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

Final Submission Due: 3/26 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 and Makefile provided to you as a starting point.

Part 3: A Restaurant!

Congratulations, you are the proud owner of a new automated restaurant! Your task is to implement a driver with a kitchen scheduling algorithm. A kitchen is defined as a device that accepts orders for dishes into a queue and processes those orders. A maximum number of orders will be given. When a kitchen is loaded, it is initially running. Each dish takes a specific amount of time to process. Finally, the kitchen is stopped when the module is unloaded.

Your kitchen must keep track of the number and type of orders in a queue. Orders can come in at any time and can be put on the queue instantaneously. An order must always be accepted if there is room.

Orders can only be processed by the kitchen one at a time, and it takes the kitchen 1 second to look inside any spot in the queue (weather it holds an order or not). Once an order is processed, the spot in the queue is cleared and the kitchen can look for other orders.

Task Specification

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

Step 1: Develop a kitchen /proc module

Develop a `/proc` entry named `/proc/kitchen`. In this project, you will be required to support a kitchen order queue of size 20. You will accept the following 4 orders:

- Caesar Salad (internally represented with a 1)
- Hamburger (internally represented with a 2)
- Personal Pizza (internally represented with a 3)
- Beef Wellington (internally represented with a 4)

As in a real kitchen, it takes varying time to process different types of orders. You kitchen must take 2 seconds to process a Caesar Salad, 3 seconds to process a Hamburger, 4 seconds to process a Personal Pizza, and a whopping 8 seconds to process a Beef Wellington. This processing time is ***in addition*** to the 1 second it must take to look in any spot in the queue for an order.

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

You will place orders on your queue by writing your order's number to the /proc/kitchen file. For example, the following will put a hamburger on the order queue:

```
$> echo 2 > /proc/kitchen
```

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

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

Step 2: Reading from `/proc/kitchen`

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

- Current spot being looked at in the queue
- Current order being processed
- Total orders processed

For example, take a look at this sample session:

```
#> insmod kitchen.ko
#> cat /proc/kitchen
```

```
Kitchen is running. Processing nothing at slot 2. Total processed is 0.
#> echo 4 > /proc/kitchen
#> cat /proc/kitchen
Kitchen is running. Processing nothing at slot 19.  Total processed is 0.
#> cat /proc/kitchen
Kitchen is running. Processing Beef Wellington at slot 0. Total processed is 0.
#> cat /proc/kitchen
Kitchen is running. Processing Beef Wellington at slot 0. Total processed is 0.
#> cat /proc/kitchen
Kitchen is running. Processing nothing at slot 2. Total processed is 1.
#> rmmod kitchen
```

The insmod and rmmod commands load and unload the kernel module, respectively. After starting the kitchen, I placed an order of Beef Wellington on the queue. However, when immediately reading from /proc/kitchen, my kitchen was not yet looking in the spot in the queue with the Beef Wellington. Finally, it looks in the right spot and "processes" the Beef Wellington for 8 seconds, so I was able to read from /proc/kitchen twice and get the same processing status.

Step 3: Use a kthread

You must use a kthread to allow the kitchen 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 kitchen kthread is modifying a spot in your queue that you are trying to read at the same time? What happens if you are trying to add an order to the queue at the same time that the kitchen 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 orders processed.**

Full Project Submission Procedure

You will need to submit the following files:

- remember.c (not the .ko or the Makefile)
- kitchen.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.)