

Lab 2

SYSTEM CALLS and Programming

ITSC205: Operating Systems Internals

NAME: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Table of Contents

[Lab Outcome(s) 2](#_Toc472385055)

[Reading 2](#_Toc472385056)

[Introduction 2](#_Toc472385057)

[1.0 Linux Shell - Interpreter 3](#_Toc472385058)

[2.0 Programming -Compiler 4](#_Toc472385059)

[3.0 Debuger -dbg 7](#_Toc472385060)

[4.0 System Calls 9](#_Toc472385061)

**EVALUATION**:

|  |  |  |
| --- | --- | --- |
| Differentiate Linux shells and its environment | 5 |  |
| Run and Compile C programs | 10 |  |
| Run and analyze C program - pointers | 10 |  |
| Use **gdb** to debug C code | 10 |  |
| Identify System Calls within C programs | 15 |  |
| TOTAL MARK | 50 |  |

Lab Outcome(s)

* Use Linux operating systems shells (Interpreter) to interact with the kernel services
* Explore Linux Operating systems kernel using kernel debuggers and C programming language
* Use system calls and write programs to invoke services provided by the kernel

Reading

* Textbook chapter 2, sections 2.2 User and Operating System Interface and 2.3 System calls. Chapter 20 (Linux system ) section 20.2.1

Introduction

**System call** is a special function (subroutine) in GNU C library that performs a call to the system (kernel). It is the transition (interface) between user and kernel space. When a program in user space makes a system call, the arguments are packaged up and passed to the kernel, which takes over execution of the program until the call completes. Most of Linux system calls numbers are declared in /usr/include/asm/unistd\_64.h. The transition from user to kernel mode is different in each architecture. Intel may handle differently this transition than ARM processors.

Linux system includes user-mode programs to initialize and manage the system and a program required to interact with the user (shell). The most important user utility is the shell which is the program that interprets commands.

1. Shell – Interpreter \_\_\_/5

The shell is an interface between user and operating system. It is **program** that **interprets commands** line by line.

When Linux system boots, the last program started by **init** script is the ***shell.***

The shell expects an input from standard input (STDIN) such as the keyboard then parses the input line into individual tokens (“words” separated by whitespace) , it organizes the tokens into an array of parameters that are passed to exec () functions. The fork() system call clones a new process (child process) which will execute the program. The execvp() function provide an array of pointers that represent the available list of arguments to the new process (child process). The first argument is the pathname of the program to be executed.

**Environment variables**

Each program, when started, receives a set of *command line parameters* and a set of *environment variables*. To list all environment variables, use command **env**. To find the value of the environment variable PATH, *pipe* the output of command **env** to input of command **grep**

**env | grep PATH**

There are different types of shells and each shell has its own prompt and environment.

1. Use the command **cat /etc/shells** to display current shells supported by Linux.
2. Use the following commands to find the current shell
   1. **echo $0** to find out the name of the program running.
   2. Use the command **echo $SHELL**.
   3. Use the command ***ps*** to list processes.
3. What is the default shell for the current user? Find out by editing the file /etc/passwd
4. To terminate the current shell session you can type the command **exit** or press **Ctrl-d.**
5. Each shell has its own environment. Use the **set | more** or **env | more** commands to verify each shell environment and identify the different system variables. You can also use echo $VARIABLE\_NAME to verify the content of individual variable

Demo to instructor the following:

* 1. In Bash display and explain the purpose PS1 and PATH variables
  2. Modify PS1 variable to modify the prompt
  3. Modify the variable PATH by adding a dot(.) to current path
  4. Display and explain the purpose of the hidden file called .bashrc

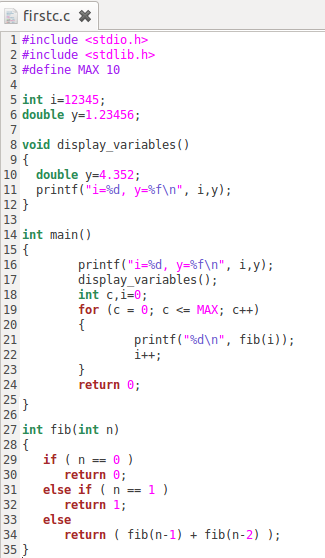
|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

2.0 C Programming \_\_\_\_/5

Most operating systems are written in C programming Language. The objective of this section is to identify C program components, compile and run C code.

1. **Identify C program components**

Use a Linux editor to type the following C program:



1. Add comments to the code to identify and explain briefly the purpose of the following C code components:
   1. Header Files
   2. Constants
   3. Global and Local variables
   4. User-defined functions
   5. Library Functions
2. Save the file as: **firstc.c**
3. Use gcc to compile the program
4. Run the code using ./a.out and option –o and demo and explain the results to the instructor

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

1. **Analyze Compiler Process \_\_\_/5**

Use the respective ***gcc*** options with the ***firstc.c*** code in the right order to analyze the output of compiler process:

* 1. Pre-processor
  2. Assembly code
  3. Object code
  4. Executable

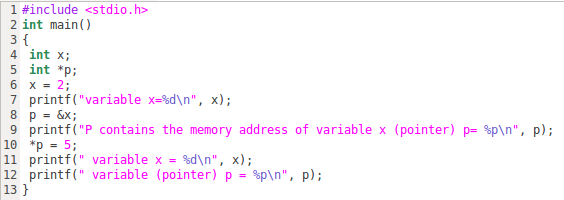
Demo the results to the instructor

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

**C. POINTERS \_\_\_/10**

Access the following URL <http://users.cs.cf.ac.uk/Dave.Marshall/C/node10.html#SECTION001010000000000000000> and read the topic: **“What is a Pointer?”**

1. Type the following code:



2. Save the program as pointers.c

3. Comment the lines that contain variables, provide the variable type

4. Compile and run the program. What is the differences between

p=&x and \*p=5

5. Modify the above pointers.c code as follows:

a. Create a second integer variable ***y***

b. Create a second integer pointer variable ***s*** that will store the address of

the variable ***y***

1. Make variable ***y*** = 10 and display the content of the variable
2. Display the content of the pointer variable ***s***
3. Use the pointer of variable ***y*** to change its value from 10 to 15

7. Compile and run the program. Analyze the results and sketch (picture) each variable with respective content and memory address. Demo the results to the instructor.

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

**3.0 Debugger Program –gdb \_\_\_\_/10**

gdb is a debugger program that allows to verify what is going on when a program is executing or when it crashes.

To verify memory addresses and what is going on the pointers code pointer.c we will use the debugger **gdb**. Use Linux manual to read the purpose of **gdb** and the commands that can be used to trace or troubleshoot C code. For more reference you can use the following web sites:

<ftp://ftp.gnu.org/old-gnu/Manuals/gdb/html_chapter/gdb_2.html> and <ftp://ftp.gnu.org/old-nu/Manuals/gdb/html_chapter/gdb_3.html#SEC7>

1. Compile the program as follows: **gcc -g -w pointers.c**
2. Read “Compiling for debugging” from the following web site <ftp://ftp.gnu.org/old-gnu/Manuals/gdb/html_chapter/gdb_5.html#SEC17> and explain the purpose of –g option.

1. Run the program ./a.out and **analyze the code and results**. Write down the memory address of x, y, p(pointer) and s(pointer) variables.
2. Use gdb to debug the program as follows:
3. ***gdb ./a.out***
4. ***(gdb) break main –*** This will create a breakpoint at main( ) function. You will see the memory address of main( )
5. ***(gdb) run*** *-* It will start the program
6. ***(gdb) n*** - This means next to see the next statement

Verify the content of the variables

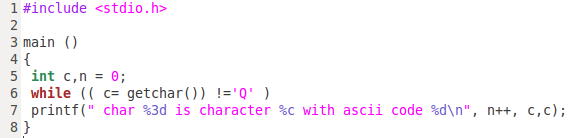
1. ***(gdb) print x*** It verifies x content
2. ***(gdb) print y*** It verifies y content
3. ***(gdb) print p*** It verifies p content (This is a pointer the content of this variable should be the address pointing to x variable)
4. ***(gdb) print s*** It verifies s content (This is a pointer the content of this variable should be the address pointing to y variable)
5. ***(gdb) n*** It will move to the next statement . Repeat this command to analyze the variable’ content in each step.
6. ***(gdb) x address*** ( replace address with the memory address of variable p (pointer). Memory addresses are in hexadecimal system you need 0x ) e.g ***(gdb)*** ***x*** 0xbffff6f8 this should display the content of this memory address
7. Use ***(gdb) x*** address to analyze the addresses of the pointers p and s and its contents.
8. Use the previous gdb commands to analyze the results of each step or statement of the pointers.c program and sketch the variables with respective memory address and content.
9. Demo the results to the instructor.

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

4.0 System calls \_\_\_\_/15

Every time a program makes a **system call**, the system has to change from user mode to kernel mode. Strace program can be used to display **system calls** when a program is running.

1. Compile and run the following C program



* 1. Comment each line explaining the purpose of each line in the program
  2. Compile this program using the option -o with gcc.
  3. Run the program analyze the results, what is the output of this program?
  4. Use the command **strace -c ./executable** (Replace the program-name with the executable file). Press the following characters A,z,2,Q and capture the screen that contains the output.
  5. Identify and write down the system calls used by this program.
  6. Run the program and demo the results to the instructor

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

**Trivial SHELL program**

Download **trivialshell.c** program from **D2L.**  This program illustrates several key concepts on **how the shell works**. Section 3 of Linux man contains information about exec( ) function : **man 3 exec**

1. This program will generate warning messages during compilation. Use the **warning option** with gcc to display them. These warnings will not affect the results of the program.

2. Run the program. While running the program issue some commands with options and arguments. Terminate the program by pressing Ctrl-c keys.

3. What is the purpose of this program?.

4. Use the command **strace -c ./program-name** (Replace the program-name with the executable file). Issue some commands with options and arguments. Terminate the program by pressing Ctrl-c keys. Identify and write down the **system calls** used by this program.

5. Demo and explain the results to the instructor

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |