Concurrency Control in Distributed Environment

Ch 8 [BenA 06]

Messages
Channels
Rendezvous
RPC and RMI
Distributed System

- No shared memory
- Communication with messages
- Tightly coupled systems
  - Processes alive at the same time
- Persistent systems
  - Data stays even if processes die
- Fully distributed systems
  - Everything goes
Communication with Messages (4)

- Sender, receiver
- Synchronous/asynchronous communication

\[X = f(\ldots);\]
\[\text{send } X \text{ to } B\]
\[X: 10\]

\[\text{OS kernel} \quad \text{send} \quad \text{DC}\]

\[\text{receive } X \text{ from } A\]
\[Y = f(X);\]
\[X: 10\]

\[\text{DC} \quad \text{receive} \quad \text{OS kernel}\]
Message Passing

- **Synchronous communication**
  - Atomic action
  - Both wait until communication complete

- **Asynchronous communication**
  - Sender continues after giving the message to OS for delivery
  - May get an acknowledgement later on
    - Message received or not

- **Addressing**
  - Some address for receiver **process**
    - Process name, id, node/name, ...
  - Some address for the communication **channel**
    - Port number, channel name, ...
  - Some address for requested service
    - Broker will find out, sooner or later
      - After message has been sent?
    - Service address not known at service request time

Usual case

prosessi
kanava
palvelu
meklari

14.3.2011
Copyright Teemu Kerola 2011
Synchronization levels (10)

**Process A**

- **X**: 10
- **Y** = f(\(X\))
- **send** X to B

**Process B**

- **received** X from A
- **Y** = f(\(X\))
- **send** X to B

**OS kernel**

---

**reliable comm**

---

**asynchronous?**

---

Copyright Teemu Kerola 2011
Synchronization levels (2/5)

Process A

asynchronous?

...  
X = f(..);

send X to B

...

X: 10

send

OS kernel

Data Com

Process B

...  
receive X from A

Y = f(X);

...

X: 5

receive

DC

OS kernel

14.3.2011
Copyright Teemu Kerola 2011
Synchronization levels (3/5)

**Process A**

asynchronous?

...  
X = f((..));

send X to B

...

X: 10

OS kernel

**Process B**

...
receive X from A

Y = f(X);

...

X: 5

DC

receive

OS kernel

Copyright Teemu Kerola 2011
Synchronization levels (4/5)

... 
X = f(...);
... 
send X to B
... 
X: 10

OS kernel

Process A

OS kernel

receive X from A
Y = f(X);
... 
X: 5

Process B

... 
reliable comm

send X to B
Synchronization levels (5/5)

Process A

...  
X = f(..);

send X to B

...

X: 10

OS kernel

synchronous?

Process B

...  
receive X from A

Y = f(X);

...

X: 5

OS kernel

send

DC

receive
Message Passing

• **Symmetric communication**
  – Cooperating processes at same level
  – Both know about each others address
  – Communication method for a fixed channel

• **Asymmetric communication**
  – Different status for communicating processes
  – Client-server model
    • Server address known, client address given in request

• **Broadcast communication**
  – Receiver not addressed directly
  – Message sent to everybody (in one node?)
  – Receivers may be limited in number
    • Just one?
    • Only the intended recipient(s) will act on it?
Wait Semantics

- **Sender**
  - Continue after OS has taken the message
    - Non-blocking send
  - Continue after message reached receiver **node**
    - Blocking send
  - Continue after message reached receiver **process**
    - Blocking send

- **Receiver**
  - Continue only after message received
    - Blocking receive
  - Continue even if no message received
    - Status indicated whether message received or not
    - Non-blocking receive

Usual case
Message Passing

• Data flow
  – One-way
    • Synchronous may be one-way
    • Asynchronous is always one-way
  – Two-way
    • Synchronous may be two-way
    • Two asynchronous communications

• Primitives
  – One message at a time
  – Need addresses for communicating processes
  – Operating system level service
  – Usually not programming language level construct
    • Too primitive: need to know node id, process id, port number,…
Channels

- History of languages utilizing channels
  - Guarded Commands
    - Dijkstra, 1975
  - Communicating Sequential Processes
    - CSP, Hoare, 1978
  - Occam
    - David May et al, 1983
    - Hoare as consultant
    - Inmos Transputer
Guarded Commands (Dijkstra)

- Way to describe predicate transformer semantics
- Communication not really specified
- Guarded command
  - Condition or guard
  - Statement

```
x, y = X, Y  -- statement (unguarded)
do  -- loop command, loop terminates when x = y
    x ≠ y →
    if  -- conditional command (itself guarded)
        x > y → x := x-y  -- guarded statement in the if
        y > x → y := y-x
    fi
    od
print x ; -- another statement, also unguarded
```

predikaatti-muunnos-semantiikka
greatest common divisor

http://en.wikipedia.org
Communicating Sequential Processes – CSP (Hoare)

- **Language** for modeling and analyzing the behavior of concurrent communicating systems
- A known group of **processes** A, B, …
- Communication:
  - output statement: B!e
    - evaluate e, **send** the value of e to B
  - input statement: A?x
    - receive the value from A to x
  - input, output: **blocking** statements
  - output & input: “distributed assignment”
    - Communicate value from one process to a variable in some other process
CSP communication

- **Input/output statements**
  - Destination!port \((e_1, \ldots, e_n)\);
  - Source?port \((x_1, \ldots, x_n)\);

- **Binding**
  - Communication with **named processes**
  - Matching types for communication

- **Example:**  
  ```
  Copy ( West => Copy => East )
  ```

  **West:**
  ```
  do true ->
  Copy!c;
  ... 
  od
  ```

  **Copy:**
  ```
  do true ->
  West?c;
  East!c;
  od
  ```

  **East:**
  ```
  do true ->
  Copy?c;
  ... 
  od
  ```

14.3.2011
Copyright Teemu Kerola 2011
OCCAM Language

- Communication through **named channels**
  - Globally defined
    - Somewhere, in advance
  - Each channel has one sender and one receiver
    - both processes in some nodes

- Transputer
  - Multicomputer
    - E.g., 100 node Hathi-2 in ÅA
  - Automatic message routing for channels
  - Programmed with OCCAM
OCCAM Example

How to bind processes to nodes? 8 vs. 100 nodes?
How to bind channels to processes, physical system?
  – 4 physical ports (N, S, E, W) in each processor

PROC Copy (CHAN OF BYTE West, EAsks, East)
BYTE c1, c2, dummy;  -- buffer size = 2
SEQ
  West ? c1  -- West has 1st byte
  WHILE TRUE
    ALT
      West ? c2  -- West has new byte
      SEQ
        East ! c1  -- send previous byte
        c1 := c2  -- copy to buffer c1
      SEQ
        EAsks ? dummy  -- East wants a byte
        SEQ
          East ! c1  -- send previous byte
          West ? c1  -- receive next one

Discuss
Inmos Transputer

- B0042
- 2D array
- 10 boards 420 cpu’s
- 30 boards 1260 cpu’s

http://www.cs.bris.ac.uk/~dave/transputer.html

14.3.2011 Copyright Teemu Kerola 2011
Channels

- Communication through named channels
  - Typed, global to processes
  - Programming language concept
  - Any one can read/write
    (usually limited in practice)

- Pipe or mailbox
- Synchronous, one-way (?)
- How to tie in with many nodes?
  - Not really thought through! Easy with shared memory!

---

**Algorithm 8.1: Producer-consumer (channels)**

<table>
<thead>
<tr>
<th>Channel of integer ch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>producer</strong></td>
</tr>
<tr>
<td>integer x</td>
</tr>
<tr>
<td>loop forever</td>
</tr>
<tr>
<td>p1: x ← produce</td>
</tr>
<tr>
<td>p2: ch ← x</td>
</tr>
</tbody>
</table>

buffer size?
Filtering Problem

- Compress many (at most MAX) similar characters to pairs …
  - \{nr of chars, char\}
- … and place newline (\n) after every K’th character in the compressed string
- Why is it called “Conway’s problem”? … “Classic coroutine example”

Algorithm 8.2: Conway’s problem

constant integer MAX ← 9
constant integer K ← 4
channel of integer inC, pipe, outC

<table>
<thead>
<tr>
<th>compress</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>char c, previous ← 0</td>
<td>char c</td>
</tr>
<tr>
<td>integer n ← 0</td>
<td>integer m ← 0</td>
</tr>
<tr>
<td>inC ⇒ previous</td>
<td>loop forever</td>
</tr>
<tr>
<td>loop forever</td>
<td>no last char?</td>
</tr>
<tr>
<td>p1: inC ⇒ c</td>
<td>q1: pipe ⇒ c</td>
</tr>
<tr>
<td>p2: if (c = previous) and (n &lt; MAX - 1)</td>
<td>q2: outC ← c</td>
</tr>
<tr>
<td>p3: n ← n + 1</td>
<td>q3: m ← m + 1</td>
</tr>
<tr>
<td>else</td>
<td></td>
</tr>
<tr>
<td>p4: if n &gt; 0</td>
<td>q4: if m &gt;= K</td>
</tr>
<tr>
<td>p5: pipe ← intToChar(n+1)</td>
<td>q5: outC ← newline</td>
</tr>
<tr>
<td>p6: n ← 0</td>
<td>q6: m ← 0</td>
</tr>
<tr>
<td>p7: pipe ← previous</td>
<td></td>
</tr>
<tr>
<td>p8: previous ← c</td>
<td></td>
</tr>
<tr>
<td>q7:</td>
<td></td>
</tr>
<tr>
<td>q8:</td>
<td></td>
</tr>
</tbody>
</table>
Matrix Multiplication with Channels

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\times
\begin{bmatrix}
1 & 0 & 2 \\
0 & 1 & 2 \\
1 & 0 & 0
\end{bmatrix}
= 
\begin{bmatrix}
4 & 2 & 6 \\
10 & 5 & 18 \\
16 & 8 & 30
\end{bmatrix}
\]

- \(16 = (7 \ 8 \ 9) \cdot (1 \ 0 \ 1)\)
- \(30 = (7 \ 8 \ 9) \cdot (2 \ 2 \ 0)\)
- Process for every multiply-add

\[
7 \times 2 + 8 \times 2 + 9 \times 0 + 0 = 30
\]

channels

other processes
27 processes
24 channels

Column 1 contains 1 row, sends it down one element at a time

West-bound multiply-add, South-bound copy North

Contains 1 value, makes three multiply-adds, forwards values down

How to initialize everything?

How to synchronize everything?
Algorithm 8.3: Multiplier process with channels

```plaintext
integer FirstElement

channel of integer North, East, South, West

integer Sum, integer SecondElement

loop forever
p1: North ⇒ SecondElement
p2: East ⇒ Sum
p3: Sum ← Sum + FirstElement . SecondElement
p4: South ← SecondElement
p5: West ← Sum
```

- How to map processes to nodes?
- How to map channels to processes?
  - North channel of one process the South channel of some other
- North-South data flow has priority (*)
  - Waiting even when data-flow East-West available
  - Node on East may be blocked unnecessarily

Discuss
Algorithm 8.4: Multiplier with channels and selective input

integer FirstElement
channel of integer North, East, South, West
integer Sum, integer SecondElement

loop forever
  either
    p1: North → SecondElement
    p2: East → Sum
    If message from North available, do this
  or
    p3: East → Sum
    p4: North → SecondElement
    If message from East available, do this
    p5: South ← SecondElement
    p6: Sum ← Sum + FirstElement · SecondElement
    p7: West ← Sum

- Guarded statement
  - Execute one selective input statement
    - Nondeterministic selection (if both available)
    - p2 follows p1, it does not compete with p3

14.3.2011
Copyright Teemu Kerola 2011
Dining Philosophers with Channels

- Each `fork i` is a process, forks[i] is a channel
- Each `philosopher i` is a process

**Algorithm 8.5: Dining philosophers with channels**

<table>
<thead>
<tr>
<th>philosopher i</th>
<th>fork i</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean <code>dummy</code></td>
<td>boolean <code>dummy</code></td>
</tr>
<tr>
<td>loop forever</td>
<td>loop forever</td>
</tr>
<tr>
<td><code>p1</code>: think</td>
<td><code>q1</code>: forks[i] ← true</td>
</tr>
<tr>
<td><code>p2</code>: forks[i] ⇒ dummy</td>
<td><code>q2</code>: forks[i] ⇒ dummy</td>
</tr>
<tr>
<td><code>p3</code>: forks[i⊕1] ⇒ dummy</td>
<td><code>q3</code>:</td>
</tr>
<tr>
<td><code>p4</code>: eat</td>
<td><code>q4</code>:</td>
</tr>
<tr>
<td><code>p5</code>: forks[i] ← true (would false be ok?)</td>
<td><code>q5</code>: deadlock?</td>
</tr>
<tr>
<td><code>p6</code>: forks[i⊕1] ← true</td>
<td><code>q6</code>:</td>
</tr>
</tbody>
</table>

- Would it be enough to initialize each `forks[i] <= true`?
  - Do you really need `forks[i] => dummy` in fork i? Why?
Rendezvous (1978, Abrial & Andrews)

- **Synchronization with communication**
  - No channels, usage similar to procedure calls
  - One (*accepting*) process waits for one of the (*calling*) processes
    - One request in service at a time
    - Calling process must know id of the accepting process
    - Accepting process does not need to know the id of calling process
    - May involve parameters and return value

- **Good for client-server synchronization**
  - Clients are calling processes
  - Server is accepting process
  - Server is active process
  - Language construct, no mapping for real system nodes
Can have many similar clients

Implementation with messages (e.g.)
  – Service request in one message
    • Arguments must be marshalled
      (make them suitable for transmission)
  – Wait until reply received
  – Reply result in another message

### Algorithm 8.6: Rendezvous

<table>
<thead>
<tr>
<th>client</th>
<th>server</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer parm, result loop forever</td>
<td>integer p, r loop forever</td>
</tr>
<tr>
<td>p1: parm ← ...</td>
<td>q1:</td>
</tr>
<tr>
<td>p2: server.service(parm, result)</td>
<td>q2: accept service(p, r)</td>
</tr>
<tr>
<td>p3: use(result)</td>
<td>q3: r ← do the service(p)</td>
</tr>
</tbody>
</table>
Guards in Rendezvous

- Additional constraint for accepting given service call
- Accept service call, if
  - Someone requests it and
  - Guard for that request type is true
    - Guard is based on local state
- If many such requests (with open guards) available, select one randomly
- Complete one request at a time
  - Implicit mutex
Bounded Buffer in Ada

Export public ops defined before task body

**task body** Buffer is

B: Buffer_Array;

In_Ptr, Out_Ptr, Count: Index := 0;

... Buffer.Append (456);
Buffer.Append (333);
...

... Buffer.Take(x);
Buffer.Take(y);
...

begin
loop
select
when Count < Index’Last =>
accept Append(l: in Integer) do
B(In_Ptr) := l;
end Append;
Count := Count + 1; In_Ptr := In_Ptr + 1;

or
when Count > 0 =>
accept Take(l: out Integer) do
l := B(Out_Ptr);
end Take;
Count := Count - 1; Out_Ptr := Out_Ptr + 1;

or
terminate
end select;
end loop;
end Buffer;

Terminates when no rendezvous processes available? Tricky!
How to know?
No concurrent operations!
Remote Procedure Call

- Common **operating system service** for client-server model synchronization
  - Implemented with messages
  - Parameter marshalling
    - Semantics remain, implementation may change
  - Mutex problem
    - Combines monitor and synchronized messages?
      - Automatic mutex for service
    - Multiple calls active simultaneously?
      - Mutex problems solved within called service
  - Semantics similar to ordinary procedure call
    - But no global environment (e.g., shared array)
  - Two-way synchronized communication channel
    - Client waits until service completed (usually)
RPC System Structure

- Client application
  - Local response
  - Local procedure calls
- Remote server application
  - Local response
  - Local procedure call

Local application or operating system Qlocal
- Local stub
- RPC mechanism
- Remote procedure call

Local stub
- RPC mechanism
- Remote procedure call

(Qremote calls Q)

(Sta05 Fig 14.12)
RPC Module

module mname
  \textbf{op} opname(formals) \ [\textit{returns} \ result] \quad \textbf{Export public ops}

body
  variable declarations;
  initialization code;
  \textbf{proc} opname(formal identifiers) \textit{returns} result identifier
  declarations of local variables;
  statements
  end
end mname

Call: \texttt{call mname.opname(arguments)}
RPC Example: Time Server

module TimeServer

\[
\begin{align*}
\text{op get_time()} & \text{ returns int; \quad \# retrieve time of day} \\
\text{op delay(int interval);} & \text{ \quad \# delay interval ticks} \\
\end{align*}
\]

body

\[
\begin{align*}
\text{int tod} & = 0; \quad \# \text{the time of day} \\
\text{sem m} & = 1; \quad \# \text{mutual exclusion semaphore} \\
\text{sem d[n]} & = ([n] 0); \quad \# \text{private delay semaphores} \\
\text{queue of (int waketime, int process_id) napQ;} & \\
\text{## when m == 1, tod < waketime for delayed processes} \\
\text{proc get_time() returns time {} & \\
\text{\quad time} & = \text{tod;} \\
\}} \\
\text{proc delay(interval) { \quad \# assume interval > 0} & \\
\text{\quad int waketime} & = \text{tod + interval;} \\
\text{\quad P(m);} & \\
\text{\quad insert (waketime, myid) at appropriate place on napQ;} & \\
\text{\quad V(m);} & \\
\text{\quad P(d[myid]); \quad \# wait to be awakened} \\
\}}
\]

(process Clock{} on next slide)
- **Internal process**
  - Keeps the time
  - Wakes up delayed clients
- **Service RPC's:**
  ```java
  time = TimeServer.get_time();
  TimeServer.delay(10);
  ```
These routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a data packet to the server. Upon receipt of the packet, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)
callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)
callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)
callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)
callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)
callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)
callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)
Remote Method Invocation (RMI)

package example.hello;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
    String sayHello() throws RemoteException;
}

http://java.sun.com/j2se/1.5.0/docs/guide/rmi/hello/hello-world.html

- Java RPC
- Start rmiregistry
  - Stub lookup (default at port 1099)
- Start rmi server
  - Server runs until explicitly terminated by user

java -classpath classDir example.hello.Server &
start java -classpath classDir example.hello.Server
package example.hello;
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class Server implements Hello {
    public Server() {}
    public String sayHello() {
        return "Hello, world!";
    }
    public static void main(String args[]) {
        try { Server obj = new Server();
            Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);
            // Bind the remote object's stub in the registry
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Hello", stub);
            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}

Output: Server ready
package example.hello;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
    private Client() {}
    public static void main(String[] args) {
        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            Hello stub = (Hello) registry.lookup("Hello");
            String response = stub.sayHello();
            System.out.println("response: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}

Output: response: Hello, world!
Summary

- Distributed communication with messages
  - Synchronization and communication
  - Computation time + communication time = ?

- Higher level concepts
  - Guarded commands (theoretical background)
  - CSP (idea) & Occam (application)
  - Named Channels (ok without shared memory?)
  - Rendezvous
  - RPC & RMI (Java)