Advanced Practical Embedded Software

Homework 2 (70 Pts) - Due Tuesday 9/19 Midnight

Revision 2017.9.10

Guidelines

Turn in a *.pdf with your answers to this assignment to D2L. **Please format your assignment nicely.** Don't make these assignments irritating to grade.

Reading & Resources

These are reading assignments that are good for you to complete the homework for the week as well as review current materials and prepare for the upcoming content in the class.

- Operating Systems Concepts - Silberschatz http://iips.icci.edu.iq/images/exam/Abraham-Silberschatz-Operating-System-Concepts---9th2012.12.pdf
    - Chapter 2 & 3
- Linux Device Drivers - Corbet - https://lwn.net/Kernel/LDD3/
    - Chapter 2 (all) & Chapter 7 (skim/reference)
- Linux Kernel Development https://github.com/yuanhui-yang/Linux-Kernel-Development/blob/master/Linux%20Kernel%20Development%20-%203rd%20Edition.pdf
    - Chapter 5 - System Calls (All)
    - Chapter 11 - Kernel Timers (Only pgs. 222 - 224)
- Test Driven Development for Embedded C - Chapter 2 Unit Testing
    - https://doc.lagout.org/programmation/C/Test-Driven%20Development%20for%20Embedded%20C%20%5BGrenning%202011-05-05%5D.pdf

Resources:

These sections are not required but may help with doing the homework assignments.

- Ubuntu Linux Boot Procedure: https://wiki.ubuntu.com/Booting
- Kernel Modules: https://wiki.archlinux.org/index.php/kernel_modules#Blacklisting
- Kernel Timer Example: https://www.ibm.com/developerworks/linux/library/l-timers-list/?ca=drs-

Problem Set

[Problem 1] Record your Repository

Provide us a github link to your repositories (if applicable).
https://goo.gl/forms/LFe5zX8B5EvkN8N82

[Problem 2 - 30 Pts] Implement Your Own System Call

You are to create your own system call. This system call needs to support the following features:

- Print information to the kernel buffer when issued and other useful debug strings

    - Log when syscall enters, exits, the size of the buffer, and the start/completion of the sort
- Your system call needs to take a set of input parameters including a
    - Pointer to a buffer
    - Size of that buffer
    - Pointer to a sorted buffer
- Validate all input parameters
- Your system call needs to allocate dynamic memory to copy data in from user space
- Pass in memory from user-to-kernel space
- Pass memory back to user space from kernel

The buffer will need to be copied into kernel space. Your syscall will need to sort your buffer in an order of largest to smallest. Once sorted, it needs to pass the buffer back to the sorted buffer array. You should create a buffer of at least 256 int32_t data items. You can randomly generate this buffer of data elements using rand and time.

Show that your kernel module works by writing a piece of software that calls your system call numerous times. You should that:

- System call works correctly (all input parameters valid and correct)
    - Print information showing that the sort worked correctly
- System call fails (input parameters are not valid and/or correct)
    - All errors should return appropriate Error values (defined in **errno.h** and **errno-base.h**)

Your system call should be defined appropriately using the SYSCALL_DEFINE macro given your argument list size. Create a new directory for your syscall and add your directory to the sources list in the main kernel makefile. You should include screenshots of example output of your program in addition to the source code in your assignment.

[Problem 3 - 20 Pts] Create a Kernel Module

Create your own kernel module with init and exit routines that print to the kernel buffer that you have loaded or removed the module. Define necessary file macros and makefiles to install the kernel module. Your kernel module needs to use a kernel timer to wake up every 500msec. Each time the timer wakes up you should call a function that prints to kernel log buffers a count of how many times the timer has fired. Declare a statically allocated variable to track this count in your callback.

To create/initialize a timer, look for the timer functions:

```
void init_timer (struct timer_list * timer);
void setup_timer (struct timer_list * timer,
                  void (*function)(unsigned long),
                  unsigned long data);
```

Be sure to add a callback, modify or delete your kernel timer appropriately:

```
void add_timer (struct timer_list * timer);
void mod_timer (struct timer_list * timer, unsigned long expires);
int del_timer (struct timer_list * timer);
```

Confirm that your module has been loaded by running the command on your system:

```
$ lsmod | grep module-name
$ modinfo module-name
```

Get a screenshot of these two commands to show that it was installed correctly.

[Problem 4 - 20 Pts] Unit Tests

You are to now implement unit tests for the data structures you coded in homework 1. This includes both the circular buffer and the doubly linked list. You are welcome to use either cmocka (https://cmocka.org/) or unity (http://www.throwtheswitch.org/unity/)

Examples can be seen at:

- https://github.com/afosdick/ecen5013/tree/develop/tutorials/unit_tests
- https://github.com/ThrowTheSwitch/Unity/blob/master/examples/example_1/test/TestProductionCode.c

All students need to write **new** linked list unit tests. All functions written from homework 1 need to be tested. In addition, boundary conditions and special operations need to be checked. For example, a double linked list should be able to insert at the head, in the middle and at the tail. The insert and delete should return expectedly if the list is empty. Another example is search, it should be given both input data that exists and data that does not exist. Your unit tests should be documented in a README for each of the two data structures. Add your unit tests code to your git repository in a unit test directory.

For students that have already completed unit tests for the circular buffer for ECEN5813, please validate these still work in your environment. For students that have not done this yet, you will have to write unit tests for you circular buffer as well (extra practice!). However, you can defer this activity until a later time to help reduce this homework load.