Assignment 1. Hash Table and Dynamic Probe in Linux Kernel (200 points)

Related Subjects

1. Linux kernel hash table
2. Linux module and device driver
3. Kprobe
4. x86's TSC (Time stamp counter) to measure elapse time
5. Multi-threaded program.

Project Assignment

Part 1: Accessing a kernel hash table via device file interface

Linux kernel consists of several generic data structures. One of them is hash table which is based on chaining objects upon a collision. So, a hash table contains a predefined number of buckets and every bucket is a linked list which will link all objects that are hashed to the same bucket. The implementation is provided in Linux/include/linux/hashtable.h.

In this assignment, you are requested to develop a Linux kernel module which initiates a hash table of 128 buckets in Linux kernel and allows the table being accessed as a device file. We will name the table as "ht530_tbl" and the objects to be embedded in the table have a type of

```
typedef struct ht_object {
        int key;
        int data ;
} ht_object_t;
```

The hash table is implemented in kernel space as a device "ht530"and managed by a device driver "ht530_drv". The hash table "ht530_tbl" is created and a device "ht530" is added to Linux device file systems when the device driver is installed. The device driver should be implemented as a Linux kernel module and enable the following file operations:

- *open:* to open a device (the device is "ht530").
- *write:* if the input object has a non-zero data field*,* a ht_object is created and added it to the hash table. If an old object with the same key already exist in the hash table, it should be replaced with the new one. If the data field is 0, any existing object with the input *key* is deleted from the table*.*
- *read:* to retrieve an object based on an input key. If no such object exists in the table, -1 is returned and errno is set to EINVAL.
- *Ioctl:* a new command "dump" to dump all objects hashed to bucket *n.* If n is out of range, -1 is returned and errno is set to EINVAL.
- *release:* to close the descriptor of an opened device file.

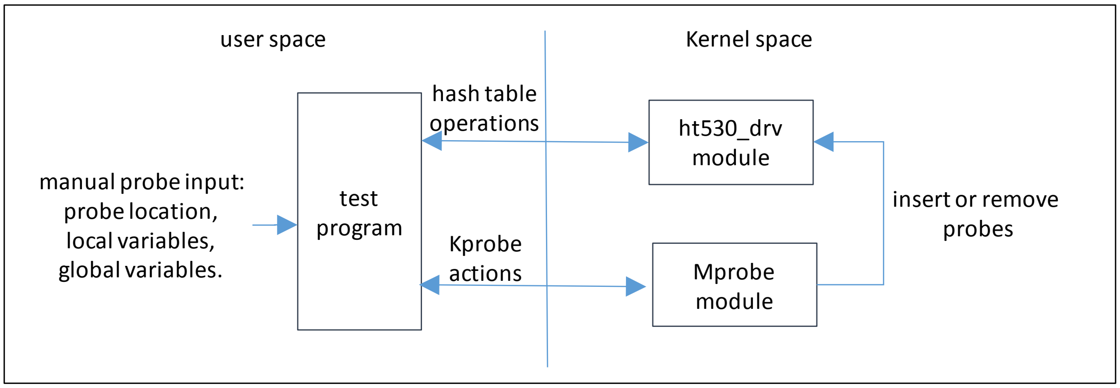
In a write call, **buf* points to a valid object and *count* gives the number of bytes of the object. The same parameters appear in read calls. However, for read function, the key value in the object pointed by **buf* is used to search the hash table. If the object is found, it is returned in the same buffer. For the *dump ioctl* command, you will need to generate a new ioctl number for it. The argument of the dump ioctl is a pointer to a buffer which is defined as

```
struct dump_arg {
        int n;          // the n-th bucket (in) or n objects retrieved (out)
        ht_object_t     object_array[8] ;  // to retrieve at most 8 objects from the n-th bucket
    } ;
```

To test your driver, a user program should be developed in which the main program creates 4 threads to populate the table with 150-200 objects and then invoke search, add, or delete operations to the table. The threads are set with different real-time priorities and consecutive file operations are invoked after a random delay. After a total of 100 search, addition, or deletion operations are done, the threads should terminate and the main program dump out all objects in the table. Note that the "ht530_tbl" and its associated objects should be deleted when the driver module is removed. So, the table is empty only at the first time your test program runs after the driver module is installed.

Part 2: Dynamic instrumentation in kernel modules

Linux has static and dynamic tracing facilities with which callback functions can be invoked when trace points (or probes) are hit. In this part of the assignment, you are required to develop a kernel module, named as "Mprobe", that uses kprobe API to add and remove dynamic probes in any kernel functions. With the module's device file interface, a user program can place a kprobe on a specific line of kernel code, access kernel information and variables. Integrated with part 1 of the assignment, you need to demonstrate the scenario depicted in the following diagram:



While exercising the hash table *ht530_tb*, your test program reads in kprobe request information from console and then invokes *Mprobe* device file interface to register a kprobe at a given location of ht530_drv module. When the kprobe is hit, the handler should retrieve few trace data items in a ring buffer such that they can be read out via Mprobe module. In the scenario, the user input request consists of the location (offset) of a source line of code in ht530_drv, the location (offset) of a local variable in the probed function's stack, and the location (offset) of a globe variable in .data or .bss sectios of ht530_drv module. The trace data items to be collected by kprobe handler include: the address of the kprobe, the pid of the running process that hits the probe, time stamp (x86 TSC), the value (4 bytes) of the local variable, and the value (4 bytes) of the global variable. Other than open and close file operations, the read and write operations of Mprobe device can be defined as:

- *write:* to unregister any existing kprobe that is registered previously by Mprobe module and to register a new kprobe. The location (offset) of the new kprobe is passed in the buffer **buf* along with the locations (offset) of a local variable and a global variable.
- *read:* to retrieve the trace data items collected in a probe hit and saved in the ring buffer. If the ring buffer is empty, -1 is returned and errno is set to EINVAL.

You can reuse the test program in part 1 for the scenario in part 2. For instance, besides the 4 threads that exercise the hash table, an additional thread can be created to receive input from console, to set up kprobes in ht530-drv, and to read out any collected data items. Using proper synchronizations, you can control how the 4 threads invoke the operations to *ht530* device file which can result in a hit at the kprobe point.

Due Date

TBD

What to Turn in for Grading

- There will be 5 bonus points each day if you submit earlier (a maximum of 20 bonus points for the assignment) and 20 points penalty per day if the submission is late.
- Failure to follow these instructions may cause an annoyed and cranky TA or instructor to deduct points while grading your assignment.
- TBD