CSci 5103 (Spring 2016)

Assignment 7

100 points

Due April 25, 2016

This part of the assignment can be done in a group of up to *two students.*

You will need access to the Linux VM set up for you on the CSELabs machines. You can certainly do this assignment on your personal computer, but YOU MUST MAKE SURE that it will run on the CSELabs Linux VM. You will lose significant points if your assignment does not compile/run on the CSELabs Linux VMs. You will receive an email from the Operator containing instructions for accessing your Linux VM.

Getting started

1. Log in to your VM
2. Copy the examples folder into your home directory

```
$ cd; cp /usr/local/linux-module-examples.tgz .
```

3. Extract the examples

```
$ tar xvfz linux-module-examples.tgz
```

4. Try out the example modules under the helloworld, scull & scullpipe folders

---

Overview

In this problem you are asked to write a character device called **scullbuffer**, which provides a bounded buffer of fixed size to synchronize any number of producer and consumer processes. **scullbuffer** will implement the character device interface. Both producer and consumer processes will first call open() to access the device. At end of their use of the device, they will call release() to indicate that the session has ended. The buffer will store multiple items, where each "item" will be a block of size 32B (bytes).

Producer process

- A producer process will open the device(scullbuffer) in "**write**" mode.
- The producer will call write() to deposit an item into the buffer in device memory. The write() operation must try copy an item from the buffer location (in the address-space of the process) which is passed to the write() function call into the buffer in device memory.
- After depositing the required number of items, the producer will call release(), which will effectively terminate the session.
- If the buffer is full and there are some consumer processes that have the device/scullbuffer open for reading, then the producer process will be blocked (i.e. go to sleep) in the call to write().

The return value of write() can be:

1. **-1** if there was any error
2. **The number of bytes** copied from the process address-space and written into device memory if successful
3. **0** if the buffer is full, and there are no consumer processes that currently have scullbuffer open for reading.

Note: Pay attention to this, and ensure that your implementation can handle this scenario.

- A producer starts waiting (i.e. goes to sleep in the kernel) because the buffer is full and there are some consumer processes which still have the device open for reading.

- Suppose that sometime later all these consumer processes release the device without removing an item from the buffer and no consumers that have scullbuffer open for reading are left; at this point, any waiting producers must be woken up (the value 0 will be returned to the producer process).

Consumer process

- A consumer process will open the device(scullbuffer) in "**read**" mode.
- The consumer will call read() to retrieve an item from the buffer in device memory. The read() operation must try to copy the next available item from the buffer in device memory into the address-space of the process, at the buffer location specified in the read() function call.
- After consuming/retrieving the required number of items, the consumer will call release(), which will effectively terminate the session.
- If the buffer is empty and there are some producer processes which currently have the device/scullbuffer open for writing, then the consumer process will be blocked (i.e. go to sleep) in the call to read().

The return value of read() can be:

1. **-1** if there was any kind of error
2. **The number of bytes** copied from the buffer in device memory intto the process address-space if successful.
3. **0** if the buffer is empty, and there are no producer processes that currently have scullbuffer open for writing.

Note: Pay attention to this, and ensure that your implementation can handle this scenario.

- A consumer starts waiting (i.e. goes to sleep in the kernel) because the buffer is empty and there are some producer processes which still have the device open for writing.
- Suppose that sometime later all these producer processes release the device without depositing an item into the buffer and no producers that have the device open for writing are left; at this point, any waiting consumers must be woken up (the value 0 will be returned to the consumer process)

Module loading & unloading

As part of module loading/initialization, you will allocate a buffer in the kernel memory using kmalloc() to hold up to NITEMS items of 32B each. NITEMS will be a parameter to your driver and can be specified at device installation time.
As part of module unloading, you need to free up the allocated memory and clean up all the resources before exiting.

Please write a script for installing and removing the scullbuffer device and also write code for the producer and consumer processes to test your device.

Synchronization

You must use counting semaphores to implement the synchronization. (See Chapter 5 of the Linux Device Drivers book).

You need to implement the following functions: ***open()***, ***release()***, ***read()*** and ***write()***.
As part of the open() and release() function implementation, you will need to maintain the counts of the currently active producer and consumer processes.
The functions read() and write() need to perform the appropriate synchronization using counting semaphores for accessing the slots in the buffer.

Helpful hints

- The code for scullpipe will be a good starting point to implement scullbuffer.
- The default value of NITEMS can be set to a reasonable number like 20.

- For easy debugging, each item can be a character block of 32B. You may wish to distinguish between individual items (both items created by different producers and items created by the same producer). Each item created by a producer can have a common prefix and have a continuously incrementing number e.g “BLACK000001”, “BLACK000002” etc. Different producers can use different prefixes for the items.
- The producer and consumer processes can each take a command line argument which indicates the number of items which the process must produce or consume. This way, running different tests will become easier.
- You may wish to add a delay in your producer and consumer processes before trying to write or read items into scullbuffer. Concretely, after the producer and consumer processes open scullbuffer for writing/reading, they should sleep for 2-3 seconds to allow all other producers/consumers to finish executing the open() call. This way, the calls to read() and write() will block and not exit with a return status of 0 immediately.

Testcases

In your readme, provide a description of the tests, the expected output, and the way to run them.
It would be helpful for testing if you print messages for each operation that occurs, like read/write succeeding, a process going to sleep, a process being woken up from sleep, read/write failing etc.

Testcase #1

- Start a producer which will produce 50 items before exiting.
- Start a consumer which will consume try to 50 items before exiting.
- Both the producer and consumer should exit normally after producing and consuming all the items.

Testcase #2

- Start a producer which will produce 50 items before exiting.
- Start a consumer which will try to consume only 10 items before exiting.
- After the consumer calls release(), the producer should not go to sleep when trying to write items into the already full buffer, and should instead get a return value of 0 from the write() call. (Because the consumer count is 0)

Testcase #3

- Start a producer which will produce 50 items before exiting.
- Start a consumer which will try to consume 100 items before exiting.
- After the producer calls release(), the consumer should not go to sleep waiting for items to be inserted into the buffer, and should instead get a return value of 0 from the read() call (Because the producer count is 0)

Testcase #4

- Start two producers which will both produce 50 items each before exiting (total = 100 items).
- Start a consumer which will try to consume 200 items before exiting.
- All the producers and the consumer should exit normally after producing and consuming all the items.

Testcase #5

- Start a producer which will produce 50 items before exiting.
- Start two consumers which will both try to consume 25 items each before exiting.
- All the consumers and the producer should exit normally after producing and consuming all the items.

Testcase #6

- Start two producers which will both produce 50 items each before exiting.
- Start a consumer which tries to consumes 5 items before exiting.
- After the consumer calls release(), both the producers should be woken from sleep (if sleeping) and should exit gracefully.

Testcase #7

- Start two producers which will both produce 50 items each before exiting.
- Start a consumer which tries to consume 200 items before exiting.
- After the producer calls release(), the consumer should be woken from sleep (if sleeping) and should exit gracefully.

Submission requirements

1. Source files for the device & device driver (*.c and *.h)
2. Makefile for compiling the scullbuffer device driver
3. Script for installing and removing the scullbuffer device
4. Source files for the producer and consumer processes (*.c and *.h)
5. Makefile for compiling the producer and consumer processes
6. Script for running your testcases
7. A readme file

Grading criteria:

Documentation, comments & readability of source code: **10 points**
Script for installing/removing the scullbuffer device: **5 points**
Readme: **5 points**
Makefiles: **10 points**
Testcases (10 points per test case) : **70 points**