

Message Passing

K. V. S. Prasad
Dept of Computer Science
Chalmers University
22 Sep 2014

Transition

- Why do we need other models?
- Advent of distributed systems
 - Mostly by packages such as MPI
 - Message passing interface
- But Hoare 1978
 - arrived before distributed systems
 - I see it as the first realisation that
 - Atomic actions, critical regions, semaphores, monitors...
 - Can be replaced by just I/O as primitives!

Models of Communication

- Speech = broadcast
 - Synchronous communication
 - Asynchronous actions (not clocked)
 - Speaker autonomous
- Post or email = asynchronous channel (buffer)
 - Both communication and action asynchronous
 - Speaker autonomous
- Telephone = synchronous channel = 0 size buffer
 - Synchronous communication and actions
 - Only internal actions autonomous

Addressing

- Broadcast
 - Sender and/or receiver anonymous
 - Can be named (maybe) in message
- Post, email, telephone
 - Receiver named (envelope, header, number)
 - Sender need not be (but can)
- What is addressed?
 - Processes? Channels?

What do processes communicate or share?

- Data
 - Tell me what you've heard
- Resources
 - Databases – don't want inconsistent DB
 - printer – don't want interleaved printouts
- Timing signals
 - Pure timing signals: empty envelopes, beeps, etc.
- So expect (equivalents of) semaphores, etc.
- Channels can be shared between processes
 - In some languages
 - But in Erlang, e.g., only one proc can input from it

Semaphore by synchronous channels

Each user:

```
loop
  chwait => token
  crit sec
  chsignal <= token
```

Semaphore:

```
loop
  chwait <= token
  chsignal => token
```

Explanation: Only one of contending users gets the token from chwait, and the semaphore then waits till this user returns the token.

The token is just a dummy (uint type, empty envelope)

Semaphore by asynchronous channels

Each user (i):

loop

chrequest <= i

chwait(i) => token

crit sec

chsignal <= token

Semaphore:

local integer i

loop

chrequest => i

chwait(i) <= token

chsignal => token

Explanation: Channels request and signal are received by the semaphore .

Simplest to have only receiver, with asynchronous channels.

One user gets its request accepted; other requests stay in the buffer.

The accepted request says on which channel the token is to be sent.

If all the users share a wait channel, the token can be stolen.

Broadcast channel is a semaphore!

Each user (i):

loop

either

ch <= i

crit sec

ch <= done

or

ch => j

ch => done

Explanation: Did I succeed in speaking (tjing)? If not, I wait till I hear another thing.

Here the channel is used only for this semaphore. If it is used for other things too, the losing process should test what it hears till it hears done.

Examples from the book

- Producer-consumer
 - Doesn't matter whether synch/asynch
- Matrix-multiplication
 - Here, could be synchronous action : gangstepped
- Dining philosophers
 - With synchronous channels only.
 - Each fork behaves like a semaphore
 - Both deadlock and starvation seem possible!

The matrix example

$$\begin{array}{ccc} \text{--} & & \text{--} & & \text{--} & & \text{--} \\ | & 1 & 2 & 3 & | & & | & 1 & 0 & 2 & | \\ | & 4 & 5 & 6 & | & \times & | & 0 & 1 & 2 & | \\ | & 7 & 8 & 9 & | & & | & 1 & 0 & 0 & | \\ \text{--} & & \text{--} & & & & & & & & \end{array}$$

Have to do a series of dot products like

$$\begin{array}{l} [7, 8, 9] \times \begin{array}{|c|} \hline 2 \\ \hline 2 \\ \hline 0 \\ \hline \end{array} = 7*2 + 8*2 + 9*0 = 14+16+0 = 30 \end{array}$$

Factorial

$$\text{fac } 0 = 1$$

$$\text{fac } n = n * \text{fac } (n-1) \text{ -- use if parm } \langle \rangle 0$$

In the context of this program of two definitions,
an expression is evaluated as follows: for a non-canonical term, find a
matching pattern, and replace lhs by rhs

$$\text{fac } 3 = 3 * \text{fac } 2$$

$$= 3 * (2 * \text{fac } 1)$$

$$= 3 * (2 * (1 * \text{fac } 0))$$

$$= 3 * (2 * (1 * 1))$$

$$= 3 * (2 * 1)$$

$$= 3 * 2$$

$$= 6$$

Functional Programming (FP)

- (Pure) function has no side-effects
 - takes arguments and returns a value,
 - can use it wherever a value is expected
 - Not an imperative statement (command)
- In FP
 - A program is a set of definitions
 - To run this program, give it input: an expression
 - A run is the evaluation of this expression
- Anything you can do with Turing machines(TM), you can do with FP, and vice-versa

Rendezvous

- Like synchronous channel, except
 - Addressing asymmetric
 - Sender knows receiver's address (entry), not v-v.
 - The communication may involve computation and return of value by the receiver
 - So made for client-server

Ada

- Uses protected objects
 - Since the 1980's
 - though the concept was around earlier
 - Thus has the cleanest shared memory model
- Also has a very good communication model
 - Rendezvous
- Ada was decided carefully through the 1970s
 - Open debates and process of definition
- Has fallen away because of popularity of C, etc.
 - Use now seen as a proprietary secret!

Robin Milner (1934-2010)

- Turing Award 1992 for CCS, ML and LCF!
- Went on to develop pi-calculus
 - Functions as processes
- Bigraphs
- CCS uses synchronous channels to make a complete calculus (programming and reasoning)