CS 307
Assignment-2

Problem 1 – Compiling the Linux kernel from source and building a loadable module (15 marks)

In this problem, you will compile a custom Linux kernel from its source. Then you will build a simple loadable module in the new kernel and try out some ways of crashing the system. Some benefits of a custom Linux kernel are:

1. Support a wide range of hardware including the latest hardware.
2. Remove unwanted drivers from the kernel.
3. Faster boot time due to small kernel size.
4. Increased security due to additional or removed modules/drivers/features.
5. You will learn about the kernel and advanced usage.
6. Always run the cutting edge latest kernel.
7. Lower memory usage.

Note: Using a virtual machine is not necessary to compile/install a custom Linux kernel. Doing it directly on your work system can be dangerous, especially when you are writing kernel modules to deliberately crash the system.

Custom Linux Kernel

1. First, decide which Linux kernel you want to install. For this assignment, you should install 5.9 or 5.10.
2. Setup a virtual machine, preferably using VirtualBox. You can use Lubuntu, Ubuntu or some other Linux distribution. Kernel compilation takes too much space, so try to allocate at least 40-50GBs to the virtual machine.
3. Make sure your VirtualBox supports the kernel you want to build. Check the release notes here: https://www.virtualbox.org/wiki/Changelog. If not, upgrade your VirtualBox installation.
4. Download the source code of the specific kernel version you want to build.
5. To build and install a custom kernel, you can check these resources:
    a. https://help.ubuntu.com/community/Kernel/Compile
    b. https://www.linode.com/docs/guides/custom-compiled-kerneldebian-ubuntu/
    c. DuckDuckGo/Google/Bing
6. Try to reduce the size of the custom kernel and compare the size of your custom kernel with the default kernel. Report your findings.

Hint: Do you need all the modules? What about debug symbols?

Building a simple loadable module

1. After the installation is complete, you will build a simple loadable module in Linux. You can follow the steps from:
    a. http://www.iitk.ac.in/LDP/HOWTO/pdf/Module-HOWTO.pdf
    b. https://scottc130.medium.com/writing-your-first-kernel-module-98ae68edf0e
    c.http://www.hitchhikersguidetolearning.com/2018/08/19/writing-a-simple-linuxkernel-module/
    d. DuckDuckGo/Google/Bing
2. Introduce some errors in the loadable module e.g. division by 0, returning a value other than zero from init mod, etc. See if you are able to crash the system. Try atleast 3-4 ways and return values and infer the results.
3. Store the bad modules for demonstration.

Problem -2 Linux Processes (10 marks)

Create four children C1 to C4, each having a pipe to communicate with the parent, where :

[i] C1 reads from keyboard and encrypt it using a mapping table and send it to Parent (eg A converted to F, G to I etc). It reads the mapping table from a predefined file.

[ii] C2 reads data from a text file and speaks it using “espeak” linux utility.

[iii] C3 copies a given input file. Here do not use the linux “cp” utility. Code the copy function.

[iv] C4 does CPU monitoring (total %CPU used etc) and periodically sends stats to the parent.

The parent reads from the four pipes and prints on the display. (Hint: use select())
Optional : Write a script to monitor CPU utilization of the above program.

Problem -3 Multithreaded MergeSort (15 marks)

You need to implement Merge Sort using threads. The sorting should be in place and You will be provided a file of 2GB data which you need to sort using the merge sort algorithm implemented. You are free to allocate some additional memory to keep temporary data if needed.

Rules:

1. The number of threads (first argument) and size of input array (second argument) will be passed as command line arguments. Please do not hard code the number of threads or

the size of the array. Rounding the number of threads down to the nearest power of 2 is acceptable.

2. Using the size of the array, create a randomly generated integer-type array.
3. Output the initial unsorted array and the final sorted array along with the total execution time.

Example I/O:
>> ./myprogram <number of threads> <input file path> <output file path>
Total execution time : [total execution time]