Lab #1 Solutions: COMP 3000A (Operating Systems)
October 9, 2007

In this lab, we will examine some core programs of the UNIX environment as implemented on the Debian GNU/Linux 4.0 systems in the lab. These programs provide a window into the functionality of the operating system.

Please answer all questions below. There are a total of 80 marks available.

1 Part A

This section is designed to be completed in the lab. You get 10% of the total marks for attempting to do part A during assigned lab hours. Please ensure that one of the Lab instructors takes your attendance.

1. **[8]** Have the the TA mark down that you were present and attempted part A during lab hours.

1.1 The Shell

The shell or command line provides a text interface for running programs. While not as visually pleasing as a graphical interface, the shell provides a more clear representation of the functionality provided by the operating system. We will therefore use the shell for most of the work in this course.

To run a program contained in the current directory in the shell, you need to prefix the name of the command with a ./. This ./ tells the shell that the location of the command you wish to run is the current directory. By default, the shell will not search for executable commands in the current working directory. To run most system commands, the name of the command can be typed without a path specification.

1.2 Help

1. **[1]** When working in the shell, help is available on most programs in the system, especially those that are command line based. This system of help is available by using the `man` command. If one wanted to get help on the echo command, the associated command would be `man echo`. What does `man` stand for?

   A: Manual

2. **[1]** The `man` command can also be used to get help on standard C functions. Briefly (in one line), what does the C function `brk` do?

   A: change data segment size

3. **[1]** What does the `more` command do?

   A: Display files for reading

4. **[1]** How do you quit the `less` command?

   A: type the q key

1.3 Shell Basics

1. **[1]** The `which` command can be used to figure out what directory an executable program resides in. Using this command, what directory contains the `top` command?

   A: /usr/X11R6/bin

2. **[1]** `bash` is your default shell. Which of the following is another shell that can be used instead of BASH? `mv`, `cat`, `bc`, `tcsh`, or `arch`

   A: tcsh

3. **[1]** The `ls` command can be used to get a listing of the files in a directory. What options are passed to `ls` to see all of the files within a directory (including hidden files)?

   A: -a

4. **[2]** The `ps` command is used to get a list of processes which are running on the system. Start a new shell from bash by running `bash`. Does the new shell replace or run in parallel with the old BASH shell? What about `exec bash`? You can use the `ps` command to determine if bash is still running.

   A: bash runs in parallel, exec bash replaces it.

5. **[1]** Because each process is started by some other process, there is a process tree structure on the system. From within your bash terminal, determine what process started BASH by using the `pstree` command.

   A: gnome-terminal or maybe xterm (note that other answers are possible)

1.4 Processes

Each application running on a system is assigned a unique process identifier. The `ps` command shows the process identifiers for running processes. Each process running on the system is kept separated from other processes by the operating system. This information will be useful for subsequent questions.

1.5 Permissions

Your permission to access a file in Unix is determined by who you are logged in as. All files on the Unix file system (including directories and other special files) have three different sets of permissions. The first set permissions denotes the allowed file operations for the owner of the file. The second set of permissions denotes the allowed file operations for a group of users. The third set of permissions denotes the allowed file operations for everyone else. A file is always owned by someone and is always associated with a group.

1. **[1]** The `ls` command with the `-l` option can be used to show both the permissions of a file as well as the owner and group associated with the file. Permissions are listed first, followed by the owner and the group.

   Who is the owner of the `/etc` directory?

   A: root

2. **[1]** What group is associated with the `/etc/shadow` file?

   A: shadow

3. **[1]** Each user is a member of some number of groups. You can determine what groups you are part of by using the `groups` command. Based on the groups listed, would you (the "student" user in the lab) be a member of the group associated with the `/etc/shadow` file?

   A: No

1.6 Environment

The environment on both Linux and Windows contains variable - value pairs which are useful to applications running on the system. In Linux, these environment variables can be printed on the command line by referring to the variable name prefixed with a $ sign (eg: to output the value in the HELLO environment variable, one could write `echo $HELLO`).

1. **[2]** On the command line, run the following two sets of commands. Notice that the bash command will start a new shell separate from the shell that the HELLO environment variable was set in.

```
HELLO="Hi There"
bash
echo $HELLO
exit

export HELLO="Hi There"
bash
echo $HELLO
exit
```

   What does the export command seem to do?

   A: Allow the second (child) bash process to see the enviornment variable.

1.7 Dynamic Libraries

Most applications on the system do not contain all the code that they need right within the executable. Instead, dynamic libraries are loaded into the program address space when the program loads. As an example, the standard C library, which contains such functions as `printf` is loaded in at run-time.

1. **[2]** Using `ldd`, what dynamic library dependencies does the `top` command have?

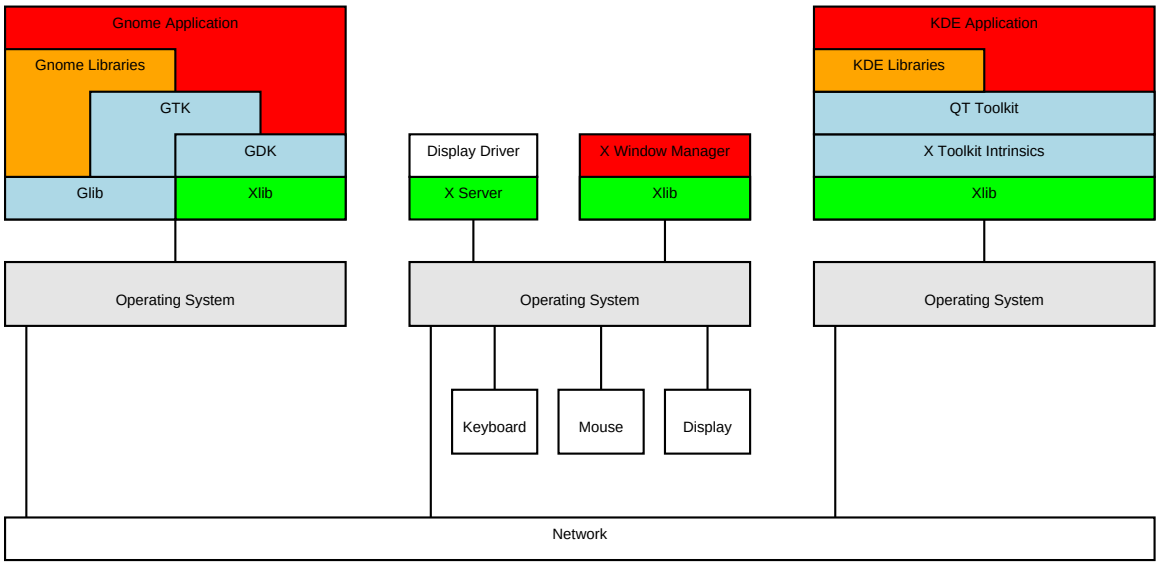
   A: linux-gate, libproc, libncurses, libc, libdl, ld-linux (Partial credit: 1/4 point for each correct library)

2. **[2]** In addition to the libraries listed as dependencies for the application `top` by `ldd`, there may be other libraries that the application loads dynamically at run-time. Retrieve the process number *PID* for the `bash` process (using `ps`) and examine the mmap file located at `/proc/PID/maps`. What other dynamic libraries have been loaded into the application while it has been running?

   A: ld, nss_compat, nsl, nss_nis, nss_files

1.8 X

Below is an architectural overview of the structure of X Window System, the graphical display environment for Linux and other UNIX systems. Microsoft Windows has a different architecture, as does Apple's MacOS X. Under Linux, core X functionality is provided by those elements coloured green. Additional graphical libraries are coloured blue and orange. Applications are coloured red[1]. The X server, which handles the video card, is not contained in the kernel but instead runs as a separate process on the system. This is in stark contrast to Windows, where video processing is done within the kernel.



The X window system, which takes care of controlling the video hardware, is separate from the window manager, which arranges the behaviour and look of windows on the screen. Under Linux, there are currently three major desktop environments built on top of the X window system, KDE, Gnome, and Xfce. Each of these have their own associated libraries and functionality.

1. **[2]** By examining the dynamic dependencies of the `xterm` command, determine whether it is a KDE application, Gnome application, Xfce application, or X application which does not depend on KDE, Gnome, or Xfce.

   A: It is a plain X application.

2. **[2]** Is `epiphany` a Gnome application, a KDE application, an Xfce application, or none of the above?

   A: The epiphany web browser is a Gnome application. The epiphany-game is a plain X application (none of the above).

[1] Image adapted from http://cnlart.web.cern.ch/cnlart/2000/001/kde-gnome2

2 Part B

This section is to be completed in your own time. Your Linux SCS computing account may be useful in solving some of the problems.

2.1 Processes

1. **[2]** What is the difference between the `fork` and `exec` function in a Unix environment?

   A: fork creates a new process, exec replaces the running program in the current process with a new program.

2. **[2]** What is a zombie process?

   A: A process which has finished running but has not had it's return value read by the person who started it yet.

3. **[8]** Give an example C program which creates a zombie process. Note that BASH by default will collect and destroy zombie processes and so you will need to avoid bash destroying the zombie process during debugging. This can be done by delaying the parent exit (using sleep is one good way to do this).

   A:

```
int main() {
  if( fork() == 0 ) {
    return 0;
  } else {
    sleep( 1 );
  }
}
```

4. **[4]** Perform the modifications to your program above to avoid creating a zombie process. List the new program.

   A:

```
int main() {
  if( fork() == 0 ) {
    return 0;
  } else {
    wait( NULL );
    sleep( 1 );
  }
}
```

2.2 Permissions

1. **[3]** Permissions on Unix are grouped into three basic file operations. What are these file operations?

   A: Read, Write, Execute

2. **[1]** What does it mean to have execute permission on a directory?

   A: Can change into the directory.

3. **[4]** What are the 6 basic file permissions within Windows?

   A:

   - **Read: Allows files or folders to be opened as read-only and to be copied.**
   - **Write: Allows the creation of files and folders; allows data to be added to or removed from files.**
   - **List Folder Contents: As per Read but also allows navigation of sub-folders.**
   - **Read and Execute: As per Read but also allows users to run executable files.**
   - **Modify: All the above as well as the permission to delete the file or folder.**
   - **Full Control: Full Control – including the ability to change ACLs.**

   No points for Modify or Write.

4. **[1]** What is the difference between the write and modify file permission in Windows?

   A: Modify allows you to delete the file or folder.

5. **[1]** Because all files are stored in a directory, under Unix permission to delete, rename, and move files is determined by the users access rights on the directory the file is contained in. What attribute on the directory prevents those who can modify a directory from deleting files (hint: help on the `chmod` command may prove useful).

   A: The Sticky bit.

6. **[4]** In Windows, a file can be associated with more than one group and each group can have different access permissions. On Unix, new groups can be created by the system administrator which are supersets of other groups. Given this, is it possible to develop an access permission scenario which would be impossible to implement in Unix but possible to implement in Windows? If yes, give an example. If no, explain why.

   A: Yes. File 1 (G1-R), File 2 (G2-R), File 3 (G1-R, G2-W).

2.3 Dynamic Libraries

1. **[6]** Dynamic libraries allow one copy of executable code to be used by many different processes on the system, without requiring that multiple copies of the code be stored on disk in different files. What are some problems that can arise when different programs use the same common DLLs (hint: "DLL Hell")?

   A: Different Versions of the DLL. No standard location for the DLL. Hard to determine when no application is installed which uses the DLL.

2.4 Environment

1. **[1]** What does the PATH environment variable do?

   A: Dictates where to look for executable programs.

2. **[2]** What environment variable tells X applications where to find the X server which it should communicate with to display the output?

   A: DISPLAY

2.5 X

1. **[3]** The architecture of X was given above, but the applications were shown running on other computers and connected over a network. Draw the architectural diagram if the applications were run on the same computer as the X server (which is the common scenario).

   A: Just put the same os under all the apps and remove the network.

2. **[2]** There are three major desktop environments for Linux, Gnome, KDE, and Xfce. Each of these environments has its own associated libraries which implement core functionality required for all applications. You can, however, run Gnome applications under KDE, KDE applications under Xfce, and so on. Many users, however, prefer to only run applications that have been designed for their desktop environment. What are two reasons for such a preference?

   A: 1. interface consistency: different environments have different UI conventions, 2. efficiency: running applications from multiple environments mean many more libraries must be loaded into RAM

3. **[5]** `cmd.exe` is the standard command interpreter in Windows XP and Vista. Choose five standard shell commands in Linux and give their standard equivalents in `cmd.exe`. For each, briefly state how similar or different it is from its Linux counterpart.