ESE 333: Real Time Operating System

Kernel pub/sub System
Project 4

1 Introduction

In this project, you are to design a pub/sub system that allows consumer and producer processes to send/receive messages between each other through the kernel. You will use *netlink* to pass data between the kernel and userspace. You are to use standard kernel's *linked list* API as a data structure and hashmap (*rhashtable*) if necessary. The project is to be done in 3 stages with a bonus 4th stage available.

2 Part 1: (Milestone) Producers, Consumers, Mutexes, and Threads

In this stage, you are to create a userspace, thread based producer-consumer program. You will modify skeleton code to implement the producer and consumer methods. You will need to make use of a mutex lock to prevent segmentation faults. Download Proj4 and modify ProdCons.c to implement the producer and consumer methods. Hint: If you want to add print statements but want to limit the printouts checking if *count % 10000 == 0* is a good way to limit printouts.

3 Part 2: Basic Netlink PubSub network

In this stage you will create a way for processes to communicate by passing data to the kernel, then the kernel must decide which userspace processes to forward the data to. Please refer to section 6 to view some tutorials about *netlink* (read with caution, the API might have changed). The Kernel and Userspace folders contain the code from netlink3.

Once you get this basic netlink demo running between one process and your kernel module, you are to extend it to support 3 processes. One of the processes is to be a publisher (producer) and the other two subscribers (consumers). The publisher will send messages and the two subscribers will receive them from the kernel accordingly.

Your kernel module must be able to distinguish between subscribers and publishers. We recommend using a 1 byte field at the beginning of your message payload to indicate attributes such as the type of the process etc.

For each process to register at the kernel driver, it has to send one message to the kernel, in that message it indicates if it is a subscriber or publisher. Once that is done, the publisher should be allowed to send further messages that the kernel module has to forward to the other two userspace subscriber processes.

Your userspace code can be contained in one program that upon start asks whether it is a publisher or subscriber (e.g. ask the user to input 0 for subscriber, 1 for publisher) or can be contained in two programs, subscriber and publisher. If it is a subscriber, it runs an infinite loop reading messages from kernel driver and prints them out after registering at the kernel module. If it is a publisher, it keeps reading user input, parses it, and sends messages to the kernel module, which will further pass them to the other 2 subscriber processes.

4 Part 3: Kernel Linked Lists and Userspace threads

In this stage, you are to modify your userspace and kernel code to support an non-fixed number of processes, each could be a publisher or subscriber.

To allow the kernel module to support as many processes as it needs, you are to use kernel's linked list (please refer to section 6) to allow dynamic addition of any new processes as publisher or subscriber. You are to keep list of each subscriber's PID and list of each publisher's PID in the kernel.

To allow userspace code to wait for messages from the kernel and send concurrently, you are required to use *pthread* (please refer to section 6) where each userspace process has two threads, one for publishing (i.e. sending data to kernel) and one for subscribing (i.e. waiting for messages forwarded from kernel).

Your kernel code must be cleaned appropriately (i.e. memory freeing) once *rmmod* is called.

5 Fourth Stage (30% bonus)

In this stage, you are to allow different topic subscriptions. Thus, when a process is subscribing, it will indicate the topic name (a byte indicating messages of different content, e.g., 'stock', 'weather', 'road conditions') on which it wants to receive messages. The topic name is set to be of size 8 bytes. You are to use *rhashtable* (please refer to section 6) with the topic name as the key. Inside each entry, it should have the linked list of subscribers that your kernel module will iterate over to send messages.

For each publisher, it must indicate the topic name on which it will publish messages.

Note: The topic name should not be transmitted with the message back to the subscribers.

6 Useful References

- ABI
- netlink1, netlink2, and netlink3
- Kernel Linked List

- Pthread
- rhashtable