Patrick Welch --- CSCE 311 --- IPC Methods

There are multiple methods for processes to communicate with other processes. Among these methods are pipes, named pipes (FIFOs), message queues, Unix Domain Sockets (UDS), and shared memory.

Pipes

Pipes are an IPC technique for passing information/data from one process to another, very much like a stream. Pipes only allow data to be passed in one direction, however, from parent to child. Pipes are useful as they are simple to use and synchronization is built into the technique.

FIFOs

FIFOs are pipes that have been given names. FIFOs can be accessed by virtually any process using the open() system call, whereas regular pipes can only be used by the process they're created in. FIFOs are useful when processes that need to communicate don't have a parent/child relationship, as that is a necessary condition when using an unnamed pipe.

Message Queues

Message queues are essentially queues/linked lists in the kernel. Differ from pipes in this way as pipes provide a stream of data. Message queues also differ from pipes in that they are bidirectional, data can be fetched in any order, messages can be read later, and they remain active until deleted. Message queues should be used when you need bidirectional communication but not enough processes are involved to make the shared memory technique worth using.

UDS

The UDS has the benefits of both the pipe and the message queue techniques. It can support streams, datagrams, and it is also bidirectional. However, with UDS you cannot have multiple readers or multiple writers. The data is strictly shared from one socket to its bounded socket. It is also resource-heavy as every message gets passed through the OS (requires mode switching).

Shared memory

Shared memory is the most efficient IPC technique. Allows multiple processes the share memory, therefore meaning that processes can communicate without having to transfer data. Also doesn't require any system calls. Best when there are many processes that need to communicate. Shared memory loses its efficiency however when there aren't many processes and the amount of memory is large. This is because it will require increasingly more semaphores which are resource-heavy.