Lecture Topics

- Today: Concurrency: Deadlock & Starvation (Stallings, chapter 6.1, 6.6-6.8)
- Next: Exam #1

Announcements

- Self-Study Exercise #5
- Project #4 (due 10/11)
- Be sure to use “cse410.cse.msu.edu” for all of the computer projects
Exam #1

- Thursday, 10/4 during lecture
- 80 minutes, 18% of course grade
- Topics:
  - computer systems overview
  - operating system overview
  - processes and threads
  - concurrency
- Study suggestions on course website

Semaphores

- A semaphore is a special variable that, apart from initialization, can only be accessed via atomic and mutually exclusive operations:
  - wait(S) -- sometimes P(S)
  - signal(S) -- sometimes V(S) or post(S)

- When a process has to wait, it will be put in a queue of processes which are blocked on that same semaphore
Semaphore operations (atomic)

Semaphore implemented as a record with two fields:

- count -- integer
- queue -- list of blocked processes

```c
struct semaphore {
    int count;
    queueType queue;
};
```

```c
void wait( semaphore s ) {
    s.count--;
    if (s.count < 0)
        // put process in s.queue, block it
};
```

```c
void signal( semaphore s ) {
    s.count++;
    if (s.count <= 0)
        // get some process from s.queue
};
```
Semaphores for Critical Sections

Initialize S to 1

repeat
    preceding section
    wait(S)
    critical section
    signal(S)
    following section
forever

The Producer/Consumer Problem

- A producer process produces information that is consumed by a consumer process
  - Example: a program produces characters that are consumed by a printer
- We need a buffer to hold items that are produced and eventually consumed
- A common paradigm for cooperating processes
Bounded Buffer (circular buffer of size $k$)

- Finite number of slots in the buffer (circular)
- Producer(s) and consumer(s) update the buffer (since in and out pointers are modified)
- Can consume **only** when there is at least one item (number $N$ of items is at least one)
- Can produce **only** when there is at least one empty slot (number $E$ of empty spaces is at least one)
General Pattern

Producer:

\[
\text{repeat}\quad\text{produce item}\quad\text{buffer}[\text{in}] = \text{item}\quad\text{in} = (\text{in}+1) \mod K\quad\text{forever}
\]

Consumer:

\[
\text{repeat}\quad\text{item = buffer}[\text{out}]\quad\text{out} = (\text{out}+1) \mod K\quad\text{consume item}\quad\text{forever}
\]

Solution using Semaphores

- Semaphore S to enforce mutual exclusion when accessing the buffer
- Semaphore N to synchronize producer and consumer (number of consumable items)
- Semaphore E to synchronize producer and consumer (number of empty slots)
Solution using Semaphores

Initialization:

\[
\begin{align*}
\text{in} &= 0; \quad \text{// Next location to put item} \\
\text{out} &= 0; \quad \text{// Next location to get item} \\
\text{S.count} &= 1; \quad \text{// CS is free initially} \\
\text{N.count} &= 0; \quad \text{// 0 slots in use} \\
\text{E.count} &= K; \quad \text{// K slots empty}
\end{align*}
\]

Producer:

\[
\begin{align*}
\text{repeat} \\
\quad \text{produce item} \\
\quad \text{wait(E)} \\
\quad \text{wait(S)} \\
\quad \text{buffer[in]} &= \text{item} \\
\quad \text{in} &= (\text{in}+1) \mod K \\
\quad \text{signal(S)} \\
\quad \text{signal(N)} \\
\quad \text{forever}
\end{align*}
\]

Consumer:

\[
\begin{align*}
\text{repeat} \\
\quad \text{wait(N)} \\
\quad \text{wait(S)} \\
\quad \text{item} &= \text{buffer[out]} \\
\quad \text{out} &= (\text{out}+1) \mod K \\
\quad \text{signal(S)} \\
\quad \text{signal(E)} \\
\quad \text{consume item} \\
\quad \text{forever}
\end{align*}
\]
Summary: Semaphores

- Semaphores: powerful tool for enforcing mutual exclusion and coordinating processes

- Uses of wait(S) and signal(S) are scattered among several processes: difficult to understand their effects

- Usage must be correct in all the processes: one bad (or malicious) process can cause the entire collection of processes to fail

---

Deadlock and Starvation

- 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
Deadlock in computing systems

- When concurrent processes share resources or communicate with one another, deadlock is a very real possibility.

- Consider processes P and Q: they are executing concurrently, and each process needs both resources A and B.

- Depending on the order of operations within each process, deadlock may occur.

---

Scenario #1

<table>
<thead>
<tr>
<th>Process P</th>
<th>Process Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get A</td>
<td>Get B</td>
</tr>
<tr>
<td>Get B</td>
<td>Get A</td>
</tr>
<tr>
<td>Release A</td>
<td>Release B</td>
</tr>
<tr>
<td>Release B</td>
<td>Release A</td>
</tr>
</tbody>
</table>
Scenario #2

Process P               Process Q

Get A                  Get B
Release A              Get A
Get B                  Release B
Release B              Release A
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>Action</th>
<th>Process Q</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₀</td>
<td>Request (D)</td>
<td>q₀</td>
<td>Request (T)</td>
</tr>
<tr>
<td>P₁</td>
<td>Lock (D)</td>
<td>q₁</td>
<td>Lock (T)</td>
</tr>
<tr>
<td>P₂</td>
<td>Request (T)</td>
<td>q₂</td>
<td>Request (D)</td>
</tr>
<tr>
<td>P₃</td>
<td>Lock (T)</td>
<td>q₃</td>
<td>Lock (D)</td>
</tr>
<tr>
<td>P₄</td>
<td>Perform function</td>
<td>q₄</td>
<td>Perform function</td>
</tr>
<tr>
<td>P₅</td>
<td>Unlock (D)</td>
<td>q₅</td>
<td>Unlock (T)</td>
</tr>
<tr>
<td>P₆</td>
<td>Unlock (T)</td>
<td>q₆</td>
<td>Unlock (D)</td>
</tr>
</tbody>
</table>
Example (continued)

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

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

Example: processes compete for RAM

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

  \begin{itemize}
  \item P1
  \begin{itemize}
  \item Request 80 Kbytes;
  \item Request 60 Kbytes;
  \end{itemize}
  \item P2
  \begin{itemize}
  \item Request 70 Kbytes;
  \item Request 80 Kbytes;
  \end{itemize}
  \end{itemize}

- 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

Four Conditions for Deadlock

- 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