Process management (lower module)

Presented by

C.Namrata Mahender

Synchronization

- Synchronize: to (arrange events to) happen at same time
- Process synchronization
  - When one process has to wait for another
  - Events in processes that occur “at the same time”
- Uses of synchronization
  - Prevent race conditions
  - Wait for resources to become available

Synchronization(cont..)

- Concurrent access to shared data may result in data inconsistency.
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes

The Critical-Section Problem

- *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.

Solution to Critical-Section Problem

1. **Mutual Exclusion**. If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress**. If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.

Solution to Critical-Section Problem(cont.)

3.Bounded Waiting. 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.

- **Assume that each process executes at a nonzero speed**
- **No assumption concerning relative speed of the *n* processes.**

Initial Attempts to Solve Problem

- Only 2 processes, P_0 and P_1
- General structure of process P_i (other process P_j)

```
do {
    entry section
        critical section
    exit section
        reminder section
} while (1);
```

- Processes may share some common variables to synchronize their actions.

Bakery Algorithm

Critical section for n processes

- Before entering its critical section, process receives a number. Holder of the smallest number enters the critical section.
- If processes P_i and P_j receive the same number, if $i < j$, then P_i is served first; else P_j is served first.
- The numbering scheme always generates numbers in increasing order ; i.e., 1,2,3,3,3,3,4,5...

Semaphores

- Generalized synchronization tools (mechanisms)
- Integer variable S is accessed only through two atomic operations *wait* and *signal*
  - Wait(S) : while S <= 0 do no-op;
    S:=S-1;
  - Signal(S) : S := S + 1;
  - S is initialized to 1 and then becomes 0 if it is held by a process in the CS
  - Only one process will be able to modify S at a time, forcing other processes to wait in the while loop
    - this will ensure mutual exclusion

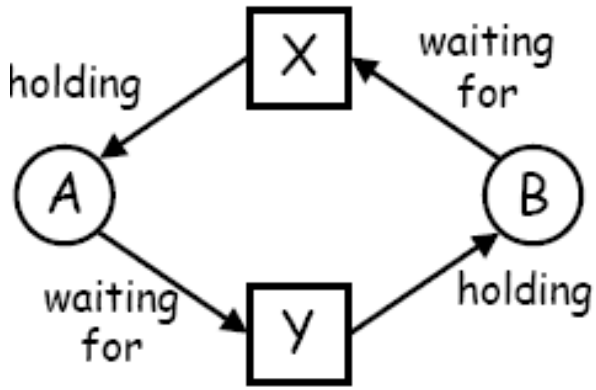
Busy Waiting

- One problem that the semaphore definition causes is that processes waiting for access to the semaphore are forced to loop continuously
  -known as busy waiting
- which means that they tie up the CPU doing nothing
  - This type of semaphore is called a spinlock

- Instead, we can force the process to block itself and while blocked, another process can gain access to the CPU
  - When the process becomes unblocked, we can revert back to it or place it back into the ready queue

What is Deadlock?

- The state of a set of permanently blocked processes
  - Unblocking of one relies on progress of another
  - But none can make progress!
- Example
  - Processes A and B
  - Resources X and Y
  - A holding X, waiting for Y
  - B holding Y, waiting for X
  - Each is waiting for the other; will wait forever



Conditions for Deadlock

- Mutual Exclusion
  - Only one process may use a resource at a time
- Hold-and-Wait
  - Process holds resources while waiting for others
- No Preemption
  - Can’t take a resource away from a process
- Circular Wait
  - The waiting processes form a cycle

How to Attack the Deadlock Problem

- Deadlock Prevention
  - Make deadlock impossible by removing a condition
- Deadlock Avoidance
  - Avoid getting into situations that lead to deadlock
- Deadlock Detection
  - Don’t try to stop deadlocks
  - Rather, if they happen, detect and resolve

Deadlock Prevention

Simply prevent any one of the conditions for deadlock

- Mutual exclusion
  - Relax where sharing is possible
- Hold-and-wait
  - Get all resources simultaneously (wait until all free)
- No preemption
  - Allow resources to be taken away
- Circular wait
  - Order all the resources, force ordered acquisition

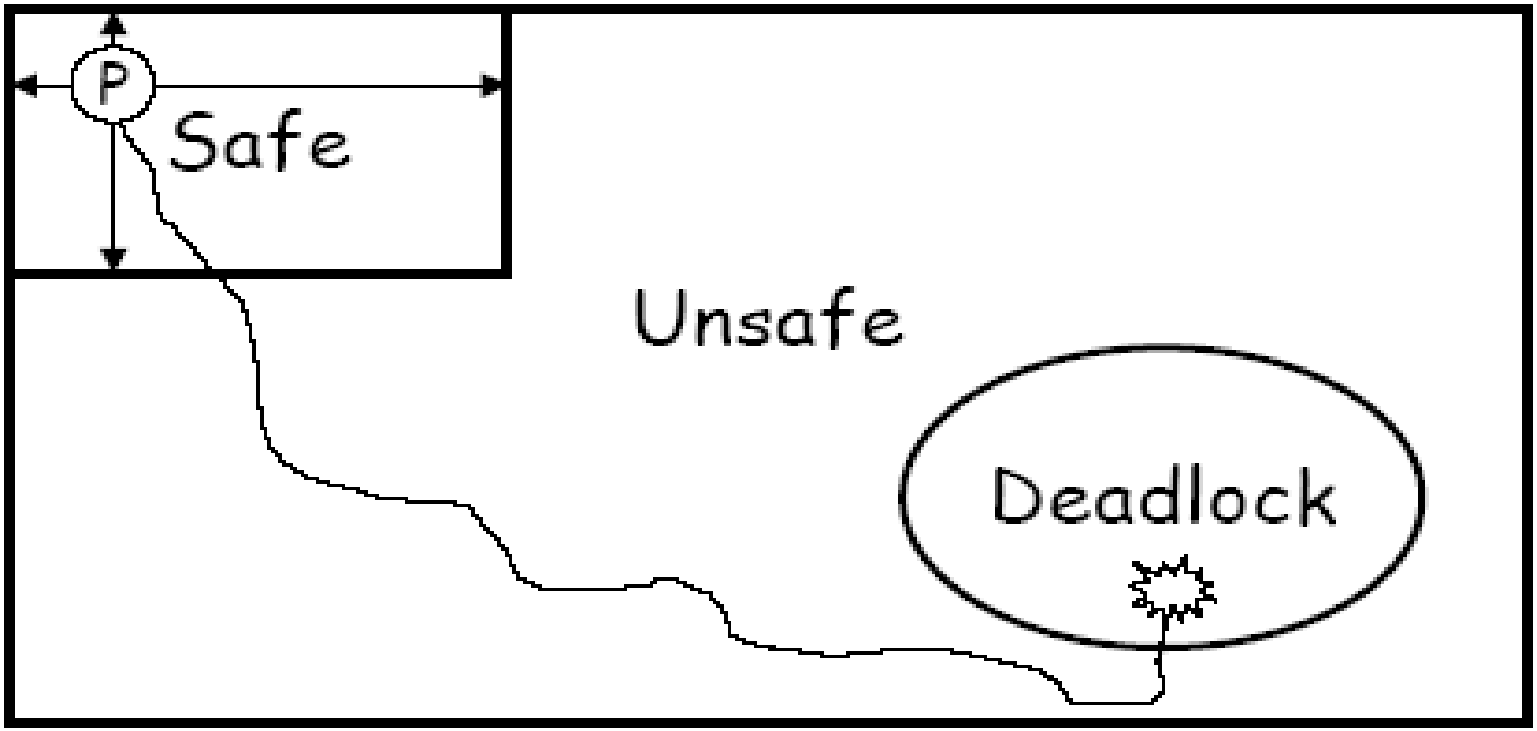
Deadlock Avoidance

- Avoid situations that lead to deadlock
  - Selective prevention
  - Remove condition only when deadlock a possibility
- Works with incremental resource requests
  - Resources are asked for in increments
  - Do not grant request that can lead to a deadlock
- Requires knowledge of maximum resource requirements

Banker's Algorithm: Concepts

- System has a fixed number of processes and resources
  - Each process has zero or more resources allocated
- System state: either safe or unsafe
  - Depends on allocation of resources to processes
- Safe state: deadlock is absolutely avoidable
  - Can avoid deadlock by certain order of execution
- Unsafe state: deadlock is possible (but not certain)
  - May not be able to avoid deadlock

Safe, Unsafe, and Deadlock States

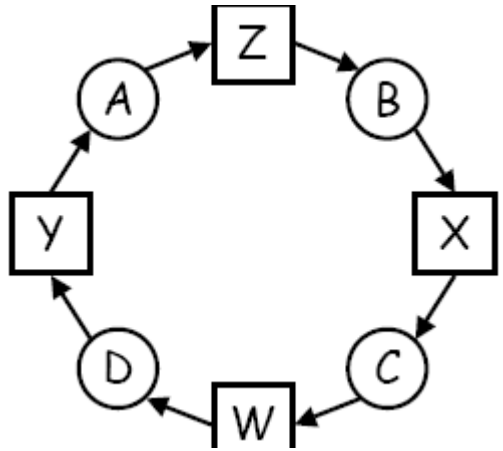


Deadlock Detection and Recovery

- Don’t do anything special to prevent or avoid deadlocks
  - If they happen, let them happen
  - Periodically, try to detect if a deadlock occurred
  - Do something (or even nothing) about it
- Reasoning
  - Deadlocks rarely happen
  - Cost of prevention or avoidance is not worth it
  - Deal with them in special way (may be very costly)

Detecting Deadlocks

- Construct resource allocation "wait-for" graph
  - if cycle, deadlock
- Requires
  - identifying all resources
  - tracking their use
  - periodically running
  - detection algorithm



Recovery from Deadlock

- Abort all deadlocked processes
  - Will remove deadlock, but drastic and costly
- Abort deadlocked processes one-at-at-time
  - Do until deadlock goes away (need to detect)
  - What order should processes be aborted?

Thank you