Outline

- Chapter 7: Process Synchronization
- Chapter 8: Deadlocks
- Eraser by Savage et al.
- Project milestone 1: Tuesday

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

- Depending on the order, the final value can be off by one

Critical Section

- Must satisfy the following requirements:
  - Mutual Exclusion: Only one process should execute in critical section
  - Progress:
  - Bounded Wait
- 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
- Spin lock or reschedule processes

Semaphore

- Wait (or P)
  - Decrement semaphore if > 0, else wait
- Signal (or V)
  - Increment semaphore
- 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

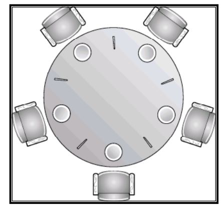
- Starvation or indefinite blocking
  - "Fairness" issue
- Indefinite wait - deadlock

Classical synchronization problems

- Bounded buffer problem
  - Producer, consumer problem
  - Can solve using semaphores
  - E.g. buffer for disk operation in your file system
- Reader-Writers problem
  - Many reader, single writer
  - E.g. your file system home work

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

Database terminology

- Atomic transaction
  - A sequence of operation either “all” happer 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

- Mutual Exclusion
- Hold and wait
- No preemption
- Circular wait
- 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 requester a priori
- No preemption
- Circular Wait
  - 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:
- Starvation:

Eraser

- Tool to dynamically detect possibility of race conditions in lock based multithreaded programs
- Key Idea: For all shared variables, for all locks in the system; check to make sure that each shared variable is covered by the appropriate number of locks
  - We don't know what variables are associated with what locks
  - Some race conditions are benign
    - Initialization
    - Read shared
    - Read-write locks
- Binary code rewrite to insert hooks
- Significant overhead: 20 to 30 times slower. Since timing is critical to threads programs, this could be an issue