Code No: 9A12601/R09

B.Tech. III Year II Semester Regular & Supplementary Examinations

Set-1

April/May - 2013

LINUX PROGRAMMING

(Information Technology)

Time: 3 Hours Max. Marks: 70

*Answer any **FIVE** Questions*

*All Questions carry **equal** marks*

- - -

1. (a) Explain the various backup utility commands with their syntax and example.

   (b) Describe the contents of a directory, explaining the mechanism by which its entries are updated by cp, mv and

2. (a) Write a shell script to display the multiplication table.

   (b) Discuss in detail about the file name substitution in shell.

3. (a) What is a file? What are the different files attributes? Explain them in detail.

   (b) Write the syntax of open function API and explain the various arguments present in it.

4. (a) What is a zombie process? Explain its importance in Unix programming.

   (b) How to terminate a process? Explain in detail.

   (c) Explain the pause and alarm functions.

5. (a) Write a program that locks resources by using named pipes.

   (b) List and explain the different APIS for message manipulation.

6. (a) Write and explain the procedure to interact a client process with server process by using semaphores.

   (b) With the help of syntax and example, explain any three APIs for shared memory.

7. (a) Write a C program to print the message by only one thread at a time by using messages(b) Give brief description about the relationship between threads, LWPs and hardware processor.

8. (a) What is a socket and explain its role in communication between client and server? **(Unit-VIII, Topic No. 8.1)**

   (b) Explain in detail about the various arguments used in bind and connect socket system calls. **(Unit-VIII, Topic No. 8.3)**

SOLUTIONS TO APRIL/MAY-2013, SET-1, QP

Q1. (a) Explain the various backup utility commands with their syntax and example.

Answer : **April/May-13, Set-1, Q1(a)**

For answer refer Unit-I, Q31(b).

(b) Describe the contents of a directory, explaining the mechanism by which its entries are updated by cp, mv and rm. Why is the size of a directory usually small?

Answer : **April/May-13, Set-1, Q1(b)**

A directory is a file consisting of list of other files. It is an inode pointing towards a data block containing list of inode - filename pairs. An inode uniquely identifies a file/ directory on any storage device. It contains the inode number, device number, device type, access mode, groupid, userid of file owner, number of hardlinks to the file, size, last access time, last modify time and last change time of a file/directory. However, the filename is just an element of the directory, but not a part of the file. In Unix, one file may have multiple names in multiple directories identified uniquely by its inode number.

If a file/directory is deleted using "rm" then the inode value corresponding to the filename for that file is set to 0 in the data block of file. The hard link count for the file is decremented. If this count becomes 0 then the data block corresponding to that file is released by the operating system.

If the file/directory is renamed using "mv" then the filename corresponding to its inode number is modified to the new file name in the data block of the directory. If the file is moved from one directory to another directory. The inode-filename pair in the data block of the directory is deleted and a corresponding entry is added in the data block of the target directory.

If a file is copied using "cp" then the data block corresponding to that file is copied to a free space in the storage device. The new file is assigned with a free inode and an inode for the new file is written to the storage device. The data block for the new file is updated by adding an inode-filename pair for that file.

Size of the directory is usually small because a directory contains only a list of inode-filename pairs consuming less space.

Q2. (a) Write a shell script to display the multiplication table.

Answer : **April/May-13, Set-1, Q2(a)**

```
# !/bin/sh
#
# Script to print a multiplication table of a given number
#
if [$# - eq 0]
then
echo "please enter the number from command line"
echo "syntax: $0 number"
echo "program  printing multiplication table of a provided number"
exit 1
fi
num = $1
j = 1
while [$j -lle 10]
do
echo "$num * $j = 'expr $j\* $num'"
j = 'expr    $j + 1'
done
```

Output

```
$bash    -f   multiply.sh    6
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
6 * 10 = 60
```

(b) Discuss in detail about the file name substitution in shell.

Answer : April/May-13, Set-1, Q2(b)

For answer refer Unit-II, Q1, Topic: File Substitution.

Q3. (a) What is a file? What are the different files attributes? Explain them in detail.

Answer : April/May-13, Set-1, Q3(a)

For answer refer Unit-III, Q1, Q4.

(b) Write the syntax of open function API and explain the various arguments present in it.

Answer : April/May-13, Set-1, Q3(b)

For answer refer Unit-III, Q23, Topic: Open ( ).

Q4. (a) What is a zombie process? Explain its importance in Unix programming.

Answer : April/May-13, Set-1, Q4(a)

Zombie Process

For answer refer Unit-IV, Q15.

Importance of Zombie Process

Zombie processes are very much important in unix programming. Parent processes in most of the programs need to obtain the basic information about its child process which have already exited. The parent process will call a function called wait( ) to wait for the child to exit and then it acquires the information. This information includes the exit status and the resource usage information about the child process. It is easy for the parent to know the exit status of the child when it is already aline. But if the child exits before the parent calls wait( ) and if the parent process is dependent on the child info, this where the problem arises. The solution would be zombie processes. They are needed in such situations.

The information can be saved off to make it available for the parent and this information is called zombie process. Now it does not make any matter if the parent calls wait( ) before or after the child exits.

(b) How to terminate a process? Explain in detail.

Answer : April/May-13, Set-1, Q4(b)

For answer refer Unit-IV, Q13.

(c) Explain the pause and alarm functions.

Answer : April/May-13, Set-1, Q4(c)

For answer refer Unit-IV, Q34.

Q5. (a) Write a program that locks resources by using named pipes.

Answer : April/May-13, Set-1, Q5(a)

Unsolved

(b) List and explain the different APIS for message manipulation.

Answer : April/May-13, Set-1, Q5(b)

For answer refer Unit-V, Q15, Q16 and Q17.

Q6. (a) Write and explain the procedure to interact a client process with server process by using semaphores.

Answer : April/May-13, Set-1, Q6(a)

For answer refer Unit-VI, Q12.

(b) With the help of syntax and example, explain any three APIs for shared memory.

Answer : April/May-13, Set-1, Q6(b)

For answer refer Unit-VI, Q6, Q7.

Q7. (a) Write a C program to print the message by only one thread at a time by using messages.

Answer : April/May-13, Set-1, Q7(a)

```
Static mutex_t locka;
void* printmsg (void* mesg)
{
    if (mutex_lock(&locka))
      perror("mutex_lock");
    else{
        printf("%s", (char*mesg); flush;
        if (mutex_unlock(&locka)) perror("mutex_
        unlock");
    }
}
int main(int argc, char* argv[ ])
{
    if(mutex_init(&locka, USYNC_THREAD, NULL))
    perror ("mutex_lock");
    while(--argc > 0)
        if (thr_create(0, 0, printmsg, argv[argc], 0, 0))
        perror ("thr_create");
    while (!thjr_join(0, 0, 0));
    if(mutex_destroy(&lockx)) perror("mutex_
    destory");
    return 0;
}
```

(b) Give brief description about the relationship between threads, LWPs and hardware processor.

Answer : **April/May-13, Set-1, Q7(b)**

For answer refer Unit-VII, Q4.

Q8. (a) What is a socket and explain its role in communication between client and server?

Answer : **April/May-13, Set-1, Q8(a)**

For answer refer Unit-VIII, Q1.

(b) Explain in detail about the various arguments used in bind and connect socket system calls.

Answer : **April/May-13, Set-1, Q8(b)**

For answer refer Unit-VIII, Q4, Topics: Bind System Call, Connect System Call.