PA#3: IPC Methods Discussion

Tanzir Ahmed

Main Goal

- Objective: To explore the details of POSIX IPC methods
- Tasks:

1) Rewrite our RequestChannel (based on FIFO) class using 2 new IPC methods:
    - Message Queues (called MQReqeustChannel)
    - Shared Memory (called SHMReqeustChannel)

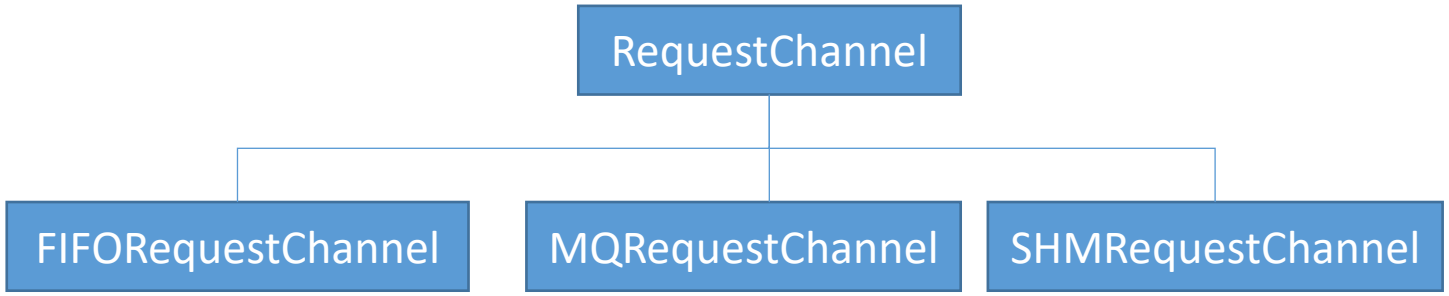
In addition, also rewrite FIFORequestChannel class with the inheritance structure in place (more about it coming up)

2) Then see the relative advantages and disadvantages of having these IPCs as the underlying methods
    - Are these new methods any faster/slower than the FIFO method originally used? Which method has the least overhead and which one has the highest
    - Can we have more number of RequestChannels compared to the case in FIFO? What are the sys admin limits now?

Whaere to Start?

The major changes are confined within your RequestChannel implementations. You now have 3 sub-types of the base RequestChannel class:

- FIFORequestChannel
- MQRequestChannel
- SHMRequestChannel



- Which channel you use?
  - Will come from the command line argument.
  - For client side:

```
./client –c <# of channels> –p <patient> -e <ecg no> -i <IPC method(f|q|m)>
```

  - For server side:

```
./dataserver -i <IPC method(f|q|m)> -m <buffer capacity>
```

- See the handout for more detailed info about the command format

What to Change Inside RequestChannel.h

- The RequestChannel is now an abstract class.
  - You have to pull out the FIFO-specific logic from it and put into the FIFORequestChannel class

```
class RequestChannel {
public :
        // things common to all types of channels
public :
      RequestChannel (...) ;
      virtual ~RequestChannel ( ) ;
      virtual int cwrite (char* buf, int buflen)=0; // writing a
message, its pure virtual
      virtual int cread (char* buf, int buflen)=0; // reading a
response, it is pure virtual as well
};
```

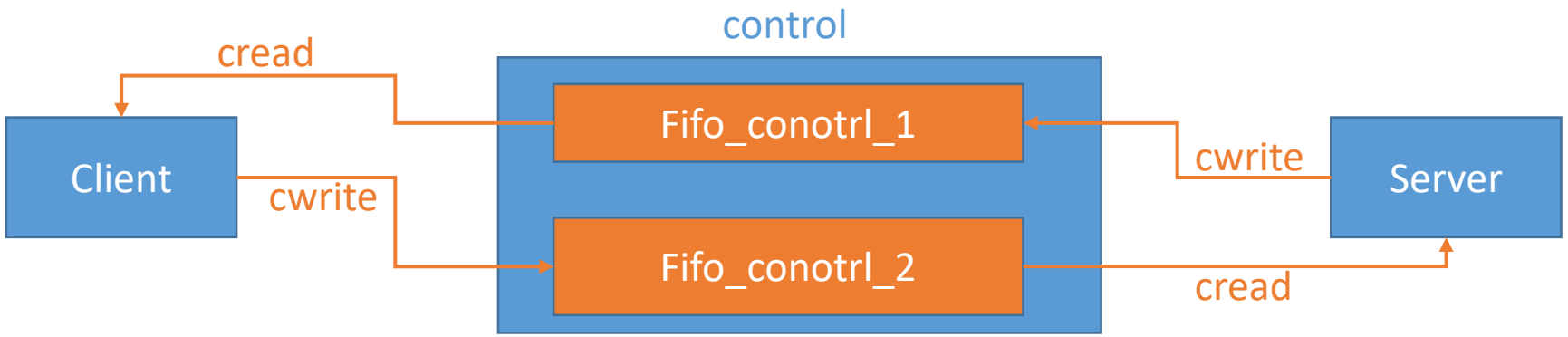
FIFO/MQ/SHM RequestChannel Classes

- They will just extend the base class:
  - And add anything specific to that particular IPC method

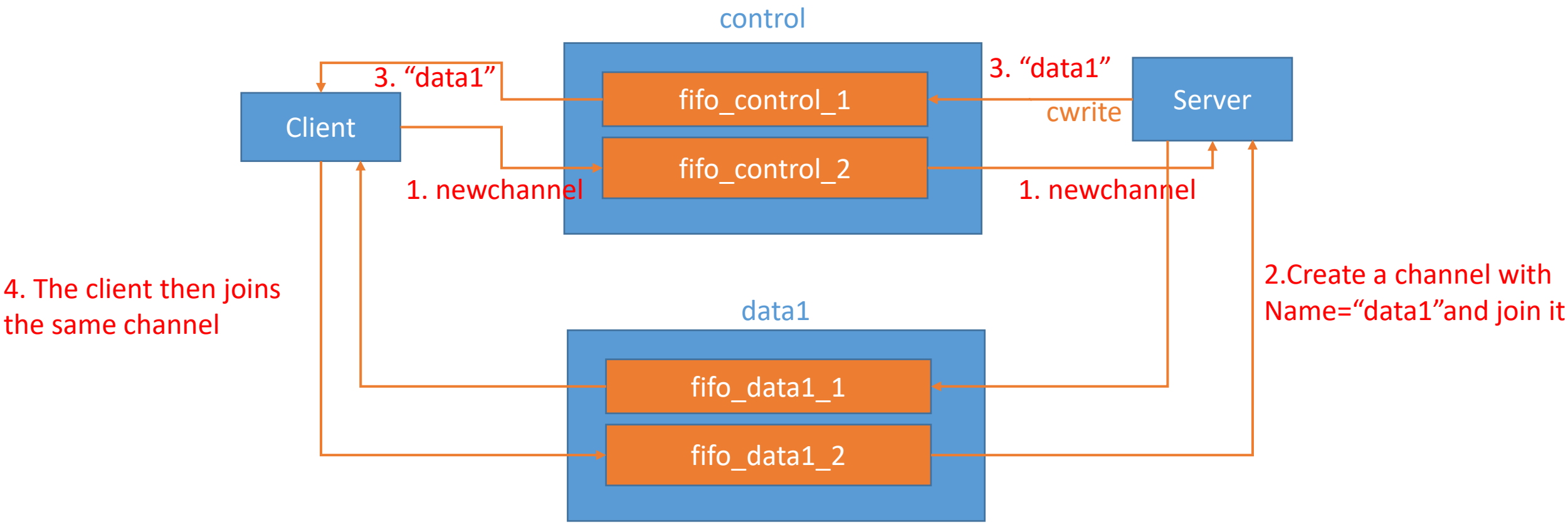
```
class (FIFO/MQ/SHM)RequestChannel:public RequestChannel {
private:
// add necessary method-specific member variables (e.g.,
msg_queue_id, shm_id, sem_id
public :
     (FIFO/MQ/SHM)RequestChannel (...) ;
     ~(FIFO/MQ/SHM)RequestChannel ( ) ;
     int cwrite (..); // overrides the virtual cwrite
function
     int cread (..);  // overrides the virtual cread()
function
};
```

Start with Understanding of the Original RequestChannel

1. Both the client and the server creates (the server creates it and the client joins it) the “control” channel
2. Underneath, there are 2 named pipes (“fifo_control_1” and “fifo_control_2”) created
   - 1 for each data direction (from client to server and the other way)

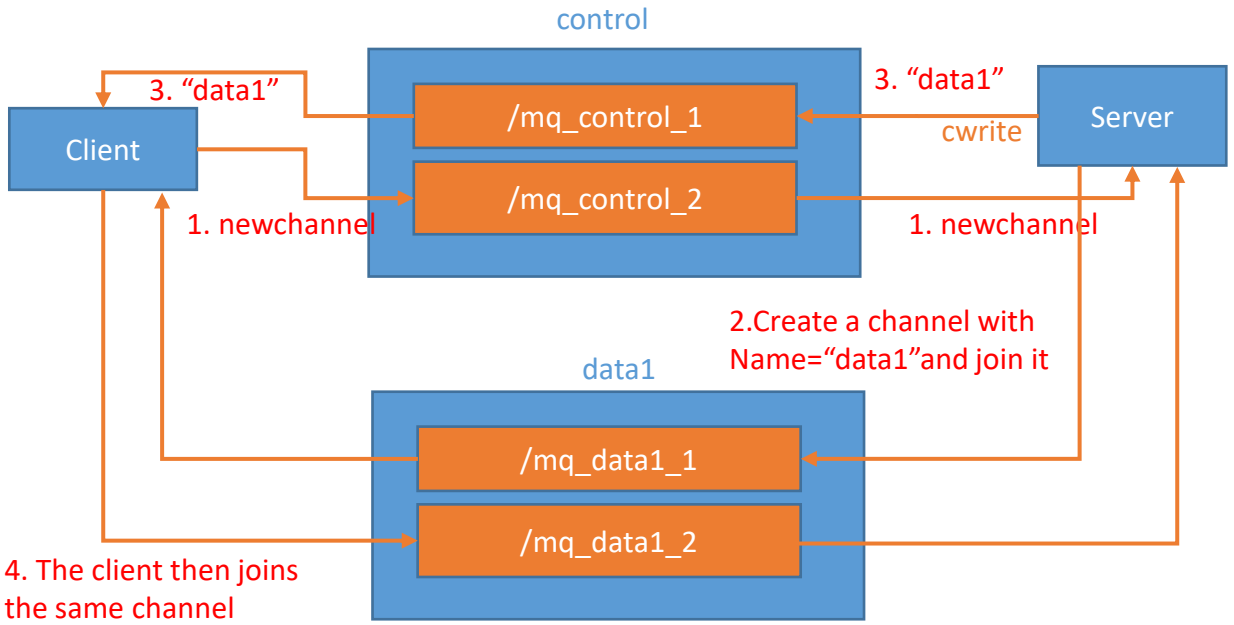


Creating A New Request Channel



How About Message Queue

- The same protocol can be used, with some small changes:
  - The names of MQ objects must start with a “/”
  - They use MQ descriptors instead of file descriptors
  - There is no separate make step, just call mq_open to achieve that
  - The cwrite and cread functions are written using mq_send and mq_recv respectively (recall that FIFO used just write and read)

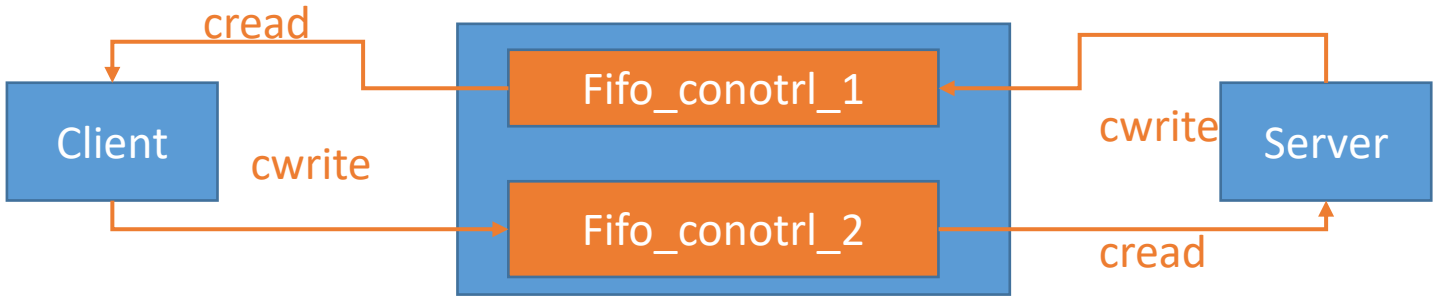


Synchronization

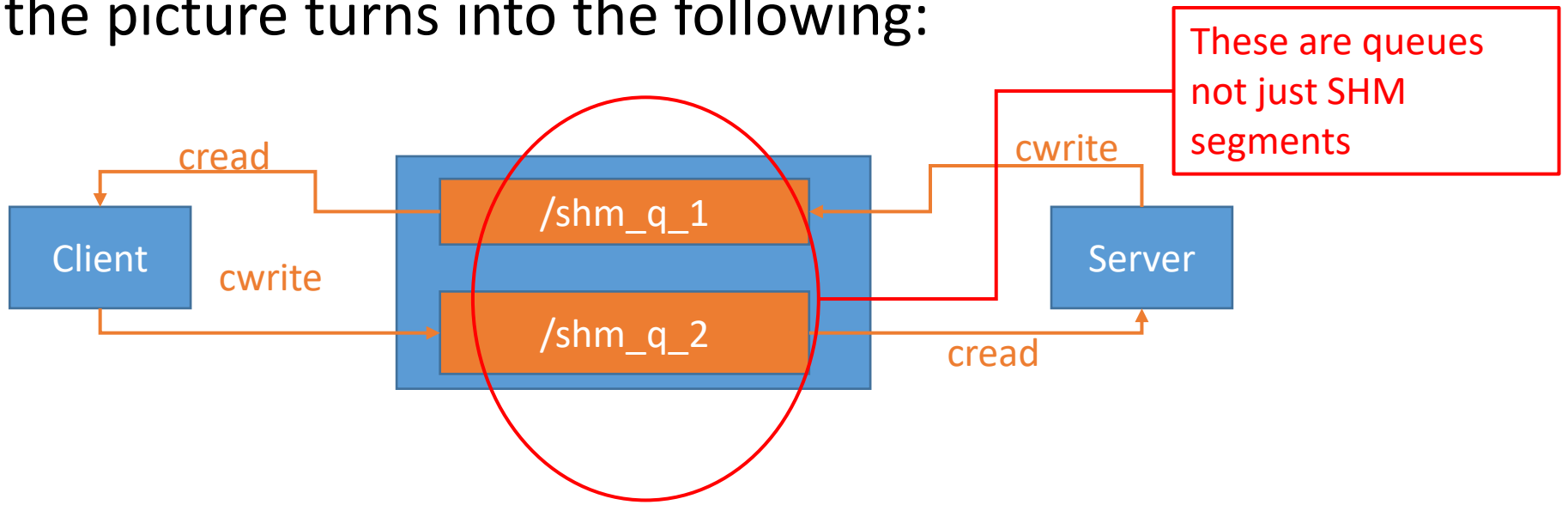
- Note that MQ's provide use with a byte-stream where send and receive functions are **synchronized** by the kernel
  - This is similar to FIFO ReqChannel
  - For that reason, MQ implementation is rather straightforward
- But messages are not synchronized in SHM
  - You need to support synchronization using **Kernel Semaphores**
  - Otherwise, **`cread()`** and **`cwrite()`** functions in SHM would be meaningless

Shared Memory RequestChannel

- This is what we had in FIFO (also in MQ)



- But in SHM, the picture turns into the following:



Shared Memory Queue (size = 1 msg) with Semaphore

```
class SHMQueue{
private:
    char* shmbuffer;
    sem_t* readerdone;
    sem_t* writerdone;
public:
    SHMQueue (char* name){ // constructor
      /* 1. create a SHM segment using the name
      and connect to it
      2. Create 2 semaphores: readerdone and
      writerdone using some string derived
     from the name (e.g., name_1 and name_2)
        */
    }
```

```
int cwrite (void* msg, int len){
   /* 1. Wait for the reader side to make sure that is done
first. This sync will happen using Kernel Semaphore */
   sem_wait (readerdone, ...);

    /* 2. now write the data */
    memcpy (shmbuffer, msg, len);

    /*3.Notify the reader that that new data is available */
    sem_post (writerdone, ...);
}

int cread (void* msg, int len){
    /*1. Wait for the writer side until data is available */
    sem_wait (writerdone, ...);

    /*2.copy from the shmbuffer to msg to recive the data */
    memcpy (msg, shmbuffer, len);

    /* 3.Notify the writer that reading is done */
    sem_post (readerdone, ...);
}
}
```

Bonus Part (12 points)

- A straight-forward way to achieve Full-duplex is to use one IPC object per direction
- The bonus part asks you to implement it using just one IPC object
  - Do not make 2 MQ object for the MQ Request Channel – rather use only one
  - The same for SHM
  - Can you use just 1 MQ for all the request channels??
- **Hint:** Look at the msqsend/recv functions for MQ (message type can be handy) and think about logically splitting the shared memory segment into two parts

Report

- The report is worth **25** points, which means that functionality is **75** points this time
- Description [5 pts]: It should describe your design briefly and also the constructors and destructors
- Data Collection [10 pts]: The most points in the report will be how effectively you collected data
  - You should fix n=100K, b=100, and then vary w within the allowed range and repeat this for each of FIFO, MQ and SHM
  - Show the results using graphs. Just having tables is not enough to earn full points
  - Use FIFO as the baseline. So, mainly you will have 2 graphs: MQvsFIFO and SHMvsFIFO
  - There should be adequate number of runs to make a as-smooth-as possible trend line

Report

- Answering Investigation Questions [10 pts]
  - Are there any difference in runtime? Can you explain the reason?
  - How many request channels can we have when we are using MQ and SHM? Are these number any different from that in FIFO?
- If you happen to do the bonus part
  - Describe your technique
  - What number of request channels are you now getting?
  - Not reporting these would result in 5 points off of the 15 bonus being deducted