# Operating Systems

## 1. Syllabus Operating System

0. Introduction Operating System

> galvin book chapter 1 and 2

1. Linux System
    - History
    - Architecture
    - Linux kernel
        - Monolithic
        - Micro-kernel
        - Modular
2. Shell
    - Types of shells
    - BASH shell
    - Linux file system
        - Linux directories
        - Absolute vs Relative path
    - File system commands
        - Directory related commands
        - Files related commands
        - File permissions/owner
        - Shell wild-card characters
    - Regular expressions
    - Advanced topics
        - IO redirection
        - Pipe
    - VI editor
        - .vimrc
    - Shell scripts
        - Variables
        - read and echo
        - if-then-else, elif, case and test command
        - while, until and for loop
        - positional parameters
        - functions
        - arrays
        - string operations
        - Directory stack
    - Shell files
        - .bashrc
        - .profile, .bash_profile
        - .bash_logout

        - .bash_history
    - Networking commands
        - ifconfig
        - netcat
        - ssh
3. System calls
    - User vs kernel mode
    - Preemptible/Non-preemptible kernel
4. CPU Scheduling
    - Pre-emptive vs non-preemptive
    - Scheduling algorithms
5. Process
    - Introduction & life cycle
    - Linux process commands: ps, top
    - Process creation: fork(), exec(), wait()
    - Orphan & zombie processes
6. Inter-process communication
    - Signals - Sending (kill command) & handling
    - Shared memory
    - Message queue
    - Pipe and Fifo
7. Threads
    - Introduction
    - Process vs Threads
    - Thread creation
    - Thread synchronization
        - Semaphore
        - Mutex
    - Producer-consumer problem
    - Deadlock
    - User vs kernel threads
8. File & IO management
    - File & File system
    - File system architecture
    - File vs Directory internals
    - Hard vs Symbolic links
    - File IO system calls
    - Disk allocation mechanism - ext2/ext3
    - Disk scheduling algorithms
9. Memory management
    - MMU introduction
    - Virtual memory / Swap area
    - Segmentation vs Paging
    - TLB and multi-level paging
    - Page fault handling
    - Page replacement algorithms

    - Dirty bit and copy-on-write

## 2. Evaluation

- Theory : 40 marks - CCEE(5-jan-2021) * MCQ based on - linux commands, - shell scripts, - linux system call, - rest conceptual question
- Lab: 40 marks * Linux commands - 10 marks * Subjective question * Bash shell scripts - 20 marks * Test case based * System call programming - 10 marks * C programming : file upload
- Internals: 20 marks - MCQ

```
(ls, cp, mv, lpr, sort, grep, cat, tac, more, head, tail, man, whatis,
whereis, locate, find, diff,
file, rm, mkdir, rmdir, cd, pwd, ln and ln –s, gzip and gunzip, zip and
unzip, tar an its variants,
zcat, cal, bc and bc –l, banner date, time, wc, touch, echo, who, finger,
w, whoami, who am
i, alias, unalias, touch, push, pop, jobs, ps, etc
```

# syllbus

- Linux Commands,
- Vi editor,
- Shell Scripting,
- Overview of OS,
- Processes,
- Scheduling & Synchronization,
- Memory management,
- File Systems,
- Case Study with Linux System Programming: Process,
- Signals,
- Semaphores & Mutex,
- Inter – Process Communication, POSIX Threads

# day1

1. OS

- it is a interface between user and hardware
- core of OS is Core OS/Kernel :
  - it doing minimul basic functionalities, like
  - 1. Process management
  - 2. file I/O managaement
  - 3. memory management
  - 4. CPU Scheduling
  - 5. hardware abstraction
  - OS also does extra functionalities like
  - 1. User interfacing
  - 2. Networking
  - 3. Protection and Security

2. Linux

- linux is inspired by unix ,which was designed by denis richtie and in 1970
- unix is based on
  - file control subsystem (I/O management) and
  - process control sub system
- its logan is :
- **"file have spaces and process have lines "**
- i.e everything is a file , examples of special files are
  - data file, directory,pipe,socket,links(shortcuts),
  - block devices , where 1 sector = 512 bytes,

  - char devices , byte by byte

- process

- system calls :

- A system call is a way for programs to interact with the operating system.
- A computer program makes a system call when it makes a request to the operating system's kernel.
- System call provides the services of the operating system to the user programs via Application Program Interface(API)

- software interrupts

## OS Learning

1. end users

- commands

2. administrator

- installation
- shell scripts

3. programmer

- system calls

4. designer

- OS Internals

## Linux Shell

1. Shell takes input from terminal i.e from end user and runs system call i.e get them executed by kernel and shows output on terminal

- terminal is the window we give input and get output ,
- internally terminal has shell

- two types of shell

- 1. GUI Shell
  - in windows: explorer.exe
  - Linux : GNOME or KDE
- 2. CLI Shell

  - windows: cmd.exe, POWER shell

  - in linux : first was bsh then changed to bash (<u>b</u>ourne <u>a</u>gain <u>sh</u>ell)

    - csh/tcsh
    - zsh

        - zsh
        - ksh --> Korn shell(unix)

2. LINUX folder structure

-

1. in boot

- linux kernal name : vmlinuz
- linux boot loader : grub

2. in bin

- executables/commands

3. in sbin

- system commands(admin)

4. in lib

- contains libraries (.so) and device drivers(.ko)
    - where ko: kernel objects

5. usr

- contains all installed programs/softwares

6. etc

- contains hardware and software configuration files of: system,application , hardware
    - source.list
- user password stored in

> /etc/passwd

7. dev

- contains device files(char and block)

8. proc

- for monitoring /dynamic config
- kernel window

9. sys

- for device drivermanagement

10. tmp

- its temporary file system(auto lost when shutdown)

11. mnt

- mount point (to see other file system)
- optional
- can mount cp or hdd using mount command

> sudo mount /dev/sdb1 /mnt

- sbd : pendrive
- sda : pc storage
- to unmount it

> sudo umount /mnt

12. root

- directory for admin user (i.e root )
-

13. home

- contains all users data like

- sunbeam
- root

- if username is sunbeam ,then it home directory is

> /home/sunbeam

14. The table below shows the section numbers of the manual followed by the types of pages they contain.

- 1. User/Linux/ Executable programs or shell commands
- 2. System calls (functions provided by the kernel)
- 3. Library calls (functions within program libraries)
- 4. Special files (usually found in /dev)
- 5. File formats and conventions eg /etc/passwd
- 6. Games
- 7. Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
- 8. System administration commands (usually only for root)
- 9. Kernel routines [Non standard]

## Linux Commands:

1. Linux version

> uname -a

2. List commandds

```
terminal > ls -l
# size of directory
```

```
terminal > ls -l -h
```

3. command to show shell

```
 >  ls /bin/*sh
 > /bin/bash  /bin/dash  /bin/rbash  /bin/sh  /bin/static-sh
```

4. to change shell [NOT RECOMMENDED]

> chsh

- i.e change shell

5. check working shell

> echo $SHELL

6. to check boot folder content

> ls /boot

7. to get kernel/cpu info

> cat /proc/cpuinfo

8. to create directory

- path -- w.r.t current directory

> mkdir

- it does'nt start with '/'
- path with wr.t to absolute path,

> mkdir /home/sunbeam/

- start with '/' : are absolute path

9. for Relative path, Special directories are

- 1. . : current directory

  > ./a.out

- 2. .. :
- 3. :

10. list contents of current directory

> ls
> e.g : ls /boot

- list content of your home directory / long view

> ls -l ~

- where -l --- long listing (detail view)
- contains type, mode/permissions,links,user, group, size,modified timestamp,name

11. to change directory

- from absolute or relative path

> cd commands

- as not start with '/', so relative
-

12. to remove empty empty directory

> rmdir

- to delete contents in directory

> rm -r dirpath

13. cat command to create and insert file content

> cat > .txt eg:

```
cat > fruits.txt
mango
banana
# use cltr + D to come out from writing
```

- to view content of the file

> cat .txt

14. copy given directory to a destination directory

> cp -r

15. move file into given directory

> mv

- also use for rename a fle

> mv -r .txt

- In linux file/directory starting with "." is hidden
- to see hidden files

> ls -a

- to see hidden file content

> cat .fruits.txt

16. IO redirection

```
> ls -l -a
# output is shown on terminal
 # now to save output in file
> ls -l -a > out.txt
# output copied to file

> cat out.txt
# we can see output of ls

# for sorting
> sort
nitin
amit
sandeep
nitin
amit
nitin
nitin
sandeep

## INPUT redirection
# input(stdin) is taken from file and output on terminal
>  sort < fruits.txt

## OUTPUT redirection
# read output from, a file and writing in another file
> sort <  fruit.txt > result.txt

# for revrse order sorting
> sort -r <   fruit.txt

## ERROR redirection
# invalid option -x
> sort -x <   fruit.txt

# error output is shown on terminal
# output is shown on terminal (stderr)



# number of standard
#stdin = 0,
# stdout = 1,
# stderr = 3
# so to write error output
> sort -x 2> err.txt
```

17.to give one command output to other commands

- for word count

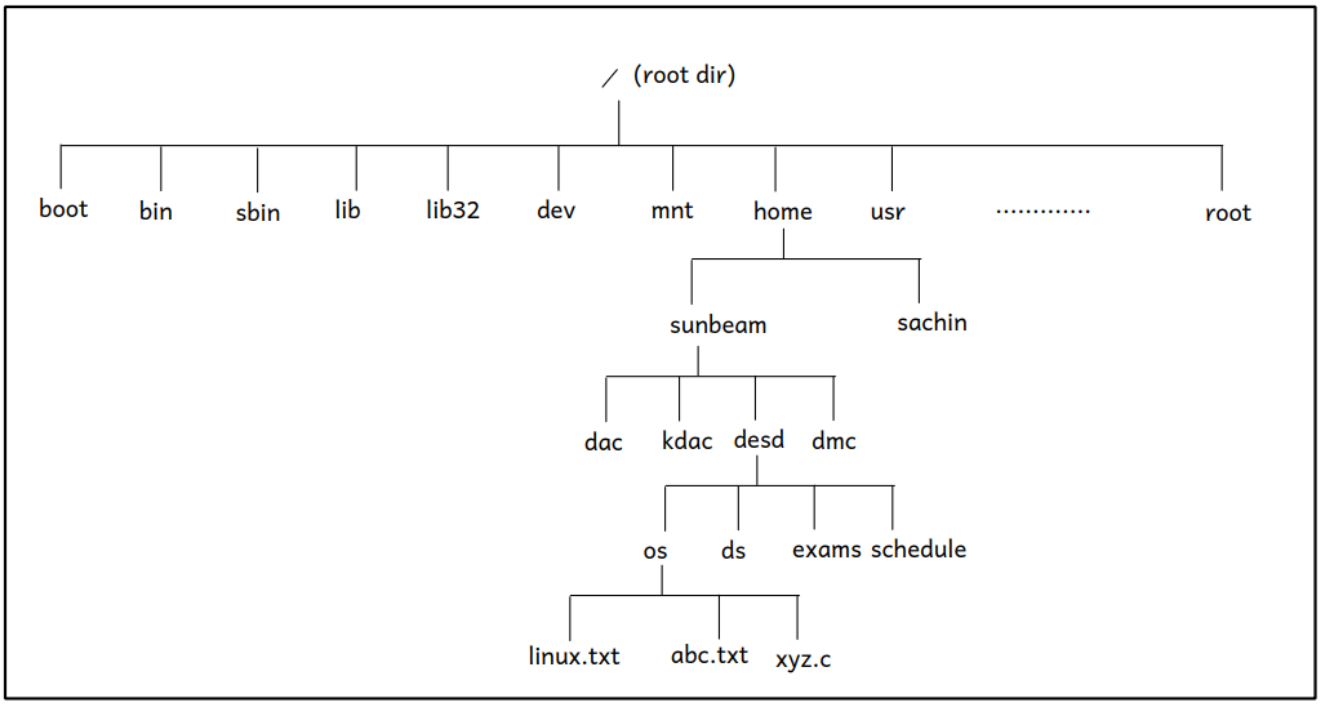
```
> wc

hello sunbeam
here
# Cltr + O
      2       3      20
#   lines    words   charcters

# to give one command output to other commands
> ls -l | wc
#left command (ls -l) output is given as input to right command(wc)
```

## Lab commands

1. LinuxFileSystemStructure our application of Tree Data structure



absolute path ==> "/home/sunbeam/desd/os/linux.txt"
relative path wr.t. sunbeam ==> "/desd/os/linux.txt"

2. Lab commands I

```
# 1.  to clear screen
> clear
# or use Cltr + L

# 2. to change dir to home directory
> cd ~
# 3. to show current working dir
> pwd

# 4. create a  directory and a sub directory,
# where -p:
>   mkdir -p  os/linux_commands

# 5.changing directory
  # - from inner directory to outer
  # use '..' + outer directory
  > cd ../../..

  # to go to root dir
  > cd /
  # to go to the previous dir
  > cd -
  # stay in current dir
  > cd .
  # go to parent dir
  > cd ..

# 6. list commands
     # list content/file names in dir
     > ls
     # list  content/file with hidden, and total data block allocated
     > ls -l -a

     # shows file/content info
     # drwxr-xr-x 3 sunbeam sunbeam 4096 Dec 11 14:51 ..
     # contians : 1)type-of-file :--> d: directory, -: for regular -
     #      2) wxr: are read and write permission
     # -l : to be displayed as a list
     > ls -l

     # here -s: show no of data block got allocated
     > ls -l -s



# 7 to make multiple  subdirectory in a dir
     > mkdir /one /two /three /four

     # make sub directories recursivly
     > mkdir -p four/five/six
     # use :  -R to display list sub-directory recursively
     > ls -R four/
```

```
# 8 . cat commands
  # to create and isnert in a file
   > cat > file1.txt
    suraj
    raj
    ram
    # use Cltr + D

   #  to show the file
   > cat file1.txt

   # now if we use same cat command , content get overwritten
   > cat > file1.txt

   #to display in reverse order
   > tac > file1.txt

   #concat multiple files
   >  cat file2.txt file3.txt

# 9 to delete a file or a directory
>  rm -r <directory>/
>  rm -r <file-name>
#
```

1. list command syntax

> ls [OPTION]... [FILE]...

- where
- ... indicate : multiple arguments allowed
- FIle: IO file , if not given , takes pwd

3. Lab Linux commands

```
# copy all text files to destination
# * : used to describe as all, if used with extension so all files of that
type
> cp ./*.txt ./../../../../
# using pipe to give output to 2nd command after |
> cat 5.txt | less
#  Sort based on ASCII value
> > sort  numbers.txt | less

# to sort number in file numerically
> sort -n  numbers.txt | less

# sorter and unique
> sort -n   numbers.txt | uniq
```

```
# head print 10 lines by default , we can set no of lines
> cat numbers.txt | head  -<number of lines>
> cat numbers.txt | head  -2

# tail to print from  last
>  cat numbers.txt | tail -<number of lines>

# can use tail and head together , pass one output as input to other
> cat numbers.txt | tail -15| head -5

#  -d : delimeter on basis on which it cut
# -f1: prints 1st field
> cat sunbeam.txt | cut -d " " -f1
> cat sunbeam.txt | cut -d " " -f2

# -i:display inode no
> ls -i -l

# we can cut any list , and give which fields , like f1,f2
> ls -i -l | cut -d " " -f1,

# path of binary
> echo $PATH

# tr : commands translates from one char to another
> echo $PATH | tr ":" "\n"

# tr : translates all small case into capital
>  echo "Suraj" | tr "a-z" "A-Z"

> echo "Suraj" | tr "a-zA-Z" "A-Za-z"
- sUraj from Suraj


# chmod use for change permission , where u: user,g: group , o: other
# where w: write, r: read,x: execute
# using u+x , u-x, we can change permission
> chmod g+w india.txt
> chmod u+rwx india.txt
> chmod o-rwx india.txt
> chmod g-rwx india.txt

# r: apply ls command
# w: we can create files and sub directory
# x: we can apply cd command on it

# there are two formats/method by which we can change mode bits of a file
i.e
# we can assign /remove access permission :
# 1. human readable : r,w,x
 used as
> chmod +rwx filepath
> chmod -rwx filepath
```

```
# 2. octal formats :
# read : 4, write : 2, execute : 1
# first digit : octal digit start with zero , so lading 0 indicate octal
contant
# second digit : access permission for user/owner
# third digit : access permission for group members
# fourth digit : access permission for other members
> chmod 0641


# create a alias for command , it is for  session
> ls -l -a -i -s
# alias command: where l = alias
> alias l="ls -l -a -i -s"
# execute alias l
> l
# command to remove alias
> unalias l
```

# day2

## Agenda

1. Advanced Linux commands
2. Shell features
3. Shell script
4. File and File system

## notes

1. **Shell wild card character** *: it is any number of any character ? : for single character

2. $? : a special shell variable , to show if your previouse commands succeeded or failed

- if $? : 0 --> success,
- !0(not zero) like 1 to 255 ---> fail

> echo $?

- i.e $? ==> exit code of previous command / program

## Linux commands part II

1. to create a file

> touch f1.txt f2.txt d.txt

- to look only .txt file

> ls *.txt

2. txt file starting with f

```
ls f*.txt
```

1. Shell wild card character

    - *: it is any number of any character
    - ? : for single character
    - for .txt with 3 character starting with f

    ```
    ls f???.txt
    ```

    - output : file.txt

    ```
    ls f?.txt
    ```

    - output f1.txt f2.txt

2. cat commands

- if no file path given , takes string as input as print output

```
cat
```

- to create insert in a file

```
cat > text.txt
```

- to append the same file

```
cat >> text.txt
```

- using combination

```
sort < text.txt > out.txt || err.txt
```

5. no of user connected

```
who
```

6. $? : a special shell variable , to show if your previouse commands succeeded or failed

- if $? : 0 --> success, !0(not zero) like 1 to 255 ---> fail

```
echo $?
```

7. test commands check a condition based on expression

```
test 12 -eq 12
```

- now we can check if it failed or success

```
echo $?
```

- where eq: equal , gt: greater than

8. for running cx program in shell

```
terminal> cat > hello.c
#include <stdio.h>
int main() {
    printf("hello world!\n");
    return 0;
}
# ctrl+D

terminal> gcc -o hello.out hello.c
terminal> echo $?
# output=0 -- success -- condition true
```

9. logical nesting of commands

- 1. &&
  - if first command successful then, run second command

  > test 12 -eq 12 && date

  - successfully run date

  > test 12 -gt 12 && date

  - failed to run date
- 2. ||
  - either 1st or 2nd command #test command have : -d : to check if it is a directory

  > test -d f1.txt && ls f1.txt

  - it only 1st command success, then run second

  > test -d fruits.txt || ls f1.txt

  - here if 1st command fails ,run second command
- 3. & (asynchronous command execution )

10. find command , is to find files , so path , by name, file name

> find ~ -name "file.txt"

11. open file from shell

> firefox

- here firefox will be opened , but shell prompt is not available
- shell is waiting for firefox to complete
- once firefox is closed, shell
- so for asyncronous use

> firefox&

12. Regular Expression

- have 3 commands type

> grep egrep

- **regular expression Wild card characters**
- 1. ^ : starts with the character
- 2. $: end with the character
  - applicable for regular expression command
- 3. [] : any single character , search is given specific range , called scan set
- 4. . : search for any single character
- 5. / : to remove special character, or use fgrep
- 6. [^...] : inverse of scan set , i.e outside this range ,any single character accepted
-

```
> cat > food.txt
this
biscuit
isnot
tasty,
but
that
cake
is
really good.
# ctrl+D

# to print a pattern in  a file , like is in a file
> grep "is" food.txt

# for is in beginning, use carrot character : '^'
> grep "^is" food.txt

>  grep "is$" food.txt

> grep "^is$" food.txt
```

- other wild cards

```
> cat > bug.txt
bag
beg
big
bug
bog
bg
b*g
# ctrl+D
# here : . --> indicates only one character between b and g
> grep "b.g" bug.txt

# here [a-z] --> valid option to appear between b and g
> grep "b[a-z]g" bug.txt

# search is specific range indicated by : []
> grep "b[aou]g" bug.txt

# to print "b*g" : i.e special character
- use / to remove special character
> grep "b\*g" bug.txt

# for no meaning of special character,search word as it is, so no wild card
meaning
> fgrep "b*g" bug.txt
```

13. Extended Regular expression character, use egrep

- grep -> only basic wildcard characters
- egrep --> basic + extended wildcard characters
- wild card character in Regular expression
- 1. 
  ```
  * : zero or more occurence of  previous character or group
  ```
- 2. 
  ```
  + : one or more occurence prev char /group
  ```
- 3. ? : 0 or 1 occurence
- 4. {n}, {m,n},{m},{n} : number of occurence of prev char /group
- 5. (w1|w2|w3) : find a word form w1,w2,w3

```
terminal> cat > big.txt
bg
big
biig
biiig
biiiig
biiiiig
biiiiiig
biiiiiiig
#ctrl+D

# zero or one
>  grep "bi*g" big.txt
- fail
## Extended category
```

```
# one or none
> egrep "bi?g" big.txt
# 1 or more
terminal> egrep "bi+g" big.txt
# exactly 3
terminal> egrep "bi{3}g" big.txt
# 3 oor more
terminal> egrep "bi{3,}g" big.txt
#3 or less than 3
terminal> egrep "bi{,3}g" big.txt
# between 3 and 5
> egrep "bi{3,5}g" big.txt

# for wit or  condition from  multiple values using |
>  egrep "(cake|biscuit|good)" food.txt
```

14. Regular Expression : flags

- 1. -c : count
- 2. -n : gives line search character found

```
> grep -c "printf" hello.c
> grep -c "b[a-z]g" big.txt

# * zero or more occurence of  previous character or group
terminal> grep -n "big" *.txt

terminal> grep -R -n "big"
# to check in all directory
> grep -R -n "goto" ~
#  grep,  recursively all files, - print line , check expression , file
path
```

15. build regex

- 1. for 10 digit mobile number, regrex

> "^[0-9]{10}$"

  - i.e start ^ [scan-set] {no on character} end $

# Shell Script using VIM Editor

## 1. VIM Editor related

- Vim Editor worlds best editor for terminal
- question on vi editor : copy on VI editor
- started as VI editor,, developed by Bill Joy
- now,we use Vim editor :VI improved editor

1. **VI editor modes :**

- 1. command mode
- press "Esc"
- 2. insert (edit)
- to insert = press "i"
- 3. write/save = :w
- 4. quit = :q
- 5. write and quit = :wq
- 6. quit without saving :q!

**cltr + s suspend and cltr + q resume**

- 7. :ls
  - to list VI files
- 8. :next

- to jumb forward to other files in VI

> :3next

  - to jump forward by 3 files
- 9. : prev

> :2prev

  - to jump back by 3 files , can be seen by :ls
- 10. : wqa
  - write and quit all
- 11. -O : open two files vertical tab

> vim -O one.h one.c

- 12. -o : open two files in horizontal tab

> vim -o one.h one.c

- 13. Cltr+ W+W
  - switch between tabs use
- 14. to copy content in file

```
use 2yy
take cursor to dest , press p
```

- 15. to open VIM editor

> vim <filename.extension>

2. **commands for Setting/customize VI**

```
# set tab stop : how many spaces
: set tabstop = 8
# set spaces on press of shift
: set shiftwidth=4

# no color
: syntax off

# color
: syntax on

# write and quit all files
: wqa

# to open two files, side by side
: vim -o one.c header.c

# set nowrap
: set nowrap

# set write all default
: set autowriteall
```

3. **for specifying setting only once , for VI** .vimrc file : content/commamds in it , are auto executed, when vi editor starts

- located in user's home directory

```
# to open .vimrc file
> vim ~/.vimrc

# to insert
: i
 set number
 set tabstop=4
 set shiftwidth=4
 set autoindent
 set nowrap
 set autowriteall
 syntax on


: wq
```

4. **data Manipulation Commands in VI editor**

- 1. to copy
    - yy --> copies current lin
    - 4yy --> copies 4 line from cursor

- :6,9y ---> copies line 6 to line 9
- yw --> copies current word (from cursor)
- 3yw --> copies 3 words from cursor

- 2. to paste
  - p ---> paste from the cursor
- 3. u ---> undo
  - for undo
- 4. cltr+R ---> redo
  - for redo
- 5. to cut
  - dd --> cut current lin
  - 4dd --> cut 4 line from cursor
  - :6,9d ---> cut line 6 to line 9
  - dw --> cut current word (from cursor)
  - 3dw --> cut 3 words from cursor
  - using cut we can delete too
- 6. to find
  - use / and filename to find

> Esc ---> :/word

- n ---> to find sub-sequent occurences

> :/printf

- to find printf

- 7. to find and replace use

> - Esc --> :%s/find/replace/#g --> replace all occurrences :%s/find/replace :%s/printf/scanf

- here can use #g for global at end

> :%s/printf/scanf/#g

- 8. Indentation

> Esc --> gg=G

- gg --> first line
- G --> last line
- = --> Indentation

- 9. Go to line

> Esc --> :34

- Jump to line 34

**10. how do you run bash command from VI editor , use Esc ---> :!command**

> Esc --> :!command

- to compile c file

> Esc ---> :!gcc -o hello.out hello.c

```
# to compile and run c program from Vim editor
Esc --> :!gcc -o one.out one.c

Esc --> :!./one.out
```

## 2. Shell Script (SS)

1. Shell script is collection of shell commands along with programming constructs lijke if-else,loop,case,functions.
2. it is interpreted language

- line by line execution
- 1. pros :
    - simplified syntax
    - quick development
- 2. cons :
    - fixed syntax (not free-form),not required spaces can give error
    - tough debugging
    - slower execution ,based on system (nowadays configure of pc better, so execution speed increased)

3. Applications of SS i.e .sh files

- 1. Installers
    - .sh files , installion command files can be made
- 2. Administrator
    - to fix some issues, having common set of instructions, use .sh file, so can be used by multiple
- 3. Automation job
    - like testing job

## 3. Shell Scripts demo

-

1. demo to write and run shell script file

- simplest shell script is set of commands
- the first line contains path of shell program to execute this script followed by #!
- when script is executed (./hello.sh), then OS invoke the shell program to execute this script
- 1. create SS

```
> vim hello.sh

#!/bin/bash
#the first line contains path of shell program to execute this script
```

```
followed by #!
#when script is executed (./hello.sh), then OS invoke the shell program to
execute this script
cal
uname -a
who

# Esc ---> :wq
```

- 2. to run SS using bash
    - used during development

> bash hello.sh

- 3. to give execute permission to SS file

> chmod +x hello.sh

- 4. now we can execute it using ./
    - using during production

> ./hello.sh

- 5. by default run with current shell(if sheband line is missing)
    - so run only by bash shell , for this in VIM , opening our .sh file
        - first line of code must be

    > #!/bin/bash

    - this command is known as Shebang line
    - it starts with # i.e comment for shell interpretor (/bin/bash)
    - so bash skip this command, but OS reads this line as its first line, as we used #! , and execute it
    - the first line contains path of shell program to execute this script followed by #!
    - when script is executed (./hello.sh), then OS invoke the shell program to execute this script

**Note :**

- 1. on Linux , group are created, which contains number of users,and those not in group are other

```

# so permission is shown in terminal as
  rwx   rwx    rwx
  user  group  other

> rw-r-----
# now change permission use chmod
```

2. demo 2 on shell varaible and operation

- shell variables created as

> var=value

- to print variable

> echo "$value"

- command substituation

```
#!/bin/bash

#while assigning value to variable do not use dollor
#no space before/after assign operator
num1=101
num2=5
# -n : skip trailing new line
echo -n "Unix is simple. "
echo "It need a "
# -e  to enable escape sequence
echo -e "Unix is very  \n user friendly"
# to access/read value of variable use $
echo "num1 value is $num1"
echo "num2 value is $num2"
# only integer arithmetic possible in shell
# for expression use expr compulsort
# using space between variable expression compulsory
expr $num1 + $num2
# traditional syntax, use backquote `` to compute and store, command
substitutation or command expansion
# can use var=$(command) also
result=`expr $num1 - $num2`
echo "sub is $result"
```

3. calculate area of rect using shell

- scale ---> for decimal points in result of expression
- bc --> indicates basic calculator for floating point calculation,
  - now bc prints on screen
- read : read input from user

```
echo "2.34 / 2"|bc

echo "scale=3;2.34 / 2"|bc
```

```
 #/bin/bash

 #calculate area of rect
 echo -n "enter length: "
 read len

 echo -n "enter breadth: "
 read br

  area=`expr $len \* $br`

  echo "area $area"

  echo -n "enter radius : "
  read rad

  # for floating point calculation, use bc --> i.e basic calculator
  # with  we can also use scale , no do decimal points it should be
accurate
  area=`echo "scale=4; 3.1415 * $rad * $rad"| bc`
  echo "area of circle : $area"
```

4. code for leap year in shell

- -eq --> equal to , -ne --> not equal to

```
 #!/bin/bash

 echo -n "enter year : "
 read y

 # c--> y%4==0 && y%100!-0 || y%400==0
 if [ `expr $y % 4` -eq 0 -a `expr $y % 100` -ne 0 -o `expr $y % 400` -eq 0
]
 then
     echo "leap year"
else
    echo "not leap year"
fi
```

5.code for loop

```
 #!bin/bash

 # print table of given number

 echo -n "enter a num: "
 read num

 i=1
```

```
while [ $i -le 10 ]
do
  res=`expr $num \* $i`
    echo "$res"
    i=`expr $i + 1`
done

for (( i=1; i<=10; i++ ))
 do
    res=`expr $num \* $i`
    echo "$res"
done
```

# day4

## notes

1. using [] in loop , is same as test command
2. tr command
3. loop use: continue and break commands in shell , for program like prime no
4. Special variable in Positional Parameters

- $0 : file name
- $1,$2...$9 :

5. shift commands: to shift/scrap first n arguments , and the further arguments are renumbered as $1,$2
6. bedefault user id = 1000, root user id = 0 [EUID]
7. Positional parameters

> terminal> ./script.sh arg1 arg2 arg3

- 1. Special variables
  - $0 --> for file name
  - $1, $2, , $9 --> for Command line argument(CLA)
  - $# ---> for count of collection/CLA
  - $* ---> for all collection element
- 2. shift command
  - to scrap/reset CLA specified in command , so we can use them again

  > shift number

8. BASH functions

```
# function declare and define
function my_func() {
}
```

```
result=$(my_func a g1 a g2
)
```

9. Array

- array declaration

> arr=( al1 al2 al3 )

- printing array element

> echo " ${arr[0]}, ${arr[1]} "

- collection of values/all elements of array

> ${arr[*]}

- count of values

> ${#arr[*]}

10. Strings

- string declaration

> str='string value'

- string length

> ${#str}

- for substring

> ${str:start_index}

- for substring count

> ${str:start_index:count}

- **special syntac for regular expression**

```
if [[ $str =~regex ]]
 then echo "true"
  fi
```

- **to find and replace words in string str**

> ${str/find/replace}

10. Directory operations

- to go to a directory and put it in stack

> pushd dirpath

- LIFO approach , using popd we go back one directory

> popd

- show directory stack

> dirs -v

## demos , Positional Parameters (in c Command line arguments )

0. syntax for loop

- 1. switch case

```
case expr in
c1)
    ...
    ;;
c2)
    ...
    ;;
c3)
    ...
    ;;
*)
    ...
esac
```

- 2. syntax for for loop

```
• for loop
# C like for loop
for (( initialization; condition; modification ))
do
    ...
done

# for-each loop

for var in collection
do
     ...
done
```

- 3. syntax for if-else

```
if [ condition ]
then
     ...
fi

# if elif fi

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

- 4. while loop

```
while [ condition ]
do
  ...
done
```

- syntax for until loop

```
until [ condition ]
do
  ...
done
```

1. using for loop for fixed array

```
#!/bin/bash

#for=each loop demo

for num in 11 22 33 44 55
do
 echo "$num"
done

~
```

2. using for and if loop

```bash
#!/bin/bash

# print all executable  files from given directory

echo -n "enter dir path : "
read dirpath

if [ -d $dirpath ]
then
  for file in `ls $dirpath`
  do
    if [ -x $file ]
    then
        echo "$file"
    fi
  done
else
   echo "invalid dir path"
fi

```

3. switch case

```bash
#!/bin/bash

# print no ofday  in a month

echo -n "enter a month "
read month

case $month in
1|jan|january)
  echo "Jan has 31 days"
  ;;
2|feb|february)
  echo "Feb has 28/29 days"
  ;;
*)
  echo "dont know "
esac

```

4. using tr command for translation from upper to lower vica-versa

```
tr - translate or delete characters
```

```
SYNOPSIS
       tr [OPTION]... SET1 [SET2]

DESCRIPTION
       Translate, squeeze, and/or delete characters from standard input,
writ-
       ing to standard output.

       -c, -C, --complement
              use the complement of SET1

       -d, --delete
              delete characters in SET1, do not translate

       -s, --squeeze-repeats
              replace each sequence of a repeated character that is listed
in
              the last specified SET, with a single occurrence of that
charac-
              ter

       -t, --truncate-set1
              first truncate SET1 to length of SET2

  [:alnum:]
              all letters and digits

       [:alpha:]
              all letters

       [:blank:]
              all horizontal whitespace

       [:cntrl:]
              all control characters

       [:digit:]
              all digits

       [:graph:]
              all printable characters, not including space

       [:lower:]
              all lower case letters

       [:print:]
              all printable characters, including space

       [:punct:]
              all punctuation characters

       [:space:]
              all horizontal or vertical whitespace
       [:upper:]
              all upper case letters

       [:xdigit:]
              all hexadecimal digits

```

```
#!/bin/bash

# print no ofday  in a month

echo -n "enter a month "
read month
month=`echo "$month" | tr "A-Z" "a-z"`

case $month in
1|jan|january)
  echo "Jan has 31 days"
  ;;
2|feb|february)
  echo "Feb has 28/29 days"
  ;;
*)
  echo "dont know "
esac
```

5. command line argument are

- $0 ---> file name
- $1,$2,..$9 ---> command line arguments
- $#--> no of arguments taken from CLI

```
#!/bin/bash
# shebang line

# addition of two numbers --passed on command line

if [ $# -ne 2 ]
then
    echo "invalid argu"
    exit
fi

result=`expr $1 + $2`
echo "result : $result"




```

```

# terminal> chmod +x demo8.sh
# terminal> ./demo8.sh 11 33

#$0 ---> name of script --> demo8.sh
# $1 --> arg1 ---> 11
# $2 ---> arg2 ---> 333
# $3,...,$9

# $# ---> count of arguments ---> 2 arguments for this program
# file name is excluded

```

6. passing all arguments in CommandLine argument, using $*

```
#!/bin/bash

# addition of all numbers passed on args

# terminal > ./demo9.sh 1 2 3 4 5
# $* ---> collection of args

sum=0

for num in $*
do
  sum=`expr $sum + $num`
done

echo "sum = $sum"

```

7. **Shift Command : mcq question**

```
#!/bin/bash

echo "arg1 : $1"

echo "arg2 : $2"

echo "arg3 : $3"
echo "arg4 : $4"

echo "arg5 : $5"
echo "arg6 : $6"
echo "arg7 : $7"
echo "arg8 : $8"
echo "arg9 : $9"
shift 9
```

```

echo "arg10 : $1"
echo "arg11 : $2"
echo "arg12 : $3"
echo "arg13 : $4"

# terminal > ./demo10.sh A B C D E F G H I J K L M N

# args: $1, $2, ..., $9
# HERE $10 AS ARGUMENT NOT AALLOWED
# 1 way then is SO USE loop
# 2 way , use shift command
# shift n command
#  shift "n" args to left (left "n" args are discarded)

~
```

8. function in shell

```
#!/bin/bash

# write a function to substract two numbers
# here echo in function are used for taking result to buffer ,again use echo after function call to print res
function substract()
{
 res=`expr $1 - $2`
 echo "$res"
}

function multiply()
{
res=`expr $1 \* $2`
echo "$res"

}

echo -n "enter two numbers"
read num1 num2
result=$(substract $num1 $num2)
echo "result : $result"

multiply $num1 $num2
```

9. array declaration

```
#!/bin/bash

# array demo
```

```

declare -a arr      # optional array declaration
arr=(11 22 33 44 55)

echo "element 0: ${arr[0]}"
echo "element 1: ${arr[1]}"
echo "element 2: ${arr[2]}"
echo "element 3: ${arr[3]}"

echo "array element count: ${#arr[*]}"

for num in ${arr[*]}
do
    echo "in for loop - ele: $num"
done

i=0
while [ $i -lt ${#arr[*]} ]
do
    echo "in while loop - ele: ${arr[$i]}"
    i=`expr $i + 1`
done
```

10. string related

```
#!/bin/bash

str1='sunbeam'
str2='infotech'

str="$str1$str2"

echo "cat string : $str"

echo "string length : ${#str} "

echo "substring of a string "
# substring ffrom index 3 to last char
echo "${str:3}"
# substring from index 7 to next 4 char
echo "${str:7:4}"

echo "compare strings "

if [ $str1 = $str2 ]
 then
    echo "equal strings"
else
   echo "two strings not equal"
fi

echo -n "enter phone number : "
```

```
read phone
# instead of regex we can use ready made syntax to  validate phone no
if [[ $phone =~ ^[0-9]{10}$ ]]
then
    echo "valid phone no : $phone"
else
    echo "invalid no "
fi

# find and replace
newstr=${str/tech/com}
```

```
# to make first digit non zero
if [[ $phone =~ ^[1-9][0-9]{9}$ ]]
```

11. to record or maintain directory track we can use

- Directory stack
- to go in and come out like : /bin --> /lib ---> /usr and come out i nreverse
- cd = change directory
- 1. so to go to dir and put it on stack use

> pushd dirpath

- 2. to pop to present dir from stack and fo to that dir

> popd

- 3.to get directory path in stack from directory stack use

  > dirs

```
sunbeam@sunbeam-Inspiron-3583:~/dac/OS/OS-module/classwork/Day4$ dirs
~/dac/OS/OS-module/classwork/Day4
sunbeam@sunbeam-Inspiron-3583:~/dac/OS/OS-module/classwork/Day4$ pushd
bash: pushd: no other directory
sunbeam@sunbeam-Inspiron-3583:~/dac/OS/OS-module/classwork/Day4$ pushd /bin
/bin ~/dac/OS/OS-module/classwork/Day4
sunbeam@sunbeam-Inspiron-3583:/bin$ dirs
/bin ~/dac/OS/OS-module/classwork/Day4
sunbeam@sunbeam-Inspiron-3583:/bin$ pushd /lib
/lib /bin ~/dac/OS/OS-module/classwork/Day4
sunbeam@sunbeam-Inspiron-3583:/lib$ dirs
/lib /bin ~/dac/OS/OS-module/classwork/Day4
sunbeam@sunbeam-Inspiron-3583:/lib$ popd
/bin ~/dac/OS/OS-module/classwork/Day4
sunbeam@sunbeam-Inspiron-3583:/bin$ popd
~/dac/OS/OS-module/classwork/Day4
```

#### afternoon session : Networking commands

12. ssh command

- ssh : secured shell
    - here data/communication is encrypted
    - based on Tcp protocol
    - ssh by default run on port 22
    - option of ssh or scp client
    - to check sssh server is on
    - where d : demain
    - most server run on bakcend no GUI , so caleed demain thread

  > systemctl status sshd

    - sshd running on port no given by

  > netstat -t lnp

    - port no of telnet : 23
    - port no of apache server : 80
    - port no of sql : 3306
    -

- telnet
    - shell, bash ,communication , not cncrypted

```

```

13.

- 1. ssh
    - encrypted communication
    - sshd server -- port=22
- 2. telnet
    - non-encry comm
    - telnet server --portno = 23
- 3. netcat command
    - create messaging app
    - to make a communication on port given : like 4321

  > terminal 1> netcat -l 4321 terminal 2> netcat localhost 4321

- 4. to get port listening
    - here,
    - -t -- tcp sockets
        - -l -- listening sockets (server sockets)

    - -n -- show port number
    - -p -- show process name

> sudo netstat -tlnp

- 5. to get ip address

> ip addr ifconfig

- 6.
- 7.
- 8.
- 9.

14. two ways to run shell

- 1. Interactive shell

- i.e bash prompt, at time of login, ssh,
- we can get interactive shell by

> cltr +alt + F1 or F2 or F3 or f4 or F5 or F6 or F7

  - one of which is gui terminal, get prompt

- 2. non interactive shell

> bash demo.sh

- internally bash command /program is executed
- to execute given script

- 3. env command
  - environment variable
  - it has import info about system
  - example
  - to get path

> $PATH

  - to get user

> $USER, $HOME

  - get running shell

> $SHELL

- get your shell prompt

> $PS1

```
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\
[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$
```

- to change env variable value , use

> export var=value

- in c main we have envp environment variable

```
   int main (int argc, char * argv[],char * envp[])
   { }
```

- after login shell aviable is called login shell

- 4. special shell script files in bash called based on type of file
    - 1. for new bash shell , when pc started i.e (login shell)
        - profile in bash shell, login in c shell ,were first used, both are same
        - use profile file mostly

> ~/.bash_profile ~/.bash_login

- 2. for bsh and bash shell
- for each login shell (GUI/CLI)
- Internally calls .bashrc

> ~/.profile

- new terminal in GUI ( interactive non-login shell)
- for each interactive shell

> ~/.bashrc

- at the time of logout

> ~/.bash_logout

- so to make changes in shell window
    - we can make changes to shell from it

> vim ~/.profile vim ~/.bashrc

- we can view them for login i.e for profile file changes use

> ssh localhost

15. ALIAS for terminal

-

1. alias for every terminal

- it can be created using

> alias c=clear

  - alias by default get destroyed when terminal is closed
  - we can remove it using

> unalias c

  - if you want to keep it running every where , so save it in .bashrc file

> vim ~/.bashrc

2. to get alias to run in running terminal/shell using shell script

> vim demo9-alias.sh

```
#!/bin/bash

alias u='echo $USER'
```

- now in termial , need to use
- it add alias to terminal

> source demo9-alias.sh

- or we can use this , to set alias to terminal

> . ./demo9-alias.sh

-

3. to switch to root login

> sudo su

- to switch back

16. for each command there is a file in bin

- can be look at it, with

> ls -l /bin/mkdir

# syllbus

- Linux Commands,
- Vi editor,
- Shell Scripting,
- Overview of OS,
- Processes,
- Scheduling & Synchronization,
- Memory management,
- File Systems,
- Case Study with Linux System Programming: Process,
- Signals,
- Semaphores & Mutex,
- Inter – Process Communication, POSIX Threads

### NOte

1. linux supportedfile system manager : (its own)ext3,fat,cdfs,ntfs
2. windows supported system manager : fat,Ntfs,cdfs
3. if you want to see liux file system in windows , use linux reader, ext2Fsd application

# day4 : OS -FIle System

## FILE System

1. file is collection of data and information on storage device

- i.e file = data(content) + metadata(information about file)
- 1. metadata is stored in inode (FCB) (FIle Control BLock)
    - contain Type,Mode,Size,User and Group,Links
    - Timestamps ,info about data blocks
    - where inode is structure
    - we can view metadata using, where filepath can be directory/file

  > stat filepath

- 2. File data is stored in
    - we have 3 types of file stytem : NTFS,FAT,ext3
    - where file system is the way of organizing file in disk

2. **File System is divided in 4 blocks** ,which are

- 1. **Boot block/Sector**
    - contains bootstap laoder
    - GRUB (grand unified bootLoader) for linux
- 2. **super block / Volume Control Block**

  - it contains info about partition, i.e remaining sections,
  - it contains metadata,
  - total no of data block present in disk space(data block) is done in super block
    - no of free data blocks
    - no used data blocks
  - there are four methos/ways by which information about free dta blocks can be kept inside super nblock referred as "free space management mechanism"
  - 1. bit vector
  - 2. liked list
  - 3. counting
  - 4. grouping
  - like label of storage, storage info , used, free space
- 3. **inode list block/ Master File Table**
  - inode structure , view using inode command
  - it contains pointer for each files in data block
  - even for empty file inode is there
- 4. **Data blocks**
  - file data

- this is general form for file system, no necessary for all file system to be same
- way of organizing file in filesystem changs FS to FS
-

3. FIle System Architecture

- 1. diagram 'File-system-arch'
- 2. some System Call API are
- 1. open ()
- 2. close ()
- 3. read ()
- 4. write()
- 5. lseek()

- called by system so called system calls

-

    3.

**mcq on system call, output type**

4. demo on system call API

- 1. to copy a file content to another file

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

```
int main(int argc, char *argv[]) {
    int fd, fs;
    char buf[32];
    int cnt;

    //0. check number of command line args

    if(argc != 3) {
        printf("invalid arguments.\n");
        return 1;
    }

    //1. open source file for reading
    fs = open(argv[1], O_RDONLY);
    if(fs < 0) {
        perror("open() failed to open source file");
        _exit(1);
    }
    //2. open destination file for writing
    fd = open(argv[2], O_WRONLY | O_TRUNC | O_CREAT, 0644);
    if(fd < 0) {
        perror("open() failed to open destination file");
        _exit(1);
    }
    //3. read few bytes from source file
    while( (cnt = read(fs, buf, sizeof(buf))) > 0 )
    {
        //4. write those bytes into destination file
        write(fd, buf, cnt);
    } //5. repeat steps 3 & 4 until end of source file

    //6. close destination file
    close(fd);
    //7. close source file
    close(fs);
    printf("file copied.\n");
    return 0;
}
//terminal> gcc -o filecopy.out filecopy.c
//terminal> ./filecopy.out ~/.bashrc copy.txt
```

- now command to compile and run file

```
terminal> gcc -o filecopy.out filecopy.c
terminal> ./filecopy.out ~/.bashrc copy.txt
```

- 1. **FILE POSITION pointer (f_pos)**

- file position is a pointer maintained for each file
- it is zero/start when we open a file

- so when a read/write happens, file position moves by that many bytes(eg 32 bytes)
- for each successive read/write, till the last byte of file, now file position becomes zero (0)

- 2.Why OS is written in C,C++, and Assemby language?

    - Linux/Unix is written in C language
    - 3 choices to write OS , C,C++ and Assembly langauge, as they are Hardware oriented language
    - Window written in C and C++
- 3. Directory

- in windows called folder and linux called directory
- internally it is a file,
- that contains directory entries for each syb directory and file in it
- 4096 byte(4kb) make a one data block, i.e its size
- let take a example as /home/sunbeam/.bashrc

1. sunbeam is a directory , it it also a special file , with inode and data block

- each file has a inode and data block
- so we want to open the directory /home/sunbeam/.bashrc

- here we firs t look in '/' and find home, and then its indoe is taken based on which it goes to home , and search in its data block sunbeam, similary takes sunbeam inode and goes to its data block and find .bashrc takes it s inode no, use it to open file

2.

- 1. for each process there is a process control block (PCB), it maintains a table called OFDT
- converting path name to inode number is called namei()
- 2. so there is a inode table in OS, as if no table, it will be slow to access inode each table, it contains/load inode no i.e for each file , size,Type
- 3. to access it/ open a file , it have a table, open file table(OFT), contains entry of inode table here to file
    - it contains pointer to file, f_pos=0,
- 4. open file descriptor table (OFDT) contains pointer/address to OFT ,also index of OFTD to the program called as File discriptor

## terms

- 5. here struct inode : inode tableentry
- 6. struct file : OFT entry
- 7. data block size (DBS) :
    - it is the memory block each file contains, with inode , i.e given to OS on creation
    - its default size, can be seen in format option
    - it is by default 4096 bytes (4kb)
    - it means , so we can set it i.e data block for file,
    - so 4kb in minimum/default size of data block size for each file
    - so this is a minimum no of bytes each file occupy
    - so higher the DBS , faster execution , but higher is wastage of space
    - size of buffer in buffer cache = size of Data block

- 8. so for each swp file , new inode is created, whenwe open a file in Vim editor
- 9. on harddisk level, write operation is done on sector basis i.e 512 bytes, it is called physical block size
- 10. for each file Data block is called it file level or logical block size
  9.

- > #pragma pack1

  - related to OS and hardware , not data block

1. commands to check

- memory info

> free -mh

- cpu info

> lscpu

# day 5

1. a file has boot block,super block, inode block, data block
2. so file logical 1st block mapping to harddisk block , this info is kept in inode
3. as inode contains info about data block, using which we can access respective block
4. VFS (Virtual File System) ---> file ---> called flat files (i.e shown as contiguous memory file)

- VFS convert file logical block to disk block using data blocks info in inode

5. to change current file position

> newpos = lseek(fd,offset,whence)

- where fd: file discriptor, offset: , whence : origin (values can be 0 (SEEK_SET),1(SEEK_CUR),2(SEEK_END) )
- in c we have

> fseek(fp,offset,whence)

6. e.g **MCQ IN EXAM**

-

0. lseek return new file position from beginning of the file

1. > lseek(fd,0,SEEK_SET)

- take f_pos to start of file
- in c library to do this, rewind(fp)

2. > lseek (fd,0,SEEK_END);

- takes f_pos to end of file

- here lseek returns file size , as goes to last bite

3. lseek(fd,0,SEEK_CUR);

- its stays in current position, so f_pos doesnt change
- it will return current file position
- same as C function, ftell()

7. f_pos is kept in OFT (Open File Table)

- if file open 4 times , then 4 time access OFT ,and have 4 f_pos

8. FIle I/O System Calls

- open() ---- fopen() --- returns FIle*
- close() ---- fclose()
- read() ---- fread()
- write() ---- fwrite()
- lseek() ---- fseek()

9. if dealing with Directory

> man 3 readdir

- 1. openat() --- opendir() --- return DIR*
    - to read directory
- 2. getdents() ---- readdir()
    - get directory
- 3. close() --- closedir()
    - close directory

- people prefer using library functon ,not syste mcall

-

10. file sywstem commands

- 1. mkdir()
- 2. stat()
- 3. rm cmd --> unlink()
- mv cmd --> link and unlink command

11. **Links in linux (shortcuts)**

- two types of link are there

1. Hard Link (another name for file)

- another directory entry pointing/ referring to same inode

```
> ln filepath linkpath
```

```
terminal> mkdir ~/links

terminal> cd ~/links

terminal> echo "first file." > one.txt

terminal> ls -l -i
## output  having link count 1
##: 8535450 -rw-r--r-- 1 sunbeam sunbeam 11 Dec 15 09:28 one.txt
terminal> ln one.txt two.txt

terminal> ls -l -i

terminal> ln one.txt three.txt

terminal> ls -l -i

# we can see link count became 3 here
8535450 -rw-r--r-- 3 sunbeam sunbeam 11 Dec 15 09:28 one.txt
8535450 -rw-r--r-- 3 sunbeam sunbeam 11 Dec 15 09:28 three.txt
8535450 -rw-r--r-- 3 sunbeam sunbeam 11 Dec 15 09:28 two.txt
```

- 1. link command

> ln filepath linkpath

- ln command internally call link() system function
- as link dir has inode and data block
- internally given inode to all link file, same as first file

- 2. rm command

> rm filepath

- internally makes call to unlink system call, it simply deletes directory entry,
- it decrement link count , if link count becomes zero, then release inode and data block of file
- shown in the below code

```
>  rm three.txt
>  ls -l -i
#output
## 8535450 -rw-r--r-- 2 sunbeam sunbeam 11 Dec 15 09:28 one.txt
## 8535450 -rw-r--r-- 2 sunbeam sunbeam 11 Dec 15 09:28 two.txt
>  cat one.txt
first file
>  cat two.txt
first file
>  rm one.txt
>  cat two.txt
first file
>  ls -l -i
```

```
total 4
8535450 -rw-r--r-- 1 sunbeam sunbeam 11 Dec 15 09:28 two.txt
>  rm two.txt
>  ls -l -i
total 0
>
```

- cannot create hardlink for directory

- 3. data recovery software,makes directory entry for your file deleted, so recovering data,
    - its dependent on file system
    - here when we delete, inode entry of deleted entry removed, data still there for reuse, till we override data over that data block, then it is completely deleted
- 4. to completely delete a dile i.e non recoverable data , file recovere possible,
    - for this,override and delete
    - there is shred command in linux for it , it garbage write file and delete it
    - so file can be recovered and data is garbage
- 5. to complete delete , with no file recoverable
    - use format, can be recovered
    - if use quick format, can be recovered
    - special utility software, zero utility format , completely 0 every file

**by default link count of directroy file is 2 ,**

**by default link count of file is 1**

- 2.
  ---

- it is good practice to use absolute path of targeted file ,while creating symbolic link
- same as windows shortcut +using link command, it creates a new special file , to keep addr/path of target file ,using

> ln -s filepath linkpath

- internally calls , symlink () system call

- here link count does not change , linked file contains path for file, so has memory of 7(name of file n bytes) bytes for the address of target file
- if main file deleted, linked file becomes useless
- here we can differentiate between targeted and linked file as diffrent file created, different inode,and datablock , not possible in hard link
- if targeted file(one.txt) is deleted, now other linked file becomes like dangling pointer , so no use for them, must be deleted

```
>  echo "new first file" > one.txt
>  ls -l -i
```

```
## output
#8535450 -rw-r--r-- 1 sunbeam sunbeam 15 Dec 15 09:55 one.txt
>  ln -s one.txt two.txt
>  ls -l -i
## output
#8535450 -rw-r--r-- 1 sunbeam sunbeam 15 Dec 15 09:55 one.txt
# 8535451 lrwxrwxrwx 1 sunbeam sunbeam  7 Dec 15 09:56 two.txt -> one.txt
>  ln -s one.txt three.txt
>  ls -l -i
## output
# 8535450 -rw-r--r-- 1 sunbeam sunbeam 15 Dec 15 09:55 one.txt
# 8535452 lrwxrwxrwx 1 sunbeam sunbeam  7 Dec 15 09:58 three.txt -> one.txt
# 8535451 lrwxrwxrwx 1 sunbeam sunbeam  7 Dec 15 09:56 two.txt -> one.txt
>  rm three.txt
>  ls -l -i
## output
# 8535450 -rw-r--r-- 1 sunbeam sunbeam 15 Dec 15 09:55 one.txt
# 8535451 lrwxrwxrwx 1 sunbeam sunbeam  7 Dec 15 09:56 two.txt -> one.txt
>  rm one.txt
>  cat two.txt
# cat: two.txt: No such file or directory
>  ls -l -i
#8535451 lrwxrwxrwx 1 sunbeam sunbeam 7 Dec 15 09:56 two.txt -> one.txt
>  rm two.txt
```

12. Free Space Management Mechanism

- when a file is requesting for free data blocks , then in which manner free data blocks get allocated for the file and in which manner info about those alocated data block can be kept inside an inode of that file
- physical data block exist in "data block " called as disk space
- total no of data block present in disk space(data block) metadata present in super block are
  - no of free data blocks
  - no used data blocks
  - there are four methos/ways by which information about free dta blocks can be kept inside super nblock referred as "free space management mechanism"
  - 1. **bit vector** : array of bits
       - in this mechanism,in super block bit vector i.e array of bits get maintained having size = total no of data block present in a data block(disk space)
       - i.e if total no of dataa block = K
       - size of bit vector = K
       - each data block has address from 0 to (K-1)
       - if nth bit = 0 ---> nth data block is free
       - if nth bit = 1 ---> nth data block is used
       - when any file is requesting for free data blco under disk allocation method bit vecotr get scanned
       - this mechanism is used with either linked allocation or index allocation
       - e.g : Extended filesystem :LINUX

- 2. **liked list**
  - in this , linked list of free data blocks get maintained inside data block itself and adress of starting data block in that list can be kept inside a super block
  - this mechanism is used with linked and index allocation
  - e.g :FAT16/FAT32
- 3. **counting**
  - in this mechanism , information about contiguous free data block can be kept in a super block
  - consist : 2 ----> 4 [count] (for block 2 to 5 contiguous free memory)
  - address of starting free data block ---> count (no of free data blocks exits in a contiguous manner including starting data block)
  - this mechanism used with contiguous allocation
- 4. **grouping**
  - in this mechanism , group of contiguous free data blocks get maintained and addresses of data blocks i nthat group get stored into the first free data block i nthat group, and address of first group data block can be kept in super block.

### 13. Disk Allocation Mechanism

- three popular mechanism :

1. **contiguous(C) allocation**

- inode contains mapping of logical block in inode with hardisk physical block

- **limitation on growing of file**, as no contiguous memory available, even in memory is available on HDD
- amount of memory required for file is available but not contiguous this is called **external /dis fragmentation**

- so moving disk block of HDD to make avaialable space contigous is called Defragmentation
- as memory in C, we can do sequential and ramdom access of memory

2. **linked allocation**

- to remove external fragmentation problem
- here each block contains data and pointer to next block, so memory for file is not contiguous
- inode contains mapping of logical block in inode with hardisk physical block(start block and end block )

- so last block can be changed, as file grows
- so here ramdom access is slow
- eg : FAT system

3. **indexed allocation**

- here, a special block is kept to maintain the information, i.e address of data blocks for a file, this block is called index block , this allocation is indexed allocation

- no extrnal fragmentation , but

- size of index block , has limitiation , so certain limitation for file to grow
- here we can do sequential and random access
- example in case of unix File System created by Dennis Richtie

- they have index box to keep data block address, ]
- here each index block has multiple entries and each entry is a index block , containing actual entries
- i.e index block pointing to multiple index block pinting to actual data block ,
- here mx file allowed 16 GB, and block size = 4kb,
- model file system: Ext2/3 --> similar to UFS , having an array of 15 memeber, not 13 of UFS , to create max file size 100GB
- so index blocked creating new index block in UFS , are called single indirect block, double indirect block ... ,
  - as no of indirect block increase ,speed decreases
  - as small files faster access , larger file slower access
- used by Linux (ext3,ext2 )

14. Journaling

- file system work in progress, for bigger file , large time req, and power failure there, so system gets down , so previous operation , a journel / log file is maintained here ,and how work is done, file system maintains log of each activity, so when system reboot file system refer the log file, and file system corruption chances reduces .

# day5 afternoon session

1. to check storage

> sudo fdisk -l

2. bebug

> sudo debugfs path

3. free space management algorithm

- super block maintains info about free inode , free data block
- there are 4 algorith of free space management :

- 1. bit vector/map
  - 4kb size of bit vector
  - 
- 2. linked list mechanism
  - each free block contains address of next free block
- 3. index mechanism
  - start block contains address of all contiguous
- 4. counting
  - no of free block are counted and listed

4. general structure of ext2/ext3 'ext3/2-arch'

- 1. create file system in linux

> sudo mkfs -t ext3 /dev/sdb1

- for window file system creation use format command

- 2. ext3 in partition has a block group which contains
  - 0. Boot block
    - common for all blocks in one partition
  - 1. super block
    - disk info
  - 2. group descriptor
    - current group info
  - 3. inode bit vector
  - 4. database bit vactor
  - 5. inode library
  - 6. data blocks

5. ext3 = ext2 + journaling

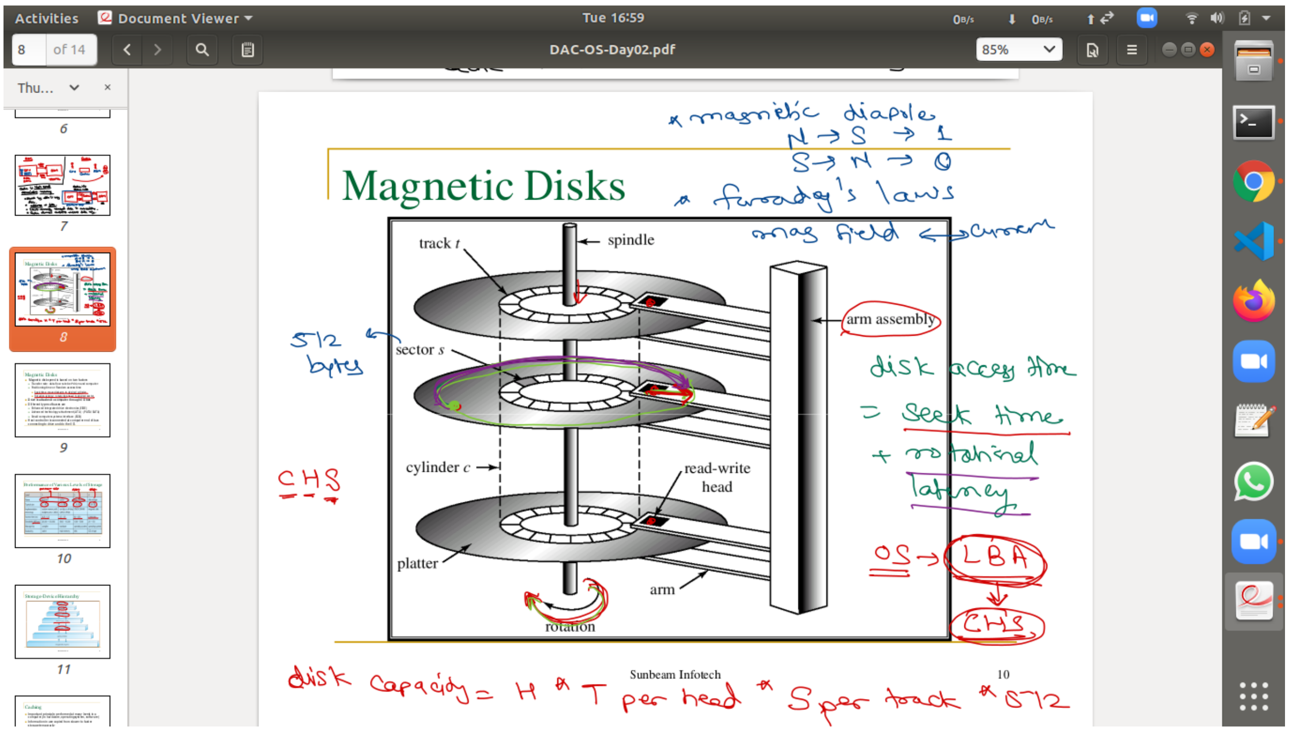
- to avoid inconsistency ,so FS doesnt get corrupt, due to power failure, solution is journaling i.e logging of FS activities , and match them on power on
- for this use journal /log file,
- for file system checks, we use tool

> fsck /dev/sdb1 + here check all directories,inode,and databases, this is slow proces

- journalling speedup checks by limiting checks to currently operated files and directories

## IMP for MCQ : 6

6. for Disk Management



Magnetic Disks

* magnetic diapole
N → S → 1
S → N → 0
* faraday's laws
mag field ⟷ current

512 bytes

CHS

disk access time
= seek time
+ rotational latency

OS → LBA
↓
CHS

Sunbeam Infotech 10

disk capacity = H * T per head * S per track * 512

- Disk access time = rotational latency + seek time
- when accessing disk , if following request are pendlng ,80,20,90,10,40,65,5
- my hdd arm pointer is at 30 , and radius from 0 to 100
- for this we have disk sheduling algorith

- 1. **FCFS : first come first served**
    - follow same order of request order

- 30 -> 80 ->20 -> 90 -> 10 -> 40 -> 65 -> -> 35

- so total seek distanxce = 50 + 60 + ... = 395

- 2. **SSTF : Shortest Seek Time First**
    - from current location of pointer , go to the nearest request
    - 30 -> 35 -> 40 -> 20 -> 10 -> 5 -> 65 -> 80 ->
    - T.S.D = 5 + 5 + 20 + ....= 130
- 3. **Scan /Elevator Alorithm**
    - magnitec tip keep moving from 0 to max , and access request as they come
    - 30 -> 35 -> 40 -> 65 -> 80 -> 90 -> 100 -> 90 -> ....->30-> 20 -> 10 ->5
    - T.S>D =
    - this algorithm is used in LINUX , by IO sub system
- 4. **Circular SCAN**
    - here magnetic pointer moves from 0 to max, then jumps back to 0
    - 30 --> 35--> 40 --> 65 --> 80 --> 90 --> 100 ---> (back to 0) --> 5 --> 10 --> 20
- 5. **Look Algorithm**
    - implementation policy of SCAN/C-SCCAN to conserve power /energy
    - move as a SCAN or C-SCAN, only when request pending ,its implementation policy

7. Process Management

- galvin book chapter 3,4,5

1. Process is a program in execution

2.

3. OS data structures for process execution

- 1. Job Queue / Process table/ Process List
    - contains all processes(PCB) in system
- 2. Ready Queue/ Run Queue
    - all processes ready to execute on cpu
- 3. waiting queue(s)
    - waiting queue per device for per syncronization object

**synchronizing object in java used by synchronized keyword is : monitor**

4. **Process Life/State Cycle (PLC)**

- running process is PLC

- 1. JOB-QUeue (new process initialized here)
- starts at Job-Queue /New --> initialized then added to
- 2. Ready Queue
- (ready for execution process)
- here cpu dispatch i.e (process goes from ready to running state) takes place
- 3. Running queue
- now CPU scheduler puts it in running state, it consumes time here, based on Round RObin algo
- if interupt occur, process go back to ready state ,
- 5. Waiting Queue
- if there is I/O operation / syncronization operation in Running state, process goes to waiting Queue (here P is sleeping)
- 6. once I/O operation finished, P go to Ready state, and then go to running
- 7. now after running process get Terminated

5. Linux Process State

- they are
- 1. **R state**
  - mean (process is Ready or Running)
  - indicated by (TASK_RUNNABLE)
  - if process has i/o request or syncronization operation
- 2. Linux(process waiting) has types
  - 1.**S state , its Interuptable sleep**
  - request not complete , but by sending signal, process in wait can also forcebly awaken, by interrupt
  - it has macro TASK+_INTERRUPTABLE
  - 2. **D state its Un-Interruptaable sleep** :
    - also refer as dormant state
    - request not complete , but by sending signal, process in wait cannot be forcebly awaken, by interrupt
    - it has macro TASK+_UNINTERRUPTABLE
  - 3. process suspended
    - process in stop state, using stop() signal(Cltr + S can be used for this in VIm and to awaken Cltr + Q )
    - macro TASK_STOPPED
- 3. **Z state(process terminated)**
  - here we use exit
  - macro TASK_ZOMBIE

6. commands to check state of process

> ps -e -o pid,state,cmd

- to let a process go to background , use Cltr + Z

  > bg

- to bring it back to foreground

> fg

7. CPU Scheduler

- decides next process to be executed on CPU

8. CPU Dispatcher

- dispatch the process on CPU i.e load its execution context (CPU regualr values) into CPU

7. Process Creation(PC)

- 1. PC depends on Operating System ,its system call

- example

- 1. in windows --> CreateProcess()
- 2. in Unux --> fork()
- 3. in BSD UNIX --> fork(), vfork()
- 4. Linux ---> fork(),vfork(), clone()
- 2. we are gonna use fork()

8. **fork()**

- creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process.
- The child process and the parent process run in separate memory spaces. *The child process is an exact duplicate of the parent process except for the following points:
- The child has its own unique process ID, and this PID does not match the ID of any existing process group (setpgid(2)) or session.

- On success, the PID of the child process is returned in the parent, and 0 is returned in the child.
- On failure, -1 is returned in the parent, no child process is created,

- 1. basic structure

```
#include <sys/types.h>
#include <unistd.h>
// pid is type def on int
//   pid_t fork(void);

main(){

 int ret;
 ret = fork();
 // it successful, return 0 to child process and pid no to pCB and parent
printf(); // run by parent and child
 if( ret == 0) {
   // true for child process
 } else{
   // true for parent
 }

}
```

- 2. basic demo

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

int main() {
   int ret;
   printf("program started.\n");
   ret = fork();
   printf("fork() returned: %d\n", ret);
   printf("program completed.\n");
   return 0;
}

// terminal> gcc -o fork.out fork1.c
// terminal> ./fork.out
```

- on bash

```
> gcc -o fork-file1.out fork-file.c
> ./fork-file1.out
program started.
fork() returned: 19890
program completed.
fork() returned: 0
```

3. demo with different tasks for parent and child process

- use to run process concurrently ,independent process

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

int main() {
    int i, ret;
    printf("program started.\n");
    ret = fork();
    printf("fork() returned: %d\n", ret);
    if(ret == 0) {
        for(i=0; i<300; i++) {
            printf("child: %d\n", i);
        }
    } else {
        for(i=0; i<300; i++) {
            printf("parent: %d\n", i);
        }
    }
    printf("program completed.\n");
    return 0;
}
// terminal> gcc -o fork.out fork2.c
// terminal> ./fork.out
```

- 5. to see process info
  - ps displays information about a selection of the active processes.
  - count of all process running

> ps -e | wc -l

  - here -o : User-defined format
  - pid :

> ps -e -o pid,state,cmd

  - to see child and parent pid

> ps -e -o pid,ppid,cmd

> ps -e -o pid,ppid,state,cmd

```
          -e :Select all processes.  Identical to -A.
    here   -o : User-defined format
  KEY    LONG         DESCRIPTION
        c      cmd          simple name of executable
               state        Process state
        C      pcpu         cpu utilization
        k      utime        user time
        o      session      session ID
        p      pid          process ID
        P      ppid         parent process ID
        s      size         memory size in kilobytes
        U      uid          user ID number
        u      user         user name
        y      priority     kernel scheduling priority
```

**MCQ:Question**

- 6. demo for orphan process

- init or systemd process id 1
- process with process id 1 is init in older linux , and systemd in newer linux
- when parent process is terminated then the child process is called orphan process , and the process is taken by init process through , a reaper process which is at /lib/systemd/systemd --user
- ownership of orphan process taken by init process pid 1
- if child process completes earlier then parent , we get [fork2.out]

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

int main() {
    int i, ret;
    printf("program started.\n");
    ret = fork();
    printf("fork() returned: %d\n", ret);
    if(ret == 0) {
        for(i=0; i<30; i++) {
            printf("child: %d\n", i);
        }
    } else {
        for(i=0; i<15; i++) {
            printf("parent: %d\n", i);
        }
    }
    printf("program completed.\n");
    return 0;
}

// terminal> gcc -o fork.out fork2.c
// terminal> ./fork.out
```

7. demo of Zombie state

- if child process completes earlier then parent , we get [fork2.out] , i.e child process is in zombie state
- as child process need a confirmation from parent process, to terminate , by system call , wait ()

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

int main() {
    int i, ret, s;
    printf("program started.\n");
    ret = fork();
    printf("fork() returned: %d\n", ret);
    if(ret == 0) {
        for(i=0; i<15; i++) {
            printf("child: %d\n", i);
            sleep(1);
        }
        _exit(3); // child exit status = 3
    } else {
        for(i=0; i<30; i++) {
            printf("parent: %d\n", i);
            sleep(1);
            if(i==15) {
                wait(&s); // get exit status of child process from its pcb
and release pcb of the child.
                printf("child exit status: %d\n", WEXITSTATUS(s));
            }
        }
    }
    printf("program completed.\n");
    return 0;
}

// terminal> gcc -o fork.out fork6.c
// terminal> ./fork.out
```

8. to avoid Zombie state using wait

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

 // wait() syscall does following
 // 1. block parent execution until any of its child is terminated
 // 2. read exit status of the child (out param of wait())
 // 3. release PCB of the child process

  // after wait() child proces is no more zombie


  int main() {
      int i, ret, s;
      printf("program started.\n");
      ret = fork();
      printf("fork() returned: %d\n", ret);
      if(ret == 0) {
          for(i=0; i<15; i++) {
              printf("child: %d\n", i);
              sleep(1);
```

```
            }
            _exit(3); // child exit status = 3
        } else {
            for(i=0; i<30; i++) {
                printf("parent: %d\n", i);
                sleep(1);
                if(i==15) {
                ret = wait(&s); // get exit status of child process from its
pcb and release pcb of the child.
                    printf("child (%d)exit status: %d\n",ret,
WEXITSTATUS(s));
                }
            }
        }
        printf("program completed.\n");
        return 0;
    }

    // terminal> gcc -o fork.out fork6.c
```

9. waitpid()

> waitpid(child_pid,&exit_status,flags ) syscall does following

1. block parent execution until any of a given child is terminated

- if arg1=-1 then wait for any child.

2. read exit status of the child (out param of wait())
3. release PCB of the child process

- returns pid of child process on success

4. flags -

- behaviour of waitpid() call
- 0 - same as wait() -- block the parent process until termination of child
- flag : WNOHANG - do not block parent process

5. if child already terminated , get its exit status and return its pid

- if no child is already terminated, returns error codde
- after wait() child proces is no more zombie

# day 6

## MCQ

1. In C language, ftell() returns the current file position of the specified stream with respect to the starting of the file. This function is used to get the total size of file after moving the file pointer at the end of the file

2. The C library function void rewind(FILE *stream) sets the file position to the beginning of the file of the given stream.

3. **to remove IPC related Objects**

- if we forget to delete mesage queue, then message queue will still remain in memory, i.e resource leakage, so to remove it, we have two command (by key and by value)

- ipcrm - remove certain IPC resources
- ipcrm removes System V inter-process communication (IPC) objects and associated data structures from the system. In order to delete such objects, you must be superuser, or the creator or owner of the object.
- command to remove ipc related object

> ipcrm -extension

```
  -M, --shmem-key shmkey
 Remove the shared memory segment created  with  shmkey after the last
detach is performed.
 -m, --shmem-id shmid
Remove  the  shared memory segment identified by shmid after the last
detach is performed.
 -Q, --queue-key msgkey Remove the message queue created with msgkey.
-q, --queue-id msgid Remove the message queue identified by msgid.
-S, --semaphore-key semkey Remove the semaphore created with semkey.
-s, --semaphore-id semid Remove the semaphore identified by semid.
```

4. what is return type/value of wait system call,

- wait()

> pid_t wait(int *wstatus);

- All of these system calls are used to wait for state changes in a child of the calling process, and obtain information about the child whose state has changed.
- A state change is considered to be: the child terminated; the child was stopped by a signal; or the child was resumed by a signal. In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed, then the terminated child remains in a "zombie" state

- wait(): on success, returns the process ID of the terminated child;
- on error, -1 is returned.

1. open ( )

- open(), openat(), and creat() return the new file descriptor, or -1 if an error occurred

6. close () : close a file descriptor

- close() returns zero on success. On error, -1 is returned,

7. read()

-

> ssize_t read(int fd, void *buf, size_t count);

- read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.
- If the file offset is at or past the end of file, no bytes are read, and read() returns zero.

- On success, the number of bytes read is returned
- On error, -1 is returned

8. write() : write to a file descriptor

> ssize_t write(int fd, const void *buf, size_t count);

- write() writes up to count bytes from the buffer starting at buf to the file referred to by the file descriptor fd.

- On success, the number of bytes written is returned
- On error, -1 is returned

9. lseek : reposition read/write file offset

> off_t lseek(int fd, off_t offset, int whence);

- lseek() repositions the file offset of the open file description associated with the file descriptor fd to the argument offset according to the directive whence as follows:

- SEEK_SET

- The file offset is set to offset bytes.

- SEEK_CUR

- The file offset is set to its current location plus offset bytes.

- SEEK_END The file offset is set to the size of the file plus offset bytes.

- Upon successful completion, lseek() returns the resulting offset location as measured in bytes from the beginning of the file.

- On error, the value (off_t) -1 is returned

10. fork () : create a child process

> pid_t fork(void);

- fork() creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process.
- On success, the PID of the child process is returned in the parent, and 0 is returned in the child.
- On failure, -1 is returned in the parent, no child process is created, and errno is set appropriately.

11. waitpid()

> waitpid(child_pid,&exit_status,flags ) syscall does following

1. block parent execution until any of a given child is terminated

- if arg1=-1 then wait for any child.

2. read exit status of the child (out param of wait())
3. release PCB of the child process

- returns pid of child process on success

4. flags -

- behaviour of waitpid() call
- 0 - same as wait() -- block the parent process until termination of child
- flag : WNOHANG - do not block parent process

5. if child already terminated , get its exit status and return its pid

- if no child is already terminated, returns error codde
- after wait() child proces is no more zombie

## notes

1. **command for executabele file**

- display information from object files.

> objdump -t exec_file.out

- ELF files are Executable Linkable Format
- which consists of a symbol look-ups and relocatable table, that is, it can be loaded at any memory address by the kernel and automatically, all symbols used, are adjusted to the offset from that memory address where it was loaded into.

- Displays information about ELF files

> readelf -h exec1.out

2. **File SysCalls**

- 1. leseek( )
    - Same as fseek()

- newpos = lseek(fd, offset, origin)
    - Internally change current file position (f_pos) stored in OFT of the file.

  - f_pos -- indicate next byte to be read/written
  - Same as newpos = fseek(fp, offset, origin)
  - offset can be +ve or -ve based on origin.
  - origin = SEEK_SET (beginning of file)
    - offset must be +ve
  - origin = SEEK_END (end of file)
    - offset must be -ve
  - origin = SEEK_CUR (current position)
    - offset can be +ve or -ve
    - +ve = forward move
    - -ve = backward move

3. Mounting

- CD/DVD -- E: -- CDFS/iso9660
- Windows/modern Linux does mounting automatically.
  - Windows: mountvol.exe
  - Linux: mount
- In Linux mounting can be done manually.

```
> sudo mount -t vfat /dev/sdb1 /mnt
  > ls /mnt
  > ...
  > sudo umount /mnt
```

# Process Management

- Process definition
- Process Life Cycle
- Pre-emptive scheduling vs Non-pre-emptive scheduling
- Process creation
- fork() syscall, using this, process based multi tasking
- orphan process
- zombie process

5. wait()/waitpid() syscall

- wait() -- UNIX syscall
- waitpid() -- Linux syscall
- All of these system calls are used to wait for state changes in a child of the calling process, and obtain information about the child whose state has changed.
- A state change isconsidered to be:

- the child terminated;
- the child was stopped by a signal; or
- the child was resumed by a signal.

- In the case of a terminated child, performing a wait allows the system to release the resources associated with the child;
- if a wait is not performed, then the terminated child remains in a "zombie" state .
- wait(): on success, returns the process ID of the terminated child; on error, -1 is returned.

## differnce between exit() and _exit() syscall

- exit() -- C library function -- <stdlib.h>
  - exit the current process -- internally calls _exit() syscall.
- _exit() -- syscall -- <unistd.h>
  - Release all resources (memory, ipc, ...) occupied by the process.
  - Write its exit status into its PCB.

## exec_system call

1. it does'nt create a process, it loads new program in calling process memory

**mcq : imp**

2. exec loads new program in calling process memory, whereas fork creates new process
3. loader that loads program in memory, is exec() itself
4. exec is a family of function, it has 7 function

- int execl()
- int execlp()
- int execle()
- int execv()
- int execvp()
- int execvpe()
- where :

- l : varaible argument list for command line arguments, to pass process
- v : argument vector/array --> commandline arguments
- p : child program will be searched in all directories mantioned in PATH variable
- e : environment variables to be given to child
  - example in C main (envp)

5. if directly call exec, a new program loaded on parent process, and it data is overloaded, as no child is created for being over writeen

- to find path of command file

> which command

- system call to ge pid of self

> getpid()

- and parent

> getppid()

- 1. using l and v and vp q

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

int main() {
    int ret, err, s;
    printf("parent started.\n");
    ret = fork();
    if(ret == 0) {
        // cal -y 2020

        // err = execl("/usr/bin/cal", "cal", "-y", "2020", NULL);

        //char *args[] = { "cal", "-y", "2020", NULL };
        //err = execv("/usr/bin/cal", args);

        char *args[] = { "cal", "-y", "2020", NULL };
        err = execvp("cal", args);

        if(err < 0) {
            perror("exec() failed");
            _exit(1);
        }
    }
    else {
        waitpid(ret, &s, 0);
        printf("child exit status: %d\n", WEXITSTATUS(s));
    }
    printf("parent completed.\n");
    return 0;
}
```

- 2. using strtok

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

 int main(){
 char cmd[512], *ptr;

 printf("cmd> ");
     gets(cmd);

  ptr = strtok(cmd, " " );
  puts(ptr);
  do {

  ptr = strtok(NULL, " " );
  puts(ptr);

   } while(ptr != NULL);

  return 0;

  }
```

- 3.

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

int main() {
    char cmd[512], *ptr, *args[32];
    int i;

    printf("cmd> ");
    gets(cmd);
    i = 0;
    ptr = strtok(cmd, " ");
    //puts(ptr);
    args[i++] = ptr;

    do {
        ptr = strtok(NULL, " ");
        //puts(ptr);
        args[i++] = ptr;
    }while(ptr != NULL);

    for(i=0; args[i]!=NULL; i++)
        puts(args[i]);

    return 0;
}
```

- 4. shell

- as shell interanlly does fork + exec , to run commands
- some commands are called bash built in or interanl commands , like exit,cd,alias,export
- external commands have their exe under /bin/
-

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    char cmd[512], *ptr, *args[32];
    int i, ret, err, s;

    printf("cmd> ");
    gets(cmd);
    i = 0;
    ptr = strtok(cmd, " ");
    //puts(ptr);
    args[i++] = ptr;

    do {
        ptr = strtok(NULL, " ");
        //puts(ptr);
        args[i++] = ptr;
    }while(ptr != NULL);

    //for(i=0; args[i]!=NULL; i++)
    //  puts(args[i]);

    ret = fork();
    if(ret == 0) {
        err = execvp(args[0], args);
        if(err < 0) {
            perror("bad command");
            _exit(1);
        }
    }
    else
        wait(&s);

    return 0;
}
```

- 
    6.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
```

```
int main() {
    char cmd[512], *ptr, *args[32];
    int i, ret, err, s;

    while(1) {
        printf("cmd> ");
        gets(cmd);
        i = 0;
        ptr = strtok(cmd, " ");
        //puts(ptr);
        args[i++] = ptr;

        do {
            ptr = strtok(NULL, " ");
            //puts(ptr);
            args[i++] = ptr;
        }while(ptr != NULL);

        //for(i=0; args[i]!=NULL; i++)
        //  puts(args[i]);

        if(strcmp(args[0], "exit") == 0 || strcmp(args[0], "bye") == 0)
            _exit(0);
        else if(strcmp(args[0], "cd") == 0)
            chdir(args[1]);
        else {
            ret = fork();
            if(ret == 0) {
                err = execvp(args[0], args);
                if(err < 0) {
                    perror("bad command");
                    _exit(1);
                }
            }
            else
                wait(&s);
        }
    }
    return 0;
}
```

- 7. to get session id, process group id ,thread id

- where session if, for each bash terminal opened, bash pid is session id
- for a no of commands running together makes a group, called process group, having a process group id
- similarly thread are created ,first being main thread, each process have multiple threads, having thread id (tig)

> ps -e -m -o pid,tid,tgid,nlwp,cmd

- output: PID TID TGID NLWP CMD

> ps -e -o pid,sid,cmd

> ps -e -o pid,pgid,cmd

- PID PGID CMD

- 8. The exec() functions return only if an error has occurred.The return value is -1,

## Inter Process Communicaton (IPC)

1. memory of one process cannot be shared with another process, so IPC
2. LInux specific IPC mechanism

- 1. Signals
- 2. Shared memory
- 3. message queue
- 4. pipe
- 5. Socket

### 3. Signals

- in it data doesnt travel from one process to another, still communication happens

- e.g : call and miscall , having specific meaning between friends, without voice data

- so there are set of predefined signals, which communicate, not actual data

- so the signals can be send by,

    - one process can send signal to another process,
    - OS to any Process,
    - but we/user cannot send signal to OS
    - default action takes plece
    1. TERM : terminate action
    2. CORE : Aborted and creates core dump
    3. STOP : suspend
    4. CONT : resume suspended
    - Default action is to stop the process.
    5. IGN : not handled, get ignored

    > man 7 signals

    - to see all signals

    > kill -l

    - Importent signals
    1. SEGINT (2)

- Cltr +C , interanlly generates SEGINT , and default action of it TERM

2. SIGTERM(15)

- default action TERM , when shutdown , SIGTERM terminates process, normally

3. SIGKILL(9)

- default action TERM
- when shutdown, SIGKILL forcibely terminates process

4. SIGSTOP (19)

- cltr + S :i.e STOP action , so process get suspended

5. SIGCONT (18)

- cltr + q , continues suspended process

6. SIGSEGV

- segment violation (due to dangling pointer), its CORE action

7. SIGCHLD (17)

- child send signal to parent , while terminating [as we dont know how to handle it till now so action is IGN]

#### Send Signals

- implemented loop.c, infinite loop and execute it
- in terminal 1

> gcc -o loop.out loop.o ./loop.out

- terminal 2

> ps -e

- find pid of loop.out

> kill -2 pid

- try signals : SIGINT, SIGTERM,SIGKILL,SIGSTOP,SIGCONT,SIGSEGV,SIGHUP,SIGCHLD

- To kill multiple processes of the same program

  > terminal> pkill -9 chrome terminal> pkill -KILL chrome

- closing terminal : SIGHUB signal is given

- i.e hang up signal - meaning parent terminal is closed
- so to ignore hangup signal, - HUP , we can use

> nohup ./loop.out

- so app doesnt close even in terminal get closed, example used in AWS
- if multiple program nunning , so how to terminate them using kill
- we cannot kill process group id ,
- we can use

> kill -signum pid kill -signame pid

> pkill -signum programname pkill -signame programname

- to kill all process

> killall

- to kill all instance of a program

> kill -9 pid

**two signals cannot be handled**

1. SIGKILL,SIGSTOP , as used in shutdown

**recieving signal**

1. signal is software counter part of hardware interrupt
2. when a process is executing , meanwhile signal is send to process, it pause current process,
- goes to table, called signal handler table (having 64 signal, handler)
- here it gets signal handler(), and implement it,
- to handle a signal ,is 2 step process
1. write signal handler
2. make its entry into signal handler table
- signal system call
- 1. demo to handle a signal

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

// signal handling demo

//step1: implement signal handler function
void sigint_handler(int sig) {
    printf("signal is caught: %d\n", sig);
}

int main() {
    int i=0;
    //step2: register signal handler (in signal handler table of
current process).
    signal(SIGINT, sigint_handler);
    while(1) {
        printf("loop: %d\n", ++i);
        sleep(1);
    }
    return 0;
}

//terminal 1> gcc -o signal.out signal1.c
//terminal 1> ./signal.out

// ctrl + C
```

```
//terminal 2> pkill -2 signal.out
```

- 2.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <signal.h>

// step1: implement signal handler function
void sigchld_handler(int sig) {
    int s;
    wait(&s); // get exit status of child process from its pcb and
release pcb of the child.
    printf("child exit status: %d\n", WEXITSTATUS(s));
}

int main() {
    int i, ret, s;
    // step2: register signal handler
    signal(SIGCHLD, sigchld_handler);

    printf("program started.\n");
    ret = fork();
    printf("fork() returned: %d\n", ret);
    if(ret == 0) {
        for(i=0; i<10; i++) {
            printf("child: %d\n", i);
            sleep(1);
        }
        _exit(0); // child exit status = 3
    } else {
        for(i=0; i<30; i++) {
            printf("parent: %d\n", i);
            sleep(1);
        }
    }
    printf("program completed.\n");
    return 0;
}

// terminal> gcc -o signal.out signal2.c
// terminal> ./signal.out
```

- 3. Signal convention
- UNIX
- signal ( ) ->

- register signal handler for given signal , Internally it make its entry into process( PCB) signal handler table
- it return s old sig handler address
- Linux

1. sigaction()

- register sig handler, enchanced signal() sys call
- signal handler are typlically registered at the start of application

## 4. Shared Memory

0. shred memory region is created in user space

- Multiple process can have pointers to shared memory region (attached processes)
- Any process can write into shared memory and other processes can read from it directly
- Fastest IPC mechanism

1. demo on using fork , a duplicate process of parent process is created, so each process have a copy of varaibles in program/process, so the out put of this program is :

> child , 1,2,3, parent , -1,-2 ,...-12

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

struct my_data {
    int count;
};

struct my_data var;

int main() {
    int i, ret, s;
    struct my_data *ptr;
    ptr = &var;
    ptr->count = 0;

    ret = fork();
    if(ret == 0) {
        for(i=0; i<10; i++) {
            ptr->count++;
            printf("child: %d\n", ptr->count);
            sleep(1);
        }
    }
    else {
        for(i=0; i<10; i++) {
            ptr->count--;
            printf("parent: %d\n", ptr->count);
            sleep(1);
        }
        waitpid(-1, &s, 0);
        printf("final count = %d\n", ptr->count);
    }
    return 0;
}

// terminal> gcc -o shm.out shm.c
// > ./shm.out
```

2. ifter using fork, how to maintain a common varaible for both process, i.e

- Shared Memory Area (SMA) : is a special memory areas, which can be shared among multiple process
- to create SMA , OS interanlly creates , shared Memory Object(SMO)
    - here in SMO we have
    1. key : like pid
    2. mode :
    3. n attached : no of process connected
    4. addr : address of SMA
    5. size : size of SMA
    6. owner : owner of SMA
- to keep reference of SMO we have share memory Table (SMT), and reference index is called share memorry id (smid)
- command to show shared memory, message queue , semaphore

> ipcs

```

----- Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

```

- runlevel command --> Print previous and current SysV runlevel
- command tells , what level running,

- if runlevel 3: we have CLI
- if runlevel 5: we have GUI

> run level

3.

```

```

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/shm.h>

#define SHM_KEY 0x1234

struct my_data {
    int count;
};

int main() {
    int i, ret, s, shmid;
    struct my_data *ptr;

    // create shared memory region
    shmid = shmget(SHM_KEY, sizeof(struct my_data), IPC_CREAT | 0600);
    if(shmid < 0) {
        perror("shmget() failed");
        _exit(1);
    }

    // get pointer to shared memory
    ptr = (struct my_data*) shmat(shmid, NULL, 0);
    if(ptr == (void*)-1) {
        perror("shmat() failed");
        _exit(1);
    }

    ptr->count = 0;

    // create child process
    ret = fork();
    if(ret == 0) {
        for(i=0; i<10; i++) {
            // child: increment the count
            ptr->count++;
            printf("child: %d\n", ptr->count);
            sleep(1);
        }
        // release shared memory pointer
        shmdt(ptr);
    }
    else {
        for(i=0; i<10; i++) {
            // parent: increment the count
            ptr->count--;
            printf("parent: %d\n", ptr->count);
            sleep(1);
        }
        waitpid(-1, &s, 0);
        printf("final count = %d\n", ptr->count);
        // release shared memory pointer
        shmdt(ptr);
        // delete shared memory region
        shmctl(shmid, IPC_RMID, NULL);
    }
    return 0;
}
```

```
> gcc -o shm.out shm2.c
> ./shm.out

> ipcs

#  Shared Memory Segments --------
  # key           shmid      owner      perms      bytes      nattch
stat
# 0x00001234 65576      sunbeam    600        4          2
>
```

4. to mark for deletion, shmdt , so we see

> ipcs

- stst = dest
- i.e mark for deletion

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/shm.h>

#define SHM_KEY 0x1234

struct my_data {
    int count;
};

int main() {
    int i, ret, s, shmid;
    struct my_data *ptr;

    // create shared memory region
    shmid = shmget(SHM_KEY, sizeof(struct my_data), IPC_CREAT | 0600);
    if(shmid < 0) {
        perror("shmget() failed");
        _exit(1);
    }
    // get pointer to shared memory
    ptr = (struct my_data*) shmat(shmid, NULL, 0);
    if(ptr == (void*)-1) {
        perror("shmat() failed");
        _exit(1);
    }

    ptr->count = 0;

    // mark shared memory region for deletion
        // shared memory region will be deleted, when no processes attached
to it.
    shmctl(shmid, IPC_RMID, NULL);

    // create child process
    ret = fork();
    if(ret == 0) {
        for(i=0; i<10; i++) {
            // child: increment the count
            ptr->count++;
            printf("child: %d\n", ptr->count);
            sleep(1);
        }
        // release shared memory pointer
        shmdt(ptr);
    }
    else {
        for(i=0; i<10; i++) {
            // parent: increment the count
            ptr->count--;
            printf("parent: %d\n", ptr->count);
            sleep(1);
        }
        waitpid(-1, &s, 0);
        printf("final count = %d\n", ptr->count);
        // release shared memory pointer
        shmdt(ptr);
        // nattached count of shm falls to zero, hence shared memory object
is deleted.
    }
    return 0;
}

// terminal> gcc -o shm.out shm2.c
// terminal> ./shm.out
```

## 5. Message Queue (MQ)

0. Process send message to kernal (copied from user space to kernel space ) and then other process recieve it from the kernel (copied from kernel space to user space).

- Slower than Shared memory

- Bi-directoional data transfer (message types will differ)
- Packet based data transfer

1. the way SMS is send from mobile to tower, tower to 2nd mobile, same way message queue works
2. process 1 to kernel (message queue object) from kernal to process 2
3. in message Queue Object (MQO) we have

- 1. KEY : key of message
- 2. COunt : of message
- 3. list : of message , i.e a pointer to queue
- 4. size : in bytes
- 5. waiting queue : mainly for reciever process

4. to create a track of message queue object(MQO), we have a message queue table(MQT) ,

- which hold reference to MQO , it index is called mqid

5. here for each message from a process ,has a message type (contains address)

- it must be long type ,and at start of message block
- as IPC developed by one developer, so sender and reciever adress ,we will know

**MCQ:**

6. if we forget to delete mesage queue, then message queue will still remain in memory, i.e resource leakage, so to remove it, we have two command (by key and by value)

- ipcrm - remove certain IPC resources
- ipcrm removes System V inter-process communication (IPC) objects and associated data structures from the system. In order to delete such objects, you must be superuser, or the creator or owner of the object.

> ipcrm

```
-Q, --queue-key msgkey
Remove the message queue created with msgkey.

-q, --queue-id msgid
Remove the message queue identified by msgid.
```

- 1. demo on message queue

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/msg.h>
```

```
#define MQ_KEY  0x4321
#define MSG_LEN 32

struct my_data {
    long type;
    char str[MSG_LEN];
};

int main() {
    int mqid, ret, s;

    // create message queue
    mqid = msgget(MQ_KEY, IPC_CREAT | 0600);
    if(mqid < 0) {
        perror("msgget() failed");
        _exit(1);
    }

    ret = fork();
    if(ret == 0) {
        // child -- sender
        struct my_data m1;
        m1.type = 123;
        printf("child: enter a string: ");
        scanf("%s", m1.str);
        printf("child: sending message ...\n");
        // send message in message queue (with type=123).
        msgsnd(mqid, &m1, MSG_LEN, 0);
    }
    else {
        // parent -- receiver
        struct my_data m2;
        printf("parent: waiting for child message.\n");
        // receive message from message queue (with type=123).
        msgrcv(mqid, &m2, MSG_LEN, 123, 0);
        printf("parent: received %s \n", m2.str);

        wait(&s);

        // delete message queue
        msgctl(mqid, IPC_RMID, NULL);
    }
    return 0;
}

//terminal> gcc msgque.c -o msgque.out
//terminal> ./msgque.out
```

## 6. PIPE

0. Unidirectional data transfer

- this is Stream Based, (i.e not predefined like message)
- Stream is a sequence of bytes, so no predefined object length
- Kernel maintains a buffer of fixed size (modern Linux default is 64 Kb), in which data is hold until,it is read by reader Process.
- pipe has two ends :
    - 1. writer end
    - 2. reader end

1. two types of pipes

- 1. Unnamed pipe (pipe)
    - communication between two related process (in same process group)
    - e.g : who | wc
    - created using pipe() syscall (system call)

```
# here , arr --> is out parameter, arr[0] -->reader end , arr[1] --->
writer end of pipe
   int arr[2];
     ret = pipe(arr);
  #  arr[0] -- file descriptor of reader end of pipe
  #  arr[1] -- file descriptor of writer end of pipe
```

- 1. demo for unnamed pipe

- for pipe, we have a pipe Object , which contains
- 1. *buf : a pointer to pipe buffer(i.e a circular queue),
- 2. read| front :
- 3. count :
- 4. waiting queue :

- whose data is referenced in inode table , with no data area

- pipe have read and write, so two entries in OFT(open file table),for read and wiite, having pointer to same inode

- in PCB, there is a OFDT(open file discriptor table), having standard entries, stdin(1),stdout(1), stderr(2), also

- reader file discriptor i.e end a pointer from OFT table(3), writer file discriptor(4),

- after creation of pipe , use fork() on it (that copies whole process,its PCB containing OFDT), this child process also have a reference to OFT, so now a pipe with four end is created (i.e child in parent pipe )

- so we must keep two ends, and close other 2 end,

- depend who sends data, close it reader ends , and someone wanna read, close writer end

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    int ret, s;
    int arr[2];

    // create pipe
    ret = pipe(arr);
    if(ret < 0) {
        perror("pipe() failed");
        _exit(1);
    }

    ret = fork();
    if(ret == 0) {
        // child -- sender
        char str1[32];
        printf("child: enter a string: ");
        scanf("%s", str1);
        printf("child: sending message ...\n");
        // send data in the pipe.
        write(arr[1], str1, sizeof(str1));

        // close pipe write end
        close(arr[1]);
    }
    else {
        // parent -- receiver
        char str2[32];
        printf("parent: waiting for child message.\n");
        // receive message from pipe.
        read(arr[0], str2, sizeof(str2));
        printf("parent: received %s", str2);

        wait(&s);

        // close pipe read end
        close(arr[0]);
    }
    return 0;
}
```

- 2. Named Pipe (fifo)
    - communication between any two proceeses
    - it is a special file denoted by (p) , created using mkfifo command/syscall
    - Writer proces open pipe in write mode and reader process open it in read node

        - File created on disk
        - Number of data blocks are always zero
    - Writer proces open pipe in write mode and reader process open it in read mode

- 0. create fifo file

```
mkfifo myfifo
```

- 1. wr_pipe.c

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

int main() {
    int fw;
    char str[40];

    printf("WR: opening fifo.\n");
    fw = open("myfifo", O_WRONLY);
    if(fw < 0) {
        perror("failed to open fifo");
        _exit(1);
    }

    printf("WR: enter a string: ");
    scanf("%s", str);

    write(fw, str, sizeof(str));
    printf("WR: data is sent.\n");

    close(fw);
    return 0;
}

//terminal> mkfifo ~/myfifo
//terminal> ls -l ~

//terminal> gcc -o wr_pipe.out wr_pipe.c
//terminal> ./wr_pipe.out
```

- 2. r_pipe.c

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

int main() {
    int fr;
    char str[40];

    printf("RD: opening fifo.\n");
    fr = open("myfifo", O_RDONLY);
    if(fr < 0) {
        perror("failed to open fifo");
        _exit(1);
    }

    printf("RD: waiting for data...\n");
    read(fr, str, sizeof(str));

    printf("RD: data is received: %s.\n", str);

    close(fr);
    return 0;
}

//terminal> gcc -o rd_pipe.out rd_pipe.c
//terminal> ./rd_pipe.out
```

-3. run

```
> ./wr_pipe.out
message
# we get messag here in rt
> ./rt_pipe.out
message
```

## 7. Socket

1. Socket is communication end point
2. socket developed by BSD UNIX
3. Socket is extension of pipe COncept

- but it is bi-directional

4. Socket has many types ( calledAddress families)

- 1. AF_UNIX
- 2. AF_INET
- 3. AF_BLUFTOUTH

5. AF_UNIX socket

- communication between two processes on same computer
- it has special file (s)

6. AF_INET (internet socket)

- communication betweeen two processes on same/ diff computers
- made up of inet socket = ip adress + port number

- it could be TCP or UDP socket , depending on underlying protocol
- in java for TCP class name Socket , Server Socket class, for UDP, datagram socket , and datagram packet

**Linux run levels**

1. user give services , based on levels/run level
2. linux run levels

-
- 0. shutdown pc --> command :

> sudo init 0

- 1. single user mode( minimum services )
- 2. multi user mode (no network in min services)
- 3. multi user mode + network
  - so server machine require this level
- 4. reserved (unused)
- 5. multi user + network + GUI
- 6. reboot pc :

3. to know current run level mode :

> runlevel

4. to change runlevel ,

> sudo init rn

- where rn : run level
- we can use to go to level 3

> sudo init 3

5. to see RAM usage on PC

> free -mh

# day7

## MCQ Q:

1. difference between Process andThread?

- in modern OS, process is like a container , that holds resources required for execution
- while thread,is a unit of execution and scheduling

2. MCQ : if Semaphore count is n ,then how many process waiting on process ?

- Answer: n

3. difference between mutex(M) and semaphore(S) ?

- 1. S is a counter so can be increment and decrement
- 2. M is a flag so can be locked and unlocked
- 3. S user for counting as well as a s binary S
- 4. M only for Mutual exclusion
- 5. M is lightweith than S
- 6. in S one process can decrement and another increment
- 7. M who lock is owner , and he can only unlock

- java syncronized block internally use Mutex
- S and M uses Spin lock hardware mechanism that is availabe in kernal space , so one process can access Semaphore at one time, so no race condition for semaphore

- test and set instruction used in Spin mechanism, to avoid race condition in S and M

4. similarly process with its segment, are divided in multiple pages

- in paging , internal fragmentation takes place ,as each page have segment of process and segment are stored in pages this is called **segmented paging**

## Multi Threading for MULTI TASKING

1. **Multi Threading**

- 1. need multiple processes, to execute multiple task concurrently , so creating multiple process for multi tasking
- 2. in thread based multi tasking , for each task one new thread stack and Tnread control block(TCB) is created in same process, while in process based multi tasking , for each task a new process is created ,so its using greater resources, as compared to thread based
- 3. so, **Thread is a Light- Weight Process**
  - i.e for each thread, there is stack section and TCB, but other sctions like text,data ,heap, etc are shared with parent process
  - so for accessing resources , in heap and data ,we dont need IPC, as its shared with threads, it is done as IPC makes tend to make System slower, so use thread for Inter Thread Communication , as far more eficient to IPC
  - thread,is a unit of execution and scheduling
- 4. each process has one default thread, called as main thread or primary thread

- check thread in a process, we use command

```
ps -e -m -o pid,tid,nlwp, cmd
```

- where -m : multi threading , nlwp : no of light weight process, tig: thread id
- 5. for each thread, a new stack for thread is created, used by main thread, here thread stack contains function activation record for that thread,
- any thread can create another thread, but convention is kept that their is no parent- child relation,
- so, all process created in a process , belong to that process

6. in LINUX, no concept of Process and Thread differentiated, they **use TASK**

- two types of taks
- 1. (thraead type)
- sharing sections with parent
- 2. (process type)
- all sections exclusive
- in linux creates, task_strut control blck (not TCB or PCB)

7. **to create a thread in linux,**

- 1. use syscall clone()
- use it based on parameter, create process and thread
- 2. library function
- in windows :win32 : CreateThread()
- in lInux/Unix/Mac: POSIX thread library (part of POSIX standand)
- POSIX :stands for Portable Operating System Interface for XWindows or XLinux
  - anything that is not windows, is Xwindows , i.e UNIX
  - it is UNIX standard
- here a library function pthread_create()

8. **thread creation is a two step process**

- 1. implement thread function / procedure
- 2. call thread creation function/ API
- in java , function executed by thread, run() method
- create a thread in ajava, thread.start

9. **pthread_create()**

> int pthread_create(pthread_t *thread, const pthread_attr_t *attr,void *(*start_routine) (void *), void *arg);

- here , parameters are :
- ( address of thread variable,attributes of a thread,address of function pointer, argument to function )
- where attibute: stack size,priority, scheduling algo,
- pthread_create - create a new thread

- The pthread_create() function starts a new thread in the calling process. The new thread starts execution by invoking start_routine(); arg is passed as the sole argument of start_routine().
- The new thread terminates in one of the following ways:

- It calls pthread_exit(3),
- It returns from start_routine().
- It is canceled ( pthread_cancel(3)).
- Any of the threads in the process calls exit(3), or the main thread performs a return from main(). This causes the termination of all threads in the process.

- On success, pthread_create() returns 0; on error, it returns an error number, and the contents of *thread are undefined.

- l is a GCC option for linking a library file.

- -lpthread in essence tells the GCC compiler that it must link the pthread library to the compiled executable.
- pthread or POSIX Threads is a standardized library for implementing threads in C

10. **demo on single thead creation**

- if only main function implemented, then a single thread is created i.e main thread , which can be seen using

> ps -e -m -o pid,tid,nlwp,cmd

- here the pid and tig for main thread is same , so this is single thread process

11. **pthead_t**

- internal structure defined, we use for thread creation pthread_t, same as pid_t

- this is an outparameter, thread id is stored in it, after thread creation using pthread_create()

- to compile this program, need a flag -lpthread, as pthread_create in not part of std library

- it is part of posix thread library

> gcc -o thread1.out thread1.c -lpthread ./thread1.out

- to check thread created

> ps -e -m -o pid,tid,nlwp,cmd

- to see process segment

> cat /proc/pid/maps

- this show concurrent execution, thread 1, 2 and main required 60 second each, so total 180 s if run separately, but using multi threading the function run concurrently , so only 60 s requires

- .Net ,create thread , thread start delegate (function pointer)

- java: thread.start

- main thread get terminated, process with thread in process get terminated

- in java same is deamon thread ,
- in .NET , background thread

12. **demo on multi thread creation**

- 1. demo to create thread

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
// fixed signature for thread function, void*
void* first_thread_func(void *param) {
    int i;
    for(i=1; i<=60; i++) {
        printf("first: %d\n", i);
        sleep(1);
    }
    return  0;
}

void* second_thread_func(void *param) {
    int i;
    for(i=1; i<=60; i++) {
        printf("second: %d\n", i);
        sleep(1);
    }
    return  0;
}

int main() {
    // internal structure defined, we use for thread creation pthread_t,
same as pid_t
    // this is an outparameter, as thread id is stored in it, after thread
creation
    pthread_t t1, t2;
    int i, ret;
    // parameters are :( address of thread variable,attributes of a
thread,address of function pointer, argument to function )
         // attibute: stack size,priority, scheduling algo,
    ret = pthread_create(&t1, NULL, first_thread_func, NULL);
    ret = pthread_create(&t2, NULL, second_thread_func, NULL);
    for(i=1; i<=60; i++) {
        printf("main: %d\n", i);
        sleep(1);
    }
    return 0;
}


// terminal> gcc thread1.c -o thread.out -lpthread
// terminal> ./thread.out
// terminal2> ps -e -m -o pid,tid,nlwp,cmd
// terminal2> cat /proc/pid/maps
```

- 2. demo to print table using thread

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

void* print_table(void *param) {
    long i, num = (long)param;
    for(i=1; i<=10; i++) {
        printf("%ld * %ld = %ld\n", num, i, num * i);
        sleep(1);
    }
    return 0;
}

int main() {
    pthread_t t1, t2, t3;
    pthread_create(&t1, NULL, print_table, (void*)10);
    pthread_create(&t2, NULL, print_table, (void*)20);
    pthread_create(&t3, NULL, print_table, (void*)30);
    printf("press enter to exit...\n");
    getchar();
    printf("bye!\n");
    return 0;
}
```

- 3. demo using pthread_join so

- calling thread i.e. main will wait for completion of t1 thread. and will collect its return values, as 2nd argument

- pthread_join - join with a terminated thread

> int pthread_join(pthread_t thread, void **retval);

- The pthread_join() function waits for the thread specified by thread to terminate.
- If that thread has already terminated,then pthread_join() returns immediately.
- On success, pthread_join() returns 0; on error, it returns an error number.
- pthread_join() will collect return value of given thread in its arg2 (out param).

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

void* print_table(void *param) {
    long i, num = (long)param;
    for(i=1; i<=10; i++) {
        printf("%ld * %ld = %ld\n", num, i, num * i);
        sleep(1);
    }
    return 0;
}

int main() {
    pthread_t t1, t2, t3;
    pthread_create(&t1, NULL, print_table, (void*)10);
    pthread_create(&t2, NULL, print_table, (void*)20);
    pthread_create(&t3, NULL, print_table, (void*)30);
    // calling thread i.e. main will wait for completion of t1 thread. and
will collect its return values, as 2nd argument
    pthread_join(t1, NULL);
    // calling thread i.e. main will wait for completion of t2 thread.
    pthread_join(t2, NULL);
    // calling thread i.e. main will wait for completion of t3 thread.
    pthread_join(t3, NULL);
    printf("bye!\n");
    return 0;
}

/*
pthread_join() will collect return value of given thread in its arg2 (out
param).
*/
```

13. **Thread termination / deletion**

- 
- When thread function is completed, thread is terminated
- A thread can exit using pthread_exit() call (in its thread function)

```
void* thread_func(void *) {
        // ...
        if(condition)
            pthread_exit(res);
        // ...
        return res;
```

- When process is terminated, all threads in that process are terminated
- A thread termination request can done using pthread_cancel() from any other thread.

14. **race condition**

- known and founded by ,Peterson's problem
- when race condition, result that comes is corrupted or unexpected

- so to avoid this race condition need synchronization
- synchronization i.e Allow or block processes/thread , so that resource can be accessed safely

1. demo on race condition

- in below done code, output shows, that code output changes, as we run the program multiple time,
- so to avoid this we need threas
- below code, variable declared globally, i.e data section , shared by threads, so both thread access same variable, and increment and decrement it,
- so output

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

int count = 0;

void* incr_thread_func(void *param) {
   int i;
   for(i=1; i<=10; i++) {
    count++;
    printf("incr: %d\n", count);
    sleep(1);
   }
   return  0;
}

void* decr_thread_func(void *param) {
   int i;
   for(i=1; i<=10; i++) {
    count--;
    printf("decr: %d\n", count);
    sleep(1);
   }
   return  0;
}

int main() {
   pthread_t t1, t2;
   int i, ret;
   ret = pthread_create(&t1, NULL, incr_thread_func, NULL);
   ret = pthread_create(&t2, NULL, decr_thread_func, NULL);
   pthread_join(t1, NULL);
   pthread_join(t2, NULL);
```

```
    printf("final count: %d\n", count);
    return 0;
}


// terminal> gcc thread4.c -o thread.out -lpthread
// terminal> ./thread.out
```

## Linux Mechanism of Synchronization

1. Semaphore

2. Mutex

3. Condition variable (event in windows)

4. **Semaphore**

- given by Dijkstra .

- simaphore is a counter ,
- having two operations :
- 1. increase operation (V)

- increase semaphore count
- icrease and if one or more process are blocked on semaphore , wake up one of the process

- 2. decrement operation (P)

- decrease Semaphore count
- if count becomes less than zero, block calling process

- two types of Semaphore

- 1. Counting S
- 2. BInary S
    - i.e count between 0 to 1

- **Application of Semaphore**

1. **semaphore as a flag /event**

- one process increment and other process decrement, semaphore, so s is a flag

2. **Semaphore for mutial exclusion**

- o for one process and 1 for another

3. **Semaphore for counting**

- here a circular-queue through which data is taken,

- one(producer) for push/upload data to queue, one(consumer) for pop/ downlaod data
- so work of semaphore, is push into buffer(queue) and pop from buffer
- there is producer consumer problem, bounded buffer problem
- use binary s and mutex

5. **Mutex**

- it is also known as Mutual Exclusion
- mutex is lightweight than binary semaphore
- mutex has two states

- 1. lock

- locking process is owner of mutex and only owner can unlock

- 2. unlock

- - if mutex is not there, no option but to use binary semaphore

- counting semaphore

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

pthread_mutex_t m;
int count = 0;

void* incr_thread_func(void *param) {
    int i;
    for(i=1; i<=10; i++) {
        // lock the mutex
        pthread_mutex_lock(&m);
        count++;
        printf("incr: %d\n", count);
        // unlock the mutex
        pthread_mutex_unlock(&m);
        sleep(1);
    }
    return  0;
}

void* decr_thread_func(void *param) {
    int i;
    for(i=1; i<=10; i++) {
        // lock the mutex
        pthread_mutex_lock(&m);
        count--;
        printf("decr: %d\n", count);
        // unlock the mutex
        pthread_mutex_unlock(&m);
        sleep(1);
    }
    return  0;
}

int main() {
    pthread_t t1, t2;
    int i, ret;
    // create mutex (with default attribs=NULL)
    pthread_mutex_init(&m, NULL);
    ret = pthread_create(&t1, NULL, incr_thread_func, NULL);
    ret = pthread_create(&t2, NULL, decr_thread_func, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("final count: %d\n", count);
    // destroy mutex
    pthread_mutex_destroy(&m);
    return 0;
}


// terminal> gcc thread4.c -o thread.out -lpthread
// terminal> ./thread.out
```

6. need Concurrent execution

- 1. Process based Multi tasking
- for this create new process, for process based multi tasking ,
- here process communication and race condition is a problem
- solution is use IPC mechanism, but when we use it
- using IPC , system performance get slow
- 2. Thread based multi tasking, is better option
    - here problem is race condition
    - solution for it,synchronization (S), but S has its own problem
    - the problem caused by S, is deadlock
    - i.e, wrong use of S, causes deadlock

7. DeadLock

- 1. **DeadLock Characteristics (DC) are**
- 1. No preemption
- process, not terminating on call

- 2. Mutual Exclusion
- locking of resource is done
- 3. Hold and wait
- resources are occupied by one process and waiting for completion
- 4. Circular wait
- multiple process in a circualr fashion , waiting for one resource,

- if all four D C occur , then its deadlock
- 2. Deadlock vs Starvation

1. Starvation ,

- process not getting CPU / IO due to low priority of process
- Process is in ready state

2. Deadlock

- process not getting resource because blocked by another process (due to deadlock characteristic)

- process is in blocked state

- 3. solution for Deadlock

- no solution for deadlock, so prevention
- so come out od D, either forcefuly takes resources ,or kill process holding resource
- in Prevention two ways

1. Deadlock Prevention

- makes system such a way, that all 4 deadlock characteristics doesnt hold

- sys V(Vips): unix version 5, it says occupy all resources, so no hold and wait

2. Deadlock Avoidence

- inform OS, about resources needed, and based on this, OS provides them,
- i.e divide in two steps: Resource request, Resource allocation
- three Algo for D A :
- 1. Safe state (check if safe to give or not)
- 2. Resource Allocation Graph( checks for circular wait)
- 3. Banker's Algo (you know what needed, )
    - developed by Edsger Dijkstra that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources.

# Memory Management

1. Multi Programming:

- loading multiple program in memory

2. how a program is loaded and address given in RAM?

- using Memory Management Unit
- when program is loaded in memory by loader, program has virtual address given by compiler ,linker
- ther are relocated and added to RAM, these address are now known as physical address
- CPU always execute a process, in it virtual address(VA) space
- when Cpu ---> VA
- it goes thorough limit () i.e VA shoul be less than (P A space)
- limit gives greater than limit, due to out of range , as dangling pointer(causing segmentation error)
- if in range of limit, then ---> base (address value is added to VA )
- this is now your PA of variable
- this process is done by MMU (Memory Management Unit)
- steps of MMU : CPU ---> (VA) ---> Limit ---> Base [VA + Base value]--- (PA)
- this all info about limit,base is stored in PCB for each process ,so base and limit different for each process

3. physical Address to process will remain same, or may change?

- changes, in case od swap operation

4. if ram filled , still process to load, so use?

- virtual memory / swap area, it is in HDD
- as all process are not active at same time, so we can swap inactive process for the time
- Steps in this:
    - 1. swap out
        - i.e taking inactive process out of ram to swap area is called swap area
    - 2. swap- in
        - getting a process back in to the memory, due to space being available/ or need
- so this process gets new address
- portion of a disk, i.e swap area is used as extension of memory , so it is called virtual memory
- here ,multiple process can be swap out

5. schemas of Memory Management based on what hardware used, and which MMU it uses:

- there are 3, based on type of MMU used :

- 1. Contiguous allocation
    - for simple MMU
- 2. Segmentation
    - Segmentation MMU
- 3. Paging
    - Paging MMU (TLV)

## 6. **for Contiguous Allocation**

- two methods are there :

- 1. Fixed position method
    - make fixed partition of a 3Mb size
    - one partition per process
    - some small process has free space, so cause internal fragmentation

  - here maximum size and no of process are limited
- 2. Variable partition method
  - per process allote memory
  - issue, when process are getting terminated
  - now for the space of terminated process ,with their address , to maintain this info , OS maintain a table,
  - table contain base and limit, for free space
  - so now if new process cames , we can put the process according to
  - 1. first fit
      - the first available space
      - faster performance
  - 2. best fit
      - occupy minimum memory partition
      - better utilization
  - 3. worst fit
      - occupy maximum memory in partition
  - **Issue and solution with CA**
  - in this case, **external fragmentation**,i.e memory available, but its not contiguous
  - solution for this, shifting all process, so max contiguous memory avaialbe , is called **Compaction**
  - but in compaction, temporarily system stop as process is shifting ,
  - so better to look contguous memory for section of a process, this reduces external Fragmentation, and this is called Segmentation
  - these are similar in Segmentation
  - now a days , Continuous Memory Management not there, as many limitation are there , only one benifit of Cont memory

## 7. Segmentation

- 1. **Define**

- here divide process into section and based on it, allocate memory

- here we look contiguous memory for section of a process rather than process itself, this reduces external Fragmentation, and this is called Segmentation

- 2. **Segmentation MMU**

- so CPU should hold, segment and virtual address for each process

- here we have a segment table, which holds for each segment,having a different limit and base

- so similar to MMU work happens,

- here we first look a segment no in segment table and get base and limit , and then use Limit and base and get Physical Address

- here is a Segment Table Base Register, (STBR ) points to current process Segment table , and loads seg MMU table contents

- here External Fragmentation is minimized , due to smaller sector are stored,
- 3. here in case of **Swapping**
- we can swap segment, in and out of memory
- and when we load segment in memory from swap area ,i.e swap in happens, when required ,this is called Demand Segmentation
- 4. **Difference**
    - 1. Segmentation is memory manage algorithm
    - 2. Demand Segmentation is virtual memory mana algorithm
- 5. **segment table** , each table contains entries of:
    - 1. base and limit
    - 2. permisiions for section (rw)
    - 3. validity bit (segment in ram : valid, and in swap area: invalid)
    - 4. dirty bit
- S used in Windows (S + paging)

## 8. **Paging**

- Linux used Paging, i.e modern Memory Management

1. concept

- 1. here ram is divided in small and equal size partition called as **frames** or physical pages
    - size of frames are paging MMU specific, linux, windows : 4kb size
- 2. similarly process with its segment, are divided in multiple **pages**
    - in paging , internal fragmentation takes place ,as each page have segment of process and segment are stored in pages this is called **segmented paging**
    - smaller the page size, lesser is internal fragmentation
- 3. pages are allocated one page per frame ,so now to keep track of them, we have a **page table**, i.e a table that keeps track of which page allocated to which frame i.e contains mapping of logical address(page) to Physical address(frames) in ram
- **page size should be equal to frame size**

2. Paging MMU

- CPU always gives logical address(i,e page address), with page table page entry,now
- ---> using paging table ,we get frame no, corresponding to page,and using base(offset) we get Physical adress (frame)
- as process have huge memories, so here we have **Page Table Base Register(PTBR)**, to maintain address/ access of/to the whole Page table, which Page MMU connect to Ram page table
- now if process want to grow, then pages in RAM grow, so page table entry increases,
- but if RAM is full, then we swap, Pages , they are called
- 1. paging out :--> swapping out of page to swap area/ Virtual memory area

- now the process want to grow , we can allocate frame to it
- 2. paging in : --> taking page from swap area to RAM, on cpu request, so also called **Demand Paging**

- here one page table entry contains

- 1. frame address
- 2. permission(rw)
- 3. validity bit (valid in ram )
- 4. dirty bit

- to keep trace

3. two level paging

- to keep track of paging table containing page entries of pages/frames scattered all over RAM ,as it requires certain memory which are kept as pages in RAM frames, we have a table or many table
- here 1st table is called to Primary PT , pointing to Secondary PT , which points to frames In RAM
- now, Virtual Address divided into 3 parts : p1, p2 and d
  - where p1: index to primary P T (of 10bytes) (if one page 4kb one page entry 4 bytes)
- p2: entries in secondary P T (pointing to address of frame) (10bytes size)
- d: no of bits required to access frame memory address (12 bytes)
- here max memory access = 2^ 32 ==> 4gb RAM access (for 32 bit OS)
- ao 64 bit OS to use => so can access
- **MMU is part of CPU chip** , pae(physical address extension )

4. Issues now, in context switch and solution

- to load page table in MMU using PTBR , will take a large amount of time
- so in Paging MMU, we contain a Hardware based , Hash Table, which is high speed associative cache memory called as **TLB (Translation Look aside Buffer)**,
  - its cache memory, as no of entries limited , 32 or 64, latest entry older deleted
  - associative : as hash table, key -value pair
- here using PTBR access main page table, i.e 2 level paging , and loads it in paging MMU table.

- now we use 3 level paging, like Linux( 2^64)

# day8

## mcq

1. (pafe fault order)

- **order of Page Fault Handling**:

1. **validity fault**(dangling pointer case)

- check if VA(Virtual Address) is in process VAS(virtual Address Space) (in some segment),
- if not send SIGSEGV (abort process)

2. **protection fault**

- check if have appropriate permission on the requested virutal address
- if not send SIGSEGV (abort process)

2.

```
# include <stdio.h>

 int main()
 {
 // "" string went to RO data section
 char *p = "SUN";
 // this write is protection fault, so runtime error
 *p = 'F';
  puts(p);
  // cout <<p
  return 0;
  } // output : segment dump : runtime error
```

3

## notes

1. Demanding page

-

-

> ps -e -o pid,vsz,rsz,cmd

- to track changes

> watch "free -mh"

- with malloc , we get contiguous memory, in virtual address/ memory, not physical pages address, after we write to that memory,i.e by OS, it get physical memory
- so, malloc allocates virtual memory, physical memory allocated by OS, on write
- for address space of 32 bit

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

int main() {

unsigned long i,size = 1 * 1024 * 1024 * 1024;
char *p = (char*) malloc(size);
printf("addr: %p \n",p);
getchar();
```

```
    for(i=0;i<size;i = i + 4096) {

    *(p + i) = 'A';
    }
    printf("physical mem allocated \n " );
    getchar();
    free(p);

    return 0;
    }
```

> gcc file.c -m32 -o file.out

- malloc allocates memory based on virtual memory space(VA)
- in Page table Entry (PTE) -validay bit values
- 1 ---> page in ram
- 0 ---> page not in RAM
- when page not in ram

1. page is swapped out
2. page in not yet loaded
3. physical page is not yet allocated (dynamic mem alloc done, OS not allocated physical address, that is accessed)
4. page is not part of process (used X for it)(so if try to access that page, leads to dangling pointer error(segmentation error))
5. when Cpu request a page and if Page Table Entry(if invalid) i.e page is not in RAM (due to one of the 4 reason), is called Page Fault

- when Page Fault happens, page fault exception handler (OS) is executed

**MCQ (pafe fault order)**

- **order of Page Fault Handling**:

1. **validity fault**(dangling pointer case)

- check if VA(Virtual Address) is in process VAS(virtual Address Space) (in some segment),
- if not send SIGSEGV (abort process)

2. **protection fault**

- check if have appropriate permission on the requested virutal address
- if not send SIGSEGV (abort process)

**MCQ**

```
# include <stdio.h>

 int main()
 {
 // "" string went to RO data section
 char *p = "SUN";
 // this write is protection fault, so runtime error
 *p = 'F';
  puts(p);
  // cout <<p
  return 0;
  } // output : segment dump : runtime error
```

3. allocate an empty frame for the page

- if none of the frame is empty, then swap out some page(victim page) and make that frame available for current page
- the process of deciding victim page, is called **page Replacement Algorithm**(PRA),
    - popular PRA are * 1. FIFO (first come) * 2. Optimal (non needed in near future) * 3. LRU (least recently used)

4. if page is on disk/swap, load it in the allocated frame
5. update PTE --> frame address, valid bit = 1
6. restart the instruction at which page fault occured

## page Replacement Algorithm

1. no of page fault, depend on PRA

- more RAM more pages kept, less page fault
- more page fault, lesser the performance

2. types of PRA

- 1. **in FIFO**
    - first in first out
    - IF no of frames increases,no of page fault increases, this is called Belady's Anomaly
    - so not used commonly
  2. **Optimal PRA**
    - here page not needed in near future, is replaced
    - in this how to know future pages ?
    - this algorith gives minimum no of page fault on paper/ theoritically, this algo cannot be implemented practically
    - so this is a ideal Algo, benchmark for other Algo
  3. **Least Recently Used** (LRU)
    - it gives better result
    - time complexity of LRU is O(n), i.e go through all pages
    - have limitation

4. Approximate LRU Algorithm, called **Second chance LRU Algo**\
   - variation of LRU
   - it has time complexity of O(1)
   - used in Linux, Mac

- to see page fault

> ps -e -o pid,vsz,rsz,maj_flt,min_flt,cmd

- disk I/O involved : major fault (maj_flt)
- no disk I/O involved : minor fault(min_flt)
- here vsz : virtual size
- rsz: real size

- Advantages of virtual memory :

1. total process size can be greater than RAM (process size > RAM )
2. can run more processeses than RAM size

- Thrashing

- disk IO (swapping) > process execution , overall system is slower
- solution is increase the RAM

- linux to create swap file

> mkswap

## paging Optimization technique

1. dirty Bit

- in Page Table Entry(PTE) we have dirty bit(d)
- it is zero, when page is loaded from disk or swap
- if d = 0,
  - page is not modified ,since loaded
  - copy of page in RAM is same as copy on disk
- it is used to check if page copy on RAM and disk are same
- if same we no need to copy from RAm to disk , in this case
- id d > 0
  - when write operation is done on page,next time page is to be swapped out
  - it must be write on disk

2. vfork

- virtual fork
- in process if we use fork, and exec then , a child process created, and with exec a new process overwritten,
- but if child process memory is less, so exec relaeses child memory, and allocates memory for process,
- so much time is wasted here ,so
- vfork invented by BSD UNIX,

- vfork is to be used, only when , we call exec() after fork()
- vfork ,says new process allocated, creates logically child/copy created,with logical PCB, by parent and wait for exec()
- now after vfork(), exec() comes,now exec() allocates a new memory and add process to it, so now
- physical child and PCB created.
- heree ,afer exec(), parent process goes back to its work

3. copy-on-write

- in case of vfork done, and exec() not called, gives unexpected result,
- so fork was created with copy-on-write
- here o nfork(), logical copy is created,
- i.e initially parent and child share same pages
- now , child wants to write the pages shared, a pysical copy of pages are created, and that address updated to page table, so actual changes to new page, this is called copy on write .
- so only page copied , which was being written, not whole process copied
- here , things are faster ,as less size page to be copied,
    - and text ,RO-data segment,are Read Only , so no need to copy them, so is faster
- no change in syntax

> man vfork

- lazy copy concept and copy on write concept same, implemention on OS, is copy on write

# # OS : DAY-01 QUIZ

Q. Computer is a ____________ which performs different functions efficiently and accurately.
A. hardware
B. machine
C. digital device
D. all of the above
**Answer: D**

Q. Which of the following is not a hardware?
A. Processor
B. Keyboard
C. Device Driver
D. Magnetic Disk
E. None of the above
**Answer: C**

Q. An OS is a _________
A. system software
B. resource manager
C. resource allocator
D. all of the above
**Answer: D**

Q. What are the functions of computer?
A. data storage
B. data processing
C. data movement
D. control
E. all of the above
**Answer: E**

Q. Which of the following is a system program?
A. compiler
B. linker
C. loader
D. assembler
E. all of the above
F. none of the above
**Answer: C**

Q. _______ converts high level programming language code into a low level programming language code.
A. An assembler
B. Compiler
C. Preprocessor
D. Linker
**Answer: B**

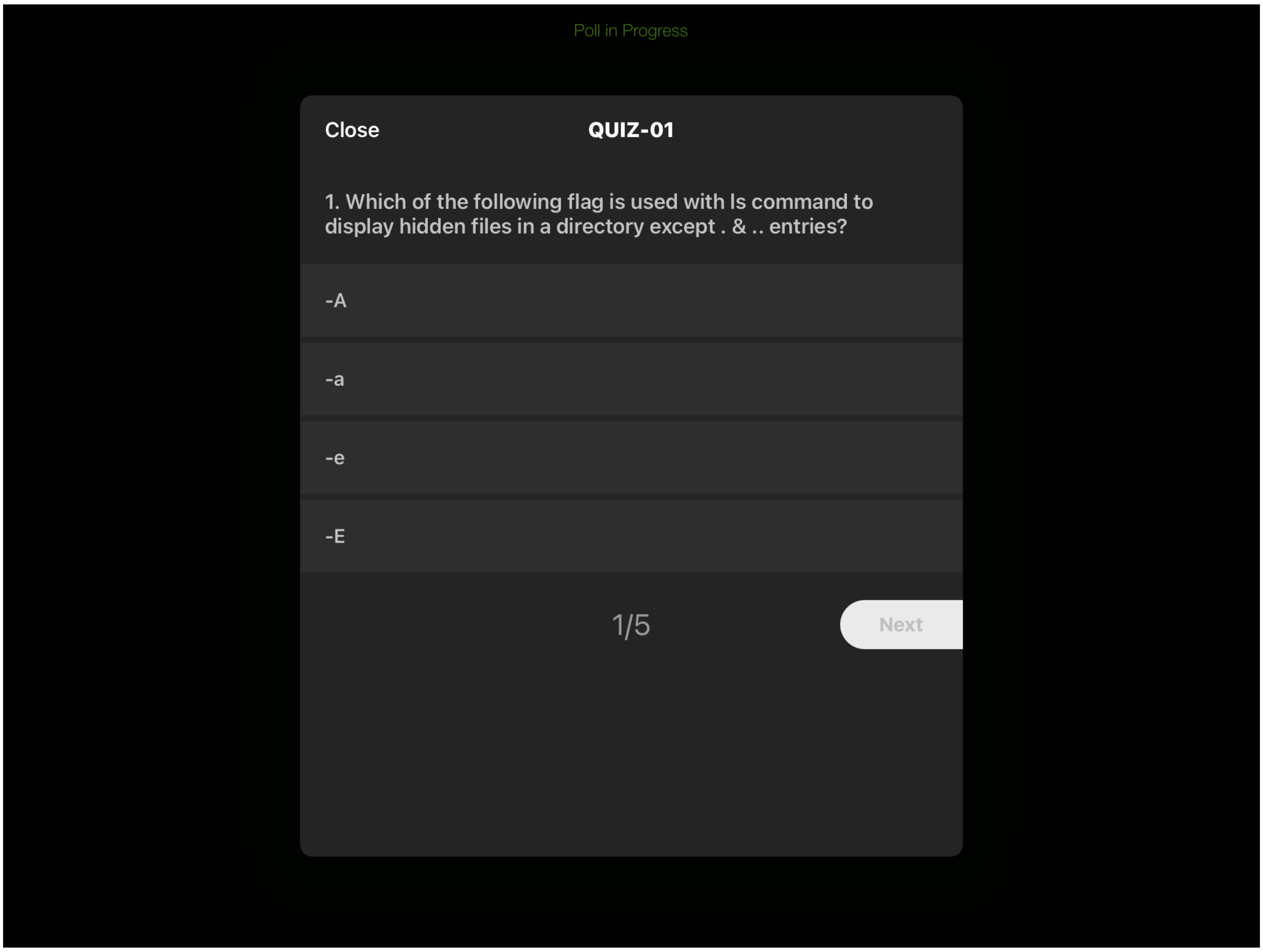
Q. Output of the linker is ______________
A. an object code
B. an executable code
C. an intermediate code
D. an assembly language code
**Answer: B**

Q. Which of the following program provides graphical user interface in Windows Operating System?
A. cmd.exe
B. explorer.exe
C. command.com
D. all of the above
E. none of the above
**Answer: B**

Q. Which of the following program is a system program?
A. Interrupt Handler
B. Device Driver
C. Loader
D. All of the above
E. None of the above
**Answer: D**

Q. Which of the following is a process?
A. program.i
B. program.o
C. program.s
D. program.out
E. None of the above
F. All of the above
**Answer: E**

# OS MCQ



Poll in Progress

Close QUIZ-01

1. Which of the following flag is used with ls command to display hidden files in a directory except . & .. entries?

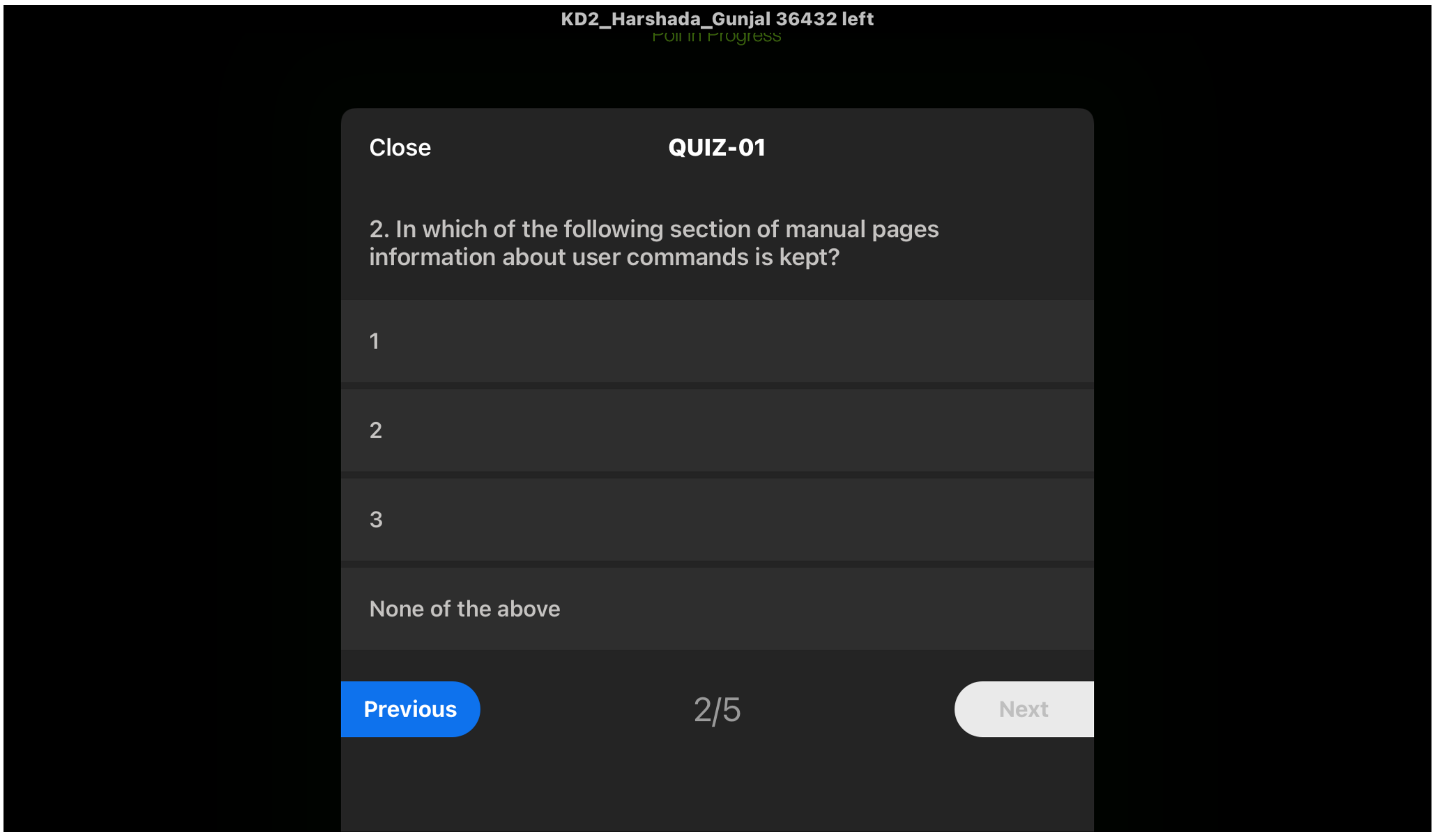
-A

-a

-e

-E

1/5 Next



KD2_Harshada_Gunjal 36432 left

Poll in Progress

Close QUIZ-01

2. In which of the following section of manual pages information about user commands is kept?

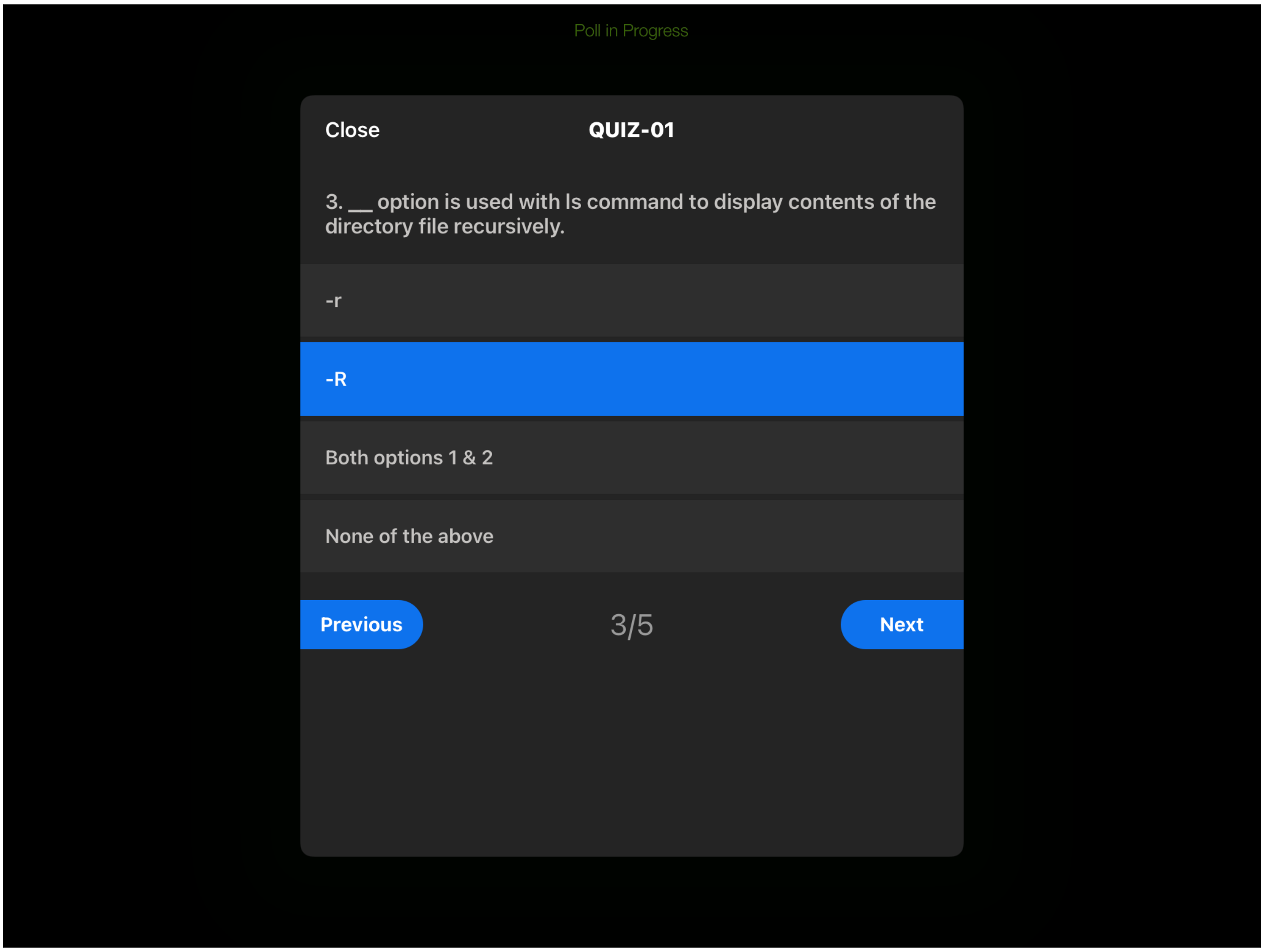
1

2

3

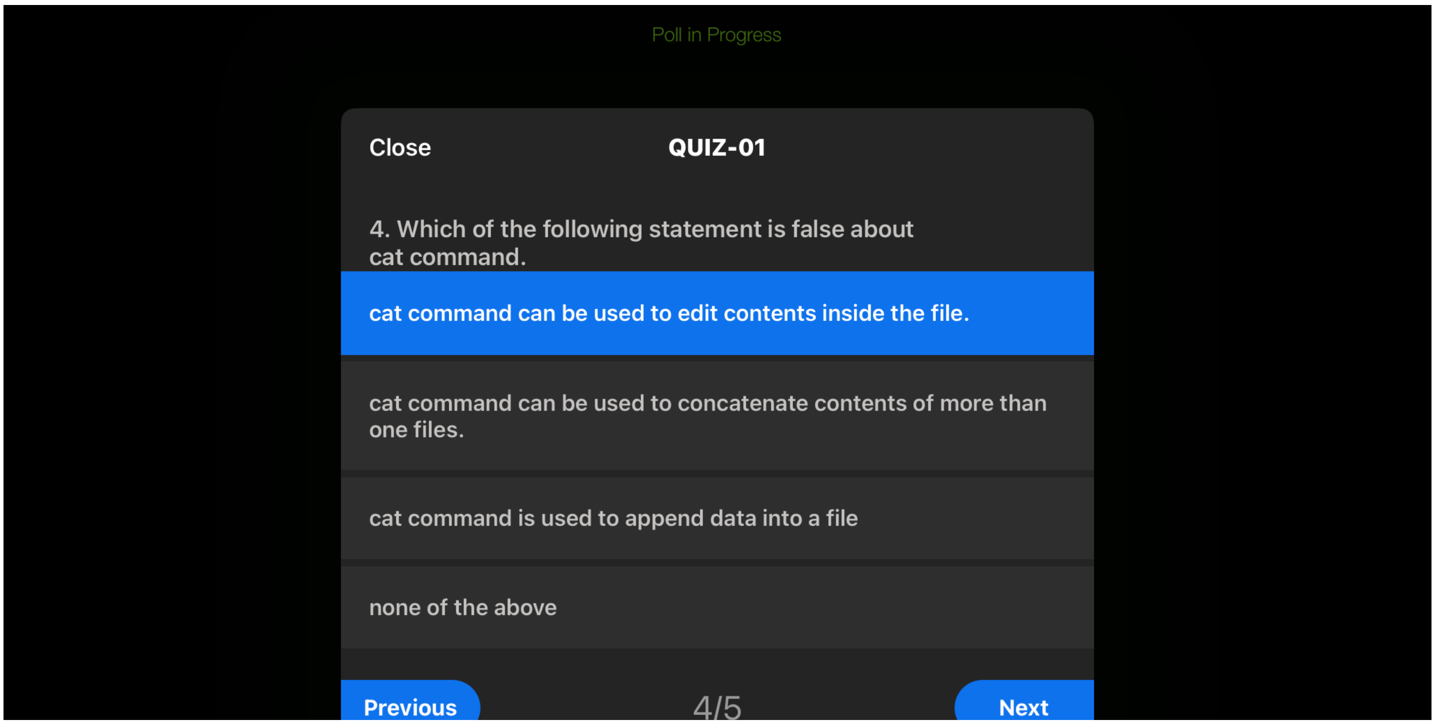
None of the above

Previous 2/5 Next



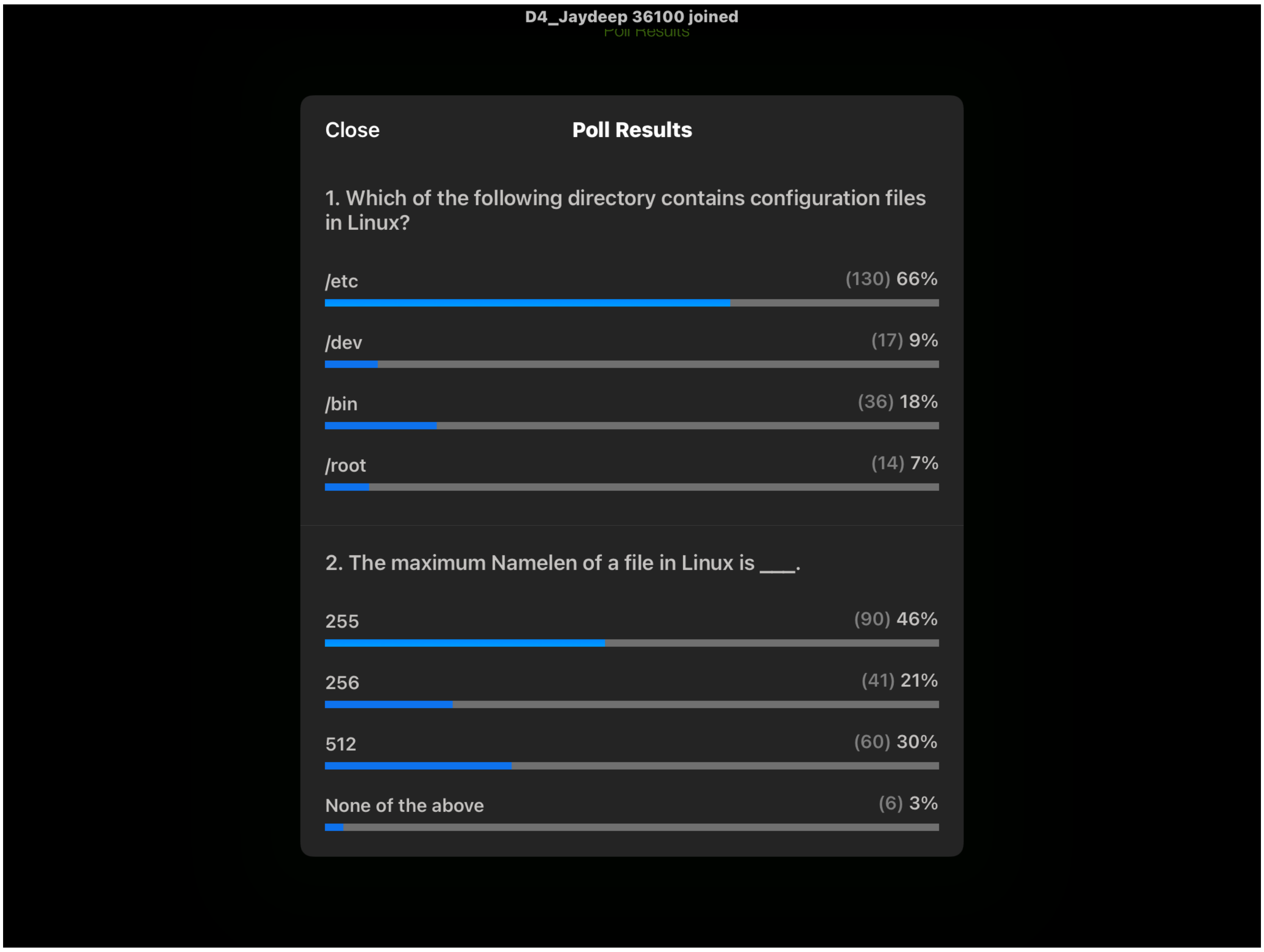
3. __ option is used with ls command to display contents of the directory file recursively.

- -r
- [✓ -R]
- Both options 1 & 2
- None of the above



4. Which of the following statement is false about cat command.

- [✓ cat command can be used to edit contents inside the file.]
- cat command can be used to concatenate contents of more than one files.
- cat command is used to append data into a file
- none of the above


D4_Jaydeep 36100 joined

Poll Results

Close Poll Results

1. Which of the following directory contains configuration files in Linux?

/etc (130) 66%

/dev (17) 9%

/bin (36) 18%

/root (14) 7%

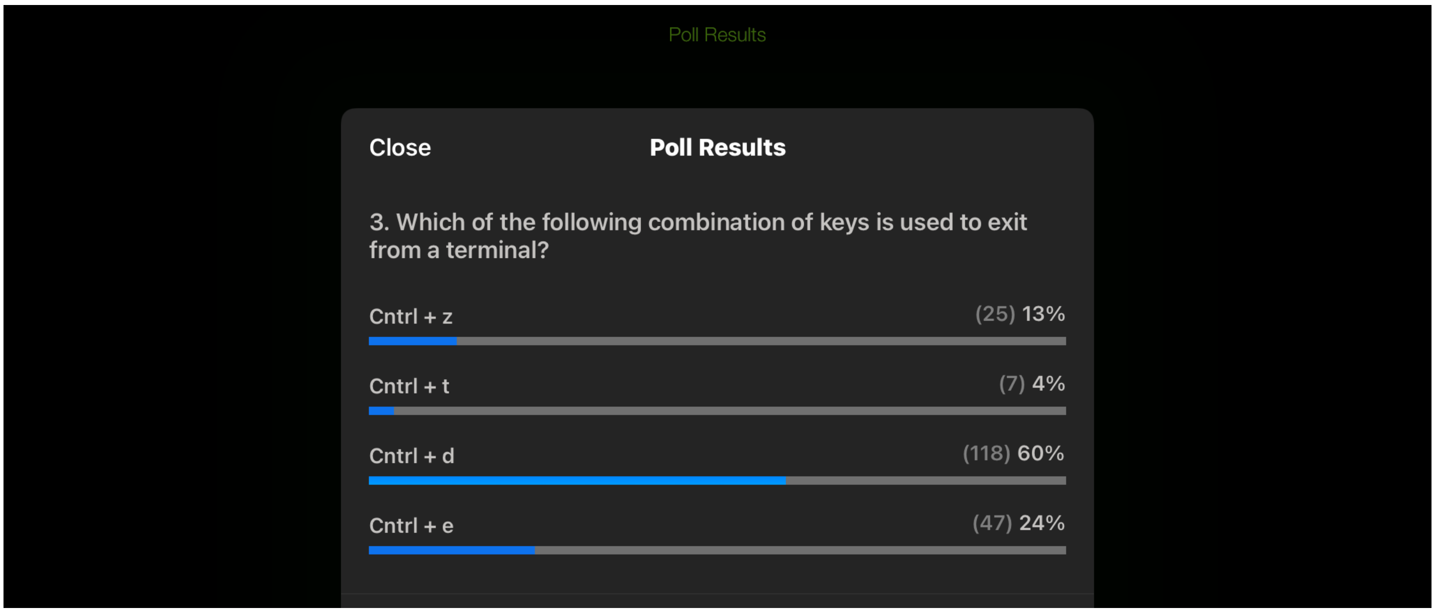
2. The maximum Namelen of a file in Linux is ____.

255 (90) 46%

256 (41) 21%

512 (60) 30%

None of the above (6) 3%


Poll Results

Close Poll Results

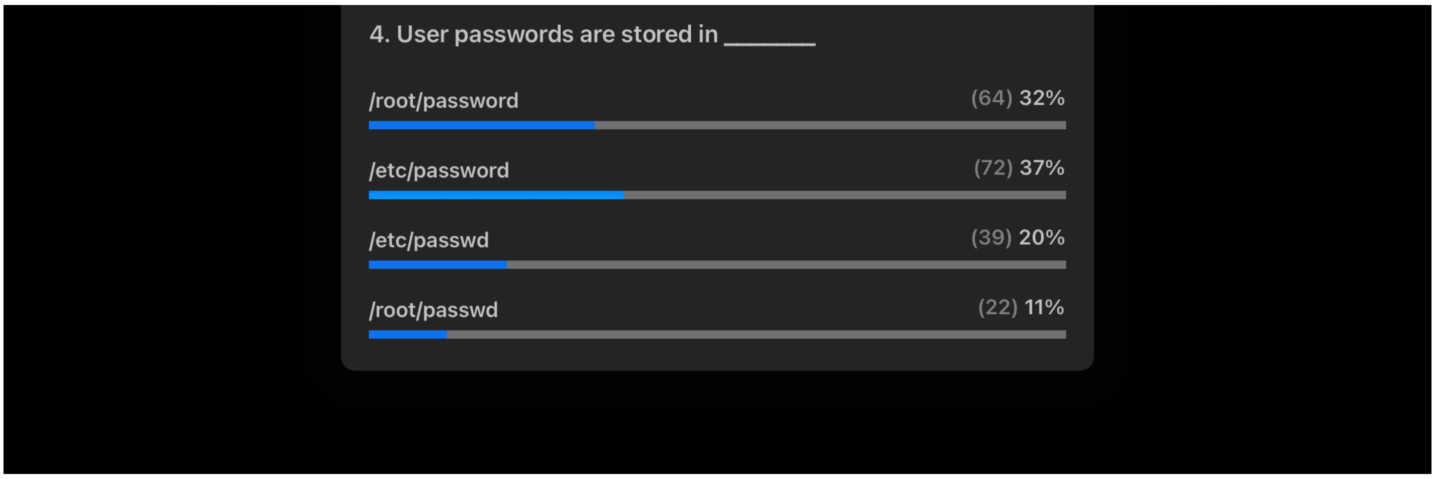
3. Which of the following combination of keys is used to exit from a terminal?

Cntrl + z (25) 13%

Cntrl + t (7) 4%

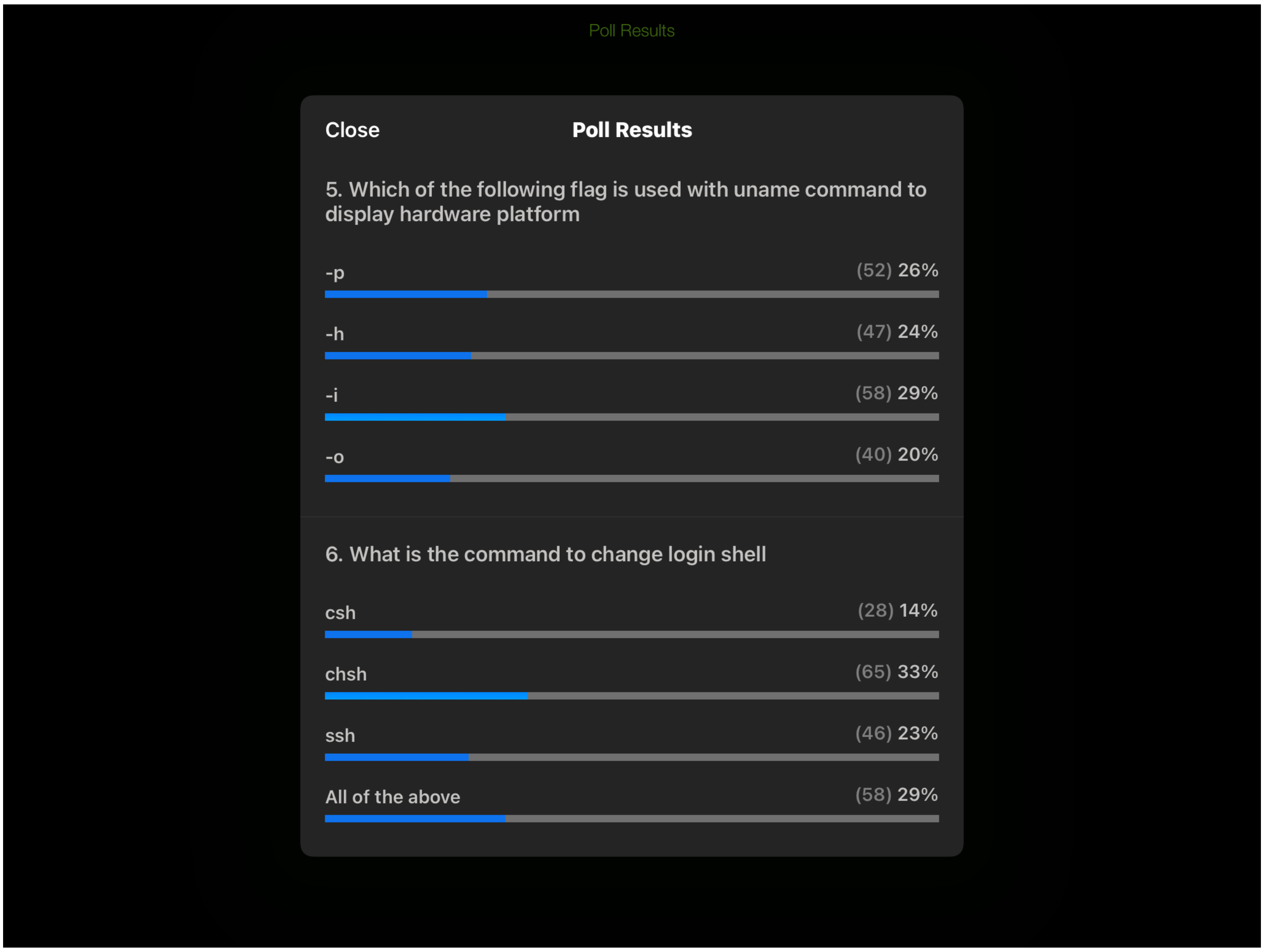
Cntrl + d (118) 60%

Cntrl + e (47) 24%



4. User passwords are stored in ________

| | |
|---|---|
| /root/password | (64) 32% |
| /etc/password | (72) 37% |
| /etc/passwd | (39) 20% |
| /root/passwd | (22) 11% |

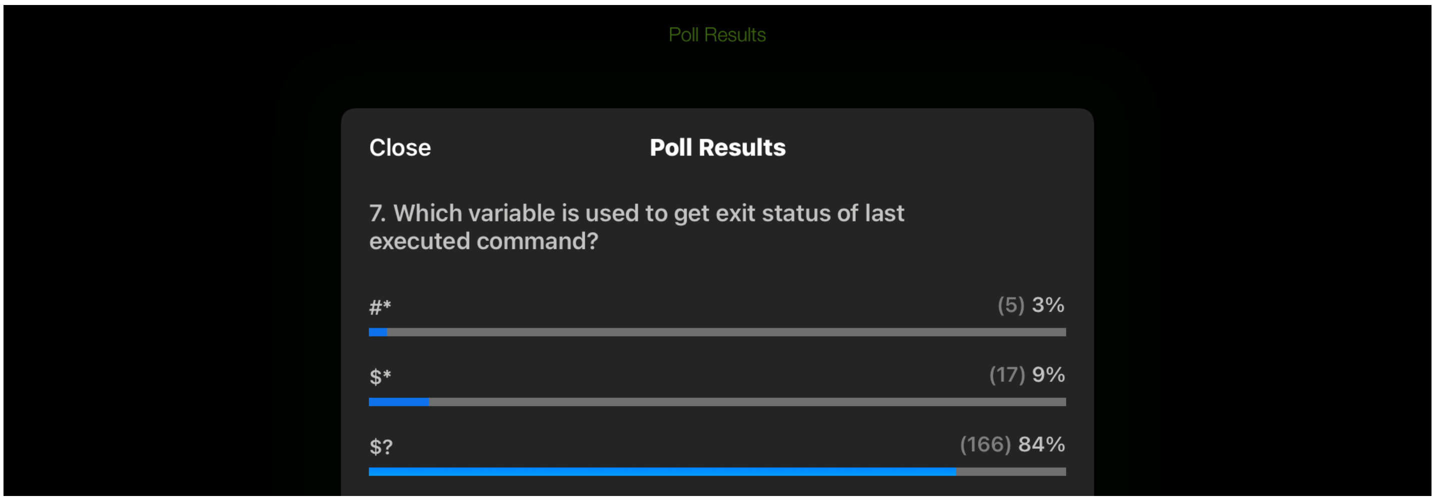


5. Which of the following flag is used with uname command to display hardware platform

| | |
|---|---|
| -p | (52) 26% |
| -h | (47) 24% |
| -i | (58) 29% |
| -o | (40) 20% |

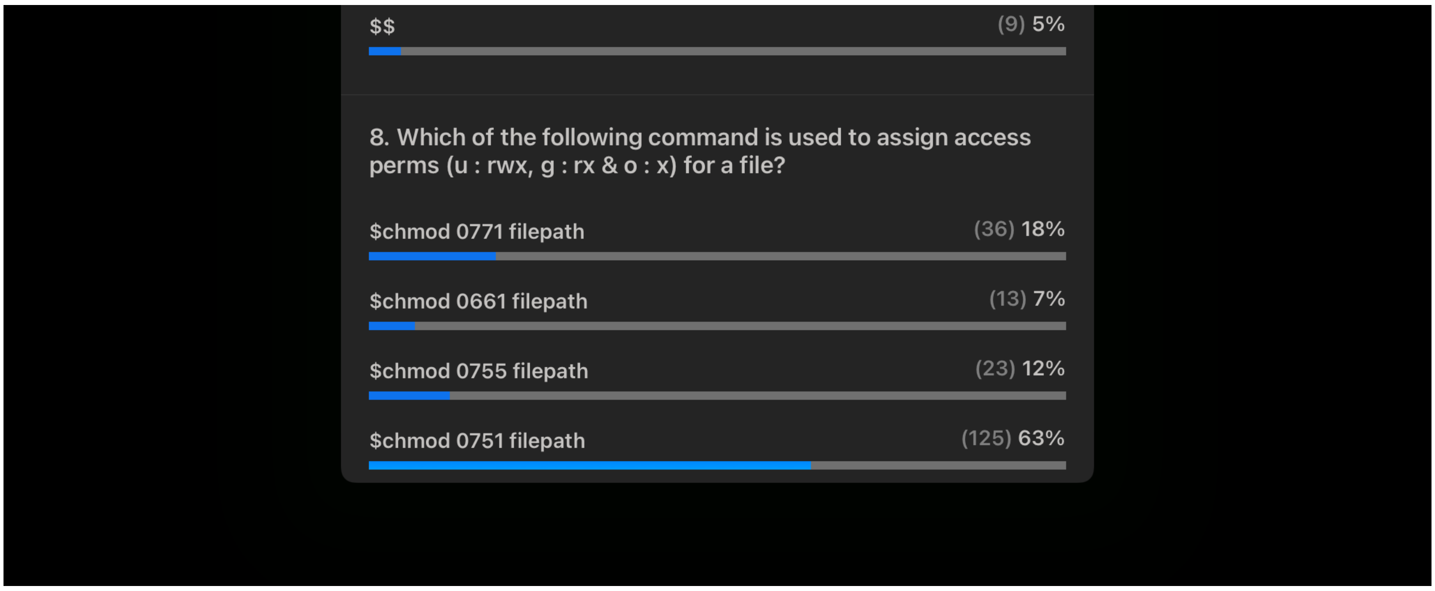
6. What is the command to change login shell

| | |
|---|---|
| csh | (28) 14% |
| chsh | (65) 33% |
| ssh | (46) 23% |
| All of the above | (58) 29% |



7. Which variable is used to get exit status of last executed command?

| | |
|---|---|
| #* | (5) 3% |
| $* | (17) 9% |
| $? | (166) 84% |



$$ (9) 5%

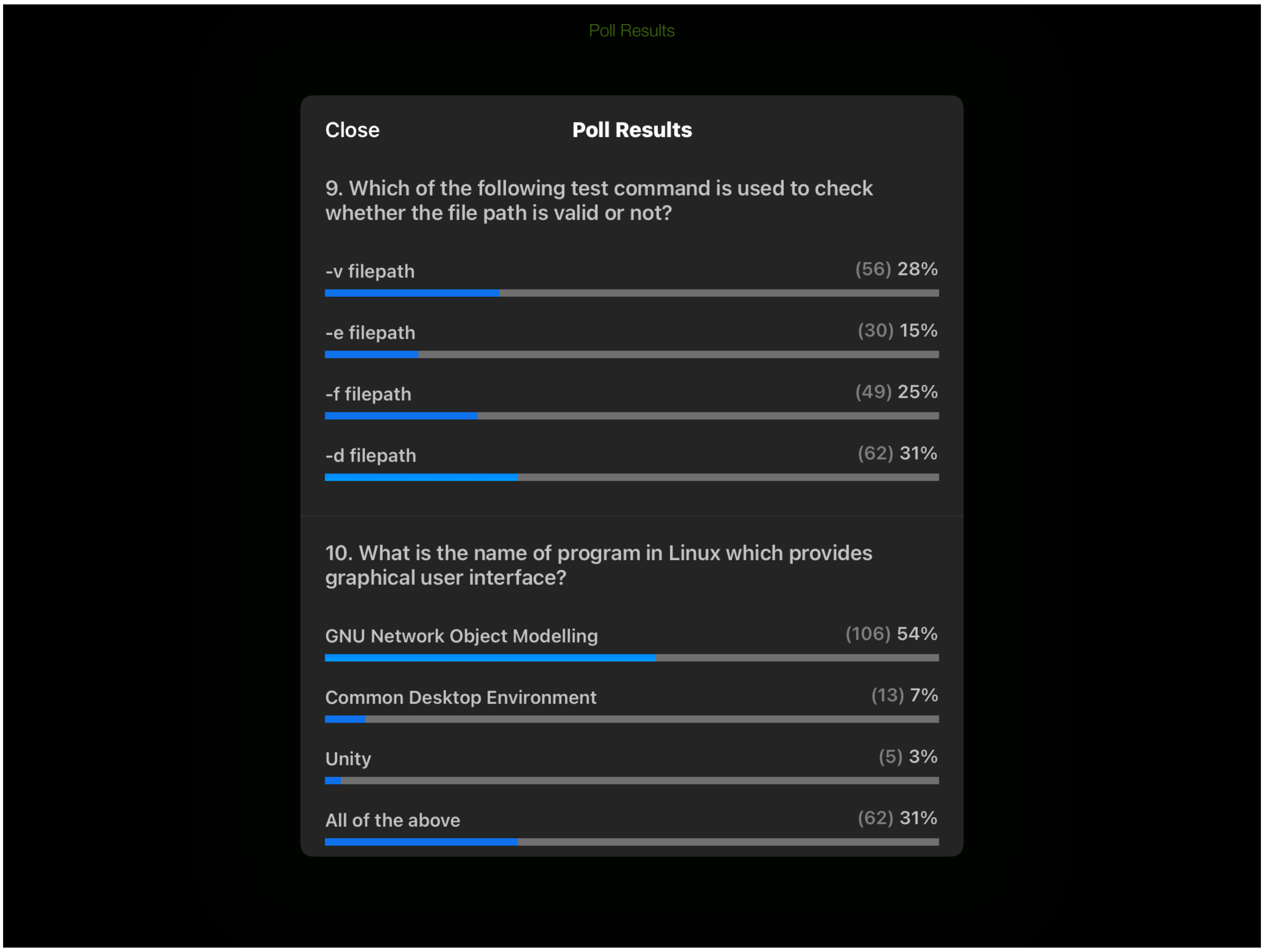
8. Which of the following command is used to assign access perms (u : rwx, g : rx & o : x) for a file?

$chmod 0771 filepath (36) 18%

$chmod 0661 filepath (13) 7%

$chmod 0755 filepath (23) 12%

$chmod 0751 filepath (125) 63%



Poll Results

Close **Poll Results**

9. Which of the following test command is used to check whether the file path is valid or not?

-v filepath (56) 28%

-e filepath (30) 15%

-f filepath (49) 25%

-d filepath (62) 31%

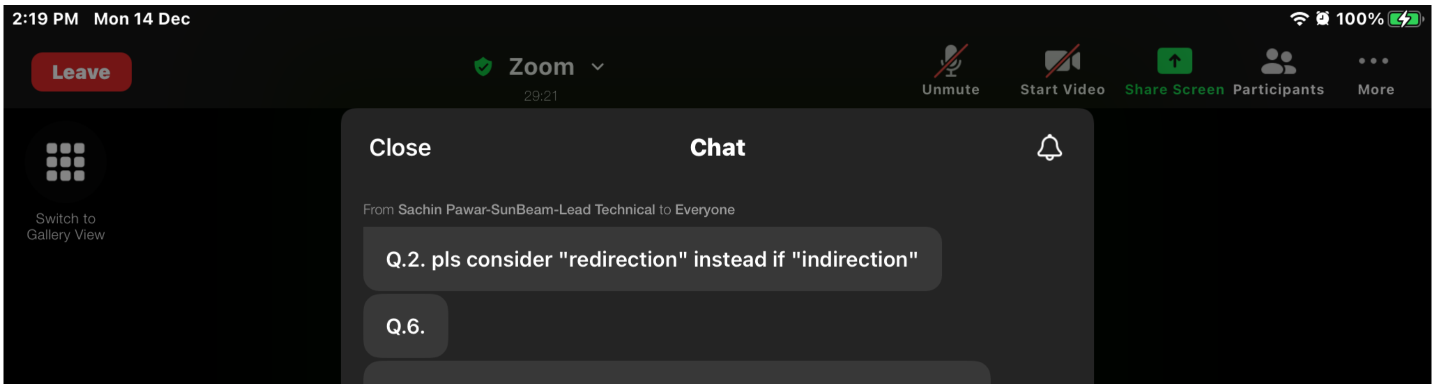
10. What is the name of program in Linux which provides graphical user interface?

GNU Network Object Modelling (106) 54%

Common Desktop Environment (13) 7%

Unity (5) 3%

All of the above (62) 31%



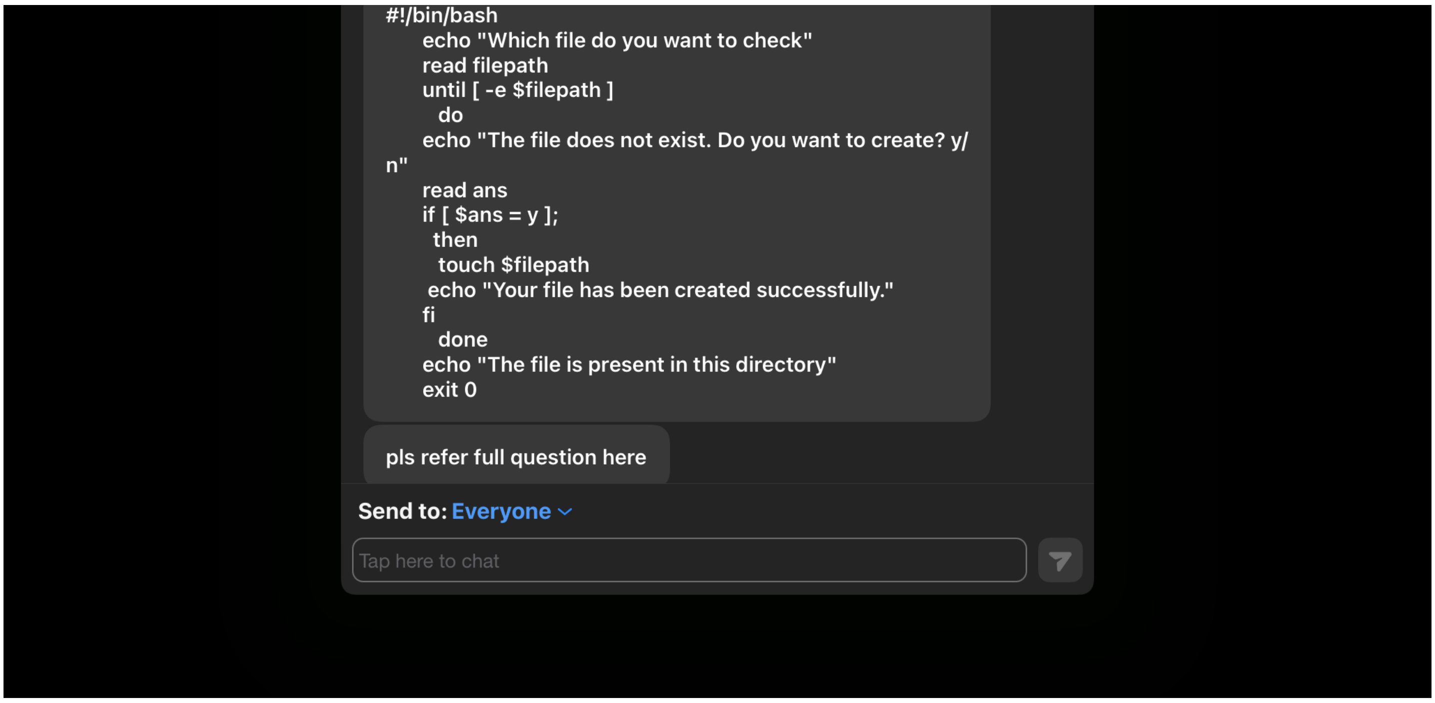
2:19 PM Mon 14 Dec 100%

Close **Chat**

From Sachin Pawar-SunBeam-Lead Technical to Everyone

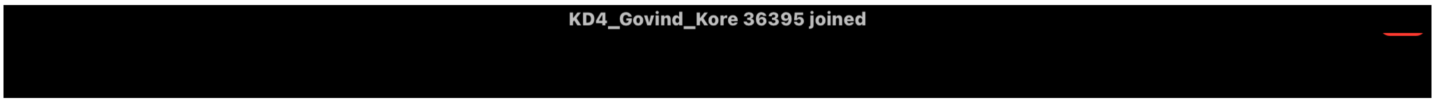
Q.2. pls consider "redirection" instead if "indirection"

Q.6.



```
#!/bin/bash
     echo "Which file do you want to check"
     read filepath
     until [ -e $filepath ]
       do
     echo "The file does not exist. Do you want to create? y/
n"
     read ans
     if [ $ans = y ];
       then
       touch $filepath
      echo "Your file has been created successfully."
     fi
       done
     echo "The file is present in this directory"
     exit 0
```

pls refer full question here



KD4_Govind_Kore 36395 joined

# QUIZ-03
Q. Which command is used to extract specific columns from the file?
A. cat
B. cut
C. grep
D. paste
**Answer: B**

Q. Which of the following is not a valid indirection operator?
A. >
B. 0>
C. 2>
D. <
**Answer: B**

Q. Which of the following command is used to display the directory attributes rather than its contents?
A. ls -l -d
B. ls -l
C. ls -x
D. ls -F
**Answer: A**

Q. Which of the following is not a communication command
A. write
B. mesg
C. mail
D. grep
**Answer: D**

Q. Which command puts a script to sleep until a signal is recieved?
A. sleep
B. suspend
C. disown
D. break
**Answer: B**

Q. What is the output of following command?
```
 #!/bin/bash
 echo "Which file do you want to check"
 read filepath
 until [ -e $filepath ]
  do
 echo "The file does not exist. Do you want to create?
 y/n"
 read ans
```

```
if [ $ans = y ];
 then
  touch $filepath
 echo "Your file has been created successfully."
fi
  done
echo "The file is present in this directory"
exit 0
```

A. it checks the existance of your entered file in the present working directory
B. it creates the file if file does not exists
C. program runs untill you create the file
D. all of the mentioned
**Answer: D**

Q. Shell script executes __________
A. commands
B. commands with its own programming language syntax
C. other binaries
D. all of the above
E. none of the above
**Answer: D**

Q. ________ is used to erase entire command line
A. Cntrl + Q
B. Cntrl + C
C. Cntrl + U
D. Cntrl + W
**Answer: C**

Q. Which of the following operation/s not handled by the Kernel?
A. Management of files on the disk
B. Carrying out data transfer file system and hardware
C. Handling of any interrupts that are issued
D. Acting as an interpreter between hardware and applications
**Answer: A**

Q. Which of the following directory contains process & kernel information files?
A. /usr/bin
B. /var
C. /mnt
D. /proc
**Answer: D**

**# QUIZ-04:**

Q. Bydefault link count of directory file is ___.
A. 1
B. 2
C. 3
D. None of the above
**Answer: B**

Q. On failure open() system call returns ____.
A. 0
B. -1
C. NULL
D. smallest non-negative integer
**Answer: B**

[ NULL is a predefined macro whose value is 0 which is typecasted into a void *
**#define NULL ((void *)0) ]**

Q. ln -s command internally makes a call to ______ system call.
A. hard_link()
B. symlink()
C. link()
D. all of the above
E. none of the above
**Answer: B**

Q. iNode of a file present in the iNode List Block is called as ______.
A. Incore iNode
B. Inmemory iNode
C. Physical iNode
D. Logical iNode
**Answer: C**
**[ iNode of a file present onto the disk (iNode List Block/Master File Table) is referred as Disk iNode/Physical iNode, whereas copy of Disk iNode in the main memory is called as Inmemory iNode/Incore iNode/Logical iNode ].**

Q. Which of the following command is used to display contents of an iNode of a file?
A. stat -i filepath
B. stat -f filepath
C. stat -a filepath
D. stat filepath
**Answer: D**

Q. In which of the following filesystem disk space allocation method there is no external fragmentation?
A. Contiguos Allocation
B. Linked Allocation
C. Direct Allocation
D. Index Allocation
**Answer: D**

Q. Which of the following free space management mechanism/s is/are used with contiguos allocation disk space allocation method?
A. bit vector
B. grouping
C. counting
D. linked list
E. Both B & C
**Answer: E**

Q. Which of the following filesystem do not supports journaling?
A. ext2
B. ext3
C. NTFS
D. JFS
**Answer: A**

Q. ______ algorithm is used in open() system call to convert filepath into iNode number.
A. iname
B. namei
C. name_convertor
D. ntoi
**Answer: B**

Q. rm command internally makes call to _____ system call.
A. remove()
B. rm()
C. rmdir()
D. unlink()
**Answer: D**

**# QUIZ-05:**
Q. On success fork() system call returns _____ to child process.
A. pid of parent
B. pid of new process
C. 0
D. -1
E. both pid of child & pid of parent
**Answer: B**

Q. Which of the following command is used to create a filesystem in Windows?
A. mkfs
B. createfs
C. format
D. all of the above
E. none of the above
**Answer: C**

Q. _____ is a system program that copies an execution context of process scheduled by the scheduler from its PCB onto the CPU registers.
A. interrupt handler
B. dispatcher
C. cpu scheduler
D. all of the above
**Answer: B**

Q. What is a daemon process?
A. process whose parent has died
B. process who has completed its execution but still has an entry in the process table
C. process which is running infinitely
D. process which runs automatically without any user interaction
**Answer: D**

Q. Which of the following disk scheduling algorithm suffers from starvation?
A. FCFS Disk Scheduling Algorithm
B. SSTF Disk Scheduling Algorithm
C. SCAN Disk Scheduling Algorithm
D. C-SCAN Disk Scheduling Algorithm
D. All of the above
E. None of the above
**Answer: B**

**SSTF**: in this algo, whichever request is close to the current position of the R/W head OR arm gets accepted and completed first, so there are quite good chance that there may exists such a request which is far away from R/W head, and this request may gets blocked, and this problem is referred as **starvation**.

Q. In which of the following case non-preemptive cpu scheduling takes place?
A. running -> ready
B. ready -> waiting
C. Both options 1 & 2
D. None of the above
**Answer: D**

Q. Consider a disk with 200 tracks and the queue has random requests from different processes in the order: **55, 58, 39, 18, 90, 160, 150, 38, 184,** Initially arm is at 100. What is an Average Seek length using **FCFS, SSTF, SCAN and C-SCAN** algorithm respecitvely.
A. 55.3, 28.5, 29.8 & 35.6
B. 56.3, 28.5, 30.8 & 35.6
C. 55.3, 27.5, 27.8 & 35.6
D. 55.3, 27.5, 28.8 & 35.6
**Answer: C**

**FCFS: (45+3+19+21+72+70+10+112+146)/9 = 498/9 = 55.3**
**SSTF: (10+32+3+16+1+20+132+10+24)/9 = 248/9 = 27.5**
**SCAN:**

Q. Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80 and 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is _________ tracks
A. 5
B. 9
C. 10
D. 11
**Answer: A**

Q. System in which multiple processes can submitted at a time is referred as
A. multi-processing
B. multi-tasking
C. multi-programming
D. multi-threading
**Answer: C**

Q. On success wait() system call returns _____.
A. 0
B. 1
C. pid of child process
D. pid of parent process
**Answer: C**

**Explanation:**
**wait() system call is used to avoid zombie state/defunct state**
**wait() system call:**
**- puases an execution of parent proces**
**- it reads an exit status of child process from its PCB and return it to the parent process**
**- it removes/destroys an entry/PCB of child process from the system i.e. from the process table**
**and it returns pid of child process to parent process on success.**

# QUIZ-06

**1.**

```
int main(void){
    printf("main() started !!!\n");
    fork();
    fork();
    fork();
    printf("main() exited !!!\n");
    return 0;
}
```

**How many number of times main() exited !!! will print in an output of a above program?**

A. 1
B. 3
C. 6
D. 8

**Answer: D**

**Explanation:**

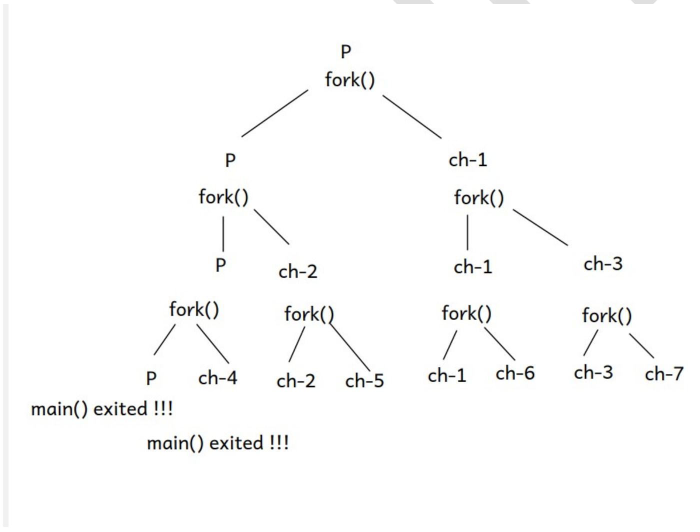
```
P

fork(): -> ch1

P                           ch1

fork():          ch2        fork():  ch3
P                ch2        ch1      ch3

fork():          fork():    fork():     fork():
P    -    ch4    ch2 - ch5  ch1 - ch6   ch3 - ch7
```



**2. Under Pipe Message Passing IPC mechanism, if processes are non-related then one process can send message to another process by using _______.**
A. pipe command (|)
B. pipe() system call
C. both options A & B
D. None of the above
**Answer: B**
- **related processes:** processes which are of same parent are called as related processes, whereas processes which are of different parents are called as **non-related processes.**
- by using pipe command (|) only related processes can communicates, whereas non-related processes can communicates with each other by using pipe() system call.

**3. Which of the following statement is false about an IPC?**
A. Shared Memory Model is faster IPC mechanism than Message Passing Model.
B. By using Pipe, Message Queue, Signals IPC mechanisms only processes running on the same system can communicates.
C. In Message Queue IPC mechanism, processes can communicates by directly sending as well recieving message packets to each other.
D. By using Socket IPC mechanism, processes can communicates with each other which are running across the machines.
**Answer: C**

- An OS maintains message queue, so first message packet/s send by any process gets submitted into the message queue and then it can be sent to the reciever process.

**4. Which of the following flag/option is used with ipcs command to display information about active message queues.**
A. -m
B. -q
C. -M
D. -s
**Answer: B**

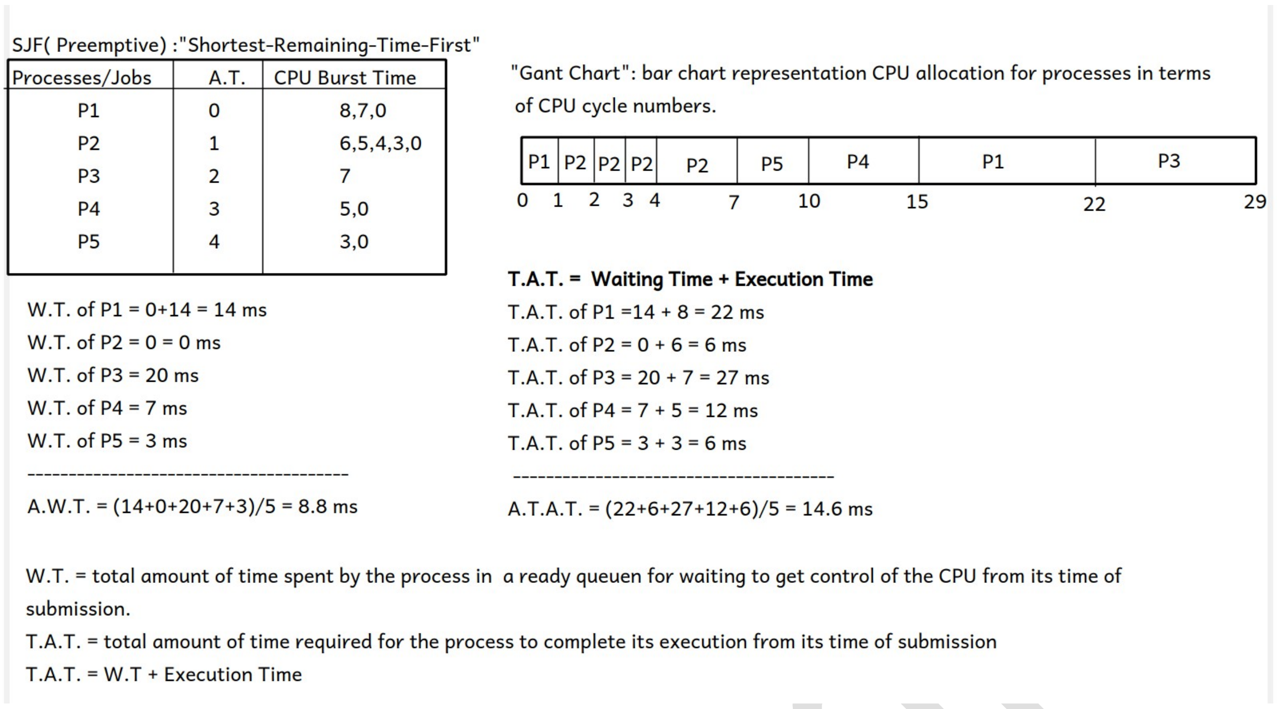
**5. Which of the following is not an IPC system call?**
A. shmget()
B. signal()
C. pthread_create()
D. kill()
**Answer: C**

**6. Which of the following CPU scheduling algorithm suffers from starvation?**
A. FCFS Scheduling
B. SJF Scheduling
C. Priority Scheduling
D. Both SJF & Priority Scheduling
E. None of the above
**Answer: D**

**7. In FCFS CPU scheduling algorithm, due to an arrival of longer processes before smaller processes, average waiting time gets decreases and overall system performance gets down this drawback is called as _____.**
A. Belady's Anamoly
B. Convoy Effect
C. Indefinite Blocking
D. Starvation
**Answer: B**

**8. Which of the following statement is false about Round Robin CPU Scheduling algorithm?**
A. RR CPU Scheduling algorithm ensures minimim response time.
B. In a RR CPU Scheduling algorithm there is no starvation.
C. In a RR CPU Scheduling algorithm, if time quantum/time slice is minimum response time also minimum.
D. RR Scheduling algorithm is non-preemptive scheduling algorithm.
**Answer: D**

**9. What is an average turn-around time for following sample input after applying SRTF (Shortest-Remaining-Time-First) CPU scheduling algorithm? (An arrival time i.e. submission time of processes are different).**
**Processes - A.T - CPU Burst Time**
**Input: P1-0-8, P2-1-6, P3-2-7, P4-3-5, P5-4-3**
A. 14.8 ms
B. 15 ms
C. 14.6 ms
D. 23 ms
**Answer: C**



SJF( Preemptive) :"Shortest-Remaining-Time-First"

| Processes/Jobs | A.T. | CPU Burst Time |
|---|---|---|
| P1 | 0 | 8,7,0 |
| P2 | 1 | 6,5,4,3,0 |
| P3 | 2 | 7 |
| P4 | 3 | 5,0 |
| P5 | 4 | 3,0 |

"Gant Chart": bar chart representation CPU allocation for processes in terms of CPU cycle numbers.

```
| P1 | P2 | P2 | P2 |  P2  |  P5  |  P4  |   P1   |   P3   |
0    1    2    3    4      7      10     15       22       29
```

W.T. of P1 = 0+14 = 14 ms
W.T. of P2 = 0 = 0 ms
W.T. of P3 = 20 ms
W.T. of P4 = 7 ms
W.T. of P5 = 3 ms
----------------------------------------
A.W.T. = (14+0+20+7+3)/5 = 8.8 ms

**T.A.T. = Waiting Time + Execution Time**
T.A.T. of P1 =14 + 8 = 22 ms
T.A.T. of P2 = 0 + 6 = 6 ms
T.A.T. of P3 = 20 + 7 = 27 ms
T.A.T. of P4 = 7 + 5 = 12 ms
T.A.T. of P5 = 3 + 3 = 6 ms
----------------------------------------
A.T.A.T. = (22+6+27+12+6)/5 = 14.6 ms

W.T. = total amount of time spent by the process in a ready queuen for waiting to get control of the CPU from its time of submission.
T.A.T. = total amount of time required for the process to complete its execution from its time of submission
T.A.T. = W.T + Execution Time

**10. Which of the following is not a CPU scheduling criteria?**
A. Waiting Time
B. Response Time
C. CPU Burst Time
D. Turn-Around-Time
**Answer: C**
- there are 5 cpu scheduling criterias:
1. cpu ultilzation (max)
2. throughput: (max) total work done per unit time
3. waiting time (min)
4. respose time (min)
5. turn-around-time (min)

CPU Burst Time/Execution Time: total amount of time spent by the process onto the CPU for completing its execution OR total no. of CPU cycles required for the process to complete its execution.

**11. Which of the following statement is false about thread & process?**
A. thread is the smallest execution unit of a process.
B. The CPU can execute only one thread of any one process at a time.
C. Process based multi-tasking is efficient than thread based multi-tasking.
D. Process based multi-tasking can be implemented by using fork() and thread based multi-tasking can be achieved by using pthread library function.
**Answer: C**

**12. If resource can acquired by more than one processes at a time then which of the following tool is used for synchronization?**
A. binary semaphore
B. mutex object
C. classic semaphore
D. all of the above
**Answer: C**

**13. In which of the following problem data inconsistency occures?**
A. Race condition
B. Critical Section Problem
C. Both options 1 & 2
D. None of the above
**Answer: C**

**14. Which of the following statement is false about multi-threading programming?**
A. in all modern operating systems bydefault one thread gets created i.e. main thread
B. if any process terminates all the threads of that process also gets terminates
C. first step of multi-threading programming is to register a thread procedure with an OS.
D. all the threads of one process executed by the CPU concurrently.
**Answer: C**

**15. Which of following is not valid multi-threading model in an OS?**
A. Many-to-many model
B. Many-to-one model
C. One-to-one model
D. One-to-many model
**Answer: D**