CPSC 457
Operating Systems

Lecture 10

Concurrency, Classical Problems

Monitors

Deadlocks

Last Time

Virtual Memory

- Page Replacement Algorithms
- Thrashing
- Kernel Memory Management
- Memory Management Considerations
- Real World Memory Management

Concurrency

- Race Conditions (and other concurrent problems)
- Critical Sections
- Atomic Operations
- Locks
  - Mutexs
  - Semaphores

This Time

Concurrency

- Some notes about Semaphores
- Spinlocks
- Classical Synchronization Problems
- Monitors
    - Monitor Construction
    - Condition Variables
    - Using Monitors
    - Java

Deadlocks

- Deadlocks
    - Resource Allocation
    - Detecting Deadlocks
    - Resolving Deadlocks

More about Semaphores

Mutexes vs Semaphores

Sequenctalization

Deadlocks and Starvation

Spinlocks

Classical Problems of Synchronization

Producer-Consumer / Bounded-Buffer

Readers-Writers Problem

Dining Philosophers

Bounded Buffer Problem

Readers-Writers Problem

Dining Philosophers Problem

Monitors

Language-Level Construct

- Code must be inserted / managed by the Language
- Split the protection of critical sections from the scheduling of access

Condition Variables

Control Access to the Monitor Lock

- `wait()` – give up access to the monitor lock and wait for someone to signal
- `signal()` – wake up a process waiting for the condition
- `broadcast()` – wake all processes waiting for the condition

Java

Every object in Java is a monitor and it’s own condition variable

- Can declare methods or code blocks synchronized
  - Code blocks allow finer grain
- Only one thread will be allowed to run in each synchronized block/method
- Use the wait and notify/notifyAll methods to signal waiting and signaling.

Deadlocks

Two or more processes are unable to proceed because each is waiting for another process (which is waiting) to proceed.

System Model

Processes

Resources

Requests

- Request
- Use
- Release

Deadlock Characterization

1. Mutual Exclusion - resources cannot be shared, a second request must be delayed
2. Hold and wait - A process must be holding a resource and waiting for another resource
3. No preemption - resources cannot be taken from a process once it has them
4. Circular waiting - For a set of processes $\{P_1, ..., P_n\}$ then P_1 must wait for P_2, which must wait for P_3 … which must wait for P_n, which must wait for P_1

Resource-Allocation Graphs

Handling Deadlocks

Preventing Deadlocks

Prevent one of the four characterizations from holding.

Avoiding Deadlocks

Decide (at runtime) which situations are safe and which will lead to Deadlocks.

Detecting Deadlocks

Look at our processes and resources and determine if they are deadlocked.

Recovering From Deadlocks

Look at our processes and resources restore them to a point where the system is free of deadlocks.

Next Time

Devices

- How the OS addresses different physical/virtual devices

File Systems

- How do we organize data on secondary storage to make sense and access it quickly