

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 Windows Command-Line Interface 5](#_Toc472385059)

[3.0 Windows Management Instrumentation WMIC 6](#_Toc472385060)

[4.0 Analyze Compilation Process 7](#_Toc472385061)

[5.0 Debugger gdb to analyze Pointers 9](#_Toc472385061)

6.0 System Calls .………………………………………………………………………………........12

L*abs must be submitted by the due date for full credit. After due date late submissions will be accepted for a period of one week (seven days) and the grade will be reduced by ten percent (10%) per day after due day.* ***Assignments that are submitted more than seven days late will receive a grade of zero (0).***

I certify that the work submitted in this assignment is my own and that it has not been taken in whole or in part from any other source. I understand that the penalty for plagiarism will include a grade of zero (0) for this assignment plus disciplinary action in accordance with SAIT policies.

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**EVALUATION**:

|  |  |  |
| --- | --- | --- |
| Differentiate Linux shells and its environment | 7 |  |
| Windows Command line interface | 8 |  |
| Windows Management Instrumentation- WMIC | 10 |  |
| Analyze compilation process | 10 |  |
| Run and analyze C program - pointers | 10 |  |
| Use **gdb** to debug C code | 15 |  |
| Identify System Calls within C programs | 10 |  |
| TOTAL MARK | 70 |  |

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. X64 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.

<https://www.kernel.org/doc/html/latest/process/adding-syscalls.html?highlight=system%20calls>

1. Shell – Interpreter \_\_\_/7

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.

1. Use Ubuntu VM or any Linux distribution to complete the following tasks

**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. ( 2 marks) What is the default shell for the current user?
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

1. (2 marks) Explore Bash variables and explain the purpose of PS1 and PATH variables
2. (3 Marks) Attach a screen capture to demo the following:
   1. Use the respective environment variable to modify the prompt to display

your-name followed by >

* 1. Add to variable PATH your home directory
  2. Display and explain the purpose of the hidden file called **.bashrc**

2.0 Windows Command-Line Interface \_\_\_\_/8

User can interact with Windows operating system via command-line interface similar to Linux shell, but with subtle syntax differences.

1. Use your host Windows machine or VM windows machine to complete the following tasks.
2. ( 3 marks) Use the respective Windows’ commands to display the following. Attach the screen capture to demo the results.
3. hostname
4. change to directory that contains system internals tools

( program installed in Lab 1 )

1. Display hidden files
2. ( 2 marks) The SET command displays all system variables at one time. Use SET to find the value of the following environment variables:
   1. ALLUSERSPROFILE=
   2. NUMBER\_OF\_PROCESSORS=
   3. PROCESSOR\_ARCHITECTURE=
   4. PROMPT=
3. ( 2 marks) Change the prompt variable to display the date followed by $. Attach a screen capture that demo results
4. (1 mark) What command will display the content of variable PATH?

**3.0 WMIC (Windows Management Instrumentation) \_\_\_/10**

Windows includes Wmic.exe utility that allows you to interact with WMI from a WMI-aware command-line shell. All WMI objects and their properties, including their methods, are accessible through the shell, which makes WMIC an advanced systems management console.

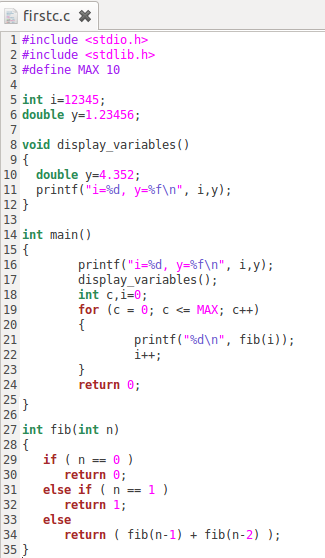
1. To switch from the standard windows shell into WMI interface do the following:
   1. Start **cmd** and type the following command: **wmic /?** to display wmic switches and objects such as BIOS, CPU, OS, etc. Every element in the system is an object. To display object properties use **get** method as follows:
   2. To verify the properties for the OS object type: **os get /?**
   3. To get OS version type: **os get version**
2. Use wmic to verify from OS object the following properties:
   1. Last Boot up time,
   2. Windows Directory and
   3. Total Virtual memory size
3. Use wmic to explore the following objects and its properties and answer the questions: (provide respective units bit , Bytes or Hrz )
   1. CPU:
      1. What is the manufacturer?
      2. What is the L2cache size?
      3. What is the MaxClockSpeed?
   2. MEMORYCHIP:
      1. How many memory banks are there in the system?
      2. What is the manufacturer?
      3. What is the speed?
   3. MEMPHYSICAL:
      1. What is the description?
      2. What is the Max Capacity?
4. You can use WMI with power shell to manage system’s objects. To verify if **Windows update** was disabled do the following:
   1. Start Windows Powershell
   2. Type : get-wmiobject win32\_service -filter "name='wuauserv'"
   3. Attach a screen capture that verifies Windows Update service and status

4.0 Analyze Compilation Process \_\_\_\_/10

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. Attach a screen capture that demo results
5. **Analyze Compilation Process**

Use the respective ***gcc*** options with the ***firstc.c*** code in the right order to analyze the output of compilation process. Write down the gcc options used to display the following:

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

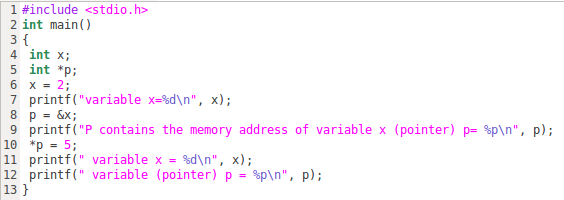
Attach screen captures to demo results

**5.0 Debugger gdb to analyze Pointers \_\_\_/10**

1. **Pointers**

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. **( 3 marks)** Compile and run the program. What is the differences between p=&x and \*p=5 ?

4. 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 memory address of variable ***y***
3. Display the content of the pointer variable ***s***
4. Display memory address of ***pointer s***
5. Use the pointer of variable ***y*** to change its value from 10 to 15
6. Compile and run the program.
7. **( 7 marks)** Attach modified code and a screen capture with results
8. **Debugger Program –gdb \_\_\_/15**

gdb (GNU debugger) is a program that allows to analyze a program while executing or when it crashes.

To verify memory addresses and 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-gnu/Manuals/gdb/html_chapter/gdb_3.html#SEC7>

1. Compile the program as follows: **gcc -g -w pointers.c**
2. **(2 marks)** 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. **(2 marks)** 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

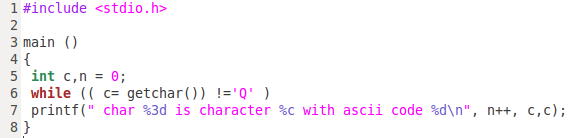
1. ***(gdb) print x*** It displays x content
2. ***(gdb) print &y*** It displays y memory address
3. ***(gdb) print p*** It displays p content (This is a pointer the content of this variable should be the address pointing to x variable)
4. ***(gdb) print s*** It displays s content (This is a pointer 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. **( 5 marks)** Attach a screen capture that demo the gdb commands you practiced and results
9. **(2 marks)** Use gdb documentation or tutorials and find gdb command that:
   1. displays the next instruction to be executed that is controlled by the program counter.
   2. displays the assembly code of this program

1. **(4 marks)** Use gdb commands to analyze the results of each step or statement of the pointers.c program and attach a screen capture that displays the **content** and the **memory address** of the following variables: x and y and pointer p and s

6.0 System calls \_\_\_\_/10

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 executable with the respective executable file generated by gcc). Press the following characters A,z,2,Q and attach a screen capture to demo results
  5. Identify system calls used by this program and use the respective header **/usr/include/x86\_64-linux-gnu/asm/unistd\_64.h** to identify respective **system call numbers**. Write down the **system calls** used by this program with the respective **system call numbers**

**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 hide warnings. 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. Attach a screen capture to demo the results of this program

5. 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.

6. Attach the screen capture that demo the system calls used by this program