COMP30112: Concurrency

Topics 4.3: Message Passing

Howard Barringer

Room KB2.20: email: Howard.Barringer@manchester.ac.uk

March 2009
Outline

Topic 4.3: Message Passing
  Background
  Synchronous Message Passing
  Asynchronous Message Passing
  Channel Selection
Topic 4.3: Message Passing

Outline

Topic 4.3: Message Passing

Background

Synchronous Message Passing
Asynchronous Message Passing
Channel Selection
The Message Passing Paradigm

- processes send and receive messages via channels
The Message Passing Paradigm

- processes send and receive messages via channels
- no need for shared memory (i.e. for shared variables)
The Message Passing Paradigm

- processes send and receive messages via channels
- no need for shared memory (i.e. for shared variables)
- distributed computation possible
The Message Passing Paradigm

- processes *send* and *receive* messages via *channels*
- no *need* for shared memory (i.e. for shared variables)
- *distributed* computation possible
- point-to-point, one-to-many (sender/receiver), many-to-one, (even many-to-many?!)
Outline

Topic 4.3: Message Passing

Background

Synchronous Message Passing

Asynchronous Message Passing

Channel Selection
Synchronous Message Passing — SMP

- assume one sender, one receiver
Synchronous Message Passing — SMP

• assume one sender, one receiver
• Port ch = new Port()
Synchronous Message Passing — SMP

- assume one sender, one receiver
- `Port ch = new Port();`
- `void send(Port ch, Message msg)`
Synchronous Message Passing — SMP

- assume one sender, one receiver
- Port `ch = new Port()`
- `void send(Port ch, Message msg)`
- `Message receive(Port ch)`
Synchronous Message Passing — SMP

- assume one sender, one receiver
- Port ch = new Port()
- void send(Port ch, Message msg)
- Message receive(Port ch)
- receiver process *blocks* until message is sent on channel
Synchronous Message Passing — SMP

- assume one sender, one receiver
- Port ch = new Port()
- void send(Port ch, Message msg)
- Message receive(Port ch)
- receiver process *blocks* until message is sent on channel
- sender process *blocks* until message is received on channel
Synchronous Message Passing — SMP

- assume one sender, one receiver
- Port ch = new Port()
- void send(Port ch, Message msg)
- Message receive(Port ch)
- receiver process *blocks* until message is sent on channel
- sender process *blocks* until message is received on channel
- i.e. synchronisation between sender/receiver processes
An FSP Model for SMP

range Message = 0..9

SENDER = SEND[0],
SEND[msg: Message] = (send[msg] -> SEND[(msg+1)%10]).


||SYS0 = (SENDER || RECEIVER)/{chan1/receive,chan1/send}. 
SYS1 = ( SENDER/{chan1/send} || RECEIVER/{chan1/receive} )

SYS2 = ( SENDER/{chan1/send} || RECEIVER/{chan1/receive} || SENDER/{chan2/send} || RECEIVER/{chan2/receive} )
Message Sequencing

PROCESS 1
send(ch1,m1)

PROCESS 2
receive(ch1)
receive(ch2)

PROCESS 3
send(ch2,m2)
Synchronous Message Passing in Java

public class Channel {

    Object chan = null;
    int ready = 0;

    public synchronized void send(Object v) throws InterruptedException {
        chan = v;
        ++ready;
        notifyAll();
        while (chan != null) wait();
    }
}
public synchronized Object receive()
    throws InterruptedException {
    while(ready==0) wait();
    --ready;
    Object tmp = chan; chan = null;
    notifyAll();
    return(tmp);
}
Outline

**Topic 4.3: Message Passing**

- Background
- Synchronous Message Passing
- Asynchronous Message Passing
- Channel Selection
Asynchronous Message Passing

- assume many senders, one receiver
Asynchronous Message Passing

- assume many senders, one receiver
- `Port ch = new Port()`
Asynchronous Message Passing

- assume many senders, one receiver
- Port ch = new Port()
- void send(Port ch, Message msg)
Asynchronous Message Passing

- assume many senders, one receiver
- `Port ch = new Port()`
- `void send(Port ch, Message msg)`
- `Message receive(Port ch)`
Asynchronous Message Passing

- assume many senders, one receiver
- Port ch = new Port()
- void send(Port ch, Message msg)
- Message receive(Port ch)
- receiver process *blocks* unless a message is queued at port
Asynchronous Message Passing

- assume many senders, one receiver
- Port ch = new Port()
- void send(Port ch, Message msg)
- Message receive(Port ch)
- receiver process \textit{blocks} unless a message is queued at port
- sender process continues execution as soon as message is sent
Modelling Asynchronous Message Passing

What we would like to do... *pseudo*-FSP with infinite lists and data:

set Message = Integer

PORT = PORT[],

PORT[] = ( send[msg: Message] → PORT[msg] ),

PORT[(hd::tl): Message List] =
  ( send[msg: Message] → PORT[(hd::tl) @ [msg]]
    || receive[hd] → PORT[tl]
  ).
In FSP ...

As usual, in practice... must use finite lists with restricted data:

range Message = 0..3
set S = {[Message], [Message][Message]}

PORT = ( send[msg: Message] -> PORT[msg] ),

PORT[hd: Message] =
  ( send[msg: Message] -> PORT[hd][msg]
    | receive[hd] -> PORT
  ),

PORT[hd: Message][tl: S] =
  ( send[msg: Message] -> PORT[hd][tl][msg]
    | receive[hd] -> PORT[tl] )..

...or even abstract away from data altogether:

\[
\text{PORT} = \text{PORT}[0], \\
\text{PORT}[i: \text{Int}] = (\text{send} \rightarrow \text{PORT}[i+1] \\
\text{when } (i>0) \text{ receive} \rightarrow \text{PORT}[i-1]).
\]

\[
\text{SENDER} = (\text{send} \rightarrow \text{SENDER}). \\
\text{RECEIVER} = (\text{receive} \rightarrow \text{RECEIVER}).
\]

\[
\text{||SYS} = (\{s[1..2]\}:\text{PORT} || s[1..2]:\text{SENDER} || \text{RECEIVER} \\
/\{\text{receive}/s[1..2].\text{receive} \}).
\]
class Port {

    Vector queue = new Vector();

    public synchronized void send(Object v) {
        queue.addElement(v);
        ++ready;
        notifyAll();
    }
}
public synchronized Object receive()
    throws InterruptedException {
    while(ready==0) wait();
    --ready;
    Object tmp = queue.elementAt(0);
    queue.removeElementAt(0);
    return(tmp);
}
Topic 4.3: Message Passing

Outline

Topic 4.3: Message Passing

Background
Synchronous Message Passing
Asynchronous Message Passing
Channel Selection
Selecting Channels - I

SELECT =
   ( when (guard1) receive1[v1:Message] -> P1[v1]
   | when (guard2) receive2[v2:Message] -> P2[v2]
   ... 
   | when (guardN) receiveN[vN:Message] -> PN[vN]).

SENDER1 = (send1[e1] -> SENDER1).
...
SENDERN = (sendN[eN] -> SENDERN).

||SYS = (  SELECT/{chan1/receive1,...,chanN/receiveN}
          || SENDER1/{chan1/send1} ...
          || SENDERN/{chanN/sendN}
          ).
GOBETWEEN = GO[0],
GO[i:0..2] = ( when (i==0) sender[msg:1..2] -> GO[msg]  
| when (i>0) req -> write[i] -> GO[0]  
).

SENDER = ( sender[1] -> SENDER  
| sender[2] -> SENDER  
).

RECEIVER = (req -> write[i:1..2] -> RECEIVER).

||SYS = (GOBETWEEN || SENDER || RECEIVER).
class GoBetween extends Thread {
    private Channel sender, req, write;
    private int i=0;

    public void run () {
        Select sel = new Select();
        sel.add(sender);
        sel.add(req);

        while (true) {
            sender.guard(i==0);
            req.guard(i>0);
            switch (sel.choose()) {
                case 1: i = sender.receive(); break;
                case 2: req.receive();
                    write.send(i); i = 0; break;
            }
        }
    }