**PA5 Report**

**Performance comparison of FIFO and MQ**

Note: the following time data is collected on the same set of data:

*n = 10000;*

*b = 20;*

*w = 50;*

*p = 10;*

whereas the capacity of message queue *m* varies from 128 to 4096.

Below is a graph that contains two lines, one for FIFO runtime and another for MQ runtime. Putting these two lines in a single graph gives me more intuition on the runtime difference.

The first thing I noticed from the above graph is the runtime for FIFO is about 40% longer than MQ. My explanation is as follows: FIFO is the IPC runs synchronously, where the OS constructs unidirectional pipes between processes and reading from the pipes in the parent process (aka client) may cause itself waiting, which is a way of synchronization managed by the OS. What’s more, new messages will not be sent through until the existing messages have been consumed, that causes even more wait time. So, these waiting time may contribute to the long execution time.

On the other hand, MQ is the IPC runs asynchronously, where the OS constructs message queues. Even though the processes who read will still wait on the message if the buffer is empty, the processes that writes in the buffer will not wait for the message to be consumed to continue writing. And the receiver may have tried several times until read succeeds. So, this asynchronous behavior plays its part on reducing the runtime.

Comparing their similarities, I found that changes of buffer sizes will not necessarily change the runtime. That’s probably because the buffer size is already large enough to contain any datapoint or new channel requests. But the runtime will be influenced significantly if the program is tested with smaller varying buffer sizes. And of course, if doing file requests, different buffer sizes will definitely have relationships with runtimes (this is already tested out in previous PA discussions).

**Maximum parameters for FIFO and MQ**

FIFO

The maximum number of worker channels is limited by how many file descriptors an environment can have and in particular, I’m running *Ubuntu* via *VirtualBox*, below is what I collected about the file descriptor limitation by running *ulimit -a* in the terminal.

A close up of text on a black background

Description automatically generated

The circled part indicates I can have 1024 file descriptors opening at the same time. Since each worker channel requires 2 fds, one for sending and one for receiving, plus the control channels and patient channels, I would conclude that *-w 500* is about the limit.

If we go beyond that, the program will dump with error message “*too many open files”.*

MQ

The maximum number of -w when using MQ is limited by the number of bytes of *POSIX message queues* in the system. Below is how I query this limit.

A screenshot of a cell phone screen with text

Description automatically generated

The circled part indicates this environment can have no more than 819200 bytes taken by message queues. Another observation is that the # of -w is not solely depended on the system limit, it also depends on the message buffer size. Because each message queue will take up *buffersize* bytes of memory, having *w* worker threads results in creating *2\*w\*buffersize* bytes of memory. In this case, with larger -w and larger buffersize, the limit will quickly be taken up.

However, I found a way to increase the system limits using the ulimit command. But the constrain is that one has to be in root mode and the command can only change the limit in that particular context, once you log out of root or switch to another directory, the limit reverts back to system numbers.

Eventually, our PT, Robert, helped me out by finding a command from stackoverflow that allows me to run the root command and navigate to my working directory at the same time, thus increasing the system limit for my working directory context.

The command is as follows, one can use it to change either max open files or message queue sizes.

*sudo sh -c "ulimit -n 65535 && exec su $LOGNAME"*

With that, I can run the program with higher flag inputs and be able to run the demo instructions.

**Cleaning up for IPC**

In both IPCs, I close the 2 file descriptors by calling the system function close(fd);

For FIFO, call the system function remove(PIPE\_NAME);

A picture containing table

Description automatically generated

In MQ, call the function mq\_unlink(Q\_NAME);

A picture containing photo, table, black, white

Description automatically generated

Jiarui Li

2020/04/22