CSCE 313.506-F18: Programming Assignment 6

Tristan Seifert

Due: Thursday, Nov. 29, 2018

1 Implementation Details

All channels descend from a common `RequestChannel` class that integrates facilities to clean up temporary files (used for FIFOs and `ftok()` calls for shared memory and message queues) and create/delete temporary files.

Temporary on-disk files are deleted automatically when the `RequestChannel` class is deallocated.

1.1 FIFOs

FIFOs are the same implementation as in the previous PAs. The maximum number of channels (w) that can be used here is approximately 2000, which corresponds with the file descriptor limit of 4096 on macOS.

FIFOs are destroyed (as well as their temporary files) when the `FIFORequestChannel` class is deleted. The performance is identical to previous programming assignment.

1.2 Message Queues

Message queues support a maximum of around 20 channels (w), due to a limit imposed on macOS. Compared to FIFOs, message queues are slightly faster by about 15%.

When the maximum amount of message queues has been reached, the program will terminate with an exception. This is because there is no clean way to otherwise handle running out of message queues: the worker threads have already been started, and the data server will have already terminated if this limit is exceeded. (It is probably possible to change this limit, but that isn't something I looked into for this assignment.)

The kernel resources for the message queue are released when the `MQRequestChannel` class is deallocated. This can cause problems if the client releases the channel before the server does (dropped messages), but to circumvent this issue, channels are released at the end of the program to give the server time to respond to any remaining messages.

Message queues are the simplest, but most efficient method of exchanging messages between processes. There are no concerns regarding synchronization like there are with shared memory, but they are still rather performant.

1.3 Shared Memory

Shared memory can support a maximum of around 20 channels (w), again due to a limit imposed on macOS. Compared to message queues, shared memory has about a 5% performance advantage, since there is no need to make syscalls to send/receive messages: there is only the cost of acquiring the semaphore for the segment, as well as additional costs associated with using atomic operations for updating the state.

These atomic operations are provided by the C++ `atomic` header: they translate to read/modify/write cycles with a `LOCK` prefix on x86_64, which makes them useable in a multiprocessor environment, and ensures consistency between threads and processes, because atomic operations bypass any CPU caches, and invoke the architecture-specific cache synchronization protocols.

Overall, shared memory is the fastest of all mechanisms for exchanging data between processes, but it is also the most difficult: variables need to be atomically modified, and care must be taken to ensure cache coherency between all cores/processors on the machine.

2 Performance Data

This data was gathered for varying values of w while $n = 10000$. The system used runs macOS 10.14.2 beta (build `18C38b`), with 64GB of RAM and dual Xeon E5-2670 v2 (10 physical cores, 20 virtual cores) processors at 2.6 GHz.

It is quite likely that message queues and shared memory would outperform FIFOs at higher numbers of w than 20, but this wasn't tested because that would require changing system-wide quotas for the maximum number of message queues and shared memory segments per process, and this is a non-trivial operation on macOS.

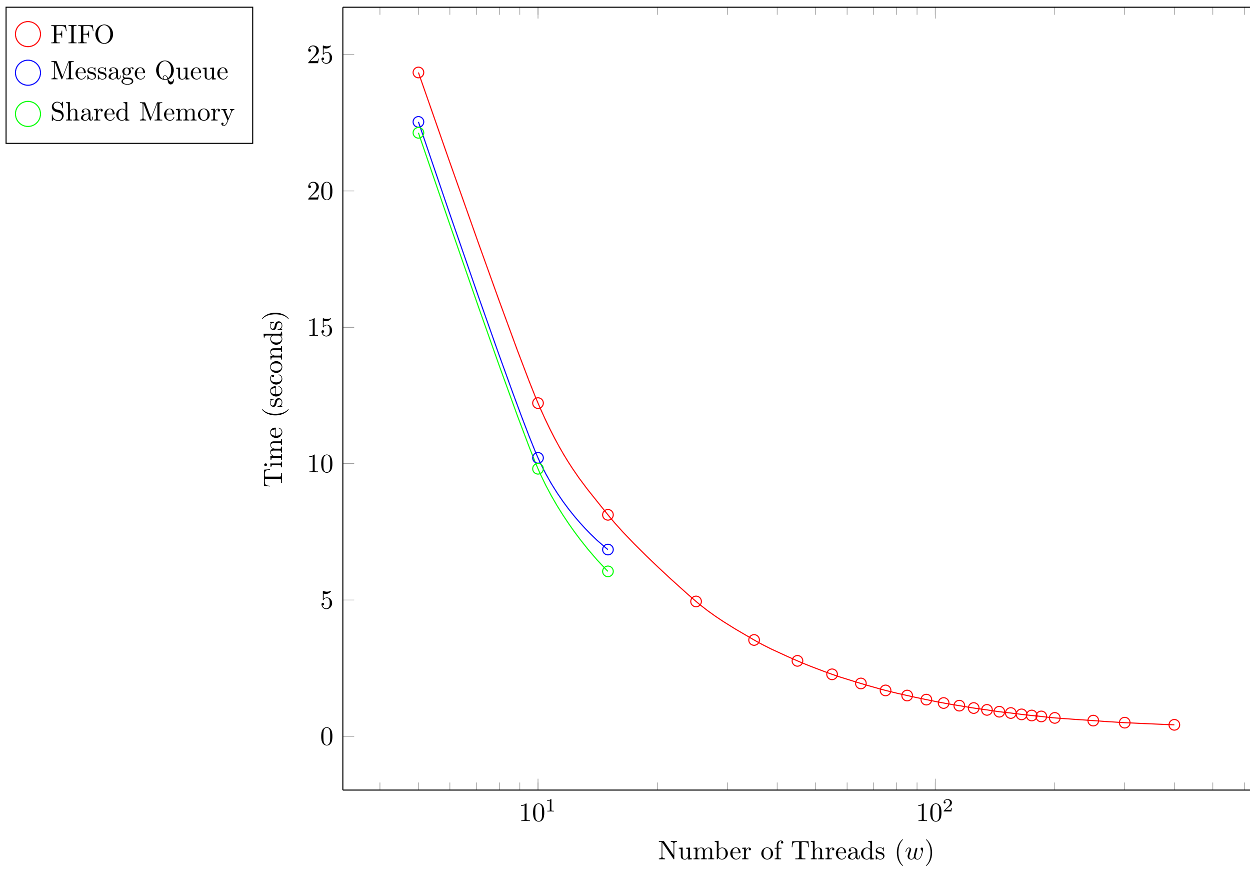

Figure 1: Runtimes for $n = 10000$, $b = 1000$, (truncated to $w = 400$)