Git ECEN 5013

Assignment 8: Char Driver and Multithreading

Github Classroom Link

https://classroom.github.com/a/YBfEJAYp

Suggested Reading:

1. Lecture 15 and 16.
2. Linux Device Drivers book chapters 1-5

Github Classroom Start Instructions

You will start with an empty project submission repository, and fill it with the content of your assignment 5 local repository using these commands on your local buildroot-assignments repository (substitute to use your assignment 6 repository if you would like to use yocto for your submission instead of buildroot.)

1. git remote rename origin assignment-5-remote
2. git remote add origin <your github classroom submission repo created with the link above>
3. git push origin master

Implementation:

1. Pick the Buildroot or Yocto implementation from Assignment 7 to use as the basis for your assignment and push to your submission repository as described above. You can use either Buildroot or Yocto for this assignment.
2. Using your assignments 3 and later repository, complete the following steps to pull in content from the aesd-assignments repository assignment-8 branch

```
git remote add aesd-assignments git@github.com:cu-ecen-5013/aesd-assignments.git
git fetch aesd-assignments
git merge aesd-assignments/assignment8
```

3. Modify the TODOs in the aesdchar.c to complete the instructions below
4. Add source code, makefile and init scripts needed to install a /dev/aesdchar device. Ultimately this device should be installed in your qemu instance, however you may also

wish to test on your host virtual machine during development. Your /dev/aesdchar device should:

a. Allocate memory for each write command as it is received, supporting any length of write request (up to the length of memory which can be allocated through kmalloc), and saving the write command content within allocated memory.
    i. The write command will be terminated with a \n character as was done with the aesdsocket application.
        1. Write operations which do not include a \n character should be saved and appended by future write operations.
    ii. The content for the most recent 10 write commands should be saved.
    iii. Memory associated with write commands more than 10 writes ago should be freed.
    iv. Write position and write file offsets can be ignored on this assignment, each write will just write to the most recent entry in the history buffer or append to any unterminated command.
    v. For the purpose of this assignment you can use kmalloc for all allocations regardless of size, and assume writes will be small enough to work with kmalloc.
b. Return the content of the most recent 10 write commands, in the order they were received, on any read attempt.
    i. You should use the position specified in the read to determine the location and number of bytes to return.
    ii. You should honor the count argument by sending only up to the first “count” bytes back of the available bytes remaining.
c. Perform appropriate locking to ensure safe multi-thread and multi-process access and ensure a full write file operation from a thread completes before accepting a new write file operation.
d. Your implementation should print the expected contents when running the drivertest.sh script.

5. Modify your socket server application developed in assignments 5 and 6 to support and use a build switch USE_AESD_CHAR_DEVICE, set to 1 by default, which:
    a. Redirects reads and writes to /dev/aesdchar instead of /var/tmp/aesdsocketdata
    b. Removes any locking and ensures all locking is performed within your /dev/aesdchar device instead.
    c. Removes timestamp printing.
6. Roll back to the sockettest.sh script used with assignment 5 (which doesn’t validate timestamps), which you can find at https://github.com/cu-ecen-5013/aesd-assignments/blob/assignment5/sockettest.sh
7. Modify your buildroot or yocto implementation to install updated versions of the aesd socket server application and aesdchar driver, and ensure your updated implementation passes when running with the sockettest.sh

Steps to enable Kernel Memory Leak Detector:

1. Make linux-menuconfig from buildroot directory
2. Enable the following options:
    - kernel hacking -> kernel debugging
    - kernel hacking -> memory debugging -> kernel memory leak detector
    - kernel hacking -> memory debugging -> Simple test for the kernel memory leak detector (This option would install a module which would have memory leaks.)
3. Save the configuration, build and run qemu.
4. If the `debugfs` isn't already mounted, mount with:
   `mount -t debugfs nodev /sys/kernel/debug/`

5. Enter the following commands on terminal to check if the kernel memory leak detector has been properly installed and is detecting memory leaks.
    - `echo clear > /sys/kernel/debug/kmemleak`
    - `modprobe kmemleak-test`
    - `echo scan > /sys/kernel/debug/kmemleak`
    - `cat /sys/kernel/debug/kmemleak` (This command is used to view memory leaks)
6. The installed module can be removed using: `rmmod kmemleak-test`

Validation:

1. Your driver should pass the drivertest.sh script provided with the assignment
2. Your qemu instance should pass the sockettest script, this time writing and reading from only the /dev/aesdchar device instead of /var/tmp/aesdsocketdata.
3. Your implementation should not crash or have memory leaks.
4. Your implementation should use proper locking methods to ensure safe access from multiple threads or processes.
5. Your project should contain build.sh, clean.sh, and runqemu.sh scripts which function as specified in previous assignments, and can successfully clean, build, and run your project without user interaction for automated build testing.

Submission:

1. Your submission repository should contain a working buildroot or yocto implementation.
2. Your buildroot or yocto implementation should reference a commit in your assignments 3 and later repository containing the code added for this assignment.