

Introduction to C and Linux

CS681 & CS392 Computer Security
Fall 2004

September 2, 2004

1 Objective

To complete most of the assignments in this course you need some basic understanding of the C Language and Unix operating system. Many of you might already be experts in C and Linux. For those who are not familiar with C and Linux please use this week as much as you can to master some fundamental concepts. If you are an expert treat this assignment as a warm up session.

2 Introduction to C

Almost all of the operating systems and most a majority of applications in use are written in the C Language. Without proper knowledge of C you might not be able to secure a system nor will you be able to write secure code. It is important that you know C if you want to be an expert in Information Assurance (and of course, do cool stuff).

2.1 Warm Up

This section is intended for those who have not programmed in C. We will not grade this section or provide solution.

PROBLEM 1 Write a program that takes a text filename and an unsigned integer n at the command line and displays the last n lines from the text file. If n is 0 or greater than the number line in the test file, your program should display an error message. (Use `argv`, `argc` for reading in command line; Use dynamic memory - `malloc()`, `calloc()`, `realloc()` and `free()`)

PROBLEM 2 Write a program to remove all comments from a C program. Dont forget to handle quoted strings and characters properly. C comments do not nest. Assume that the input is syntactically correct (i.e. it compiles without errors). You should accept the name of the C file to read from the command line.

PROBLEM 3 Write a program that declares an integer, a short integer, a long integer, a float, a double, an integer array, a character array. When executed , the program should print the memory address and size of all its variables.

PROBLEM 4 Use all available string functions (`strcpy`, `strcmp`, `strlen`, `strstr` etc.) in a program. Be creative in defining this problem. Credit will be given for small cute programs.

PROBLEM 5 What is the result of executing the following program segment? Why?

```
int i, test[2];
test[0] = test[1] = -1;
i = 0;
test[i] = i = i + 1;
printf("%d %d\n", test[0], test[1]);
```

2.2 Writing a Shell

A shell is essentially an interface between a user and an operating system. In this section we ask that you write a shell using some basic C functions such as `fork()`, `exec()`, `wait()`, and `signal()`. The purpose of this section is to get you familiar with *nix processes and how to work with them. A good reference on the subject is the free C Book (http://publications.gbdirect.co.uk/c_book/) and the Programming in C: UNIX System Calls and Subroutines using C (<http://www.cs.cf.ac.uk/Dave/C/CE.html>).

This section serves two purposes: 1) you will learn about processes and how to create them, passing arguments to them, inheriting opened files, waiting for processes to terminate, getting their exit status, background processes, and so on. 2) You will use the shell you wrote as a base to create a shell for the rest of the semester. **Note: This section is for everybody and will be graded.**

2.2.1 tinybash

We ask you to design and implement a shell called `tinybash`. This shell is similar to `bash` shell in Linux, which gives you a prompt, accepts commands and executes them. Of course, we are not expecting to create a shell that has all the features of `bash` but only a very small subset of them hence the name `tinybash`.

Command Line - The general syntax for `tinybash` command line should be:

```
command [arg1 arg2 ....] [&]
```

where the arguments inside [] are optional, and a space separates argument. The following general command line option should also be available:

& - can be appended to any argument to make it run in the background. By default all commands should run in the foreground.

Supported Commands - `tinybash` should support the following internal commands:

ls - **List** current directory contents

Synopsis - `ls [-al] [file...]`

Description For each operand that names a file of a type other than directory, `ls` displays its name as well as any requested, associated information. For each operand that names a file of type directory, `ls` displays the names of files contained within that directory, as well as any requested, associated information. The following option are available:

-a List all entries

-l List all information associated with a file.

cp - **Copy** files

Synopsis - `cp [source file] [target file]`

Description `cp` copies the contents of the source file to the target file.

mv - **Move** files

Synopsis - `mv [source file] [target file]`

Description `mv` moves the contents of the source file to the target file.

stt - **Status**

Synopsis - `stt`

Description `stt` will print the exit status of previous command.

xt - **Exit**

Synopsis - `xt`

Description `xt` will terminate `tinybash`.

Executing Commands You should use `fork()`, `exec()`, and `wait()` to execute a command. `tinybash` should only wait for completion of foreground commands before prompting for next command and it should not wait while executing background commands.

You must print process id on screen when a command is executed in the background. If a command terminates due to a signal `tinybash` should print that signal number. A CTRL-C should not kill `tinybash`, it should only kill any foreground command.

All error while executing a command should be handled gracefully.

3 Introduction to Linux

In this section you will learn about basic Linux command, their function and proper usage.

3.1 Linux Trivia

1. What is the first process a typical Linux kernel starts?
2. Explore and report your Linux box's startup procedure in detail. You should read and explain what each startup script does, what executable files they execute.
3. Find a file in the Linux box assigned to you in ISIS that contains the string Hello CS392 and report the command you used to find this file
4. What does the command `strace` do? Run any of the code you compiled in the above section with `strace` and explain the results.
5. Write a shell script to execute all executable files in the current directory and capture their output to file "out.lab0".
6. Modify your Linux box's startup such that it appends the current time and date to a file in root's directory.
7. What command would you use to see the processes that are running currently on your Linux system? Explain how you can kill (stop) a process.
8. How do you setup variables in bash? And how would you remove them with out logging out?
9. What is the command that output the last-bootup time of the system?
10. What is meant by real and effective user-IDs?
11. Categorize the following commands (See Section 3.2) in Linux as one of 1) File and Directory Management, 2) User Management, 3) Process Management, 4) Compiler and Linker, 5) Communication, 6) Editing, or 7) Miscellaneous commands. Also give a brief (one to four line description) for each command. You should describe their use and the most important/popular switches/options used with these command.

3.2 Useful Commands

This section will not be graded and is only for those who are new to Linux/*nix environment. We recommend you to study these commands, understand their purpose and learn to use them properly, this for your own good.

- | | | | |
|------------------------|----------------------|----------------------|-------------------------|
| 1. <code>grep</code> | 2. <code>find</code> | 3. <code>man</code> | 4. <code>slocate</code> |
| 5. <code>strace</code> | 6. <code>ssh</code> | 7. <code>sftp</code> | 8. <code>ls</code> |

- | | | | |
|-------------------------|-----------|----------------------|-------------|
| 9. chmod | 10. chown | 11. chgrp | 12. passwd |
| 13. useradd and adduser | 14. su | 15. vi | 16. rmdir |
| 17. whereis | 18. lsmod | 19. insmod | 20. gcc |
| 21. gdb | 22. make | 23. fdisk and cfdisk | 24. pipe |
| 25. >, >> (redirection) | 26. ln | 27. rm | 28. cp |
| 29. mv | 30. ld | 31. ftp | 32. more |
| 33. less | 34. cat | 35. tar | 36. top |
| 37. ps | 38. kill | 39. df | 40. last |
| 41. gunzip | 42. patch | 43. bzip | 44. bunzip |
| 45. mkdir | | | |

4 What and how to Handin

For this assignment you must hand section 2.2 and 3.1 on or before midnight 09/08/2004. We prefer you to e-mail you solution to vikram@isis.poly.edu with subject line "CS392 lab 0 <your name>" with out the quotes. Subject line is very important if you want to receive any credits.