Chapter Table of Contents

Chapter 2.2

Working with UNIX File System

Aim

To acquire basics and structure of UNIX file system

Instructional Objectives

After completing this chapter, you should be able to:

- Explain various UNIX system calls used to manage file system along with its syntax
- Describe the various ways of creating files in UNIX
- Explain how to create special files in UNIX
- Illustrate how to change the directory and root
- Explain the methods to change owner and mode of file
- Demonstrate the use of - stat,fstat,pipes and dup
- Illustrate how to mount and unmount a device or a file system
- Describe how to link and unlink files in UNIX

Learning Outcomes

At the end of this chapter, you are expected to:

- Summarise the functionality of UNIX system calls
- Create a file using UNIX commands
- Summarize the steps for creating a special files in UNIX
- Use cd command to change present directory
- Demonstrate the commands chown and chmod
- Describe system calls - stat,fstat,pipes and dup
- Discuss how to mount and unmount files in UNIX file system
- Perform link and unlink

2.2.1 UNIX system calls

a) Open, Read, Write, Lseek, Close

A system call is a request for the operating system on behalf of the user's program. The system calls can be defined as functions used in the kernel itself. When a process makes a call to system call, the mode of execution changes from user mode to kernel mode. The C compiler uses a predefined library that have list of all the system calls used

System calls can be classified into five major categories: file management, process control, device management, information maintenance and communication.

UNIX system offers around 200 special called system calls. The following table lists some important system call related to file system:

GENERAL CLASS	SPECIFIC CLASS	SYSTEM CALL
File Structure Related Calls	Creating a Channel	creat() open() close()
	Input/Output	read() write()
	Random Access	lseek()
	Channel Duplication	dup()
	Aliasing and Removing Files	link() unlink()
	File Status	stat() fstat()
	Access Control	access() chmod() chown() umask()
	Device Control	ioctl()

Let us begin our discussion on system calls which are related to file I/O.

The system calls related to file system allows you to open, create, and close files, read and write files, randomly access files, alias and remove files, get information about files, check the accessibility of files, change protections, owner, and group of files, and control devices.

The basic input and output operations in a file system start by opening a file. A file can be opened in a UNIX system by using "creat()" or "open()" system calls. These system calls return a file descriptor as an integer. Following are the predefined values of File descriptors:

Int- file descriptor	Used for
0	Standard input (terminal Keyboard)
1	Standard output (display screen)
2	Standard error (display screen)

open():

It's not required to open a file to access its attributes but if you want to read /write into a file, and then you have to open the file.

open()system call allows you to open a file for reading, writing, or reading and writing.

Prototype for the open() system call:

```
#include <fcntl.h>

int open(char *file_name,int option_flags [,int mode]);
```

Here file name is a pointer to the character string that represents file's pathname, option_flags represent the type of channel, and mode defines the file's access permissions if the file is being created.

The allowable option flags as defined in "/usr/include/fcntl.h" are:

Option Flags	Purpose
O_RDONLY 0	Open the file for reading only
O_WRONLY 1	Open the file for writing only
O_RDWR 2	Open the file for both reading and writing
O_NDELAY 04	Non-blocking I/O
O_APPEND 010	append (writes guaranteed at the end)
O_CREAT 00400	open with file create (uses third open arg)
O_TRUNC 01000	open with truncation
O_EXCL 02000	exclusive open

Following are some example statements to open a file that already exists:

```
intfd;                                          / The file descriptor
fd = open("/etc/passwd", O_RDONLY) ;            /Read only
fd = open("abc.txt" , O_WRONLY | O_APPEND) ;    /Like the >> symbol
fd = open("../abc.txt" , O_WRONLY | O_TRUNC) ; /Similar to the > symbol
```

The first open call above sets the file offset pointer to the beginning of the file. This pointer specifies the position in the file from where the read or write operation will take place. The second open system call opens the file 'abc.txt' for writing but only by appending, which sets the offset pointer to end of file(EOF).The third open system call truncates the file's content and position the offset pointer at the beginning.

close ():

A program automatically closes all open files before termination. But, it's a good practice to close a file explicitly when it is not required.

To close a channel, use the close () system call.

```
The prototype for close():

int close(intfile_descriptor)
```

This call deallocates the file descriptor for the opened file/process and makes it available for next open.

Read() write():

The read() system call does all input and the write() system call does all output. When used together, they provide all the tools necessary to do input and output sequentially. When used with the lseek() system call, they provide all the tools necessary to do input and output randomly.

Both read() and write() take three arguments. Their prototypes are:

```
int read(intfile_descriptor, char *buffer_pointer,unsignedtransfer_size);
int write (intfile_descriptor, char *buffer_pointer,unsignedtransfer_size);
```

- where file_descriptor identifies the I/O channel(i.e. the file on which read/write operation to be performed),
- buffer_pointer pointsto the area in memory where the data is stored for a read() or where the data is taken for a write(),
- transfer_size defines the maximum number of characters transferred between the file and the buffer.
- read() and write() return the number of bytes transferred.
- This command will read/write single character at a time,but if you don't want to process each character individually. Then you should use an array.

Let us see an example using open(), read() and write():

```
/*  ccp.c---copies a file with read and write system calls  */

     #include <fcntl.h>         /* defines options flags */
     #include <sys/types.h>     /* defines types used by sys/stat.h */
     #include <sys/stat.h>      /* defines S_IREAD & S_IWRITE  */
     #define BUFSIZE 1024

int main()
     {
int fd1, fd2;          /* file dicriptor for read and write */
       int n ;               /* Number of characters returned by read */
char buffer[BUFSIZE];  /* size of buffer for read and write operation */

       fd1 = open("/etc/passwd", O_RDONLY);
       fd2 = open("passwd.bak", O_WRONLY | O_CREAT | O_EXCL, S_IWRITE);


/* return values of read is used in write as argument */

       while (( n= read( fd1, buffer, BUFSIZE )) > 0 )
              write( fd2, buffer, n) ;

       close( fd1);
       close( fd2);

exit(0);
     }
```

lseek():

The UNIX system , ordinary file are treated as a sequence of bytes. Generally, in a file input and output operation is done sequentially. The read and write operations are done from the beginning to the end of the file. Sometimes, sequential access to a file is not efficient as its takes lots of time.

Using lseek() system call UNIX system provides a user random access to a file, that means you can read and write randomly anywhere in the file. During file I/O, the UNIX system uses a long integer, also called a File Pointer, to keep track of the next byte to read or write. This long integer represents the number of bytes from the beginning of the file to that next character. Random access I/O is achieved by changing the value of this file pointer using the lseek() system call.

The lseek call moves the file offset pointer to a specified point. lseek doesn't do any physical I/O, but it determines the position in the file where the next I/O operation will take place.

- here file_descriptor identifies the I/O channel(file for I/O)

- offset and whencework together to describe how to change the file pointer according tothe following table:

whence	new position
0	offset bytes into the file
1	current position in the file plus offset
2	current end-of-file position plus offset

If successful, lseek() returns a long integer that defines the new file pointer value measured in bytes from the beginning of the file. If unsuccessful, the file position does not change.

Following is an example using lseek():

```
     /* lseek.c  */

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

int main()
     {
intfd;
long position;

fd = open("datafile.dat", O_RDONLY);
if ( fd != -1)
          {
position = lseek(fd, 0L, 2);  /* seek 0 bytes from end-of-file */
if (position != -1)
printf("The length of datafile.dat is %ld bytes.\n", position);
else
perror("lseek error");
          }
else
printf("can't open datafile.dat\n");
close(fd);
     }
```

Self-assessment Questions

1) Choose the correct statement:
   a) C programs can directly make system calls
   b) Library functions use system calls
   c) Both (a) and (b)
   d) Library functions does not use system calls

2) When a process makes a system call, its mode changes from
   a) User to kernel
   b) Kernel to user
   c) Restricted to unrestricted
   d) Unrestricted to restricted

3) Which of the following are not system calls?
   a) chmod
   b) open
   c) lseek
   d) getcx

4) File pointer
   a) Is a long integer
   b) Is of a pointer data type
   c) Represents the position of read-write head from the beginning of file
   d) None of these

5) Open()system call lets you open a file for
   a) Reading
   b) Writing
   c) Reading and writing
   d) All of these

6) Read and write system call allows to do input and output ……
   a) Is a long integer
   b) Is of a pointer data type
   c) Represents the position of read-write head from the beginning of file
   d) None of these

7) lseek system call allows ……..access within a file
   a) Sequential
   b) Random
   c) Sequencial as well as random
   d) None of these

2.2.2 File Creation

A file can be opened by using either the "creat()" or "open()" system calls. These system call return a file descriptor that is used to handle all I/O related operations. File descriptors 0, 1, and 2 refer to standard input, standard output, and standard error files respectively.

creat():

The prototype for the creat() system call is:

Int creat(char *file_name, int mode);

- Where file_name is pointer to a null terminated character string that names the file and mode defines the file's access permissions.
- The mode is usually specified as an octal number such as 0666 that would mean read/write permission for owner, group, and others.

Creat system call creates a file, if the file named by file_name does not exist and if the file already exist, then its contents are discarded and the mode value is ignored.

Following is an example of how to use creat():

```
     /*  creat.c */

     #include <stdio.h>
     #include <sys/types.h>        /* defines types used by
sys/stat.h */
     #include <sys/stat.h>         /* defines S_IREAD &
S_IWRITE       */

int main()
     {
intfd;
fd = creat("datafile.dat");

printf("datafile.dat opened for read/write access\n");
printf("datafile.dat is currently empty\n");

close(fd);
exit (0);
     }
```

2.2.3 Creation of special files

Normally special files get created during Linux distribution installation process, but you can also create special files yourself if you want to. Device file is also known as a “device node”. Before creating a special file, you need to gather some information. Say /dev/zero is an example of a special file. To know more about this file use ls command as following

;

```
$ ls – l  /dev/zero
```

```
crw-rw-rw 1 root root 1, 5  may1  11.23  /dev/zero
```

“c” – specifies that it is a “character device”. “b” is used for “block device”

“1, 5” – specifies “major device number” and “minor device number”

Basically the major number tells the Linux kernel which device driver to talk to, and the minor number tells the device driver which device you're talking about. For example, for a disk, disk controller is the major device number and the disk is the minor device is the disk.

With this information, you can create your own device node by using *mknode* command as following:

```
$ mknodmy_device  c 1 5
```

This creates a new file my_device in current folder, which does exactly the same thing as /dev/zero. This file contain information about device type, major number and minor number.

Similarly, you can use ls command to get above information about a device file and you can create your own device node/ special file using mknode command.

mknod() returns zero on success, or -1 if an error occurred (in which case, errno is set appropriately).

Self-assessment Questions

8) Create system call…….
   a) Opens a file that already exists
   b) Opens a file that does not exist by creating a new file
   c) Opens a file that does exist, its contents are discarded and the mode value is ignored.
   d) All of these

9) Mknode is used to create
   a) A new file
   b) A new device file
   c) A new directoy
   d) All of above

10) Mknod() returns _____ on success.
   a) 1
   b) 2
   c) 3
   d) 0

11) Which command is used to move to the parent directory of the current directory:
   a) $cd . .
   b) $ cd.
   c) $ cd
   d) $ cd /

12) Which command is used to move to the root directory:
   a) $cd . .
   b) $ cd.
   c) $ cd
   d) $ cd /

13) Which command is used to return to your home directory, enter:
   a) $cd . .
   b) $ cd.
   c) $ cd
   d) $ cd /

14) Options for access mode of a file can be…
   a) Read
   b) Write
   c) Execute
   d) All of these

15) The command ls o+wx /home/amit

a) Will give read and write permission to owner

b) Will give write and execute permission to owner

c) Will give read and execute permission to owner

d) None of these

2.2.4 Changing directory and root

```
$cd
```

This command changes your current directory location. By default, your UNIX login session begins in your home directory.

To switch to a subdirectory (of the current directory) named*myfiles*, enter:

```
$ cdmyfiles
```

To switch to a directory named /home/abc/xyz_doc, enter:

```
$ cd  /home/abc/xyz_doc
```

To move to the parent directory of the current directory, enter:

```
$ cd . .
```

To move to the root directory, enter:

```
$ cd /
```

To return to your home directory, enter:

```
$ cd
```

2.2.5 Changing owner and mode

File ownership provide a secure method of storing the files. When you login to a UNIX system, you are assigned a owner ID and group ID. The chown command changes the owner of a file. If you want to you this command, you need to login as root.

The basic syntax of chown is as follows:

```
$ chown user filelist
```

Here user can be either the name of a user on the system or the user id of a user on the system.

Following example –

```
$ chown Amit final_marks
```

This chown command will change the owner of file 'final_marks' to Amit. Root has the capability to change ownership of any file but a normal user can only change the ownership of files that they own.

Every file in UNIX has the following attributes as shown in figure 2.2.3 –

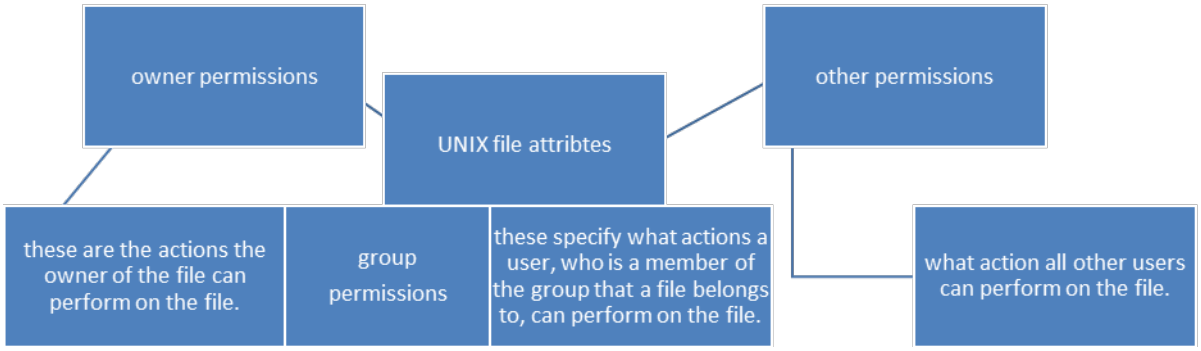

Figure 2.2.1: UNIX file attributes

There are permissions associated with each file/directory (i.e. access mode).Permissions are broken in three groups as shown in figure 2.2.4:

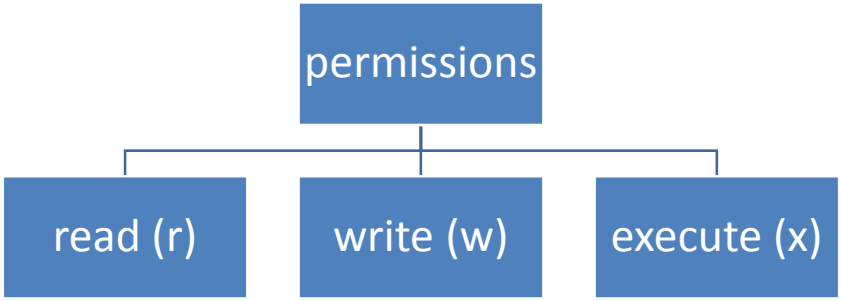

Figure 2.2.2: Permissions associated with a file/directory

To see, what are the permissions associated with a file, you can use ls command with –l option as follows:

```
$ ls –l /home/amit

-rwxr-xr--  1  amit   users 1024  may 1 12.30 my_file

drwxr-xr--- 1  amit   users 1024  may 1 12.30 my_dir
```

Here first column represent the permissions associated with a file/directory.

Characters (2-4): represent the permissions for the file's owner

Characters (5-7): represent the permissions for the group file's owner

Characters (8-10): represent the permissions for everyone else

We can analyze from the output of the above command that owner have read (r), write (w) and execute (x) permissions. The group has read(r) and execute(x) permissions but no write (w) permission. Other world has only write (w) permissions.

Changing permissions:

To change permissions associated with a file/directory you can use chmod command.

This command can be used in two modes:

- Symbolic mode
- Absolute mode

Symbolic mode:

This mode uses the symbols to add, delete, or specify the permissions you want to set for a file. Following is the symbol table used to change permissions

Chmod operator	Description
+	Add the permissions to a file/directory
-	Removes the designated permission(s) from a file/directory.
=	Set the permissions

For example the command:

```
$ lso+wx /home/amit
```

We have seen in the earlier example that others had only read (r) permission. This command will give owner write(w) and execute(x) permission.

Absolute mode:

In absolute mode, each permission is given a value as mention in following figure 2.2.5:

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

If you type following command:

```
$ ls 777 /home/amit
```

Here now the others will get write(w) as well as execute(x) permission.

2.2.6 Stat and fstat

The i-node data maintains the information about a file. If you want to use the information in the i-node in your program, you can access this information with the stat() and fstat() system calls. stat() - return the information in the i-node for the file named by a string

fstat() - return the information in the i-node for the file named by a file descriptor.

```
The prototypes for stat() and fstat() are:

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

int stat(file_name, stat_buf)
char *file_name;
struct stat *stat_buf;

intfstat(file_descriptor, stat_buf)
intfile_descriptor;
struct stat *stat_buf;
```

Here file_name names the file as an ASCII string and file_descriptor namesthe I/O channel and therefore the file.

2.2.7 Pipes

pipe() system call creates a communication channel between two processes. You can connect the standard input of one process to the standard output of the other process.

```
The prototype for pipe() is:

int pipe (intfile_descriptors);
```

2.2.8 Dup and dup2

The dup() system call duplicates an open file descriptor and returns the new file descriptor with following properties:

- The new file descriptor refers to the same open file.
- Both file descriptors share the same file pointer.
- Both the files have same access mode(rwx)

Has the same access mode, whether read, write, or read and write.

Prototype for the dup () system call: int dup(intfile_descriptor);

```
  /*  dup.c
demonstrate redirection of standard output to a file.
 */

int main()
     {
intfd;
fd = open("abc.txt",O_WRONLY | O_CREAT);
close(1);         /* close standard output  */
dup(fd);       /* fd will be duplicated into standard out's slot */
close(fd);        /* close the extra slot */
printf("Hello, world!\n");    /* should go to file foo.bar */
exit (0);         /* exit() will close the files */
     }
```

2.2.9 Mounting and unmounting file system:

The **mount** system call connects/mounts a storage device or file system to an existing file system hierarchy. This call allow the user to access the data from storage device or/file system as a file system instead of disk blocks.

The syntax of mount system call is as following:

Mount (special pathname, directory pathname, options)

Here

Special pathname: is the name of device special file, that will mounted on file system.

Directory pathname: is the location in the existing file system directory where file system of the device special file will be mounted. It is also known as a mount point.

Options- if the file system to be mounted as read only.

Example: suppose, you have logged in UNIX system as a user. The file system appears like in figure 2.2.5

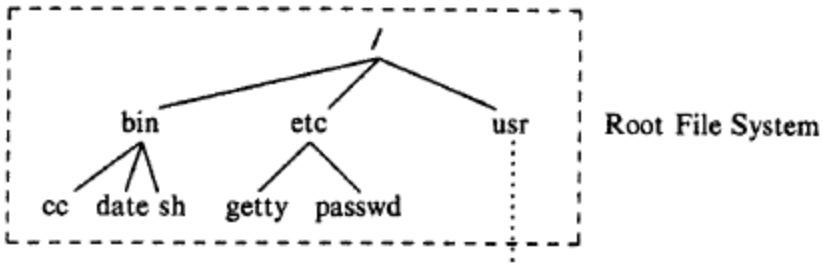

Figure 2.2.3: file system where a user has logged in.

Say, you want to mount the device disk1 to current user file system. The file system of disk1 under dev directory appears as shown in figure 2.2.6

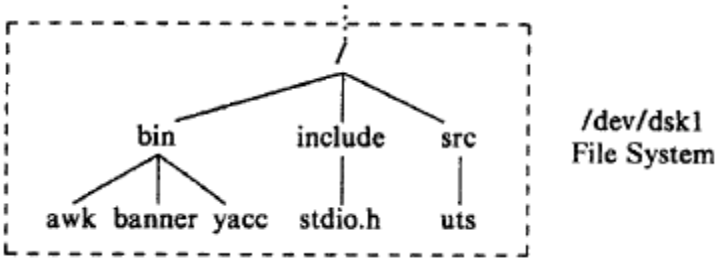

Figure 2.2.4: /dev/disk1 file system

The file system contained in /dev/disk1 is attached to /user in the existing file system tree as shown in fugure 2.2.7

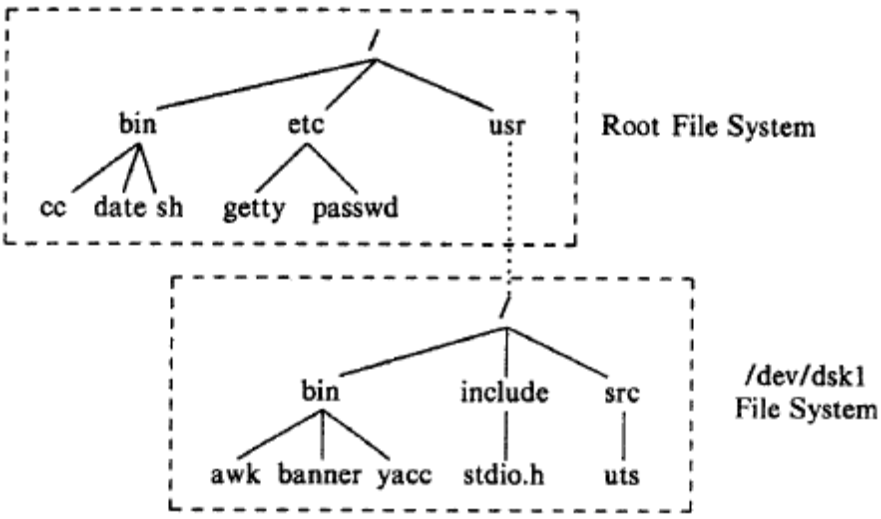

Figure 2.2.5: mounting /dev/disk1 file system to /user under existing root file system

The mount command is used to invoke the mountsystem call that mounts the file system in the "/lev/dsk" onto the directory "/usr".

The **umount** system call disconnects/ unmounts a storage device or file system from an existing file system hierarchy.

The syntax of unmount system call is:

Unmounts (spectial pathname)

Here pathname specifies the storage space or file system to be unmounted.

A file system cannot be unmounted when it is busy - for example, when there are open files on it, or when some process has its working directory there. If you want to unmountsfile system in the "/dev/dsk" onto the directory "/usr", following system call will be used:

unmount(/dev/disk1)

2.2.10 Link and Unlink

Link (file1, file2)

The **link** system call links the file1 to a new name (i.e. file2) in the file system directory structure.

File1- name of existing file

File2- name of target file

Now, file2 shares the same inode as file1.therefore, if you make any changes in any of the file, they will be visible in other file. This kind of link is also known as hard link.

Unlink:

The *unlink* system call removes a directory entry for a file.

The syntax for the *unlink* call is

unlink(pathname);

here*pathname*- name of the file to be *unlinked* from the directory hierarchy.

Example:

unlink(file2);

Self-assessment Questions

16) Choose the correct statement:

a) The function stat refers a file by its name

b) The function stat refers a file by its file descriptor

c) The function fstat refers a file by its name

d) The function fstat refers a file by its file descriptor

17) Mounting a file system results in loading of:

a) Super block

b) Boot block

c) Inode table

d) All of these

18) When a file is aliased

a) A new directory entry is created

b) The inode number is shared

c) A new inode is created

d) None of these

Summary

- A system call is a request for the operating system on behalf of the user's program.
- System calls can be classified into five major categories: file management, process control, device management, information maintenance and communication.
- A file can be opened by using either the "creat()" or "open()" system calls. These system call return a file descriptor that is used to handle all I/O related operations.
- File ownership provide a secure method of storing the files.
- The i-node data maintains the information about a file.
- Pipe system call creates a communication channel between two processes.
- The mount system call connects/mounts a storage device or file system to an existing file system hierarchy.

Terminal Questions

1. Explain various UNIX system calls used to manage file system along with its syntax.
2. Describe the various ways of creating files in UNIX. How to create special files in UNIX?
3. Explain the method/commands to change owner and mode of a file.
4. Demonstrate the use of - stat, fstat, pipes and dup
5. Illustrate with appropriate example commands, how to mount and unmount a device or a file system.
6. Describe how to link and unlink files in UNIX.

Answer Keys

Self-assessment Questions	
Question No.	Answer
1	c
2	a
3	d
4	b
5	d
6	a
7	a
8	d
9	b
10	d
11	a
12	d
13	c
14	d
15	b
16	a and d
17	c
18	b

Activity

Activity Type: Offline **Duration: 30 Minutes**

Description:

Students should be divided into groups to design a ‘capital gain tax calculator’ in Excel for the A.Y. 2016-17.

Bibliography

e-References

- This website was referred on 3rd May 2016 while developing content for Unix file system http://www.csie.ntnu.edu.tw/~ghhwang/course_slices/OS/Unix_System_Calls.pdf
- This website was referred on 3rd May 2016 while developing content for Unix file system http://www.cs.uofs.edu/~bi/2003f-html/cs352/syscalls-file.htm
- This website was referred on 3rd May 2016 while developing content for Unix file system http://www.tutorialspoint.com/unix/unix-pipes-filters.htm
- This website was referred on 3rd May 2016 while developing content for Unix file system http://www.tutorialspoint.com/unix_system_calls/unlink.htm

External Resources

- Maurice J. Bach, The Design of Unix Operating System, (2010) Pearson Education
- S. Prata, Advance UNIX, a Programmer's Guide, (2011), BPB Publications, and New Delhi,
- B.W. Kernighan & R. Pike, The UNIX Programming Environment, (2009) Prentice Hall of India.
- Jack Dent Tony Gaddis, Guide to UNIX Using LINUX, (2010) Vikas/ Thomson Pub. House Pvt. Ltd.

Video Links

Topic	Link
The Linux File System	https://www.youtube.com/watch?v=2qQTXp4rBEE
Directory structure of the UNIX file system	https://www.youtube.com/watch?v=PEmi550E7zw

Notes: