CS153: Deadlock

Chengyu Song

Slides modified from Harsha Madhyvasta, Nael Abu-Ghazaleh, and Zhiyun Qian
Administrivia

- **Lab**
  - Lab1 is due this Sunday
  - Demo sessions next week
- Little book of semaphores
  - First 4 chapters
- Midterm reviews
  - Next week, bring your questions
Administrivia

• Homework
  • Homework1 is due today

• Hints:
  • Q1: list all possible cases you can come up with
  • Q2: differentiate user-level threads, kernel-level threads, and kernel threads
  • Q3: ++i is NOT atomic
Recap: synchronization

- Race condition can happen between threads with shared data
  - Concurrent
    - At least one write
  - Synchronization is necessary to provide
    - Safety, liveness, performance
- Mechanisms
  - Alternation, locks, semaphores, monitors, C/V, etc
The deadly embrace!

- Synchronization is not trivial
  - Incorrect use of synchronization can block all processes
  - You have likely been intuitively avoiding this situation already
- More generally, processes that allocate multiple resources generate dependencies on those resources
  - Locks, semaphores, monitors, just represent the resources they protect
  - If one process tries to access a resource that a second process holds, and vice-versa, they can never make progress
Deadlock

- Deadlock is a problem that can arise:
  - When processes compete for access to limited resources
  - When processes are incorrectly synchronized
- Definition:
  - **Deadlock** exists among a set of processes if every process is waiting for an event that can be caused only by another process in the set
Conditions for deadlock

• Deadlock can exist if and only if all the following four conditions hold:

  1. **Mutual exclusion**: at least one resource must be non-sharable

  2. **Hold and wait**: there must be one process holding one resource and waiting for another resource

  3. **No preemption**: resources cannot be preempted (critical sections cannot be aborted externally)

  4. **Circular wait**: There must exist a set of processes \([P_1, P_2, P_3, \ldots, P_n]\) such that \(P_1\) is waiting for \(P_2\), \(P_2\) for \(P_3\), etc.
Dining lawyers

- Each lawyer needs two chopsticks to eat, each grabs chopstick on the right first.
Deadlock can be described using a resource allocation graph (RAG)

- A set of vertices $P = \{P_1, P_2, ..., P_n\}$ of processes
- A set of vertices $R = \{R_1, R_2, ..., R_m\}$ of resources
- A directed edge from a process to a resource, $P_i \rightarrow R_j$, means that $P_i$ has requested $R_j$
- A directed edge from a resource to a process, $R_i \rightarrow P_j$, means that $R_j$ has been allocated to $P_i$
- Each resource has a fixed number of units
Conditions for deadlock: RAG

- If a RAG has no cycles, deadlock **cannot exist**
- If a RAG has a cycle, deadlock **may exist**
Waits-for graph

- If all resources are single unit and all processes make single requests, then we can replace the RAG with a simpler waits-for graph (WFG)
  - A set of vertices $P=\{P_1, P_2, \ldots, P_n\}$ of processes
  - A directed edge $P_i \rightarrow P_j$ means that $P_i$ has requested a resource that $P_j$ currently holds
- If the graph has no cycles, deadlock **cannot exist**
- If the graph has a cycle, deadlock **exists**
Dealing with deadlock

- There are four approaches for dealing with deadlock:
  - Ignore it – how lucky do you feel?
  - Prevention – make it impossible for deadlock to happen
  - Avoidance – control allocation of resources
  - Detection and Recovery – look for a cycle in dependencies
Deadlock prevention

• Ensure that at least one of the necessary conditions cannot happen
  • **Mutual exclusion**
    • Make resources sharable (not generally practical)
    • Lock-free data structures
  • **Hold and wait**
    • Process cannot hold one resource when requesting another
    • Process requests, releases all needed resources at once
Deadlock prevention (cont.)

- Ensure that at least one of the necessary conditions cannot happen
  - **Preemption**
    - OS can preempt resource (costly)
    - `trylock` - may lead to *livelock*
  - **Circular wait**
    - Impose an ordering (numbering) on the resources (locks) and request them in order (popular implementation technique)
Deadlock avoidance

• Provide information in advance about what resources will be needed by processes to guarantee that deadlock will not happen
• System only grants resource requests if it knows that the process can obtain all resources it needs in future requests
• Avoids circularities (wait dependencies)
Deadlock avoidance (cont.)

• Tough
  • Hard to determine all resources needed in advance
  • Good theoretical problem, not as practical to use
Banker's algorithm

- The classic approach to deadlock avoidance for resources with multiple units
- Assign a credit limit to each customer (process)
  - Maximum credit claim must be stated in advance
- Reject any request that leads to a dangerous state
  - A dangerous state is one where a sudden request by any customer for the full credit limit could lead to deadlock
  - A recursive reduction procedure recognizes dangerous states
Banker's algorithm

• In practice, the system must keep resource usage well below capacity to maintain a resource surplus
  • Rarely used in practice due to low resource utilization
Banker's algorithm simplified

- Avoidance through scheduling
Detection and recovery

- If we don't have deadlock prevention or avoidance, then deadlock may occur
- We need two algorithms
  - One to determine whether a deadlock has occurred
  - Another to recover from the deadlock
Deadlock detection

- Algorithm
  - Maintain a RAG or WFG
  - Traverse the graph looking for cycles
- Expensive, for both maintaining and traversal
- Only invoke detection algorithm depending on
  - How often or likely deadlock is
  - How many processes are likely to be affected when it occurs
Deadlock recovery

- Abort all deadlocked processes
  - Processes need to start over again
- Abort one process at a time until cycle is eliminated
  - System needs to rerun detection after each abort
- Preempt resources (force their release)
  - Need to select process and resource to preempt
  - Need to rollback process to previous state
  - Need to prevent starvation
Deadlock summary

• Deadlock occurs when processes are waiting on each other and cannot make progress
  • Cycles in Resource Allocation Graph (RAG)
• Deadlock requires four conditions
  • Mutual exclusion, hold and wait, no resource preemption, circular wait
• Four approaches to dealing with deadlock:
  • Ignore it, prevention, avoidance, detection and recovery
Additional concurrency bugs

- Atomicity-violation bugs
  - Time-of-check-to-time-of-use (TOCTTOU)
- Order-violation bugs
- Compiler optimizations
  - The `volatile` keyword
- etc.
For next class ...

• Midterm review 1
• Textbook
  • All chapters that are covered so far