed \n");

exit(1);

}
```

```
}

return 0;

}

void dine(int n)
{

printf("\nPhilosopher % d is thinking ", n);



pthread_mutex_lock(&chopstick[n]);



pthread_mutex_lock(&chopstick[(n + 1) % NUM_CHOPSTICKS]);



printf("\nPhilosopher % d is eating ", n);


sleep(3);



pthread_mutex_unlock(&chopstick[n]);



pthread_mutex_unlock(&chopstick[(n + 1) % NUM_CHOPSTICKS]);
```

```
printf("\nPhilosopher % d Finished eating ", n);

}
```

Output :

```
Philosopher  2 is thinking
Philosopher  2 is eating
Philosopher  3 is thinking
Philosopher  5 is thinking
Philosopher  5 is eating
Philosopher  1 is thinking
Philosopher  4 is thinking
Philosopher  4 is eating
Philosopher  2 Finished eating
Philosopher  5 Finished eating
Philosopher  1 is eating
Philosopher  4 Finished eating
Philosopher  3 is eating
Philosopher  1 Finished eating
Philosopher  3 Finished eating
```

Conclusion : Thus we have successfully implemented Banker’s Algorithm and the concept of Dining Philospher’s Problem using C++.

Name: Meet Brijwani

Roll no: 14

Batch: S11

EXP 8:

Aim: Write a program to demonstrate the concept of dynamic partitioning placement algorithms i.e. Best Fit, First Fit, Worst Fit.

Theory:

In the operating system, the following are four common memory management techniques.

Single contiguous allocation: Simplest allocation method used by MS-DOS. All memory (except some reserved for OS) is available to a process.

Partitioned allocation: Memory is divided into different blocks or partitions. Each process is allocated according to the requirement.

Paged memory management: Memory is divided into fixed-sized units called page frames, used in a virtual memory environment.

Segmented memory management: Memory is divided into different segments (a segment is a logical grouping of the process' data or code).In this management, allocated memory doesn't have to be contiguous.

Most of the operating systems (for example Windows and Linux) use Segmentation with Paging. A process is divided into segments and individual segments have pages.

In Partition Allocation, when there is more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.

When it is time to load a process into the main memory and if there is more than one free block of memory of sufficient size then the OS decides which free block to allocate.

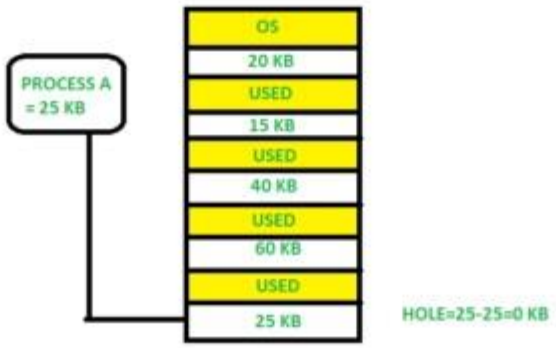
There are different Placement Algorithm:

A. First Fit

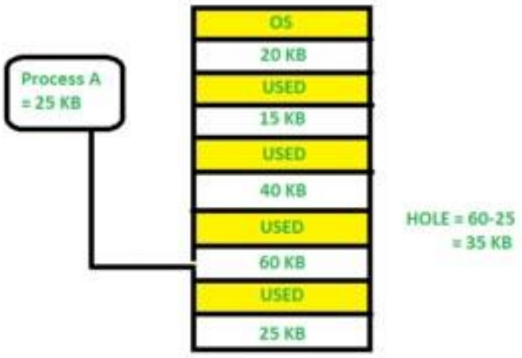
B. Best Fit
C. Worst Fit
D. Next Fit

1. First Fit: In the first fit, the partition is allocated which is the first sufficient block from the top of Main Memory. It scans memory from the beginning and chooses the first available block that is large enough. Thus it allocates the first hole that is large enough.

2. Best Fit Allocate the process to the partition which is the first smallest sufficient partition among the free available partition. It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.



3. Worst Fit Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory. It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to process.



4. Next Fit: Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

Is Best-Fit really best?

Although best fit minimises the wastage space, it consumes a lot of processor time for searching the block which is close to the required size. Also, Best-fit may perform poorer than other algorithms in some cases. For example, see the exercise below.

Sl.No.	Partition Allocation Method	Advantages	Disadvantages
1.	Fixed Partition	Simple, easy to use, no complex algorithms needed	Memory waste, inefficient use of memory resources
2.	Dynamic Partition	Flexible, more efficient, partitions allocated as required	Requires complex algorithms for memory allocation
3.	Best-fit Allocation	Minimizes memory waste, allocates smallest suitable partition	More computational overhead to find smallest split
4.	Worst-fit Allocation	Ensures larger processes have sufficient memory	May result in substantial memory waste

5.	First-fit Allocation	Quick, efficient, less computational work	Risk of memory fragmentation

Comparison of Partition Allocation Methods:Exercise: Consider the requests from processes in given order 300K, 25K, 125K, and 50K. Let there be two blocks of memory available of size 150K followed by a block size 350K.

Which of the following partition allocation schemes can satisfy the above requests?

A) Best fit but not first fit.

B) First fit but not best fit.

C) Both First fit & Best fit.

D) neither first fit nor best fit.

Solution: Let us try all options.

Best Fit:

300K is allocated from a block of size 350K. 50 is left in the block.

25K is allocated from the remaining 50K block. 25K is left in the block.

125K is allocated from 150 K block. 25K is left in this block also.

50K can’t be allocated even if there is 25K + 25K space available.

First Fit:

300K request is allocated from 350K block, 50K is left out.

25K is allocated from the 150K block, 125K is left out.

Then 125K and 50K are allocated to the remaining left out partitions.

So, the first fit can handle requests.

So option B is the correct choice.

Program:

a)First Fit:

```
#include<stdio.h>

// Function to allocate memory to
// blocks as per First fit algorithm
void firstFit(int blockSize[], int m, int processSize[], int n)
{ int i, j;
```

```
// Stores block id of the //
block allocated to a process
int allocation[n];

// Initially no block is assigned to any process
for(i = 0; i < n; i++)
{
   allocation[i] = -1;
}

// pick each process and find suitable blocks
// according to its size ad assign to it
for (i = 0; i < n; i++) //here, n -> number of processes
{
    for (j = 0; j < m; j++) //here, m -> number of blocks
   {
      if (blockSize[j] >= processSize[i])
      {
         // allocating block j to the ith process
         allocation[i] = j;

         // Reduce available memory in this block.
         blockSize[j] -= processSize[i];

         break; //go to the next process in the queue
      }
   }
}

printf("\nProcess No.\tProcess Size\tBlock no.\n");
for (int i = 0; i < n; i++)
{ printf(" %i\t\t\t", i+1);
   printf("%i\t\t\t\t",
   processSize[i]);
   if (allocation[i] != -1)
      printf("%i", allocation[i] + 1);
```

```
        else
           printf("Not Allocated");
        printf("\n");
    }
}

// Driver code
int main()
{
    int m; //number of blocks in the memory int n;
    //number of processes in the input queue int
    blockSize[] = {100, 500, 200, 300, 600}; int
    processSize[] = {212, 417, 112, 426}; m =
    sizeof(blockSize) / sizeof(blockSize[0]); n =
    sizeof(processSize) / sizeof(processSize[0]);
    firstFit(blockSize, m, processSize, n);

    return 0 ;
}
```

b)Best Fit:

```
#include<stdio.h>

void main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
static int barray[20],parray[20]; printf("\n\t\t\tMemory
Management Scheme - Best Fit"); printf("\nEnter the
number of blocks:"); scanf("%d",&nb); printf("Enter the
number of processes:"); scanf("%d",&np);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
    {
printf("Block no.%d:",i);
       scanf("%d",&b[i]);
```

```
    }
printf("\nEnter the size of the processes :-\n");
for(i=1;i<=np;i++)
    {
        printf("Process no.%d:",i);
        scanf("%d",&p[i]);
    }
for(i=1;i<=np;i++)
{
for(j=1;j<=nb;j++)
{
if(barray[j]!=1)
{ temp=b[j]-p[i];
if(temp>=0)
if(lowest>temp)
{ parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
barray[parray[i]]=1;
lowest=10000;
}
printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
for(i=1;i<=np && parray[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i])
;
}
```

c) Worst Fit:

```
#include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int processSize[], int
processes)
{
    // This will store the block id of the allocated block to a process
    int allocation[processes]; int occupied[blocks];
```

```
// initially assigning -1 to all allocation
indexes // means nothing is allocated
currently for(int i = 0; i < processes; i++){
allocation[i] = -1;
}

for(int i = 0; i < blocks; i++){
   occupied[i] = 0;
}

// pick each process and find suitable
blocks // according to its size ad assign to it
for (int i=0; i < processes; i++)
{
int indexPlaced = -1;
for(int j = 0; j < blocks;
j++)
{
   // if not occupied and block size is large enough
   if(blockSize[j] >= processSize[i] && !occupied[j])
      {
         // place it at the first block fit to accomodate process
         if (indexPlaced == -1)
            indexPlaced = j;

         // if any future block is larger than the current block where //
         process is placed, change the block and thus indexPlaced
         else if (blockSize[indexPlaced] < blockSize[j])
            indexPlaced = j;
      }
   }

   // If we were successfully able to find block for the process
   if (indexPlaced != -1)
   {
```

```
}

            // allocate this block j to process p[i]
            allocation[i] = indexPlaced;

            // make the status of the block as occupied
            occupied[indexPlaced] = 1;

            // Reduce available memory for the block
            blockSize[indexPlaced] -= processSize[i];
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < processes; i++)
    {
        printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n",allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

// Driver code
int main()
{
    int blockSize[] = {100, 50, 30, 120, 35}; int processSize[] =
    {40, 10, 30, 60}; int blocks =
    sizeof(blockSize)/sizeof(blockSize[0]); int processes =
    sizeof(processSize)/sizeof(processSize[0]);
    implimentWorstFit(blockSize, blocks, processSize,
    processes);

    return 0;
}
```

Output:

a)First Fit:

```
PS C:\Meet\COLLEGE\LAB ASS\Aoa> cd "c:\Meet\COLLEGE\LAB ASS\Aoa\" ; if ($?) { gcc firstfit.c -o firstfit } ; if ($?) { .\firstfit

Process No.     Process Size    Block no.
 1                      212                     2
 2                      417                     5
 3                      112                     2
 4                      426                     Not Allocated
PS C:\Meet\COLLEGE\LAB ASS\Aoa>
```

b)Best Fit:

```
PS C:\Meet\COLLEGE\LAB ASS\Aoa> cd "c:\Meet\COLLEGE\LAB ASS\Aoa\" ; if ($?) { gcc bestfit.c -o bestfit } ; if ($?) { .\bestfit }

                        Memory Management Scheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:3

Enter the size of the blocks:-
Block no.1:10
Block no.2:20
Block no.3:30
Block no.4:30
Block no.5:40

Enter the size of the processes :-
Process no.1:7
Process no.2:9
Process no.3:2

Process_no      Process_size    Block_no        Block_size      Fragment
1               7               1               10              3
2               9               2               20              11
3               2               3               30              28
PS C:\Meet\COLLEGE\LAB ASS\Aoa>
```

c)Worst Fit:

```
PS C:\Meet\COLLEGE\LAB ASS\Aoa> cd "c:\Meet\COLLEGE\LAB ASS\Aoa\" ; if ($?) { gcc worstfit.c -o worstfit } ; if ($?) { .\worstfit }

Process No.     Process Size    Block no.
1                       40              4
2                       10              1
3                       30              2
4                       60              Not Allocated
PS C:\Meet\COLLEGE\LAB ASS\Aoa>
```

Conclusion: Thus we have successfully implemented dynamic partitioning placement algorithms.

Experiment no: 09.

Name: Meet Brijwani
Roll no: 14
Batch: S11

Aim: Write a program in C demonstrate the concept of page replacement policies for handling page faults eg: FIFO, LRU

Theory:

Page replacement policies are an integral part of virtual memory management in operating systems. When a program accesses data or instructions, the operating system loads the corresponding pages into physical memory (RAM) from secondary storage (usually a hard disk). However, physical memory has limited capacity, so not all pages can reside in memory simultaneously. When a program requests a page that is not in memory, a page fault occurs, triggering the need for a page replacement policy to determine which page to evict from memory to make room for the new one. Two commonly used page replacement policies are First-In-First-Out (FIFO) and Least Recently Used (LRU). Let's delve into these policies and understand their mechanisms and trade-offs.

First-In-First-Out (FIFO):
FIFO is one of the simplest page replacement algorithms. It evicts the oldest page in memory, based on the assumption that the page that has been in memory the longest is least likely to be needed in the near future.

Mechanism:
When a page fault occurs and memory is full, the operating system selects the page that entered memory earliest (the oldest page) for replacement.
The selected page is evicted from memory, and the new page is loaded in its place.

The page table is updated accordingly.
Advantages:
Simplicity: FIFO is easy to implement and understand.
Low Overhead: The overhead of maintaining data structures is minimal.
Disadvantages:
Belady's Anomaly: FIFO can suffer from Belady's anomaly, where increasing the number of frames can actually increase the number of page faults.
Poor Performance: FIFO does not consider the access history of pages, leading to suboptimal performance in many cases, especially when the access patterns are irregular.
Least Recently Used (LRU):
LRU is based on the principle that the page that has not been accessed for the longest time is least likely to be used in the near future.

Mechanism:
LRU keeps track of the time of the last access for each page.
When a page fault occurs, the operating system selects the page that was least recently accessed for replacement.
The selected page is evicted from memory, and the new page is loaded in its place.
The page table is updated accordingly, and the access time of the new page is recorded.
Advantages:
Optimality: LRU provides better performance than FIFO in terms of reducing the number of page faults in many scenarios.
Flexibility: LRU can adapt to varying access patterns by considering the access history of pages.
Disadvantages:
Implementation Complexity: Implementing an efficient LRU algorithm requires maintaining a data structure to track the access times of pages, which can be resource-intensive.
High Overhead: The overhead of maintaining access times for each page can be significant, especially in systems with a large number of pages.
Comparison and Trade-offs:

Optimality: LRU is generally considered more optimal than FIFO because it takes into account the actual access history of pages rather than just the order of arrival. However, implementing a true LRU algorithm can be complex and resource-intensive.
Overhead: FIFO has lower overhead compared to LRU since it does not require tracking access times for each page. However, this simplicity comes at the cost of potentially poorer performance.
Belady's Anomaly: FIFO can suffer from Belady's anomaly, where increasing the number of frames can paradoxically increase the number of page faults. LRU does not suffer from this anomaly.
Adaptability: LRU is more adaptable to varying access patterns since it considers the actual access history of pages. FIFO, on the other hand, may perform poorly in scenarios with irregular access patterns.

Program:

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

#define MAX_FRAMES 3 // Maximum number of page frames
#define MAX_PAGES 10 // Maximum number of pages in reference
string
// Function prototypes void fifo(int pages[], int n, int frames); void lru(int
pages[], int n, int frames);

int main() {     int pages[MAX_PAGES];
    int n, frames;

    printf("Enter the number of pages: ");     scanf("%d", &n);

    printf("Enter the page reference string: ");     for (int i = 0; i < n; i++) {
scanf("%d", &pages[i]);
    }

    printf("Enter the number of frames: ");     scanf("%d", &frames);

    printf("\nFIFO Page Replacement:\n");     fifo(pages, n, frames);
```

```
    printf("\nLRU Page Replacement:\n");     lru(pages, n, frames);
    return 0;
}

void fifo(int pages[], int n, int frames) {     int frame[MAX_FRAMES];
    int page_faults = 0;     int current_frame = 0;

    for (int i = 0; i < frames; i++) {         frame[i] = -1; // Initialize frames as
empty
    }

    for (int i = 0; i < n; i++) {         int page = pages[i];         int found = 0;

        // Check if page is already in a frame         for (int j = 0; j < frames;
j++) {             if (frame[j] == page) {                 found = 1;                 break;
            }
        }

       // Page fault: replace the oldest page
        if (!found) {             printf("Page %d caused a page fault.\n", page);
frame[current_frame] = page;             current_frame = (current_frame +
1) % frames; // Circular queue
            page_faults++;
        }

        // Display current state of frames         printf("Frames: ");         for
(int j = 0; j < frames; j++) {             printf("%d ", frame[j]);
        }         printf("\n");
    }

    printf("Total page faults: %d\n", page_faults);
}

void lru(int pages[], int n, int frames) {     int frame[MAX_FRAMES];
    int page_faults = 0;     int counter[MAX_FRAMES] = {0};

    for (int i = 0; i < frames; i++) {
```

```
        frame[i] = -1; // Initialize frames as empty
    }

    for (int i = 0; i < n; i++) {        int page = pages[i];        int found = 0;

        // Check if page is already in a frame        for (int j = 0; j < frames;
j++) {            if (frame[j] == page) {                found = 1;
counter[j] = i + 1; // Update counter to indicate recent access
                break;
            }
        }

        // Page fault: replace the least recently used page        if (!found) {
printf("Page %d caused a page fault.\n", page);            int min_counter =
counter[0];            int min_index = 0;            for (int j = 1; j < frames; j++)
{                if (counter[j] < min_counter) {                    min_counter =
counter[j];
                    min_index = j;
                }
            }
            frame[min_index] = page;            counter[min_index] = i + 1; //
Update counter to indicate recent access            page_faults++;
        }

        // Display current state of frames        printf("Frames: ");        for
(int j = 0; j < frames; j++) {            printf("%d ", frame[j]);
        }        printf("\n");
    }

    printf("Total page faults: %d\n", page_faults);
}
```

Output:

```
Enter the number of pages: 10
Enter the page reference string: 1 2 3 4 1 2 5 1 2 3
Enter the number of frames: 3

FIFO Page Replacement:
Page 1 caused a page fault.
Frames: 1 -1 -1
Page 2 caused a page fault.
Frames: 1 2 -1
Page 3 caused a page fault.
Frames: 1 2 3
Page 4 caused a page fault.
Frames: 4 2 3
Page 1 caused a page fault.
Frames: 4 1 3
Page 2 caused a page fault.
Frames: 4 1 2
Page 5 caused a page fault.
Frames: 5 1 2
Frames: 5 1 2
Frames: 5 1 2
Page 3 caused a page fault.
Frames: 5 3 2
Total page faults: 8
```

```
LRU Page Replacement:
Page 1 caused a page fault.
Frames: 1 -1 -1
Page 2 caused a page fault.
Frames: 1 2 -1
Page 3 caused a page fault.
Frames: 1 2 3
Page 4 caused a page fault.
Frames: 4 2 3
Page 1 caused a page fault.
Frames: 4 1 3
Page 2 caused a page fault.
Frames: 4 1 2
Page 5 caused a page fault.
Frames: 5 1 2
Frames: 5 1 2
Frames: 5 1 2
Page 3 caused a page fault.
Frames: 3 1 2
Total page faults: 8
```

Conclusion:

In summary, page replacement policies such as FIFO and LRU play a crucial role in virtual memory management by determining which pages to evict from memory when page faults occur. While FIFO is simple to implement and has low overhead, it may perform poorly in scenarios with irregular access patterns and can suffer from Belady's anomaly. LRU, on the other hand, provides better performance by considering the actual access history of pages, but it comes with higher implementation complexity and overhead. The choice between FIFO and LRU depends on factors such as the system's requirements, workload characteristics, and available resources.

Experiment no: 10.

Name: Meet Brijwani
Roll no: 14
Batch: S11

Aim: Write a program in C to do disk scheduling - FCFS, SCAN, C-SCAN

Theory:

File management and I/O (Input/Output) management are crucial components of operating systems responsible for handling the storage and retrieval of data from disk storage devices efficiently. Disk scheduling algorithms play a vital role in optimizing I/O operations by determining the order in which disk requests are serviced. Among the various disk scheduling algorithms, First-Come, First-Served (FCFS), SCAN, and C-SCAN are widely used. Let's delve into these concepts and understand how each algorithm works and its implications in disk management.

File Management:
File management involves organizing and managing files on disk storage devices to facilitate efficient storage, retrieval, and manipulation of data. It encompasses various operations such as file creation, deletion, access control, and directory management. A file system is responsible for managing these operations and maintaining the structure and integrity of files and directories.

I/O Management:
I/O management deals with managing Input/Output operations between the CPU, memory, and I/O devices such as disks, printers, and network interfaces. Disk I/O operations are particularly significant as they involve relatively slow mechanical devices compared to the CPU and memory. Disk scheduling algorithms are employed to optimize the order in which

disk requests are serviced to minimize seek time and maximize disk throughput.

Disk Scheduling Algorithms:
1. First-Come, First-Served (FCFS):
FCFS is the simplest disk scheduling algorithm, where disk requests are serviced in the order they arrive. It operates on the principle of fairness, as requests are processed based on their arrival times without any consideration for their locations on the disk.

Mechanism:
When a disk request arrives, it is added to the end of the request queue.
The disk scheduler services requests in the order they are queued.
Each request is processed sequentially, starting from the innermost track to the outermost track of the disk.
Implications:
FCFS is easy to implement and ensures fairness in servicing requests.
However, it may lead to longer seek times, especially if requests are scattered across the disk, resulting in poor disk performance.
2. SCAN (Elevator) Algorithm:
SCAN, also known as the elevator algorithm, simulates the movement of an elevator moving up and down a building. It services requests in one direction until reaching the end of the disk, then reverses direction and continues servicing requests in the opposite direction.

Mechanism:
The disk head starts from one end of the disk and moves towards the other end while servicing requests along its path.
When it reaches the end of the disk, it reverses direction and starts moving towards the opposite end, servicing requests along the way.
SCAN prevents the disk head from unnecessarily traversing the entire disk by changing direction at the disk boundaries.
Implications:
SCAN reduces the average seek time by prioritizing requests closer to the current position of the disk head.
However, it may result in starvation for requests located at the extremes of the disk if there is a continuous stream of requests in one direction.

3. C-SCAN (Circular SCAN) Algorithm:
C-SCAN is an enhancement of the SCAN algorithm that overcomes the potential starvation issue by treating the disk as a circular buffer. After reaching one end of the disk, the disk head jumps to the other end without servicing requests, ensuring fairness in request servicing.

Mechanism:
Similar to SCAN, the disk head moves in one direction, servicing requests until reaching the end of the disk.
Instead of reversing direction immediately, C-SCAN jumps to the other end of the disk without servicing requests.
It then continues servicing requests in the same direction, preventing starvation for requests located at the disk boundaries.
Implications:
C-SCAN provides fairness in request servicing by preventing starvation for requests at the extremes of the disk.
However, it may result in slightly higher average seek times compared to SCAN due to the jump between disk ends.

Program:

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

#define MAX_REQUESTS 100 // Maximum number of disk requests
#define MAX_CYLINDERS 200 // Maximum number of cylinders on the
disk

// Function prototypes void fcfs(int requests[], int n, int initial_position);
void scan(int requests[], int n, int initial_position, int cylinders); void
c_scan(int requests[], int n, int initial_position, int cylinders);

int main() {     int requests[MAX_REQUESTS];
    int n, initial_position, cylinders;

    printf("Enter the number of disk requests: ");     scanf("%d", &n);
```

```
    printf("Enter the disk requests: ");     for (int i = 0; i < n; i++) {
scanf("%d", &requests[i]);
    }

    printf("Enter the initial position of the disk head: ");     scanf("%d",
&initial_position);

    printf("Enter the total number of cylinders on the disk: ");
scanf("%d", &cylinders);

    printf("\nFirst-Come, First-Served (FCFS):\n");     fcfs(requests, n,
initial_position);

    printf("\nSCAN:\n");     scan(requests, n, initial_position, cylinders);

    printf("\nC-SCAN:\n");     c_scan(requests, n, initial_position,
cylinders);

    return 0;
}

void fcfs(int requests[], int n, int initial_position) {     int total_seek_time =
0;

    printf("Sequence of servicing requests:\n");     for (int i = 0; i < n; i++) {
printf("%d ", requests[i]);         total_seek_time += abs(requests[i] -
initial_position);         initial_position = requests[i];
    }

    printf("\nTotal seek time: %d\n", total_seek_time);
}

void scan(int requests[], int n, int initial_position, int cylinders) {     int
total_seek_time = 0;     int direction = 1; // 1 for right, -1 for left

    printf("Sequence of servicing requests:\n");
```

```
    // Sort requests to service in ascending order     for (int i = 0; i < n - 1;
i++) {         for (int j = 0; j < n - i - 1; j++) {             if (requests[j] >
requests[j + 1]) {                 int temp = requests[j];                 requests[j] =
requests[j + 1];                 requests[j + 1] = temp;
            }
        }
    }

    // Find the index where the disk head should reverse direction     int
reverse_index = 0;     while (reverse_index < n &&
requests[reverse_index] < initial_position) {
        reverse_index++;
    }

    // Service requests in one direction     for (int i = reverse_index - 1; i
>= 0; i--) {         printf("%d ", requests[i]);         total_seek_time +=
abs(requests[i] - initial_position);         initial_position = requests[i];
    }

    // Service requests in the reverse direction     for (int i = reverse_index;
i < n; i++) {         printf("%d ", requests[i]);         total_seek_time +=
abs(requests[i] - initial_position);         initial_position = requests[i];
    }

    printf("\nTotal seek time: %d\n", total_seek_time);
}

void c_scan(int requests[], int n, int initial_position, int cylinders) {     int
total_seek_time = 0;

    printf("Sequence of servicing requests:\n");

    // Sort requests to service in ascending order     for (int i = 0; i < n - 1;
i++) {         for (int j = 0; j < n - i - 1; j++) {             if (requests[j] >
requests[j + 1]) {                 int temp = requests[j];                 requests[j] =
requests[j + 1];                 requests[j + 1] = temp;
            }
```

```
        }
    }

    // Service requests in one direction     for (int i = 0; i < n && requests[i]
< initial_position; i++) {         printf("%d ", requests[i]);
total_seek_time += abs(requests[i] - initial_position);         initial_position
= requests[i];
    }

    // Jump to the beginning of the disk     printf("%d ", 0);
total_seek_time += initial_position;

    // Service requests in the reverse direction     for (int i = n - 1; i >= 0
&& requests[i] >= initial_position; i--) {         printf("%d ", requests[i]);
        total_seek_time += abs(requests[i] - initial_position);
initial_position = requests[i];
    }

    printf("\nTotal seek time: %d\n", total_seek_time);
}
```

Output:

```
Enter the number of disk requests: 5
Enter the disk requests: 98 183 37 122 14
Enter the initial position of the disk head: 53
Enter the total number of cylinders on the disk: 200

First-Come, First-Served (FCFS):
Sequence of servicing requests:
98 183 37 122 14
Total seek time: 469

SCAN:
Sequence of servicing requests:
37 14 98 122 183
Total seek time: 208

C-SCAN:
Sequence of servicing requests:
14 0 183
Total seek time: 222


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

File management and I/O management are essential components of operating systems, responsible for efficient storage and retrieval of data from disk storage devices. Disk scheduling algorithms such as FCFS, SCAN, and C-SCAN play a crucial role in optimizing disk I/O operations by determining the order in which disk requests are serviced. While FCFS provides simplicity and fairness, SCAN and CSCAN aim to minimize seek times and prevent starvation for requests located at the extremes of the disk. The choice of disk scheduling algorithm depends on factors such as disk workload characteristics, system requirements, and performance considerations.