EEL-4736/EEL-5737 Principles of Computer System Design

Homework #1

Assigned: 8/21/2017; Due: 5 pm on 9/4/2017 - To be done individually

This assignment consists of two parts and will prepare you with an environment that will be used for development of design assignments (and a final project) throughout the semester. The assignments/project will be based on a file system design of increasing complexity, based on the Linux FUSE file system, and the Python programming language. **Part A** will guide you through the setup and configuration of a virtual machine environment with Linux for development and testing of your assignments/project; it will also expose you to the Python language through a brief tutorial and test. The goal is to get you prepared with the environment that will be used throughout the semester, and to ensure you are able to go through the basic steps in programming and use of this environment. **Part B** will guide you through examples to learn about the Linux system call interface and the architecture of the FUSE file system.

Part A

The setup consists of a virtual machine running the Linux operating system, the FUSE (file systems in user space) module, and Python bindings for FUSE. It is assumed that you have an x86-based Windows, Linux or Mac laptop, 4GB+ of RAM, and around 16GB of free disk space to set up this environment. If you are unsure if your computer meets these requirements, please contact the TA or instructor. If you have any problems during these steps, it is suggested that you use the class discussion list to ask and answer questions. That way you may get a response from the TA, instructor or fellow students, and others who might have the same question will benefit from it too.

1) Install a virtual machine monitor in your computer. There are two choices of free virtual machine technologies suggested – VirtualBox (www.virtualbox.org), or you can also use VMware Player (www.vmware.com). You only need one of these installed and either will work for the class, but VirtualBox is recommended if you don't have a virtual machine in your computer yet.

2) Download a Ubuntu Linux 16.04 64-bit desktop edition ISO from Ubuntu website. Creating a new virtual machine and installing the operating system is straightforward. Just use the default options in VirtualBox/VMware and follow default the installation steps. It is recommended that you have these at least 1 GB of RAM and 8 GB of hard disk for the VM.

3) Boot up your virtual machine and familiarize yourself with the Ubuntu environment. If you are unfamiliar with UNIX or Linux, it is highly recommended that you go through a tutorial. There are various tutorials online on sites such as http://www.ee.surrey.ac.uk/Teaching/Unix/, http://www.tldp.org/LDP/intro-linux/html/, and YouTube.

4) Install software pre-requisites for the file systems in user space (FUSE); in a command line terminal, type:

```
sudo apt-get update
sudo apt-get install libfuse-dev git python-setuptools
```

5) Download and install the fusepy Python-based FUSE file system, following the instructions from: https://github.com/terencehonles/fusepy (hint: use the git clone command with this URL to download the entire repository).
Test the FUSE file system with the example memory.py code. Memory.py implements a simple in-memory, one-level file system. You can test it by running a sequence of commands like the following (in the directory where you downloaded memory.py to). On one terminal window, mount the FUSE file system:

```
cd fusepy
mkdir fusemount
cp examples/memory.py .
python memory.py fusemount
```

Note: when you mount the file system, it may print warning messages regarding logging – don't worry about it. if you get a permission denied error like "failed to open /etc/fuse.conf: Permission denied", change the permissions of the file.

```
sudo chmod 755 /etc/fuse.conf
```

On another terminal window/tab, create a file on the mounted file system:

```
echo "hello world" > fusemount/hello.txt
ls –l fusemount
cat fusemount/hello.txt
```

Unmount the FUSE file system:

```
fusermount –u fusemount
```

6) Go through the following Google Education Python tutorial and complete the basic exercises by following these URLs:

https://developers.google.com/edu/python

https://developers.google.com/edu/python/exercises/basic

Part A Questions:

A.1 Summarize: what is FUSE, why it is useful, and how is it architected. You should research appropriate resources on the Internet to formulate your answer, but use your own words in your answer – do not plagiarize. Also, make sure to include references.

A.2 What happens when you mount and unmount the FUSE file system?

A.3 Where in hardware (e.g. memory, disk) is the "hello world" string stored after the echo command runs? Through what abstraction (e.g. virtual memory, file system) is this content accessible? Explain.

Part B

In this section, you will use the setup you created in Part A to observe the use of the system call interface and the architecture of FUSE (File Systems in User Space), as described below.

You will use two mechanisms to inspect the system call interface and FUSE:

a) You will use the "strace" Linux command, which captures and displays all system calls issued by an application, showing the system call type, arguments, and return values. Type "man strace" on your Linux machine and/or search the Internet for "strace" to learn more about this command

b) You will use the output of the memory.py FUSE file system, which provides a trace of operations that take place in the FUSE file system. FUSE essentially provides an application programming interface (API) and a program that runs along with the O/S kernel to allow the logic of a file system to be implemented by a user-level application. When you mount a FUSE file system, all file operations that reference names in the file system namespace rooted at the directory you mounted (e.g. /home/user/fusemount) are redirected from the OS kernel to the application that is bound to the FUSE file system (memory.py in this example), and each file system call is interpreted by the user-level application. Search the Internet for FUSE to learn more about this system; sources of information and pointer include the FUSE project web page and the Wikipedia entry for FUSE:

https://github.com/libfuse/libfuse

http://en.wikipedia.org/wiki/Filesystem_in_Userspace

Mount the memory file system as explained in Part A. Enable debugging in memory.py by passing "debug=True" parameter in the source code where the fuse object is instantiated.

```
fuse = FUSE(Memory(), argv[1], foreground=True, debug=True)
```

In another terminal, run the following commands under "strace" so you can capture a trace of system calls (the following command will trace only file descriptor-related system calls and show all arguments and return values). Run one command at a time, take the time to inspect the output logs of both strace and memory.py at each step to answer the questions below.

Make sure that you are in the fusemount directory from Part A, you use the "cd" command to change directory.

```
strace -v -e trace=desc ls
strace -v -e trace=desc touch myfile.txt
strace -v -e trace=desc echo "hello" > hello.txt
```

return to your home directory

```
cd ~
strace -v -e trace=desc cat <"path to the fusepy directory relative to home dir">/fusemount/hello.txt
```

example:

strace -v -e trace=desc cat fusepy/fusemount/myfile.txt

Part B Questions:

B.1 Select one of the system calls shown in the strace output and describe the following components of the system call you chose:

name, arguments, return value; caller, callee; layer of the caller (app, O/S), and layer of the callee.

B.2 Describe in one sentence the purpose of the following commands: ls, touch, echo.

B.3 For the third "strace" (which executes the "echo" command), which functions of the memory.py program are executed and, what is the return value for each function that is called. (Hint: you have to look at the output of memory.py as well as the source code of memory.py to answer this question)

B.4 Referring to Figure 2.5 (p. 54), what are three important instruction references that are at play in this exercise? Describe one instance where a change of environment reference occurs. (Hint: consider both applications and the O/S kernel; read the FUSE documentation https://github.com/libfuse/libfuse to get a better understanding of the different execution environments)

B.5 Consider the first command "ls". Inspect the "open()" system calls at the output of strace (ignore the other system calls). a.) Explain in one sentence, what does open() do? b.) Why are there several "open()" calls that refer to files in pathnames that begin with "/lib" and "/usr/lib"? c.) Which "open()" system call(s) result in call(s) to the FUSE file system?

B.6 Consider the fourth command "cat". Inspect the "open()" system calls at the output of strace (ignore the other system calls). a.) What is the absolute path name of the file you are creating in the context of the kernel's file name resolver? b.) What is the absolute path name in the context of the FUSE name resolver?

Expected Submission format:

Turn in through CANVAS a zip file containing the following:

1. A PDF document with answers for questions in Part A and B

2. Python code solutions for Step 6 in Part A (named list1.py, string1.py, wordcount.py)