

Project 3 Specification

Kernel Module Programming

Kernel Time and Elevator Scheduling

Assigned: October 14, 2009, 11:00am

Due: November 9, 11:59:59pm

Language Restrictions: C only

Purpose

The purpose of this project is dually to introduce you to the realm of kernel programming and to alert you to the concurrency of the kernel and the importance of synchronization at the kernel level.

There are two independent parts to this project. The first part of this project serves as a gentler introduction to kernel programming. It will introduce you to how to utilize the /proc file system to provide users with information. Furthermore, you'll be introduced to the kernel's notion of time. The second portion of the project will introduce you to the notion of scheduling algorithms and will serve as a more thorough introduction to kernel module programming, as well as /proc file system utilization.

Ultimately, this project should bestow upon partakers knowledge regarding the importance of synchronization, the methods and tools available to implement correct synchronization, and the usage of the /proc file system to export kernel information to user space.

Problem Statement

Design and implement a kernel module that displays to the user the value of the kernel variable, **xtime**. This display must be created by the use of the /proc file system kernel interface. This display must be read-only, namely, it should not be modifiable from user space. The display shall be name **/proc/currenttime**. To demonstrate the correctness of your module, you must provide a user-space application that reads from this procfile and displays to the user the value of **xtime**.

Additionally, design and implement a set of system calls and a kernel module that implements an elevator scheduling algorithm. Any elevator scheduling algorithm may be used. To exercise the correctness and efficiency of your algorithm, as well as to demonstrate the functionality of your module and system calls, you will also implement a set of user-space applications.

Questions

1. What is the **xtime** variable used for in the kernel?
2. Why is the resolution of the function **gettimeofday()** different from the values you expose in **/proc/currenttime**?
3. How is I/O scheduling similar to elevator scheduling?
4. Map the analogy:
 - a) A passenger represents a _____ in terms of a hard-drive.
 - b) An elevator represents a _____ in terms of a hard-drive.
5. If an elevator models a storage device, provide an analogy for a memory device relative to the elevator. Is elevator scheduling appropriate for memory devices?
6. What is a **jiffy**?

Project Task

Part 1: xtime Kernel Module

Here, you are tasked with implementing a kernel module. This module should:

- Create a /proc file system entry named **/proc/currenttime** upon being loaded using *insmod*
- **/proc/currenttime** should show two integral values delimited by a single space:
 - xtime.tv_sec
 - xtime.tv_nsec
- Remove **/proc/currenttime** when *rmmmod* is invoked

Example usage:

```
$> cat /proc/currenttime
1234567 22334231
$> echo 1 > /proc/currenttime
bash: /proc/currenttime: Permission denied
```

Additionally, a user-space program should be written. This program should output the value of the function **gettimeofday()** as two integers delimited by a space. It should also output the values exposed in **/proc/currenttime** on a separate line as two integers.

Example output:

```
$> ./xtime_test
gettimeofday: 1234566 22334230
xtime: 1234567 22334231
```

Part 2: Elevator Scheduling

Introduction

The task here is to implement an elevator scheduling algorithm. An elevator, for the sake of this project, is defined as a device that may only move up and down. An elevator is stopped if there are no requests to process or if it is currently loading passengers. It has a starting floor, a current load, and a current floor. All elevators are assumed to start at floor 1 upon creation. A maximum load and a maximum floor will be given as assumptions. An elevator also takes a finite amount of time to move between floors and to collect passengers from a floor- this does not happen instantaneously.

Passengers constitute the only load allowed within an elevator. A passenger always counts as a single unit of load. No distinction is made between passengers. Passengers randomly appear on a floor of their choosing. They always have in mind the floor they wish to go to. A passenger may not choose an elevator they prefer- they must board the first one that can accept them. Once they board the elevator, they may only get off when the elevator arrives at the floor they initially requested- a passenger may not depart the elevator earlier than this. Once a passenger arrives at their destination floor, they cease to exist. Passengers wait infinitely, until the elevator arrives to collect them- they never get bored and choose to use the stairs.

A building has a number of elevators and a number of floors. The elevators must be aware of the maximum number of floors, else they would collide with the ceiling. The elevators in the building must coordinate in order to transport as many passengers as possible.

Task Specification

This is a classic exercise in modeling multiple consumers and multiple producers. The producers produce passengers and the consumers are the elevators. There are many pieces needed to provide a complete implementation and these will be detailed in turn below.

Begin by creating a kernel module. This module must implement the above description. More precisely:

- Develop a representation of elevators. In this project, you will be required to support up to a maximum of **3** concurrent elevators.
 - Each elevator has a maximum load of **30** passengers.
 - An elevator must wait for **2** seconds when transitioning from one floor to the next.
 - An elevator must wait for **1** second while collecting passengers.
 - Two elevators may NOT collect passengers from the same floor at the same time.
 - An elevator may not go from floor **MAX_FLOOR** to floor 0 in one increment- it must go through floors **MAX_FLOOR - 1, MAX_FLOOR - 2, ..., 1, 0**, in order.
- A building has a maximum of **20** floors.
- Passengers indicate a starting floor and a destination floor.
- Multiple passengers may issue requests at the same time.
- Multiple passengers may wait on the same floor.

- The module must provide a /proc entry named **/proc/elevator**
 - For each active elevator, print:
 - The elevator number
 - The elevator's direction of movement (UP, DOWN, LOADING, STOPPED)
 - The current floor the elevator is on
 - The next floor the elevator intends to service
 - The number of passengers the elevator is carrying
 - If LOADING, the elevator is carrying the number of passengers it had before moving to the next floor
 - For each floor of the building, print:
 - The number of passengers waiting on that floor
 - Print the total number of passengers that have been serviced
 - Serviced is defined as the number of passengers that have departed from the elevator, or ceased to exist, if you will

Once you have a complete kernel module, you must modify the kernel by adding a set of system calls. These system calls will be used by a user-space application to control your elevators and create passengers. They should have the following prototypes and functionality:

- **int start_elevator(int elevator_number)**
 - Description: Activates elevator **elevator_number** for service. From that point onward, **elevator_number** exists and will begin to service requests as detailed above.
 - Returns:
 - 2 If **elevator_number** is already active.
 - 1 If **elevator_number** is out of range, i.e. less than 0 or greater than **MAX_ELEVATORS_SUPPORTED**
 - 0 Otherwise.
- **int issue_request(int start_floor, int destination_floor)**
 - Description: Creates a passenger at **start_floor** that wishes to go to **destination_floor**.
 - Returns:
 - 1 If the request is not valid.
 - A request is not valid if:
 - Either **start_floor** or **destination_floor** are out of range.
 - Note: **start_floor == destination_floor** is a valid request.
 - 0 Otherwise.
- **int stop_elevator(int elevator_number)**
 - Description: Deactivates elevator **elevator_number**. At this point, this elevator will process no more requests. However, before an elevator ceases to exist, it must offload all its current passengers. Only after the elevator is empty may it be deactivated.
 - Returns:
 - 3 If **elevator_number** is already in the process of deactivating.
 - 2 If **elevator_number** does not exist.
 - 1 If **elevator_number** is out of range.
 - 0 Otherwise.

Once you've implemented your system calls, you must interact with two user-space applications that allow communication with your kernel module. These are as follows:

- **producer.c**
 - This program will enter an infinite loop and issue **random** requests, **three** per second, until it is aborted by pressing Ctrl-C.
- **consumer.c** <--start **M** | --stop **M**>
 - This program expects one flag and one argument:
 - If the flag is --start, then the program must start elevator number **M**.
 - If the flag is --stop, then the program must stop elevator number **M**.
 - If there are no arguments, too many arguments, or incorrect flags, the program must inform the user regarding the correct usage of the program.
 - If the number **M** is out of range, the program should inform the user of the maximum number of elevators supported.
 - *: Additional error messages may be printed by the program to signal all possible error returns that the system calls return: elevator already active, elevator already deactivated, elevator shutting down.

producer.c and **consumer.c** will be provided to you.

Allowed Assumptions and Additional Comments

- You won't have to implement anything to provide read-only access to **/proc/currenttime**.
- **xtime** is an existing kernel variable that you may look up.
- **MAX_FLOORS = 20**
- **MAX_ELEVATORS_SUPPORTED = 3**
- **REQUESTS_PER_SECOND = 3**
- **MAX_LOAD = 30**
- **WAIT_SECS_BETWEEN_FLOORS = 2**
- **WAIT_SECS_ELEVATOR_LOADING = 1**
- An arbitrary number of requests may be issued on a given floor. This means that you cannot in advance predict the number of requests that will be waiting on a given floor.
- All elevators have the same maximum load.
- Initialize an elevator as follows:
 - Direction: STOPPED
 - Current load: 0
 - Current floor: 0
- Elevators need not run in parallel, only concurrently:
 - This means that each elevator need not be scheduled on a separate thread. You need to explicitly and sequentially coordinate the elevators to maximize passenger throughput.
 - Concurrently means requests are being issued while your elevators service passengers.
- As stated before, you may use any scheduling algorithm to implement passenger request processing.

Extra Credit

The top five submissions as measured by the above evaluation procedure will receive +10 points to their project 3 grade. The metric to optimize is: **total passengers serviced**.

Project Submission Procedure

As this project is potentially non-portable, you will be required to schedule for a project demonstration. The project will be graded on the spot during this project demonstration period. A week before the project is due, registration for demonstration will open. You will be given 15 minutes to present your project to me. This demonstration will take place by the computer you used to implement your project. You will be required to demonstrate:

- Project source files
 - System calls, module sources, user-space sources
- Project makefiles
- Successful installation of part 1
- Information stored in **/proc/currenttime**
- Successful removal of part1
- Successful installation of part 2
- Execution of an arbitrary number of consumers and producers for **5** minutes
- Information stored in **/proc/elevator** once per minute
- Successful removal of part 2

Any demonstration failing to install a portion of the project successfully during their allotted time will receive a **0** for that portion of the project. Be absolutely sure that you can at least install and remove all kernel modules before attempting to demonstrate. As before, any project that fails to compile will also receive a **0** for that portion of the project.

The grader, at their discretion, may choose to question you on project components. You should be able to demonstrate an understanding of the project implementation.