Project name

P. Rossi

December 28, 2021

Contents

NOTE: IT IS MANDATORY TO FILL IN ALL THE SECTIONS AND SUBSECTIONS IN THIS TEMPLATE

1 Project data

- Project supervisor(s): Federico Reghenzani
- Describe in this table the group that is delivering this project:

Last and first name	Person code	Email address
Davide Li Calsi	10613807	davide.li@mail.polimi.it

- Describe here how development tasks have been subdivided among members of the group, e.g.:
  - Being the only member of the group, I did everything myself.
- Links to the project source code; Put here, if available, links to public repos hosting your project: https://github.com/DavideLiCalsi/AOS

2 Project description

2 pages max please

- What is your project about?
- Why it is important for the AOS course?

For those who choose to work on an open source project, please put here any reference/copy of messages exchanged with project maintainers to **identify the subject of the pull request**.

My project was about the im plementation of a Publish/Subscribe inter-process communication mechanism in Linux. The idea was the implementation of a Linux kernel module for that specific purpose. Processes should be able to create a new topic by writing its name to a special character file /dev/newtopic. Once a topic has been created, processes can specify a signal code for that topic and/or subscribe to it and/or publish a message for that topic. Whenever new content is published, the subscribers receive a signal of the same type as the one stored in /dev/topics/<topic_name>/signal_nr.

The project is revelant for the AOS course because it consists in writing a Linux kernel module, i.e. you are expanding your OS by adding new functionalities. It was a chance to write some code that executes in priviliged mode, which (I learned the hard way) requires a lot of attention. Furthermore, while working on the project I had to apply several concepts seen during lectures such as kernel memory allocation or kernel synchronization, dealing with a lot of data strucures mentioned in the course. Moreover, I had the chance to actually implement some character device drivers.

2.1 Design and implementation

Describe here the structure of the solution you devised. Note, don't put major parts of the source code here; if you can, put hyperlinks to existing repos.

For those who choose to work on an open source project, please put here an history (mail messages/github issues etc..) of the interaction with the development team that helped you identify such design and the code reviews that helped you improve it.

Those that have chosen to present a paper can exclude this section.

IMPLEMENTATION

The core structure was suggested to me by my supervisor. First, my module creates the character driver /dev/newtopic. A process can simply write a string to it to register a new topic. This triggers the add_new_topic(char*) function, which registers the topic by creating the struct topic_subscribe. This struct contains all key information related to the new topic. Furthermore, add_new_topic creates 4 more character drivers under the folder /dev/topics/<topic_name>:

- **subscribe:** any process that wishes to subscribe to a topic simply needs to write any string to it
- **signal_nr:** special file used to set or retrieve the signal that the system should send to the topic's subscribers
- **subscribers:** contains a list of pids, one for each task that subscribed to the topic
- **endpoint:** write to it to publish new content, read from it to retrieve the latest content

For each of these files, I wrote 4 functions that are called when specific operations are executed on the file: one for file opening, one for file reading, one for file writing, and finally one for file releasing. They were all register as fields of the appropriate struct file_operations, stored in the struct topic_subscribe.

For the purpose of storing the pids of each topic's subscribers I used the pre-implemented struct list_head (list_head source code), since it provided a flexible, easy-to-use and bug-free implementation of bidirectional linked lists. The list of subcribers is also stored in the struct topic_subscribe.

The module was also designed to prevent the most common errors (though some less evident bugs might still be present of course). e.g it is impossible to create two topics with the same name, to subscribe twice to the same topic...

Finally, the head of the source file defines some macros that limit some quantity in the module. For instance MAX_SUBSCRIBERS is the maximum number of subscribers that can subscribe to a topic. These macros can and should be appropriately tuned, according to the real-life scenario in which we plan to adopt the IPC mechanism.

As for locking, I used two different mechanisms. To regulate concurrent access to the list of subscribers, I used read/write spinlocks (rwlock.h), because the module naturally has a good read/write pattern when it comes to accessing that data structure. Infact whenever a task accesses the subscribe file it meaans (and can only) write the list of subscribers. Also, if the same data structure is accessed from the subscribers file, or after a write to the endpoint file, someone must be trying to read the full list. Thus I decided to use rwlocks in order to allow for a higher degree of concurrency, since many readers are allowe to read at once (and I assume subscriptions will less frequent than reads).

On the other hand, I used regular spinlocks to lock the signal number field and the message buffer of each topic.

3 Project outcomes

3.1 Concrete outcomes

Describe the artifacts you've produced, if possible by linking to repo commits. For those who choose to work on an open source project, please put here the **URL to your final pull request**.

Those that have chosen to present a paper can include a link to the slides.

I created a linux kernel module source file that you can compile to obtain a functional kernel module for inter-process communication using Publish/Subscribe.

3.2 Learning outcomes

What was the most important thing all the members have learned while developing this part of the project, what questions remained unanswered, how you will use what you've learned in your everyday life? Please also indicate which tools you learned to use.

Examples:

- Foo learned to write multithreaded applications, he's probably going to create his own startup with what she has learned. She also learned how to debug with gdb.

- Bar learned how to interact with the open source community, politely answering to code reviews and issuing pull requests through Git.

First of all, I have learned the pain of making mistakes while working directly in kernel mode. I had to give up on several virtual machines because of some mistakes, and also reinstalled my OS once (because I was overconfident and tried my first module on my machine).

I have also learned how to extend my Linux distro, enriching it with new useful features. This skill is valuable to me, since I like the idea of customizing my machine in the way I like. I think I will have a good time (painful but good) writing more kernel modules in the future, creating more and more functionalities for my personal interest.

Finally, I have learned more about the OS internal mechanism, especially character special files. I had heard of special files for years, so it was nice to learn how this mechanism that I used quite a few times works under the hood.

3.3 Existing knowledge

What courses you have followed (not only AOS) did help you in doing this project and why? Do you have any suggestions on improving the AOS course with topics that would have made it easier for you?

Of course, every course in which I had to write C code was useful for this project (Fondamenti d'Informatica, Algoritmi e Principi dell'Informatica...). The ACSO course was also quite helpful because it had already provided the notion of kernel module and even few toy examples to write one. However , of course most of the concepts that I had to reuse were from the AOS course. I don't think you should add further topics for this project's development.

3.4 Problems encountered

What were the most important problems and issues you encountered? Did you ever encountered them before?

- At first I had some issues with character device drivers. I did not immediately understand how to register ones, overall the data structures required to create a file and the file_operations functions sounded a bit messy to me, but after some researches and trial-and-error I succesfully figured it out.
- Mistakes. A mistake in user space results in a segmentation fault et similia. A mistake in kernel space... some were not that disruptive, and only resulted in scary error alerts, but I had to setup a couple virtual machines.
- Writing the appropriate read functions for the struct file_operations. At first I did not know that I had to return 0 when the reading operation was complete, or that I had to update the offset. Therefore, whenever I would attempt to read the subscribers list for example, the cat command would keep on reading the same data in an infinite loop.
- Adquate locking. Selecting the most appropriate locking mechanism cause a bit of trouble, becasue I was slightly confused. Also I was unsure about where and when to lock in the code.

4 Honor Pledge

(**This part cannot be modified and it is mandatory to sign it**)

I/We pledge that this work was fully and wholly completed within the criteria established for academic integrity by Politecnico di Milano (Code of Ethics and Conduct) and represents my/our original production, unless otherwise cited.

I/We also understand that this project, if successfully graded, will fulfill part B requirement of the Advanced Operating System course and that it will be considered valid up until the AOS exam of Sept. 2022.

Davide Li Calsi