Cushing 208 usage….

- One of the machines was compromised while students were installing the OS
  - Sshd exploit to attack machines in Virginia Tech
  - Precautions:
    - Always use a real root (and other user passwords)
    - Lets use XXXXXXXX for the root password
    - Turn on the security while installing Linux

Outline

- Chapter 7: Process Synchronization
  - Critical sections

- Chapter 8: Deadlocks

  A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set

Process Synchronization

- Cooperating processes (threads) sharing data can experience race condition
  - Outcome depends on the particular order of execution
  - Hard to debug; may never occur during normal runs

Register1 = counter	Register2 = counter
Register1 = Register1 + 1	Register2 = Register2 - 1
counter = Register1	counter = Register2

- The final value of the shared data depends upon which process finishes last.

- To prevent race conditions, concurrent processes must be **synchronized**.

Critical Section

- *n* processes all competing to use some shared data
- Each process has a code segment, called *critical section*, in which the shared data is accessed.
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section
- Must satisfy the following requirements:
  - Mutual Exclusion: Only one process should execute in critical section
  - Progress: Scheduling decisions cannot be postponed indefinitely
  - Bounded Wait: A bound must exist 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.
- Remember that synchronization techniques themselves do not guarantee any particular execution order

Approaches

- Software based
  - flag[i] = true;
  - turn = j
  - while (flag[j] && turn == j);
  - …..
  - flag[i] = false;
  - Bakery algorithm for multi-process solution
- Hardware assistance
  - Disable interrupts while accessing shared variables
    - Works for uniprocessor machines
  - TestAndSet and Swap atomic instruction

Semaphore

- Wait (or P)
  - Decrement semaphore if > 0, else wait
- Signal (or V)
  - Increment semaphore
- Counting semaphore – integer value can range over an unrestricted domain
- Binary semaphore – integer value can range only between 0 and 1; can be simpler to implement
  - Also known as mutex locks
- Can implement a counting semaphore S as a binary semaphore
- Semaphores provide mutual exclusion
- Spinlocks - CPU actively waits wasting CPU resources. One optimization is to schedule the process to sleep and have the Signal wake the process. Higher overhead

Deadlocks and Starvation

- Starvation – indefinite blocking. A process may never be removed from the semaphore queue in which it is suspended
  - "Fairness" issue
- Deadlock – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes

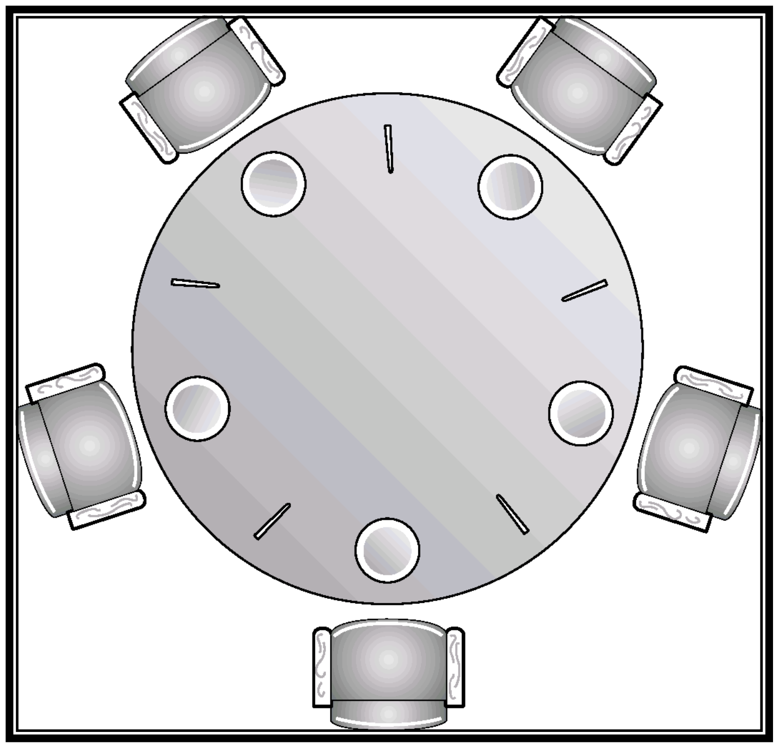
Classical synchronization problems

- Bounded buffer problem
  - Producer, consumer problem
  - Can solve using semaphores
  - E.g. buffer for disk operation in file systems

- Reader-Writers problem
  - Many reader, single writer

Dining Philosopher problem

- Each process thinks for random intervals, picks up both forks and eats for random interval. Cannot eat with one fork



Monitors

- Higher level language construct
- Implicitly locks an entire function

- Java *synchronized* and *notify* mechanisms

Database terminology

- Atomic transaction
  - A sequence of operation either "all" happen or none at all
  - Either "committed" or "aborted"
  - If aborted, transaction is rolled back
  - Log based recovery where each operation is logged. On failure, the log is played back in reverse
    - Redo log
    - Undo log
  - Shared or exclusive
  - Growing and shrinking phase
- Serializable atomic transactions
  - More later

Deadlocks

- Conditions for deadlock:
  1. **Mutual exclusion:** only one process at a time can use a resource.
  2. **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
  3. **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.
  4. **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by
  5. *$P_2, \ldots, P_{n-1}$* is waiting for a resource that is held by P_n, and P_0 is waiting for a resource that is held by P_0.
- Deadlock avoidance protocols
  - Ensure that the above condition cannot happen simultaneously
  - Detection and recovery
  - Laissez-faire - typical OS's assume deadlocks are rare, and detection and avoidance expensive

Deadlock prevention

- Mutual Exclusion
  - Some resources are not mutual - read sharing
- Hold and Wait
  - Whenever a process requests new resource, it does not hold other resources
    - All resources are requested a-priori
- No preemption
- Circular Wait
  - impose a total ordering of all resource types; always request resources in increasing order
- Bankers algorithm: Don’t give out resources unless you can satisfy all outstanding requests
- Avoiding deadlocks can lead to low utilization

Recovery

- Terminate process
  - Abort all deadlocked processes
  - Abort one at a time till cycle is eliminated
- Selecting the victim: Number of resources held by the process
- Rollback transactions: return to some safe state, restart process for that state.
- Starvation: same process may always be picked as victim, include number of rollback in cost factor.