Shared Memory

In computer programming, shared memory is a method by which program processes can exchange data more quickly than by reading and writing using the regular operating system services. For example, a client process may have data to pass to a server process that the server process is to modify and return to the client. Ordinarily, this would require the client writing to an output file (using the buffers of the operating system) and the server then reading that file as input from the buffers to its own work space. Using a designated area of shared memory, the data can be made directly accessible to both processes without having to use the system services. To put the data in shared memory, the client gets access to shared memory after checking a semaphore value, writes the data, and then resets the semaphore to signal to the server (which periodically checks shared memory for possible input) that data is waiting. In turn, the server process writes data back to the shared memory area, using the semaphore to indicate that data is ready to be read.

Other forms of interprocess communication (IPC) include message queueing, semaphores, and sockets.

Whereas two processes using sockets to communicate may live on different machines (and, in fact, be separated by an Internet connection spanning half the globe), shared memory requires producers and consumers to be co-resident on the same hardware. But, if your communicating processes can get access to the same physical memory, shared memory will be the fastest way to pass information between them.

Shared memory may be disguised under different APIs, but on modern Unixes the implementation normally depends on the use of mmap(2) to map files into memory that can be shared between processes. POSIX defines a shm_open(3) facility with an API that supports using files as shared memory; this is mostly a hint to the operating system that it need not flush the pseudofile data to disk.

Because access to shared memory is not automatically serialized by a discipline resembling read and write calls, programs doing the sharing must handle contention and deadlock issues themselves, typically by using semaphore variables located in the shared segment. The issues here resemble those in multithreading (see the end of this chapter for discussion) but are more manageable because default is not to share memory. Thus, problems are better contained.