CS60038: Assignment 1

Building and installing the Linux kernel and developing a loadable kernel module
(Submission Deadline: September 03, 2023 EOD)

3. You need to submit the assignment through CSE Moodle: https://moodlecse.iitkgp.ac.in/. The course joining key is: STUAOSD

4. Only one member from each group should submit the assignment solution as a single zip file.

5. Mention the name and roll numbers of the group members in your submission.

6. Please document your implementation properly. You should include a README file explaining how to compile and run your code, as well as the test cases that you have tested.

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.

You can download the 64-bit Ubuntu 20.04 LTS Desktop Image and use that in a Virtual Machine (VM). Please note that all the assignments will be evaluated on this platform and kernel version only.

For the assignment concerning configuring and building of the Linux kernel, you need to use kernel version 5.10.191.

Part A: Configuring and building Linux kernel

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

You need to make the following configurations using **`menuconfig`**, and then build and install the compiled kernel over the Ubuntu 20.04 LTS Desktop version. You need to download the kernel source from https://www.kernel.org/ only. Do not use any other third-party download servers.

1. Remove NUMA memory allocation, scheduler, and emulation
2. Remove Kyber I/O Scheduler
3. Include multipath TCP (MPTCP)

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

NUMA stands for non-uniform memory architecture (or access). It is a type of physical memory design used in shared memory processing (or symmetric multiprocessing)architectures. This allows the processor to have faster access to its local memories in comparison to the shared or non-local memory. If NUMA is enabled in the BIOS; you can see the list of nodes using the command **`numactl --hardware`**.

Kyber I/O scheduler is an I/O scheduler intended to work for fast multi-queue storage devices. You can check whether the kernel version you have includes the Kyber I/O scheduler by checking the directory `/lib/modules/<your kernel version>/kernel/block/` [Replace your kernel version with the kernel version on which you are checking. To know the kernel version you can use the command **`uname -a`**]. If it is present you will see an entry like **`kyber-iosched.ko`** present there. For more information regarding such schedulers and their working look into this blog.

For more information on MPTCP, you first need to check the version of MPTCP that you are using in your system. To check that use the command **`dmesg | grep MPTCP`**. Alternatively, to have more fun, you can also check with the command **`curl http://www.multipath-tcp.org`.**

Part B: Assignments on Loadable kernel module

Objective: In this part of the assignment, you need to develop a loadable kernel module for doing various jobs inside the kernel space. In addition to this, you need to handle concurrency, mutual exclusion, memory management, process management, and io-control. You need to use 64-bit Ubuntu 20.04 Desktop with Kernel 5.10.191 (which you have compiled and installed on your VM).

In this assignment, you need to write a loadable kernel module (LKM) that provides the functionality of a **Deque** (double-ended 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_<roll_nos>`**. This path will be world-readable and writable. A userspace program will interact with the LKM through this file.

A user-space process can interact with the LKM in the following manner only.

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

Step 2. Write one byte of data to the file to initialize the heap.

i) The first byte should contain the maximum capacity N of the deque. The size N should be in between 1 to 100 (including 1 and 100). If N is not within this range, produce an EINVAL error, and the LKM is left uninitialized.

Step 3. Next, you need to implement write calls that should pass integers to be inserted in the queue (one integer at a time). LKM will insert the integer on the left of the queue if it is odd and to the right of the queue if it is even. LKM must produce an invalid argument error in case a wrong argument is given (i.e, unsupported type). On a successful write call, LKM will return the number of bytes written (4 bytes for 32-bit integers). LKM will produce EACCES error for any excess write calls (> N) and return -EACCES.

Step 4. As integers are written one by one in Step 3, they are inserted into the deque. In between, there might also be intermediate read calls as explained in Step 5.

Step 5. Read calls should read the integer from left only. LKM will produce EACCES error for any excess read calls (when size of deque 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 -EACCES.

Step 6. Userspace process closes the file.

LKM 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. Test your LKM implementation with multiple user-space processes writing and reading data over a Dequeue.