Lecture Topics

- Today: Deadlock
  (Stallings, chapter 6.1-6.6)
- Next: I/O Management
  (Stallings, chapter 11.1-11.9)

Announcements

- Unclaimed exam booklets available
- Self-Study Exercise #10
- Project #9 (due 11/30)
- Project #10 (due 12/7)
Stallings

- Deadlock: a situation in which two or more processes are unable to proceed because each is waiting for the others to do something.

- Starvation: a situation in which a runnable process is overlooked indefinitely by the scheduler; although it is able to proceed, it is never chosen.

Deadlock

- Deadlock: permanent blocking of a set of processes that either compete for system resources or communicate with each other.

- Deadlock involves conflicting needs for resources by two or more processes.

- No efficient solution for the general case.
Traffic example

- Consider an intersection with stop signs on each corner (a four-way stop).
- General rule: a driver yields to the car on the driver’s right.
- Works well if two or three cars are involved.
- May lead to deadlock if four cars are involved.
Formalization

- Car 1 needs a and b
- Car 2 needs b and c
- Car 3 needs c and d
- Car 4 needs d and a
- Deadlock is possible, but not inevitable
Formalization

- Car 1 has a, needs b
- Car 2 has b, needs c
- Car 3 has c, needs d
- Car 4 has d, needs a
- Given this resource allocation, deadlock exists

Resource Types

- Reusable resource: can be used by only one process at a time; not depleted by use
  - processors
  - primary storage

- Consumable resource: created (produced) and destroyed (consumed)
  - interrupts
  - messages
Reusable Resources

- Examples: processors, I/O channels, primary and secondary storage, files, databases, and semaphores
- Process obtains resources for its own use, later releases those resources for use by other processes
- Simple example: deadlock occurs if each process holds one resource and requests the other

Example: P and Q compete for D and T

<table>
<thead>
<tr>
<th>Process P</th>
<th>Process Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>Action</td>
</tr>
<tr>
<td>p_0</td>
<td>Request (D)</td>
</tr>
<tr>
<td>p_1</td>
<td>Lock (D)</td>
</tr>
<tr>
<td>p_2</td>
<td>Request (T)</td>
</tr>
<tr>
<td>p_3</td>
<td>Lock (T)</td>
</tr>
<tr>
<td>p_4</td>
<td>Perform function</td>
</tr>
<tr>
<td>p_5</td>
<td>Unlock (D)</td>
</tr>
<tr>
<td>p_6</td>
<td>Unlock (T)</td>
</tr>
</tbody>
</table>
Example (continued)

Sequence 1:  \text{P0 P1 P2 P3 Q0 P4 P5 P6 Q1 ...}
(processes complete)

Sequence 2:  \text{P0 P1 Q0 Q1 Q2 P2}
(processes deadlock)

Example: processes compete for RAM

- Space is available for allocation of 200 Kbytes, and the following sequence of events occur:

- Deadlock occurs if both processes progress to their second request
Consumable Resources

- Created (produced) and destroyed (consumed) by a process
- Examples: interrupts, signals, messages, and information in I/O buffers
- Deadlock may occur if "Receive" is blocking
- May take a rare combination of events to cause deadlock – difficult to debug

Example: message passing

Deadlock occurs if "Receive" operation is blocking: each process is waiting for the other process

```
P1
  ...
  Receive (P2);
  ...
  Send (P2, M1);

P2
  ...
  Receive (P1);
  ...
  Send (P1, M2);
```
Resource Allocation Graphs

- Circles: processes
- Squares: resources
- Dots: instances of resources

(a) Resource is requested

(b) Resource is held

(c) Circular wait

(d) No deadlock
Review: traffic example

- Car 1 has a, needs b
- Car 2 has b, needs c
- Car 3 has c, needs d
- Car 4 has d, needs a
- Given this resource allocation, deadlock exists

Graph for traffic example
Four Conditions for Deadlock

Decisions about system (policies):

- Mutual exclusion
- Hold-and-wait
- No preemption

Circumstances (consequence of policies):

- Circular Wait

---

Mutual exclusion
- only one process may use a resource at a time

Hold-and-wait
- A process may hold allocated resources while awaiting assignment of other resources

No preemption
- No resource can be forcibly removed from a process holding it
Four Conditions for Deadlock

- **Circular wait**
  A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain.

Deadlock: Three Approaches

- **Deadlock Prevention**
  - Adopting a policy that eliminates one of the conditions

- **Deadlock Avoidance**
  - Making appropriate dynamic choices based on the current state of resource allocation

- **Deadlock Detection**
  - Detect deadlock and take action to recover
Deadlock Prevention

Design system so that deadlock cannot occur:

- indirect method: prevent one of the three necessary conditions (mutual exclusion, no preemption, or hold and wait)

- direct method: prevent circular wait

Deadlock Prevention

- No Mutual Exclusion
  - Cannot be disallowed

- Hold and Wait
  - Prevent by forcing a process to request all required resources at one time
  - Prevent by blocking the process until all requests can be granted simultaneously
Deadlock Prevention

- **Hold and Wait**
  - Inefficient: process may have long wait for all resources to be available; resources held for long periods without being used
  - Impractical: may not be able to predict which resources will be required.

Deadlock Prevention

- **No Preemption**
  - Prevent by forcing a process to release any current resources whenever it makes a new request which cannot be granted immediately (must re-request resources)
  - Prevent by allowing OS to preempt other processes (take resources away)
  - Useful only when resource state can be saved and restored later.
Deadlock Prevention

- **Circular Wait**
  - Define a linear ordering of resource types
  - If a process has been allocated resources of type \( R \), then it may subsequently request only those resources of types following \( R \) in the ordering
  - May be inefficient, slowing down processes and denying resource access unnecessarily

Deadlock Avoidance

- A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to a deadlock
- Requires knowledge of future process requests
Two Approaches to Deadlock Avoidance

- Process Initiation Denial: do not start a process if its demands might lead to deadlock

- Resource Allocation Denial: do not grant an incremental resource request to a process if this allocation might lead to deadlock

Deadlock Avoidance

Restrictions on its use:

- Maximum resource requirement must be stated in advance

- Processes under consideration must be independent; no synchronization requirements

- There must be a fixed number of resources to allocate

- No process may exit while holding resources
Process Initiation Denial

- Processes required to declare maximum need for each type of resource.
- OS keeps track of those maximum needs and current allocations.
- When a new process enters the system, the OS checks to see if the maximum needs of all current processes, plus the new process, can be met. If not, process not allowed.

Ex: Process Initiation Denial

- Assume a system has 100 Kbytes of RAM
- Current processes:
  - P1 has 30 Kbytes (declared max of 50 Kbytes)
  - P2 has 10 Kbytes (declared max of 40 Kbytes)
- New process enters system:
  - P3 (declares max of 35 Kbytes)
- OS will not allow P3 to enter system, since sum of all declared maximums exceeds total amount of RAM in system (based on worst case scenario)
Resource Allocation Denial

- Strategy referred to as the banker’s algorithm
- State of the system is the current allocation of resources to processes
- A safe state is where there is at least one sequence that does not result in deadlock
- Approach: system only moves from one safe state to another safe state

Resource Allocation Denial (cont)

- Currently in safe state
- When a process makes a request for resources, check to see if the resulting state is still a safe state
- Resulting state is safe? Grant request
- Resulting state is not safe? Deny request, block process
Example #1

Current state:

<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

P2 requests one unit of R1 and one unit of R3

Proposed state:

<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Is the proposed state safe?

Yes – processes can complete: P2, P1, P3, P4
Example #1 (cont)

(a) Initial state

(b) P2 runs to completion

(c) P1 runs to completion
Example #1 (cont)

Current state:
P1 requests one unit of R1 and one unit of R3

Example #2

Current state:
P1 requests one unit of R1 and one unit of R3
Example #2 (cont)

Unsafe state: contents of V cannot satisfy any of the four processes (assuming they all request their remaining resources) – request denied

Not deadlocked, but potential for deadlock exists
## Exercise

Current system state:

<table>
<thead>
<tr>
<th></th>
<th>Claim</th>
<th>Alloc</th>
<th>Need</th>
<th>Avail</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>0 1 0</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2 0 0</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>3 0 2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2 1 1</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>0 0 2</td>
</tr>
</tbody>
</table>

a) What is R (resource vector)? \[ R: 9 \ 5 \ 7 \]
b) Is the current state safe? Yes: P1, P3, P2, P0, P4

## Exercise (cont)

Assume P1 requests (1,0,2)

Proposed system state:

<table>
<thead>
<tr>
<th></th>
<th>Claim</th>
<th>Alloc</th>
<th>Need</th>
<th>Avail</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>0 1 0</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2 0 0</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>3 0 2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2 1 1</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>0 0 2</td>
</tr>
</tbody>
</table>

a) Is the proposed state safe? Yes: P1, P3, P2, P0, P4
Exercise (cont)

Assume P0 requests (1,2,0)

Proposed system state:

<table>
<thead>
<tr>
<th></th>
<th>Claim</th>
<th>Alloc</th>
<th>Need</th>
<th>Avail</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

a) Is the proposed state safe? No

Deadlock Avoidance: limitations

- Maximum resource requirement must be stated in advance
- There must be a fixed number of resources to allocate
- No process may exit while holding resources
- Processes must be independent; no synchronization requirements
Deadlock Detection

- Resource requests are granted to processes whenever possible
- OS periodically checks for deadlock (by identifying circular wait situations)
- OS then recovers from deadlock

Strategies once deadlock detected

- Abort all deadlocked processes
- Back up each deadlocked process to some previously defined checkpoint, and restart all process (original deadlock may re-occur)
- Successively abort deadlocked processes until deadlock no longer exists
- Successively preempt resources until deadlock no longer exists
Selection Criteria Deadlocked Processes

- Least amount of processor time consumed so far
- Least number of lines of output produced so far
- Most estimated time remaining
- Least total resources allocated so far
- Lowest priority

Table 6.1. Summary of Deadlock Detection, Prevention, and Avoidance
Approaches for Operating Systems [ISP 088]

<table>
<thead>
<tr>
<th>Approach</th>
<th>Resource Allocation Policy</th>
<th>Different Scheme</th>
<th>Major Advantages</th>
<th>Major Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prevention</td>
<td>Conservative, non-meetsSuspect</td>
<td>Resolving all situations in order</td>
<td>- Works well for processes that perform a single burst of activity</td>
<td>- No guarantee necessary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Sufficient delay in process initiation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Resource allocation may be less than necessary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Preemptive resource allocation may be necessary</td>
</tr>
<tr>
<td>Avoidance</td>
<td>Mid-level between arbitration and prevention</td>
<td>Maximizing the situation matrix</td>
<td>- Can prevent many resource issues problems solutions</td>
<td>- Deadlocks incremental resource requests</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Resource requirements may not be known at OS startup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Policies subject to web-based intervals</td>
</tr>
<tr>
<td>Detection</td>
<td>Very liberal, require minimum resource use possible</td>
<td>Involved critically to stop for deadlock</td>
<td>- No delay in process initiation</td>
<td>- purse resource handling</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Limited supervision license</td>
</tr>
</tbody>
</table>