Project 02

Due Dec 11, 2020 at 11:59pm **Points** 100 **Questions** 4
Available Nov 26, 2020 at 12am - Dec 12, 2020 at 11:59pm 17 days
Time Limit None **Allowed Attempts** Unlimited

Instructions

Introduction to Linux Kernel Modules and Character Device Drivers

In this project, you will implement a disk simulator that you will then turn into a character based device for which you will subsequently write a driver and install it in Linux as a kernel module.

This project must be implemented on Linux, since it uses Linux-specific ways to implement both the drivers and kernel modules.

There are `CMakeLists.txt` build configuration files included for building the code in steps 1 and 3, but the kernel module in task 2 must be built in a special way that is not supported by `cmake`. Instead, in this step you will need to use the also provided `Makefile`; just run `make` without preconfiguring it with `cmake`.

You must submit incremental solutions for each of the steps. Submissions that include the functionality from the further steps will be rejected.

This quiz is no longer available as the course has been concluded.

Attempt History

	Attempt	Time	Score
KEPT	Attempt 2	1,565 minutes	70 out of 100
LATEST	Attempt 2	1,565 minutes	70 out of 100
	Attempt 1	15,643 minutes	0 out of 100 *

* Some questions not yet graded

Score for this attempt: **70** out of 100

Submitted Dec 11, 2020 at 7:49pm
This attempt took 1,565 minutes.

Question 1

Not yet graded / 30 pts

Step 1

Description | Submission

Implementation

Implement a disk simulator that allows for writing and reading information using logical linear addresses. The code in **`disk.zip`** archive provides the declaration of the disk geometry and the prototypes of the functions that you must implement. There is also some demonstration code and seed code for testing. You must not change the data structures, but you can augment the testing if needed.

There are details of the implementation in the comments accompanying the code.

BlanchardSeanProject02.zip (https://cilearn.csuci.edu/files/2406164/download)

Question 2

Not yet graded / 15 pts

Step 2

Description | Submission

Description

In this task, you will implement a timer-based tester of the disk simulator that you implemented in the previous steps.

To get familiar with Linux timers, download the archive **`timer.zip`**, examine the code, and then compile it. Run the program and make sure that you understand what it does, and how it does it.

Using the code from the `timer` project enhance your test program for the disk simulator, so that it periodically calls your disk simulator in response to a random timeout. Each time the timeout occurs (or - in other words - the timer expires) the timeout handler should generate a random disk reference (a logical block number), a random number of blocks to write or read (toss a coin using `rand()%2`), and some random content. The generated content to write should be printable; for example, you can use the function provided in the previous project.

Each time a write or read action is performed, your test program should print:

- its type (read or write),
- written or read content,
- logical block number,
- corresponding cylinder, head, and sector, and
- time of the operation.

BlanchardSeanProject02.zip (https://cilearn.csuci.edu/files/2406517/download)

Question 3

Not yet graded / 40 pts

Step 3

Description | Implementation | Notes | Submission

Description

In this part, you will incorporate your simulated device in a Linux device driver for a block-based device.

Start with the tutorial **Implementing Linux Device Drivers.**

Then, incorporate the code that you implemented in Step 1 with the tutorial code, so that the driver interfaces with the the simulated disk.

BlanchardSeanProject02.zip
(https://cilearn.csuci.edu/files/2406570/download)

Unanswered

Question 4

Not yet graded / 15 pts

Step 4

Description | Submission

Description

In this task, you will extend the timer-based tester of the disk simulator that you implemented in Step 2 so it can work as a writer or as a reader depending on the provided command line argument. With that extension you can run a number of readers and writers from different terminals. That will test your implementation thoroughly ensuring that concurrent clients are not destructive to each other.

If the tester is started with "`-r`" on the command line, then it works in the reader mode.

```
$ ./test_cidev -r
```

If the tester is started with "`-w`" as a command line argument, then it works in the writer mode.

```
$ ./test_cidev -w
```

The application should check the status of the device before undertaking any action. If the device is busy, then it can be neither read nor written; otherwise, any operation is allowed.

Quiz Score: **70** out of 100

This quiz score has been manually adjusted by +70.0 points.