Project 3 Specification
Kernel Module Programming
Kernel Modules, Mutual Exclusion, and Scheduling

Final Submission Due: 3/27 at 11:59:59pm

Language Restrictions: C only

Purpose

This project introduces you to the nuts and bolts of kernel programming, concurrency, and synchronization in the kernel. This project is divided into two parts and worth 40 points. This project may be completed in partners or by yourself.

Part 1: remember Kernel Module

In Unix-like operating systems, the /proc interface is often used to set or read kernel information. For example, take a look at /proc/cpuinfo and /proc/meminfo using the following commands:

```
$> cat /proc/cpuinfo

$> cat /proc/meminfo
```

Even though these proc files look like ordinary files, they are actually virtual kernel drivers! These modules perform actions based on whether a user reads or writes to them.

You will write your own proc module called remember that

1. allows the user to write a string (max length 80)
2. allows the user to read back the string that was just added

For example:

```
$> echo "Hello there" > /proc/remember
$> cat /proc/remember
Hello there
$>
```

Please use the example proc module (hello_proc.c) and Makefile provided to you as a starting point. Change the name of the module from helloworld to remember, create a global string of size 80 to hold the string written, and use that global variable whenever a read is performed.

You can use the original 4.9 kernel or the 5.5.5 kernel.

Part 2: Your very own Virtual Printer

Congratulations, you are the proud owner of a new virtual printer called the penguin printer (a.k.a "penguin"). Your task is to implement a printer scheduling algorithm. A printer is defined as a device that accepts print jobs for documents into a queue and processes those documents. A maximum number of jobs will be given. When a printer is loaded, it is initially running. Each job takes a specific

amount of time to process. Finally, the printer is stopped when the module is unloaded.

Your printer must keep track of the number and type of jobs in a queue. Jobs can come in at any time and can be put on the queue instantaneously. A job must always be accepted if there is room. Jobs can only be processed by the printer one at a time, and it takes the printer 1 second to look inside any spot in the queue (whether it holds a job or not). Once a job is processed, the spot in the queue is cleared and the printer can look for other jobs.

Task Specification

This is a classic exercise in modeling consumers and producers. The producer produces jobs and the consumer is the printer. There are many pieces needed to provide a complete implementation discussed below.

Step 1: Develop a penguin printer /proc module

Develop a `/proc` entry named `/proc/penguin`. In this project, you will be required to support a printer job queue of size 10. You will accept the following 4 jobs:

- 1-page document (internally represented with a 1)
- 2-page document (internally represented with a 2)
- 3-page document (internally represented with a 3)
- 4-page document (internally represented with a 4)

As in a real printer, it takes varying time to process different types of jobs. Your printer must take 1 second to process a 1-page document, 2 seconds to process a 2-page document, 3 seconds to process a 3-page document, and 4 seconds to process a 4-page document. This processing time is ***in addition*** to the 1 second it must take to look in any spot in the queue for an job.

Finally, the printer can break down. It also must accept the following job:

- maintenance (internally represented with a 5)

Maintenance takes a whopping 10 seconds, in addition to the 1 second it must take to look in any spot in the queue for a job.

(Hint – you can "process" for a given amount of seconds by using the ssleep() function.)

You will place jobs on your queue by writing your jobs's number to the /proc/penguin file. For example, the following will put a 2-page document on the order queue:

```
$> echo 2 > /proc/penguin
```

If the queue is full, the printer code should return –ENOMEM and the job should not be placed on the queue.

```
$> echo 2 > /proc/penguin
bash: echo: write error: Cannot allocate memory
```

Step 2: Reading from /proc/ penguin

In addition to accepting orders and commands, your printer must be able to display status information by reading the device. Specifically, when someone reads from `/proc/penguin`, they should see:

- Current spot being looked at in the queue
- Current job being processed
- Total jobs processed

For example, take a look at this sample session:

```
$> insmod penguin.ko
$> cat /proc/penguin
Processing nothing at slot 2. Total jobs processed: 0.
$> echo 4 > /proc/penguin
$> cat /proc/penguin
Processing nothing at slot 8. Total jobs processed: 0.
$> cat /proc/penguin
PriProcessing 4-page document at slot 0. Total jobs processed: 0.
$> cat /proc/penguin
Processing 4-page document at slot 0. Total jobs processed: 0.
$> cat /proc/penguin
Processing nothing at slot 2. Total jobs processed: 1.
$> rmmod penguin
```

The insmod and rmmod commands load and unload the kernel module, respectively. After starting the printer, I placed 4-page document on the queue. However, when immediately reading from /proc/penguin, my printer was not yet looking in the spot in the queue with the 4-page document. Finally, it looks in the right spot and "processes" the 4-page for 4 seconds, so I was able to read from /proc/penguin twice and get the same processing status.

Step 3: Use a kthread

You must use a kthread to allow the printer to run in the background and process orders. Please look at my example, which starts a kthread on module load.

Step 4: Mutual Exclusion

You may have been getting lucky up until now, but what happens if the printer kthread is modifying a spot in your job queue that you are trying to read at the same time? What happens if you are trying to add a job to the queue at the same time that the printer is taking one away? If you have any global variables in your code that can be accessed by more than one thread at a time, you must protect the reading and writing of those variables with mutexes (semaphores with only 0 and 1 values). If not, your entire driver could randomly crash. The probability of this goes up with the more cores you have in your virtual machine. Use kernel mutexes to protect critical sections of code where you read or write to global variables.

Extra Credit

The top five submissions as measured by the above evaluation procedure will receive +5 points to their project 3 grade. The metric to optimize is: **total jobs processed.**

Full Project Submission Procedure

You will need to submit the following files:

- remember.c (not the .ko or the Makefile)
- penguin.c (not the .ko or the Makefile)
- The filled-in Project3-README.docx. (If you are a Mac user, please keep the format as Word or else convert to a pdf. I have problems grading Pages documents on a non-Mac.)