Week 2 (Jan 18 - 22)

Deadlocks

Resources

- Usually limited
- Examples
  - Printers
  - Semiphores
  - Memory
  - Tables
- Processes need access to resources
- Two types of resources
  - Pre emptive: Can be taken away with no ill effects
  - Non- preemptable: Will cause failure if taken away
- Sequence of events to use resource
  1. Request
  2. Use
  3. Release
- Can't use if request is denied
  - Options:
    - Block and Wait
    - Continue (if possible) without resource
    - Process fails with error code
  - Some of these options can prevent deadlock

Deadlocks

- When do deadlocks happen?
  - Processes are granted exclusive access to resources
  - Each deadlocked process needs a resource held by another deadlocked process
- What is deadlock?
  - A set of processes is deadlocked if each process in the set is waiting for an event that only another process can make happen
  - Usually the event is a release of a resource
  - In deadlock, none of the processes can:
    - Run
    - Release resources
    - Be awakened
- Four conditions for deadlock (Coffman Conditions)
  1. Mutual Exclusion
     - Each resource assigned to at most one process
  2. Hold and Wait
     - Processes holding resources can request more resources

3. No preemptive
   - Previously granted resources can not be taken away
4. Circular Wait
   - Circular chain of processes each waiting for a resource held by another process in the chain

- These 4 conditions are necessary and sufficient for deadlock

How OS deals with Deadlock

- Ostrich method (Ignore it)
- Detect and Recover
- Dynamically avoid
- Prevent

Ostrich Method (ignore)

- Reasonable if deadlocks rare
  - Cost of prevention is high
- Unix/Windows take this approach
  - Resources plentiful
  - Deadlocks rare
  - Typically handled by rebooting
- Tradeoff: Convenience vs Correctness

Avoid Deadlock

- Find ways to detect and prevent deadlocks
- Stop them from happening in the first place

Deadlock detection using graphcs

- Hold requests in table
- Graph contains cycle => deadlock
- Use DFS to find cycles

Recover from deadlock

- Through Preemption
  - Take resources from another process
  - Depends on nature of resource and process
- Rollback
  - Checkpoint a process periodically
  - Use saved state to restart process in deadlock
  - May present problem if process affects outside world
- Killing process
  - Crude and Simple
  - Kill one process in deadlock
  - Other processes can get its resources
  - Pick process that can most easily be rerun

Prevent completely

- Ensure one of necessary and sufficient conditions not true
  - Mutual Exclusion
  - Circular wait
  - Hold and Wait
  - No Pre-emption
- Not always possible

Attacking Mutual Exclusion

- Some devices (printers) can be spooled
  - Only printer daemon accesses actual printer
  - Spool contents as file
- Principle
  - Avoid assigning resources unless necessary
  - As few processes as possible claim resource

Attacking Hold and Wait

- Require processes to request resources before starting
  - Never wait for what it needs
  - Problems
    - Cannot always know what it needs
    - Takes up resources other processes could be using
  - Variation
    - Give up all resources before making a new request
      - Granted all previous resources as well as all new resources

Attacking No Pre-emption

- Usually not viable
- May work for some resources
  - Forcibly take away memory pages, suspending process
  - Process may resume with no ill effects

Attacking Circular Wait

- Assign an order to resources
- Always acquire resources in numerical order
  - Need to acquire all at once
- Prevents circular wait
  - Processes holding resource n can't wait for m (m $<$ n)
- No way to complete cycle

What does FreeBSD do?

- What resources are of issue?
  - Locks and Semaphores
  - Physical resources usually not a concern
- Goal: Prevent deadlocks
  - Low cost
  - Simple
  - Flexible

Two Phase Locking

- Phase One
  - Process tries to lock all data it needs at one time
  - If needed data is locked, start over
- Phase Two
  - Perform updates
  - Release lock
- Similar to request all at once
- Used in databases
- Avoids hold and wait

Starvation

- Some schedulers may cause jobs to be postponed indefinately
  - e.g. shortest first schedule may cause a long job to never complete even when not blocked
- Starvation can lead to deadlock
  - Starved process may hold resources
  - If not used/released in reasonable time, shortage may cause deadlock

Livelock

- Processes run, but make no progress
  - Example:
    - P0 has A ; P1 has B
    - P0 wants B ; P1 wants A
    - Both realize deadlock
    - P0 drop A ; P1 drop B
    - P0 gets B ; P1 gets A
    - ...
- Real life example
  - Ethernet transmission collisions
    - On collision wait and try again