

Class Notes Sept 6 2006.

### **Synchronization hardware.**

Synchronization hardware solves the problem of mutual exclusion for multiple CPU's, however it still gives us the annoyance of having a busy wait. The TestAndSet instruction is usually not a privileged instruction, so any process can execute it, but there is a friendlier way to do mutual exclusion in processes. This is to do a system call, and request mutual exclusion from the operating system. If this is done, the waiting process can be put in a wait state, and other processes can run while it is waiting. The resources used by a process are typically longer term than those used by the OS, so it makes sense to use TestAndSet in the OS (it is absolutely necessary sometimes) and some other mechanism at the process level.

Several mechanisms have been created for use at the process level. The most universal is the semaphore. The semaphore is an integer variable that can be accessed only through the functions wait and signal. (It is essentially a class.) Wait and Signal are actually system calls, so they are executed by the operating system. Wait causes the integer to be decremented, Signal causes it to be incremented. The semaphore is never allowed to become negative, so if a Wait operation were to make the semaphore negative, the requesting process is forced to wait.

Each semaphore has a queue of processes that are waiting for it. When Signal is executed, the queue is examined, and if it is non-empty, one process is activated and removed from the queue. Otherwise the semaphore is incremented.

Here are some problems that are pretty easy to solve using semaphores:

1. Protecting critical regions
2. Bounded buffer problem
3. Dining philosophers
4. Readers and Writers