CS60038: Assignment 1

Building and installing the Linux kernel and developing loadable kernel modules

Assignment Date: 7th August 2019
Deadline: 28th August 2019, EOD

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.

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 3 different ways and install it in the system. As we have demonstrated the process of setting or changing the TCP congestion control algorithm in the kernel during the tutorial class, the students need to configure the modules in their kernel build in the following three different ways.

1. Build CUBIC TCP as the loadable kernel module (LKM) and make DCTCP as the default congestion control algorithm
2. Do not build any loadable TCP congestion control module.
3. Do not disturb the default congestion module, however, make all the modules available to change from userspace application.

The students need to submit the config file and a report describing the changes they are observing after installing these kernel 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: 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`.

A userspace process can change the congestion control algorithm using the setsockopt system call. For more details go through `man 7 tcp`.

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.

Assignment 1

In this assignment, the students need to write a loadable kernel module which performs sorting of an array of N objects inside the kernel mode. Your LKM should be able to handle two types of objects, a) 32-bit integer, b) string (null-terminated) (max length 100byte). 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 sorting algorithm
  - i) The first byte should contain either 0xFF or 0xF0. It informs the expected object type for sorting. Here 0xFF means objects are 32bit integers and 0xF0 means null-terminated string with a maximum length of 100 bytes. Other than these values will cause an error of type EINVAL and LKM left uninitialized.
  - ii) The second byte should contain the length N of the array to be sorted. The length 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 N write call should pass N objects. LKM must produce an invalid argument error in case of wrong argument (i.e, unexpected object or object size). On successful write call LKM will return the number of remaining objects. LKM will ignore any excess write call silently. However, any read call before completion of N write produces EACCES error.
- **Step 4.** Once N objects are pushed inside the LKM, it will sort the objects and be ready to give output to the process.
- **Step 5.** Next N read call should return N objects in nondecreasing order. (In case of string, $\texttt{strcmp}(\texttt{str}_i, \texttt{str}_{i+1}) \texttt{ <= 0}$ should hold).
- **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 which builds a binary search tree inside the kernel space for a userspace process. The LKM also needs to support various ioctl call 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 any time it wants.

Command	Description
PB2_SET_TYPE	This command initializes the LKM with the expected object type. If LKM is already initialized for the current process, this command resets the LKM and reinitializes again. It expects a pointer to char as value where the pointer points to a single character which stores value either 0xFF or 0xF0. These values represent the same meaning as in assignment 1. 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 with code EACCES.
PB2_SET_ORDER	This command set the order (i.e. inorder (default), preorder or postorder) of output and resets the output cursor to the root of the BST. It also expects a pointer to char as the value like PB2_SET_TYPE. However, the value of the character can be 0x00, 0x01 or 0x02 which represents inorder, preorder or postorder respectively. Invalid value causes an error in LKM and sets error number to EINVAL. On success, this command returns 0.
PB2_GET_INFO	This command returns the information about the nodes of the tree. PB2_GET_INFO command expects a pointer to a structure object of type **struct obj_info** as the value. On success, LKM returns the size of the BST and fills the various fields of the pointer passed by the process. On error, LKM sets appropriate error no.
PB2_GET_OBJ	This command searches an object in the BST. It expects a pointer to an object of type **struct search_obj** as value. This function always returns 0. However, it fills the field **found** in the **search_obj** accordingly.

The definition of ioctl commands are as follows:

```
#define PB2_SET_TYPE    _IOW(0x10, 0x31, int32_t*)
#define PB2_SET_ORDER   _IOW(0x10, 0x32, int32_t*)
#define PB2_GET_INFO    _IOR(0x10, 0x33, int32_t*)
#define PB2_GET_OBJ     _IOR(0x10, 0x34, int32_t*)
```

The definition of structures are as follows:

```
struct obj_info {
    int32_t deg1cnt; //number of nodes with degree 1 (in or out)
    int32_t deg2cnt; //number of nodes with degree 2 (in or out)
    int32_t deg3cnt; //number of nodes with degree 3 (in or out)
    int32_t maxdepth;//maximum number of intermediate nodes from root to a
leaf
    int32_t mindepth;//minimum number of intermediate nodes from root to a
leaf
};

struct search_obj {
    char objtype;    // either 0xFF or 0xF0 represent integer or string
    char found;      // if found==0 then found else no found

    int32_t int_obj; // value of integer object. valid only if objtype == 0xFF
    char str[100];   // value of string object. valid only if objtype == 0xF0
    int32_t len;     // length of string object. valid only if objtype == 0xF0
};
```

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 object type in the kernel.
3. Insert object using the write system call on the file descriptor as many time as want. This call is exactly the same as the write system call in assignment 1. However, there is no bound on the number of times the user program can call the write system call. LKM verifies the arguments in write system call and adds a new node in the BST.
4. To get the object stored in LKM BST, userspace program needs to perform a read system call on the file descriptor. Every read system call returns a single object from the BST. Consecutive read system calls return consecutive nodes as per the order set by ioctl system call with command PB2_SET_ORDER. For example, if a userspace program wants to sort the object then it has to insert the objects in the LKM, then calls the ioctl system call with command PB2_SET_ORDER and value 0x00. After that, it performs consecutive read system call to get the sorted list.
    - The first read system after a write system call or a ioctl with command PB2_SET_ORDER, returns the first node in the list (i.e. the root in preorder, smallest element in inorder and postorder).
    - The read system call returns 1 in case of successful execution. However, it returns 0 after it finishes the traversal of the tree.
    - A read system call returns 0 if there is no node.
5. When all the operations are done, the userspace process closes the file and LKM releases all the resources allocated for the process.

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 assumption. If you find anything ambiguous in the assignment statement, do not hesitate to post it to the piazza.

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