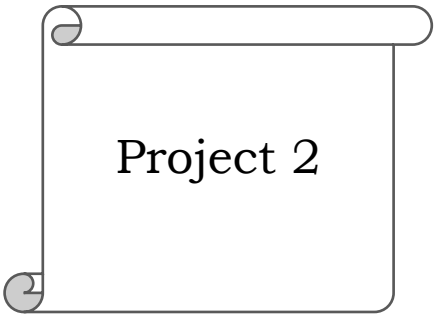
Amirkabir University of Technology

Department of Computer engineering and information technology

Operating systems



Authors:

Kasra Mojallal

Kasra Farrokhi

Professor:

Dr. Nastooh Taheri Javan

Overview of the project:

In this project you have three major parts, first is to learn about Linux Kernel and its module programming, after that you will implement some scripts and the last part is about XV6. These parts are separate and there is no need to do them in a sequence.

It is essential to have a report, so that you answer the questions that are asked throughout the project.

- ## Part 1 – Linux Kernel

In this part you will learn about Linux Kernel and its module programming.

1.1

First thing to do is that you should learn from the internet how to compile and install Linux Kernel on your system. When you do so, you will learn how to install it with your features and modules so you can have a Kernel installed fully compatible with your needs.

The whole point of this part is to go throw the phases of compile and installing a kernel on your system but since the installation takes a lot of time, you just have to go throw the parts and not the last installation part.

You have to write a 3-4 paragraph report of how this task was done and answer this question after.

- what is Kernel?
- what is user mode and kernel mode?
- what happens if ext4 driver is removed?

1.2

This task is pretty easy, you just have to write a simple hello world module and run it.

Here you find a complete guide.

For the purpose of this part you just have to read the first two sections of this guide.

You have to write a 3-4 paragraph (detailed) report of how the task is done and also you should put the final (.c) code and your Makefile on the project folder. Last but not least, you have to answer these questions as well:

- what would happen if there was no driver module?
- what is the difference between **insmod** and **modeprobe**?
- what are init_module and cleanup_module?

1.3

For this part you have to write a module that for all the processes gives this information:

pid: name: state:

parent_pid: parent_name: parent_state:

- Part 2 – Scripts

In this part you should learn some Linux scripts. Please write your answers in the report.

1. What is the command to create three directories that dir3 is a sub directory of dir2 and dir2 is a sub directory of dir1?
2. Use cat to create a file which has the following content: one-two-three-four-five
3. Can you access the root directory and why?
4. Where is the cat command stored?
5. Create two variables, first one with the value Dumb, and the second with the value do, then use echo to printout Dumbledore.
6. Using to character, execute the previous command again.
7. How can we store the users of the bash in a sorted list with the name bashusers.txt?
8. How can we write a command to execute two commands with a sequence?

- Part 3 – XV6

In this part you have to implement ticket lock in XV6

3.1

You are all familiar with Critical section, you know that when a process enters that section, no other processes should be allowed to enter their section for the obvious reasons. To handle this problem, the process has to acquire a lock when it enters that section and release it when it is done what it was doing.

This situation is handled with acquire and release functions in XV6, which causes busy waiting. On the other hand, there are two other functions, acquiresleep and releasesleep which do not have the busy waiting problem.

That being said, there is another option too. Any processor which wants to acquire a lock, is given a ticket. Every time processor acquires a ticket number, it servers the processes which owns that ticket number and the process can lock the processor and release it after it is done.

This ticket lock method follows the FIFO rule so there is no starvation, and that is a big advantage over the spinlock method.

In order to implement this method, you have to take a look at spinlock.c and spinlock.h, and then figure out how to do it.

Such as spinlock, you need to have ticketlock.c and ticketlock.h. (you have to create them)

You have to add two system calls. ticketlockInit, to initialize the ticket lock method and ticketlockTest to run this method.

You are provided with the test code.

Last but not least, in order to add to the number of ticket, you should use inline assembly so that it runs atomic. (it should be added to x86.h)

3.2

With the use of ticketlock, you should implement readers-writers lock with the advantage for readers. To do this you have to add two system calls called rwinit and rwtest.

rwtest is same as the one you learned in class that has the algorithm for accessing the shared data when a reader comes or when a writer comes.

If you read the test file provided for you, you realize that when you start, you are asked to enter a series of ones and zeros (ones for writers and zeros for reader) that show the sequence of readers and writers. (the first number must always be 1)

By doing this, the processor knows due to its access it must request for rwlock. You should assume that the shared data is a variable that is added to it, every time a writer, writes to it.

You are also provided with the test file for this part as well.

It must be noted that, since every one's code is different from others, provided codes might need some change. Feel free to change it as you like but the result must be the same.

If you have any questions feel free to contact us:

Questions 1 and 3:

Email: kasramojallal1@gmail.com Telegram: @kasramojallal

Question 2:

Email: farrokhi.kasra@gmail.com Telegram: @kasrafarrokhi