PA#5: Interprocess Communication Mechanisms

Due: Tuesday 4/21/20 at 11:59pm

Introduction

Two programming assignments PA2 and PA4 have made heavy use of the `FIFORequestChannel` class, which was pre-written and given to you along with the server code. If you did look at them, you may have noticed that the `FIFORequestChannel` class uses a mechanism called "named pipes" or "FIFOs" to communicate between the two sides of the channel. However, FIFOs are only one of several different IPC mechanisms, each of which have their own particular uses that make them suited to particular applications. In this programming assignment, we are going to expand our toolbox by learning about 2 "new" IPC mechanisms in addition to named pipes: *message queues* and *shared memory.*

Background

Message Queues

While pipes provide a single unidirectional byte stream between two processes, message queues are slightly more sophisticated in the sense that they can be used between 2 or more processes. There are POSIX library functions as found in `mq_overview` for message queue opening/creation, sending messages, receiving messages, closing the message queue, deleting the message queue, and modifying the message queue's attributes. You may be able to use default attributes for this assignment. However, keep in mind that those defaults may vary by system. Visit the man pages for `mq_overview` (note that this is POSIX IPC, not the System V IPC, which is the older way) for how to check and set default message queue attributes. In the end, you will have a class called `MQRequestChannel` which you can use in place of `FIFOReqeustChannel` such that all communication between the server and client is throught message queues instead of FIFO.

The Assignment

Code

You have to start off of your code from PA4. We are assuming that you have a working PA4. If that is not the case, please contact us for a working version of PA4.

You then have to make up 3 versions (i.e., really just 3 modes of running the same code) of your PA5 each using a separate IPC-method-based request channels: FIFO, message queue, and shared memory. Call these versions `PA5_FIFO`, `PA5_MQ`, respectively. You should have an abstract `RequestChannel` class and 3 sub-classes:

- `FIFORequestChannel`

- `MQRequestChannel`

The API of base class `RequestChannel` should be as follows:

```
class RequestChannel {
public:
    /* some public constants */
    typedef enum {SERVER_SIDE, CLIENT_SIDE} Side;

protected:
    /* all IPC mechanisms will need a unique ‘‘name’’,
    so it is common to all */
    string my_name;
    /* all channel will need to know which side it is */
    Side my_side;
public:
    /* CONSTRUCTOR/DESTRUCTOR */
    RequestChannel (const string _name, const Side _side){
        my_name = _name, my_side = _side;
    }
    virtual ~RequestChannel(){}

    /* The following 2 pure virtual (i.e., =0 means pure) means
    they must be overridden in subclasses
    (e.g., FIFOReqeustChannel, MQRequestChannel) */

    virtual int cread (void* ptr, int len)=0;
    virtual int cwrite (void* ptr, int len)=0;

};
```

Here, the first one `FIFOReqeustChannel` would be taken “almost” directly from PA4, except that you need to extend the base class `RequestChannel` and add required member variables. It must also override the `cread` and `cwrite` functions. The following code snipped shows it partially. Note that

```
class FIFORequestChannel: public RequestChannel {  // extends the base class
    private:
        /* the following are FIFO specific, because
        you know you are going to need 2 fd's */
        int rfd;   // read pipe descriptor
        int wfd;   // write pipe descriptor

        string rfname; // read pipe name
        string wfname; // read pipe name
    public:

        /* CONSTRUCTOR/DESTRUCTOR */
        FIFORequestChannel (const string _name, const Side _side);
        ~FIFORequestChannel();

        int cread (char* ptr, int len);
        int cwrite (char* ptr, int len);

};
```

Compiling and Running Your Code

There should be only 1 `makefile` to compile everything together.

Take an additional runtime argument option "-i" which should get one of:

- "f" for FIFO
- "q" for message queue

The following is an example command to run PA5:

```
./client -n <requests/person> -b <bounded buffer size>
           -w <number of request channels>
           -m <buffer capacity>
           -i <f|q>
```

You must resolve the derived type of `RequestChannel` class in the runtime using `polymorphism` and `run-time binding` in C++. That means that based on the value of argument "i", you must choose what type of `RequestChannel` you are going to use. The following is block of code that show how to acheive that:

```
   RequestChannel* channel;

   if (i == "f"){
       channel = new FIFOReqeustChannel (name, side);
   }else if (i == "q"){
       channel = new MQReqeustChannel (name, side);
   }
```

Clean Up

You must clean up all IPC objects from the kernel memory and all temporary files you created. You can check the IPC objects currently persisting in your system by listing the `/dev/mqueue` directory or by running the `ipcs` command in shell. In addition, you should clean all heap-allocated objects.

Report

- Gather timing data on the same set of n, b, w, m arguments on each of `PA5_FIFO` and `PA5_MQ`. Note that you need to modify the default capacity of the message queue to test out different m values that are larger than the default.
- Present a performance comparison of the different IPC mechanisms based on this data, and attempt to provide explanation for any differences and similarities.
- Present the results in separate graphs using `PA5_FIFO` (i.e., PA4) as the baseline for comparison.
- What is the maximum w and thus the max number of `RequestChannels` that you can use for each IPC? How much more can you go beyond the limit in `PA4`? (recall that the limit imposed by how many file descriptors each user can have.

- What are some of the limits encountered by each class, either due to the specific implementation or to operating system limitations, and how does the program behave when it encounters them?
- Describe the clean-up activity you have done for each IPC

What to Turn In

Turn in a single zip file `PA5.zip` containing the report, all the class files (separated into `.h` and `.cpp`) and a `makefile`. Note that the same `makefile` should build all request channel versions (FIFO and MQ).

Bounus - worth 50% points

In the bonus part, the task is to implement another request channel type based on Shared Memory, which let us assume, is called `SHMRequestChannel`.

Shared memory is exactly what it sounds like: a segment of memory that can be read and modified (depending on its configuration) by multiple different processes. You will notice that there are no system calls for reading and writing the shared memory segment. This is because a shared memory segment is semantically identical to any other memory segment, such as can be obtained from `malloc`, except for the IPC and synchronization considerations. One can read and modify it using `memset, strcpy, memcpy`, or just about any other memory-reading/writing operations.

While requesting a shared memory segment from kernel is similar to what we did in FIFO and MQ, the main differences comes from kernel-provided synchronization or lack there of.