CS60038: Advances in Operating Systems Design

Autumn 2022

Assignment 1

Building and installing the Linux kernel and developing a loadable kernel module

Assignment Date: 19 August 2022
Submission Date: September 18 2022 EOD (Via Moodle)

The objective of this assignment is to get hands-on experience in building the Linux kernel from the source. This will help us to get familiar with the process of configuring, building, compiling, installing, and booting up a kernel. The second part of this assignment aims to develop a basic loadable kernel module.

For the assignment concerning configuring and building of Linux kernel or developing the Loadable kernel Module, it is recommended that you use kernel version **5.6.9**. For downloading the tarball of the kernel you can run the following command from the terminal.

```
wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.6.9.tar.gz
```

A Virtual Machine (VM) with 64-bit Ubuntu 20.04 OS is already provided to you. If in case you are doing the assignment from your own VM, you can download the 64-bit Ubuntu 20.04 Desktop Image from this link. Please note that all the assignments will be evaluated on this OS and kernel version only.

Part A: Configuring and building the Linux kernel

Objective: In this assignment, students need to configure, build, and install a Linux kernel from the source.

Students need to configure, compile and install a fresh kernel in the system. As we have demonstrated the process of setting or changing the different configurations in the kernel from the `menuconfig` during the tutorial class, the students need to configure the modules in their kernel build in the following three different ways.

1. Remove AppArmor support
2. Remove DCCP protocol

3. Update default TCP congestion control algorithm to Reno

The students need to submit the config file and a report describing the changes they are observing after installing these kernels in the system.

AppArmor protects Linux systems from insecure or untrusted processes by running them in restricted confinement, while still allowing processes to share files, exercise privilege, and communicate with other processes. You can check whether the kernel version you have includes AppArmor support by using the following command: `cat /sys/kernel/security/apparmor/profile`

DCCP is a general-purpose transport protocol that provides the establishment and maintenance of unreliable packet flow and congestion control. Use the following command for checking if DCCP is disabled: `grep -r dccp /etc/modprobe.d/* | grep -i "/bin/true"`

TCP Reno detects lost packets early as half of the current congestion window is saved and it skips slow start by going directly to the congestion avoidance algorithm. To check the current TCP congestion control algorithm, use the command: `cat /proc/sys/net/ipv4/tcp_congestion_control`

Part B: Assignments on Loadable kernel module

Objective: In this part of the assignment, the students need to develop a loadable kernel module for doing various jobs inside the kernel space. In addition to this, the students need to handle concurrency, mutual exclusion, memory management, process management, and io-control.

Please note that all the assignments will be evaluated on Ubuntu 20.04 LTS and kernel 5.6.9 version only.

In this assignment, you need to write a loadable kernel module (LKM) that provides the functionality of a Priority-Queue of max **Capacity ≤ N** inside the kernel mode. Your LKM should be able to handle 32-bit integers. Upon insertion of this LKM, it will create a file at the path **/proc/partb_1_<group_no>**. (Group No is already shared with you in the spreadsheet of VM details). This path will be world-readable and writable. A user-space program will interact with the LKM through this file.

A userspace process can interact with the LKM in the following manner only.

Step 1. It will open the file (**/proc/partb_1_<group_no>**) in read-write mode.

Step 2. Write one byte of data to the file to initialize the Priority-Queue.

i) The first byte should contain the maximum capacity N of the Priority-Queue. The size N should be between 1 to 100 (including 1 and 100). Other than these values produce an EINVAL error, and LKM is left uninitialized.

Step 3. Next, write calls should pass integers first and then their priority in the proc file. (1 at a time). Thus at max, there can be 2N+1 lines in the proc file. LKM will insert these elements in the Priority-Queue based on the associated priority of the element.

- **An element with a priority of lower integer value is inserted first in the Queue and dequeued first before an element with a priority of higher integer value.**
- **If two elements have the same priority, they are served according to their order of insertion in the queue. (In this case, an element that is inserted earlier in the queue is dequeued first).**
- **The associated priority value is always a positive integer i.e., 1, 2, 3 ….**

LKM must produce an invalid argument error in case of a wrong argument (i.e, unexpected type). On a successful write call, LKM will return the number of bytes written (4 bytes for 32-bit integers). LKM will produce an EACCES error for any excess write calls (> 2N+1) and return -1.
Step 4. As integers are written one by one in Step 3, they are inserted into the Priority Queue. In between, there also might also be intermediate read calls as explained in Step 5.
Step 5. Read calls should read the integer with the priority of lower integer value first. With each read call, the element gets popped out from the Priority-Queue. LKM will produce an EACCES error for any excess read calls (when the size of Priority Queue is 0). In case of a successful call, it will return the number of bytes read (4 bytes), and in case of an error, it will return -1.
Step 6. Userspace process closes the file.

Your LKM implementation should be able to handle concurrency and separate data from multiple processes. Also, no userspace program should be able to open the file more than once simultaneously. However, the userspace process should be able to reset the LKM (for the said process) by reopening the file. LKM needs to free up any resources it allocated for the process when it (the process) closes the file.