1. **Do we implement POSIX or System V IPC mechanisms for this PA? The lecture slides went over System V and the documentation indicates that we should be using POSIX.**

   You must use POSIX IPC mechanisms.

POSIX vs. System V

POSIX is the current UNIX standard API whereas System V is an older standard API that was the first commercial version of UNIX. For PA4, you are expected to use POSIX utilities to implement all IPC functionality.
The man pages linked in the document (and here: mq_overview(7), shm_overview(7), sem_overview(7)) include an overview of the POSIX utilities and contain links to the related functions for the methods.
If you are unsure if the function you are using is POSIX or System V, check the man pages. If the method is under *man 2* (i.e., appended with (2), like semget(2)), it is a system call, or part of the System V API; if the method is under *man 3* (i.e., appended with (3), like sem_open(3)), it is a C library function, or part of the POSIX API. The only exception is mmap(2)/munmap(2) - allocation/deallocation of a shared memory segment can be done using mmap(2)/munmap(2).
We want you to use POSIX utilities because they are simpler to understand functionally and interface better.

2. **Can we get more clarity on how to implement multiple data point and file transfers when the "-c" flag is specified?**

Data Channel Requests

Multiple Data Points

For requesting multiple data points, you request the same information through each channel. If you have 3 data channels and you request 1000 data points, you would collect 1000 data points for each channel and write them to a file (**you must write each channel to one file, like x1.csv for the first channel, x2.csv for the second channel, etc.**)

File Transfers

For file transfers, each data channel must collect roughly *s*/*c* bytes, where *s* is the size of the file and *c* is the number of data channels. Your last channel can't just be responsible for the last chunk of remaining bytes.
You can either iterate over each channel, sending a file message requesting your buffer capacity, up until the last file message, where you would request remaining bytes. Or you can determine the number of bytes needed per channel, then iterate over each file message per channel where the last message sent over the channel will request the remaining bytes of its segment.

3. **The instructions say the file should be evenly split across the channels, but with the given buffer size. If the buffer size is < size / num_channels then wouldn't this cause the read to be incomplete because bytes would get cut off in each message? So do we just have to guess a large enough buffer size for this to work?**
   That's a corner case. You can either make however many full requests you can, or send partial requests through each channel (the server should support that).

4. **Do we need to change the server.cpp functions to be able to support channels other than FIFO? Do we need to rewrite the functions?**

   You will have to change the server to work with all types of channels. Notice that functions in the server starter code takes `FIFORequestChannel*` as an argument, and `process_newchannel_request(...)` in the server only creates a `FIFORequestChannel*` upon a `NEWCHANNEL_MSG` from the client. You must also modify the server to process the *"-i"* command line argument.

5. **Do you have to specify the attributes when connecting to an existing message queue or only upon the original creation?**
   You must specify the same attributes on all calls to mq_open. You specify attributes when calling mq_open by defining your own mq_attr struct object. Do not rely on system defaults.

6. **Is the NULL option for mq_open okay to use? Or should we create our own attributes? NULL uses the system's default so I'm wondering if this is okay and if not where can we find info on what to set the attributes to.**
   You are recommended to create the struct. First off, you can get away with one message, so you don't need 10 (the default), and also, your message queue should only store as many bytes as it needs to (which will be your buffer capacity from client - passed along as a parameter in the constructor).

7. **How do I get priority for the message queues from the MQRequestChannel read and write?**
   We don't care about priority in this particular PA. In mq_send, put that parameter as 0, and in mq_receive, put that parameter as NULL.

8. **I'm working on the message queue and whenever I try to get a single data point, I end up getting a junk value and then execution hangs. I'm passing in attributes when I create the queue and it shows it being created successfully so I'm having a hard time debugging the issue.**
   For mq_send, the length of the buffer you're writing must be less than or equal to the queue's mq_msgsize attribute. The main thing to note is for mq_receive - you can't request less than mq_msgsize bytes from it (even when the number of bytes written in the message is less than mq_msgsize bytes). You need to choose a sufficiently large number for the *msg_len* parameter in mq_receive that will never be smaller than buffer capacity, *"m"*. 8 kB is what is recommended.

9. **Is anyone familiar with what a bus error is? I'm getting a bus error when running anything related to shared memory.**
   This is caused by not including ftruncate to populate the shared memory segment in the file system.