Lecture 9: Message Passing 1

Adam Hawley

February 22, 2019

Contents

1 Message-Based Coordination

Shared-variable coordination is based on *normal* variable access by multiple processes. Message-passing coordination requires new primitives. Abstractly, these are seen as:

- `send(message)`
- `receive(message)`

A message can never be received (read) before it has been sent (written). This means that `receive(message)` is potentially **blocking**.

2 Evaluating Message-Based Coordination

There are a number of ways to describe different approaches:

- Models for send and receive
- How tasks are identified
- The relationship between tasks
- Types of messaages that can be sent

3 Possible Models for Sending

3.1 Aynchronous

Sender does not wait, `send()` returns *immediately* but sent messages must be buffered.

3.2 Synchronous

Sender blocks until receiver can receive, vice-versa. Message can pass directly from sender to receiver, this means no buffer is necessary. Sometimes this approach is called *rendezvous*.

3.3 Remote Invocation

Sender blocks until receiver is ready (Synchronous). This time, when the message is received, the receiver will flag an acknowledgement or reply from the reviver. This is also known as an *extended rendezvous*.

4 Task Naming

How do senders and receivers refer to each other?

- Direct
- Indirect

- Symmetric
- Asymmetric

4.1 Direct

Tasks have names. This is used to identify end points, e.g send(message) to Joe.

4.2 Indirect

An intermediate is used (e.g. a *mailbox* or a *channel*). These names can be statically named.

4.3 Symmetric

Sender specifies receiver and vice versa. For example:

- `send(message) to task`
- `receive(message) from task`

4.4 Asymmetric

Send to named receiver, receive from any sender:

- `send(message) to task`
- `receive(message)`

5 Task Relationships

What is the relationship between senders and receivers?

- One-to-one (can be synchronous or buffered)
- One-to-many
  - A **multi-cast** communication
  - Sometimes inaccurately called a *broadcast* which is one-to-everybody
  - Requires indirect naming
- Many-to-many (multiple multi-casts)

6 Message Types

These can be limited to a set of predefined types or unlimited (all types/classes can be communicated). Low-level languages are more likely to limit message type.

SEE LECTURE FOR OCCAM