CS60038: Assignment 1

Building and installing the Linux kernel and developing loadable kernel modules

Assignment Date: 16th September 2020
Due Date: 07th October, 2020

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 18.04 Desktop Image from this link. Please note that all the assignments will be evaluated on this platform and kernel version only.

For the assignment concerning configuring and building of Linux kernel, it is recommended that you use the kernel version 5.2.6 [as demonstrated during the tutorial]. 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-5.2.6.tar.xz
```

Part A: Configuring and building Linux kernel

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

Students need to configure a fresh kernel in two different ways and install it 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 two different ways.

1. Do not include any Bluetooth Subsystem Support.
2. Update and make **Reno** as the default TCP congestion control algorithm.

The students need to submit the config file and a report describing the changes they are observing after installing these kernels in the system. For more details about system information, the students are advised to look at the **`/proc/sys/net/ipv4/`** file system and the system command **`sysctl`**.

For your information: The default congestion control algorithm can be changed system-wide using the **`sysctl`** command with the key **`net.ipv4.tcp_congestion_control`**. However, one can only change the algorithms which are listed in **`net.ipv4.tcp_available_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.

You can download the 64-bit Ubuntu 18.04 Desktop Image from this link and use that in a VM. Please note that all the assignments will be evaluated on this platform and kernel version only.

Assignment 1

In this assignment, the students need to write a loadable kernel module that provides the functionality of a heap (of size ≤ N) inside the kernel mode. Your LKM should be able to handle 32-bit integers. Moreover, the heap can be min-heap or max-heap. Upon insertion of this LKM, it will create a file at the path `/proc/partb_1_<roll no>`. This path will be world-readable and writable. A userspace 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_<roll no>**) in read-write mode.

Step 2. Write two bytes of data to the file to initialize the heap.

i) The first byte should contain either 0xFF or 0xF0. It informs the expected heap type. Here 0xFF means min-heap and 0xF0 means max-heap. Other than these values will cause an error of type EINVAL and LKM left uninitialized.

ii) The second byte should contain the size N of the heap. The size N should be in between 1 to 100 (including 1 and 100). Other than these values produce an EINVAL error, and LKM left uninitialized.

Step 3. Next, write calls should pass integers to be inserted in the heap (1 at a time). LKM must produce an invalid argument error in case of 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 EACCES error for any excess write calls (> N) and return -1.

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

Step 5. Read calls should read the min/max integer by extract-min or extract-max according to the min-heap or max-heap respectively. LKM will produce EACCES error for any excess read calls (when size of heap 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.

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.

Assignment 2

Students now need to develop an LKM that supports various ioctl calls to set and retrieve various configurations and internal information respectively. The following table describes the ioctl commands and their task. Here, upon insertion of this LKM, it will create a file at the path **/proc/partb_2_<roll no>**. This path will be world-readable and writable. A userspace program will interact with the LKM through this file. A userspace program can fire ioctl system call any time it wants.

Command	Description
PB2_SET_TYPE	This command initializes the LKM with the expected heap type and size. If LKM is already initialized for the current process, this command resets the LKM and reinitializes again. It expects a pointer to a structure of type **struct pb2_set_type_arguments,** which contains two integers **heap_size** and **heap_type.** The **heap_type** values 0 and 1 represent min-heap and max-heap respectively. Invalid value or inaccessible pointer causes error and sets error number to EINVAL. On success, it returns 0. No other operation can be performed (including read or write) before performing this system call. Every other system call on the LKM returns error (-1) with code EACCES.
PB2_INSERT	This command inserts an integer into the heap. It takes a pointer to integer as the input value. Invalid value causes an error in LKM and sets error number to EINVAL. On success, this command returns 0. If the heap is full, it returns error (-1) with code EACCES.
PB2_GET_INFO	This command returns the information about the heap. PB2_GET_INFO command expects a pointer to a structure object of type **struct obj_info** as the value. On success, LKM returns 0, and fills the various fields of the pointer passed by the process. On error, LKM sets appropriate error no and returns -1.
PB2_EXTRACT	This extracts the max or min according to the type of the heap and gives the result by filling the fields of the pointer to **struct result** structure passed by the process. This function returns 0 on success and -1 on error.

The definition of ioctl commands are as follows:

```
#define PB2_SET_TYPE    _IOW(0x10, 0x31, int32_t*)
#define PB2_INSERT      _IOW(0x10, 0x32, int32_t*)
#define PB2_GET_INFO    _IOR(0x10, 0x33, int32_t*)
#define PB2_EXTRACT     _IOR(0x10, 0x34, int32_t*)
```

The definition of structures are as follows:

```
struct pb2_set_type_arguments {
	int32_t heap_size; // size of the heap
	int32_t heap_type; // heap type: 0 for min-heap, 1 for max-heap
};


struct obj_info {
    int32_t heap_size; // current number of elements in heap (current size of the heap)
    int32_t heap_type; // heap type: 0 for min-heap, 1 for max-heap
    int32_t root; // value of the root node of the heap (null if size is 0).
    int32_t last_inserted; // value of the last element inserted in the heap.
};

struct result {
    int32_t result;         // value of min/max element extracted.
    int32_t heap_size;     // current size of the heap after extracting.
};
```

A userspace process interacts with the LKM in the following manner only.

1. It will open the file (**/proc/partb_2_<roll no>**) in read-write mode and obtain the file descriptor.
2. Userspace process calls the ioctl system call with command PB2_SET_TYPE and appropriate value to set the expected heap type and size in the kernel.
3. Insert object using the ioctl system call with command PB2_INSERT as many times as you want. However, there is no bound on the number of times the user program can call this insert operation. LKM verifies the arguments in the system call and inserts the input into the heap. In case the heap is full, an appropriate error is returned.
4. To perform extract-min or extract-max, the userspace program uses ioctl system call with command PB2_EXTRACT. The system call returns 0 in case of successful execution and populates the **struct result,** whose pointer is passed while making the call.
5. When all the operations are done, the userspace process closes the file and LKM releases all the resources allocated for the process.
6. In between the userspace program can call the ioctl system call with command PB2_GET_INFO to retrieve information and current state of the heap.

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.

Important: The students have to submit the LKM source code and Makefile. In every source code file, students need to comment their name, roll number, and the kernel version they have used to build the LKM at the beginning of the file.

Do not make any assumptions. If you find anything ambiguous in the assignment statement, do not hesitate to post it on Microsoft Teams.

DO NOT COPY CODE FROM OTHER GROUPS. WE WILL NOT EVALUATE A SOLUTION IF FOUND COPIED. ALL INVOLVED GROUPS WILL BE AWARDED ZERO.

CLARIFICATIONS

Part B, Assignment 1

0xFF and 0xF0 are not strings, they are hexadecimal representations.

LKM will produce EACCES error for any read calls before the heap is initialized.

If the read call tries to read less than 4 bytes, then also an element will be removed from the heap (extract-min or extract-max). Only the requested number of bytes will be copied to the userspace buffer, and the rest will be ignored. For the next read calls, the next element will be extracted.
read call will always return the number of bytes read.

There should be separate heap for every user process.

Part B, Assignment 2

The struct obj_info.heap_size will give the current heap size.

```
struct obj_info {
    int32_t heap_size; // current number of elements in heap (current size of the heap)
    int32_t heap_type; // heap type: 0 for min-heap, 1 for max-heap
    int32_t root; // value of the root node of the heap (null if size is 0).
    int32_t last_inserted; // value of the last element inserted in the heap.
};
```

There should be separate heap for every user process.