More on Operating Systems

by David G. Messerschmitt

Supplementary section for Understanding Networked Applications: A First Course, Morgan Kaufmann, 1999.

Processes and Objects

Typically a network application is structured as interacting objects, and those objects are partitioned not only among hosts, but also among processes within the same host. Each process is an independent center of activity that interacts with other processes on the same host or other hosts. Each process thus consists of exactly one active object and a collection of passive objects (this statement will be amplified shortly). Two processes can support two active objects, whether on the same or different hosts. This allows concurrent objects, with a couple motivations:

- *Intrinsic application requirements*. Objects executing on two different hosts or in two different processes within a host can be concurrent—the latter through time slicing. This allows objects to satisfy concurrent requests intrinsic to application requirements, such as accommodating multiple users.
- *Performance*. Concurrent active objects on different hosts can increase task throughput through parallelism and pipelining. Concurrent active objects in the same host can improve performance by mitigating the effects of blocking on external requests, since other active objects are not blocked.

Processes and concurrent active objects in the same host don't directly reduce the time to completion or throughput of repetitive tasks; in fact, the context switches between processes consumes significant processing cycles and actually reduces the total processing available to applications. Rather, processes enable applications to accommodate the need for concurrent tasks as demanded by intrinsic application requirements, or by blocking in external interactions.

Example: A server host may serve many clients concurrently by dedicating a separate process to each client. Clients can then make demands on the server at the same time or almost the same time, and the complexities of dealing with the resulting concurrent tasks are handled transparently to the application developer by the operating system. A process is also valuable when one server interacts with another because blocking may result in considerable inactive time. Other processes on the same host can likely continue useful work while one process is blocked.

Inter-Process Communication

Processes need to communicate if applications are to be partitioned among processes. In fact, in order for objects within different process to collaborate, their respective processes must communicate. Process-to-process and object-to-object communication have similar options, such as those discussed in Chapter 12. Processes can send messages to one another (including sessions), or one process can do a *remote procedure call (RPC)* on another process. One function of the OS is to manage the details of process-to-process communication, such as the queueing of messages or creating the shared context of a session.

Internet transport protocols like TCP and UDP support communication between processes on different hosts. They provide a way to address a message to a process within a host. The specific addressing mechanism is the *port number*, described in Chapter 19.

Just as with local and distributed objects, an important design goal for processes is *location transparency.* Process-to-process communications should be transparent to whether they are local or remote. This way, when an application is divided up into processes, it won't matter later if the mapping of processes to hosts is changed. This is an important tool for scalability—processes that share a host can later be assigned to different hosts to improve performance. Some operating systems allow processes to communicate using shared memory—one process puts data in the memory and the other reads it—which violates location transparency but contributes to performance.

Review

Object interaction across two or more processes requires process-to-process communication. Abstract communication services among processes are transparent to whether the processes are on the same or different hosts. The remote procedure call (RPC) is the process-to-process equivalent to the remote method invocation (RMI) for interacting objects, but processes can also communicate by messages or a session. One approach to scalability is to split processes on a single host to multiple hosts.