**Operating System Concepts**

**Process Synchronization**

• A cooperating process is one that can affect or be affected by the other processes executing in the system.

•Cooperating processes may either directly share a logical address space (that is, both code and data), or be allowed to share data only through files. The former case is achieved through the use of *lightweight processes or threads. Concurrent access to shared data* may result in data inconsistency.

• In this lecture, we discuss various mechanisms to ensure the orderly execution of cooperating processes that share a logical address space, so that data consistency is maintained.

**Cooperating Processes**

• The concurrent processes executing in the operating system may be either independent processes or cooperating processes.

• A process is *independent* if it cannot affect or be affected by the other processes executing in the system.

• On the other hand, a process is *cooperating* if it can affect or be affected by the other processes executing in the system.

• There are several reasons for providing an environment that allows process cooperation:



**Race condition**

• When two or more processes are executing concurrently, the result can depend on how the two instruction streams are interleaved. (i.e., the final result of both cooperating processes depends on who runs when)

* Correct results a matter of luck
* They occur rarely …. But they do occur
* Debugging race conditions is not fun

– 1st Difficulty: Finding the Condition

– 2nd Difficulty: Reproducing the Result

**The Critical-Section Problem**



• The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section.

• Thus, the execution of critical sections by the processes is mutually exclusive in time.

• The critical-section problem is to design a protocol that the processes can use to cooperate.

• Each process must request permission to enter its critical section.

• A solution to the critical-section problem must satisfy the following three requirements:

1. **Mutual Exclusion:** If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress:** If no process is executing in its critical section, and there are some processes that want to enter into their own critical sections, then the decision of whom to enter must not be postponed indefinitely.
3. **Bounded Waiting:** There exist a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

**Inter Process Communication**

*(IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system.*

*This allows a program to handle many user requests at the same time.*

*Since even a single user request may result in multiple processes running in the operating system on the user's behalf, the processes need to communicate with each other.*

*The IPC interfaces make this possible. Each IPC method has its own advantages and limitations so it is not unusual for a single program to use all of the IPC methods.*

**Shared Memory:**

*Shared Memory* is an efficient means of passing data between programs. One program will create a memory portion which other processes (if permitted) can access.

A process creates a shared memory segment using shmget()|.

The original owner of a shared memory segment can assign ownership to another user with shmctl().

It can also revoke this assignment.

Other processes with proper permission can perform various control functions on the shared memory segment using shmctl().

 

shared memory is a method by which program [process](http://whatis.techtarget.com/definition/process)es can exchange data more quickly than by reading and writing using the regular operating system services.

For example, a [client](http://searchenterprisedesktop.techtarget.com/definition/client) process may have data to pass to a [server](http://whatis.techtarget.com/definition/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 [buffer](http://searchcio-midmarket.techtarget.com/definition/buffer)s 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](http://searchenterpriselinux.techtarget.com/definition/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.

**Message Passing:**

 

Communication takes place by exchange of messages.

If P & Q wish to communicate, they need to:

– Establish communication link between them

– Communication link can be uni/bi directional, and associated with a single pair of communicating processes

– Exchange messages via send (message), receive (message)

*Direct Communication*

– Explicitly name the recipient or sender of the communication

send(P, message);

receive(Q, message);

*Indirect communication*

– send(A, message), and receive(A, message) to mailboxes

– Mailboxes have unique ID, processes can communicate if they share the mailbox, can be owned by OS or process itself, and are lost if the owner process dies (not if owned by OS)