Project 2 Specification
Kernel Module Programming
Kernel Compilation, Kernel Modules, and Scheduling

Final Submission Due: 3/25 at 11:59:59pm

Language Restrictions: C only

Purpose

This project introduces you to the nuts and bolts of kernel compilation, kernel programming, and concurrency and synchronization in the kernel. This project is divided into three parts.

Part 1: Compile a Kernel

(See Project 2A Specification)

Part 2: `remember` Kernel Module

(See Project 2A Specification)

Part 3: A Restaurant!

Congratulations, you are the proud owner of a new virtual restaurant! Your task is to implement 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 not running (although it can accept orders). A kitchen starts processing orders with a start command. Each dish takes a specific amount of time to process. Finally, a kitchen must be stopped before it can be 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 (whether 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
```

In addition to accepting orders 1-4, your kitchen should accept two additional numbers as commands:

- 0 : start the kitchen
- -1: stop the kitchen

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:

- Kitchen status: running or not running
- Current spot being looked at in the queue
- Current order being processed
- Total orders processed

(If the kitchen is not running, the last two pieces of information do not need to be displayed.)

For example, take a look at this sample session:

```
$> insmod kitchen.ko
$> cat /proc/kitchen
Kitchen is not running.  Total processed is 0.
$> echo 0 > /proc/kitchen
$> echo 4 > /proc/kitchen
$> cat /proc/kitchen
Kitchen is running. Processing nothing at slot 2. Total processed is 0.
$> 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.
$> echo -1 > /proc/kitchen
$> cat /proc/kitchen
Kitchen is not running.  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 in response to a start command. Please look at my example, which starts a kthread on module load. You will need to modify the kthread to process a kitchen queue and instead start/stop the kthread through commands written to `/proc/kitchen`.

Extra Credit

The top five submissions as measured by the above evaluation procedure will receive +5 points to their project 2 grade. The metric to optimize is: **total orders processed.**

Halfway Project Submission Procedure
By the halfway point, you should have parts 1 and 2 finished. (Worth 20%)
(See Project 2A Specification)

Full Project Submission Procedure
By the final submission date, part 3 should be finished. (Worth 40%)

. You will need to zip up the following files for submission:

- A folder called Part 3 containing:
  - The kitchen.c file **(not the .ko file)**
  - The Makefile for kitchen
- The README text file, which should contain:
  - The names of all the members in your group
  - A listing of all files/directories in your submission and a brief description of each
  - Instructions for compiling your programs (NOTE: use the Makefile provided for you)
  - Instructions for running your programs/scripts
  - Any challenges you encountered along the way
  - Any sources you used to help you write your programs/scripts