PA3: IPC Methods

Rubric

1. [10 pts] Building the class hierarchy and FIFORequestChannel
    a. [2 pts] FIFO as a subclass of the base request channel class. Note that the destructor, `cread()` and `cwrite()` functions all need to be virtual for this to work. Deduct 5-10 points if any of these functions are not virtual.
    b. [5 pts] The base class should have as much of the common things to itself as possible. Repeating thighs unnecessarily in the subclasses moves it away from the OOP principles. Deduct 3-4 points if common things (e.g., name() function, two IPC objects' names, the file descriptors) are still in subclasses (e.g., FIFO, MQ, or SHM channels). Things specific to the derived classes should be in them. For instance, each subclass should have their own destructors, constructors, `cwrite()` and `cread(), open_ipc()` functions
    c. [3 pts] Makefile should have entries added for FIFO/MQ/SHM RequestChannels
2. [15 pts] MQRequestChannel:
    a. [5 pts] Constructor: Create 2 MQ objects with names derived from the channel name and by calling the `mq_open(name, O_RDWR|O_CREAT, ..)`. These modes are necessary so that this function does not fail depending on which order the scheduler follows to run the client side and the server side. For instance if you always create the channel from the server and just connect to it in read only mode from the client, you must then make sure that the server reaches the constructor first in some way. This dual mode in the above makes the constructor avoids such order dependency and makes sure that your program always works
    b. [5 pts] Destructor: Should destroy both MQ IPC objects (i.e., mq_close()) and delete them permanently (i.e., mq_unlink()). Check using "ls /dev/mqueue"
    c. [5 pts] send and receive functions: send is straightforward. In the receive you should either hardcode 8KB as the receive buffer size when calling mq_recieve() or query MQ buffer size from the system and use the return value. Either method is acceptable because 8KB is the default size in almost all linux systems
3. [20 pts] SHMRequestChannel (SHMRC):
    a. [10 pts] Constructor: The two objects used here are not barebone Shared Memory segments. Rather they are kernel-semaphore-protected SHM segments that provide necessary synchronization for send and receive functions. Call this class SSHM (i.e., synchronized SHM) or SHMQ (SHM queue), which does a Producer-Consumer synchronization using 2 kernel semaphores: one indicating whether the sender is done and the other indicating whether the receiver is done. Therefore, the SHMRCl constructor just creates 2 SSHMs, where each SSHM constructs 1 SHM segment (i.e., by calling shm_open, ftruncate, and mmap()) and 2 kernel semaphores protecting it with right initialization (i.e., sender_done = 0, recv_done = 1)

   b. [5 pts] Destructor: SHMRC destroys the two SSHM objects. Each SSHM destroys the SHM segment (i.e., `shm_close, munmap(), shm_unlink()`) and the 2 semaphores (i.e., `sem_close(), sem_unlink()`)
   c. [5 pts] Send and Receive: They call the respective send and receive functions of the SSHM class who first does `sem_wait()`, then copy the message in/out and finally does `sem_post ()` to signal the other side

4. [30 pts] Adapting client.cpp to work with the class hierarchy
   a. [5 pts] Adding getopt arguments "-i" for IPC method and "-c" for number of channels in the client.cpp
   b. [3 pts] Making a child process using fork() and executing the server using exec(). Sending buffer capacity "-m" and ipc method "-i" to the server when doing fork() + exec()
   c. [2 pts] Requesting and connecting to all the `c` channels of the type specified by "-i"
   d. [5 pts] Requesting 1 data point through each of these channels when typed the following command:
      `./client -p <patient no> -t <time stamp> - e <ecg no> -c <no of channels> -i <ipc method>`
   e. [5 pts] Requesting and timing 1K data point through each of these channels when typed the following command:
      `./client -p <patient no> - e <ecg no> -c <no of channels> -i <ipc method>`
   f. [5 pts] Requesting and timing a file split through the channels (if file size is s, each channels should collect roughly s/c bytes for this one) when typed the following command:
      `./client -f <file name> -i <ipc method> - m <buffer capacity>  -c <no of channels>`
   g. [5 pts] Proper clean up: Deleting all heap-allocated objects, sending QUIT_MSG through all channels and waiting for the server to exit using `wait()/waitpid()`
5. [10 pts] Adapting server.cpp to work with the class hierarchy
   a. [5 pts] Should accept the "-i" argument for the IPC method, create the correct base class object using it for both the main function and the new channel function
   b. [5 pts] All methods should be changed to accept and work with just the base class's pointer and they should be able to work with derived classes' objects accordingly
6. [15 pts] Report: Design description. Comparison of the 3 methods using timing of 1K data points and split file requests. You are not required to make a graph. Include a youtube link to your demo video, which is prepared using the following instructions.

Demo Instructions

1. Show the structure of all your RC classes according to points 1, 2 and 3 of the rubric. Demonstrate that you have:
    a. Followed OOP principles
    b. Fulfilled requirements for the IPC methods
2. Show your client.cpp all the changes you have made and describe why you have written your code in the way you have. We want to see that you understood and then used the lecture video instead of blindly following.
3. Show your server.cpp in the same manner and demonstrate your understanding of it.
4. Run the following to request 1K data point through each of the **c=5 channel**s under each method and show the time taken:

```
$./client -p 10 -e 1 -c 5 -i f
$./client -p 10 -e 1 -c 5 -i q
$./client -p 10 -e 1 -c 5 -i m
```

They all should have the same output except for debug messages

5. Run the following to request a 1 MB or larger file split through c=50 channels under each method and show the time taken. Demonstrate that the received file matches the source. Also pull your code to show that you split the file through these channels.

```
$truncate -s 1000000 1MB.dat
$./client -f 1MB.dat -c 50 -i f
#diff received/1MB.dat BIMDC/1MB.dat
$./client -f 1MB.dat -c 23 -i f # requesting through 23 channels
#diff received/1MB.dat BIMDC/1MB.dat
$./client -f 1MB.dat -c 50 -i q
#diff received/1MB.dat BIMDC/1MB.dat
$./client -f 1MB.dat -c 35 -i f # requesting through 35 channels
#diff received/1MB.dat BIMDC/1MB.dat
$./client -f 1MB.dat -c 50 -i m
#diff received/1MB.dat BIMDC/1MB.dat
$./client -f 1MB.dat -c 19 -i f # requesting through 19 channels
#diff received/1MB.dat BIMDC/1MB.dat
```