Processes and Sockets

Spring 2012
Models for Concurrent Programming

**Shared Memory**

- Analogy: two processors in a computer, sharing the same physical memory

Concurrent modules A and B interact by reading & writing shared state in memory

![Shared memory](image)

**Message Passing**

- Analogy: two computers in a network, communicating by network connections

A and B interact by sending messages to each other through a communication channel

![Message passing](image)
Message Passing Example

Modules interact by sending messages to each other

- Incoming requests are placed in a **queue** to be handled one at a time
- Sender doesn’t stop working while waiting for an answer to its request; it handles more requests from its own queue
- Reply eventually comes back as another message

<table>
<thead>
<tr>
<th>Account 1</th>
<th>Account 2</th>
<th>Account 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>bal: $50</td>
<td>bal: $200</td>
<td>bal: $50</td>
</tr>
</tbody>
</table>

- deposit $100 to account 1
- withdraw $100 from account 2
- deposit $100 to account 1
- get balance of account 1

- get bal
- dep $100
- dep $100

Queue for Account 1

- bal 1250
Message Passing Has the Same Risks

Message passing doesn’t eliminate race conditions

- Suppose the account state machine supports \texttt{get-balance} and \texttt{withdraw} operations (with corresponding messages)
- Can Alice and Bob always stay out of the OVERDRAWN state?

\begin{itemize}
  \item Alice
    \begin{itemize}
      \item get-balance
        \begin{itemize}
          \item if balance > \$75, withdraw \$75
        \end{itemize}
    \end{itemize}
  \item Bob
    \begin{itemize}
      \item get-balance
        \begin{itemize}
          \item if balance > \$50, withdraw \$50
        \end{itemize}
    \end{itemize}
\end{itemize}

\begin{itemize}
  \item Lesson: need to carefully choose the atomic (indivisible) operations of the state machine – withdraw-if-sufficient-funds would be better
\end{itemize}
Message Passing with Threads

Use a synchronized queue for message-passing between threads

- interface java.util.concurrent.BlockingQueue is such a queue
  - ArrayBlockingQueue is a fixed-size queue that uses an array representation
  - LinkedBlockingQueue is a growable queue (no FULL state) using a linked-list representation

No `take` transition in EMPTY state, so a thread that tries to `take` from an empty queue must `block` (wait) until it can

- ArrayBlockingQueue is a fixed-size queue that uses an array representation
- LinkedBlockingQueue is a growable queue (no FULL state) using a linked-list representation
Lists, Sets, and Maps can be made thread-safe by a wrapper function

- \( t = \text{Collections.synchronizedSet}(s) \) returns a thread-safe version of set \( s \), with a lock that prevents more than one thread from entering it at a time, forcing the others to block until the lock is free
- So we could imagine synchronizing all our sets:

\[
\text{thumbnails} = \text{Collections.synchronizedSet(} \text{new HashSet<Thumbnail>(})()
\]

This doesn’t fix all race conditions!

- Doesn’t help preserve invariants involving more than one data structure
More Thread-Safe Classes

Objects that never change state are usually* thread-safe

- **Immutable** objects never change state
  - e.g., java.lang.String is immutable, so threads can share strings as much as they like without fear of race conditions, and without any need for locks or queues

* Caveat: some apparently immutable objects may have hidden state: e.g. memoizing (caching) method return values.
Processes Vs. Threads

Process

Tries to make the program feel like it has the whole machine to itself; like a fresh computer has been created, with fresh memory

It is an instance of a running program that is isolated from other processes on the same machine

Thread

Simulates making a fresh processor inside the computer, running the same program and sharing the same memory as other threads in process

It is a locus of control inside a running program (i.e. position in the code + stack, representing the current point in the computation)
Client Server Pattern

Just what it’s name implies
Sockets

a network interface is identified by an IP address
➢ (or a hostname, which translates into an IP address)
➢ examples: 127.0.0.1, localhost; web.mit.edu

an interface has 65536 ports
➢ numbered from 0 to 65535

a server process binds to a port (the listening port)
➢ clients have to know which number it’s binding to.
➢ Some numbers are well-known (port 80 is the standard web server port, port 22 is the SSH port, port 25 is the standard SMTP email server port).
➢ When it’s not a standard port for the kind of server, you just treat it as part of the address
**Sockets**

The listening port is just used to accept incoming client connections.

- Once the connection is accepted, the server creates a new socket for the actual connection, with a fresh port number (unrelated to the listening port number).
- Both the client and server sockets have port numbers.