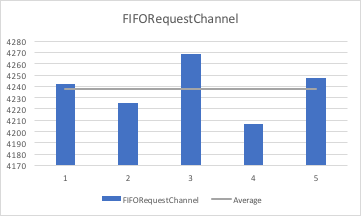
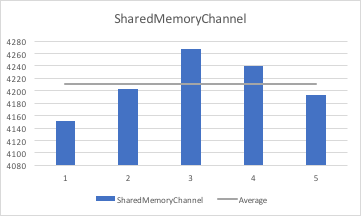
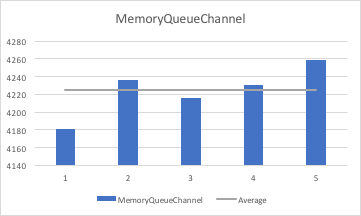
Gather timing data on the same set of n, b, w arguments on each of PA6 FIFO, PA6 MQ,

and PA6 SHM.







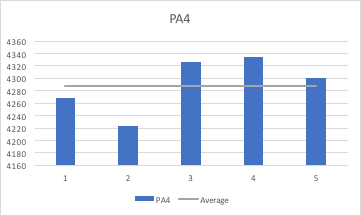
Present a performance comparison of the different IPC mechanisms based on this data,

and attempt to provide explanation for any differences and similarities.

* It Seems like there isn’t much time difference in running time of each type of request channel. However, as it shown above, there is tiny difference in average time. The shared memory channel took least amount of time on it’s performance, and FIFO channel took more time on its performance relatively. However, they have almost same time range of the performance.

Present the results in separate graphs using PA6 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. Now that we are

not using file descriptors for MQ or SHM, has that situation changed?)

* The maximum w for FIFORequestChannels in MAC OS is 993 when I ulimit -n 2000, but the default maximum number of w is 126.
* The maximum w for MemoryQueueChannel in Mac is 39. However, when I run this in Virtual Box which has Ubuntu as OS, It was able to run with over 2000 number of w.

The maximum w for SharedMemoryChannel in Mac is 3. However, when I run the same code in the Ubuntu, I was able to run it more than 500 number of w.

For mac, except for the FIRORequestChannel, MemoryQueueChannel and ShareMemoryChannel has much lesser capacity of w compared to PA4.

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?

* FIFORequestChannels

It’s not concurrency safe within a process, which mean it could cause the race condition. If race condition happened, the program will stop operating.

* MemoryQueueChannel

Once it reachs the limit, the program will stop operating. Moreover, since the number of limit could be varied based on the operating system. It would not be safe to used across the OS.

* ShareMemoryChannel

Since there is no guarantee of order between sending and receiving processes, it could be messy when it has too many request channels. Moreover, since this uses the kernel semaphores, the operation of the program also depends on the capacity of the semaphores.

Describe the clean-up activity you have done for each IPC

* FIFORequestChannels
* Removed the pipes that we used using remove funtions
* MemoryQueueChannel

msgctl(msgid, IPC\_RMID, NULL) in the deconstructor msgid has the id of message queue id.

* ShareMemoryChannel

Used shmdt(data);

shmctl(shmid, IPC\_RMID, 0); in deconstructor of SHMBoundedBuffer.