Concurrent processes and programming (cont’d)

Language mechanisms for synchronization

A concurrent language extended from a sequential language adds additional constructs to provide:

- Specification of concurrent activities
- Synchronization of processes
- Interprocess communication
- Nondeterministic execution of processes

Synchronization mechanisms and language facilities

<table>
<thead>
<tr>
<th>Synchronization methods</th>
<th>Language facilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared-variable synchronization</td>
<td></td>
</tr>
<tr>
<td>semaphore</td>
<td>shared variable and system call</td>
</tr>
<tr>
<td>monitor</td>
<td>data type abstraction</td>
</tr>
<tr>
<td>conditional critical region</td>
<td>control structure</td>
</tr>
<tr>
<td>serializer</td>
<td>data type and control structure</td>
</tr>
<tr>
<td>path expression</td>
<td>data type and program structure</td>
</tr>
<tr>
<td>Message passing synchronization</td>
<td></td>
</tr>
<tr>
<td>communicating sequential processes</td>
<td>input and output</td>
</tr>
<tr>
<td>remote procedure call</td>
<td>procedure call</td>
</tr>
<tr>
<td>rendezvous</td>
<td>procedure call and communication</td>
</tr>
</tbody>
</table>
Message passing synchronization

- The only means of communication in distributed systems
- Implicit synchronization: messages can be received only after they have been sent
- Non-blocking send, blocking receive: *asynchronous* message passing
- Blocking send, blocking receive: *synchronous* message passing

Asynchronous message passing:

- Is an extension of the semaphore concept to distributed systems
- Send operations assume that the channel has an unbounded buffer
- Example: *pipe* and *socket*

Synchronous message passing:

- No buffering of messages in the communication channel
- *rendezvous* between send and receive
- Examples: Communication Sequential Processes (CSP), Remote Procedure Call (RPC) - asymmetrical communication, Ada rendezvous - symmetrical communication
Interprocess communication and coordination

- Distributed IPC and process coordination are based on message passing
- Dependent on the ability to locate communication entities: role of the name service
- Three fundamental message passing communication models:
  - message passing
  - request/reply (RPC)
  - transaction communication
- Distributed process coordination examples:
  - distributed mutual exclusion
  - leader election

Message passing communication

- Messages are collections of data objects
- Their structure and interpretations are defined by the peer applications
- Communicating processes pass composed messages to the system transport service
Basic communication primitives:

- `send(destination, message)`
- `receive(source, message)`

where source or destination = (process name, link, mailbox, port)

process name (global PID) - direct communication primitive
link (connection) - direct communication primitive
mailbox - indirect communication primitive many-to-many
port - indirect communication primitive many-to-one

Message synchronization and buffering:

```
sender  source kernel  network  destination kernel  receiver
  1  →  2  →  message  →  3  →  4 request
     8 ←  7 ←  ack ←  6 ←  5 reply
```
1. **Nonblocking send**: 1+8
2. **Blocking send**: 1+2+7+8
3. **Reliable blocking send**: 1+2+3+6+7+8
4. **Explicit blocking send**: 1+2+3+4+5+6+7+8
5. **Request and reply**: 1-4, service, 5-8

At the receiving site **blocking** is quite explicit: blocked for message arrival

Implicit buffer space:

- in sender’s kernel
- in receiver’s kernel
- in the communication network

**Pipe and socket APIs**

- Used in both UNIX and Windows
- **Pipes**: implemented with finite-size, FIFO byte stream buffer maintained by the OS kernel
  - created with the pipe system call, which returns two descriptors (one for writing, one for reading)
  - data in pipes are uninterpreted byte sequences
  - are *anonymous*
– variation: named pipes - use the semantics of ordinary
files for opening, communicating processes need not exist
concurrently
– use limited to a single domain within a common file system
(except named pipes under Windows)

• **Socket** is a communication endpoint of a communication link
managed by the OS’s transport system

– modeling network I/O based on conventional file I/O
– created by the socket system call
– used for file-oriented read/write operations
– used for communication-specific send/receive operations
– communicate over various network protocols, for example
  TCP, UDP, (raw) IP
– socket descriptor is a logical communication endpoint
  (LCE); it must be associated with a physical communication
  endpoint (PCE): for example host network address and
  transport port in case of TCP or UDP
Connectionless socket communication:

- Peer process
  - Socket
  - Logical communication endpoint (socket, LCE)
  - Bind
  - Physical communication endpoint (PCE)
  - sendto / recvfrom

- Peer process
  - Socket
  - Logical communication endpoint (socket, LCE)
  - Bind
  - Physical communication endpoint (PCE)
Connection-oriented socket communication:

**Server**
- socket
- bind
- listen
- accept
- read
- write

**Client**
- socket
- connect
- write
- read

Arrows indicate the flow of the process:
- Rendezvous
- Request
- Reply
Secure Socket Layer

Goals:

- **Privacy** in socket communication
- **Integrity** of socket data
- **Authenticity** of servers and clients using asymmetric public-key cryptography

SSL consists of two protocols:

- **Handshake protocol**
  - establishing the **write keys** and **MAC secret** (message authentication check) $\rightarrow$ **master secret**
  - Validating the authenticity of clients and servers
  - Client of the Record Layer protocol

- **Record Layer protocol**
  - Fragmentation, compression/decompression
  - Encryption/decryption of message records
### SSL Handshake protocol

**CLIENT**

<table>
<thead>
<tr>
<th>Message</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientHello</td>
<td>randomC, CipherSuites</td>
</tr>
<tr>
<td></td>
<td>randomS, CipherSuite, session ID</td>
</tr>
<tr>
<td></td>
<td>server public key</td>
</tr>
<tr>
<td></td>
<td>encrypted pre-mastersecret</td>
</tr>
<tr>
<td>ClientKeyExchange</td>
<td></td>
</tr>
<tr>
<td>ChangeCipherSpec</td>
<td>hashed message and secret</td>
</tr>
<tr>
<td>Finished</td>
<td></td>
</tr>
<tr>
<td>SocketMessage</td>
<td></td>
</tr>
</tbody>
</table>

**SERVER**

<table>
<thead>
<tr>
<th>Message</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerHello</td>
<td></td>
</tr>
<tr>
<td>ServerKeyExchange</td>
<td></td>
</tr>
<tr>
<td>ChangeCipherSpec</td>
<td></td>
</tr>
<tr>
<td>Finished</td>
<td></td>
</tr>
<tr>
<td>SocketMessage</td>
<td></td>
</tr>
</tbody>
</table>