

Non-Determinism

©2007 Dept Computer Science, Australian National University

Selective Waiting

- Dijkstra's guarded commands:

```
if x <= y -> m := x  
□ x >= y -> m := y  
fi
```

selection is
non-deterministic!

- The programmer needs to design the alternatives as 'parallel' options:
 - all cases need to be covered and overlapping conditions need to lead to the same result
- Extremely different philosophy: 'C'-switch:

```
switch (x) {  
  case 1: r := 3;  
  case 2: r := 2; break;  
  case 3: r := 1;  
}
```

- the sequence of alternatives has a crucial role.

©2007 Dept Computer Science, Australian National University

Selective waiting in Occam2

```
ALT
  Guard1
    Process1
  Guard2
    Process2
...
```

- Guards are referring to boolean expressions and/or channel input operations.
- The boolean expressions are local expressions, i.e. if none of them evaluates to true at the time of the evaluation of the ALT-statement, then the process is stopped.
- If all triggered channel input operations evaluate to false, the process is suspended until further activity on one of the named channels.
- Any Occam2 process can be employed in the ALT-statement
- The ALT-statement is non-deterministic (there is also a deterministic version: PRI ALT).

©2007 Dept Computer Science, Australian National University

Selective waiting in Occam2

```
ALT
  NumberInBuffer < Size & Append ? Buffer [Top]
    SEQ
      NumberInBuffer := NumberInBuffer + 1
      Top             := (Top + 1) REM Size
  NumberInBuffer > 0 & Request ? ANY
    SEQ
      Take ! Buffer [Base]
      NumberInBuffer := NumberInBuffer - 1
      Base            := (Base + 1) REM Size
```

- Synchronization on input-channels only:
 - to initiate the sending of data (Take ! Buffer [Base]), a request need to be made first (Request ? ANY)
- CSP (Hoare) also supports non-deterministic selective waiting

©2007 Dept Computer Science, Australian National University

Message-based selective synchronization in Ada95

- Forms of selective waiting:

```
select_statement ::= selective_accept |
                  conditional_entry_call |
                  timed_entry_call |
                  asynchronous_select
```

- underlying concept: Dijkstra's guarded commands
- `selective_accept` implements ...
 - wait for more than a single rendezvous at any one time
 - time-out if no rendezvous is forthcoming within a specified time
 - withdraw its offer to communicate if no rendezvous is available immediately
 - terminate if no clients can possibly call its entries

©2007 Dept Computer Science, Australian National University

Selective_Accept in Ada95

```
selective_accept ::= select
                    [guard] selective_accept_alternative
                    { or [guard] selective_accept_alternative
                    [ else sequence_of_statements ]
                    end select;
```

```
guard ::= when <condition> =>
```

```
selective_accept_alternative ::= accept_alternative |
                                delay_alternative |
                                terminate_alternative
```

```
accept_alternative ::= accept_statement [ sequence_of_statements ]
```

```
delay_alternative ::= delay_statement [ sequence_of_statements ]
```

```
terminate_alternative ::= terminate;
```

©2007 Dept Computer Science, Australian National University

Ada95: Select-or

```
select
  accept ... do ...
end ...
or
  accept ... do ...
end ...
or
  accept ... do ...
end ...
or
  accept ... do ...
end ...
...
end select;
```

- If none of the named entries have been called, the task is suspended until one of the entries is addressed by another task.
- The selection of an accept is non-deterministic, in case that multiple entries are called.
 - The selection can be controlled by means of the real-time systems annex.
- The select statement is completed, when at least one of the entries has been called and those accept-block has been executed.

©2007 Dept Computer Science, Australian National University

Ada95: Guarded Select-or

```
select
  when <condition> =>
    accept ... do ...
  end ...
or
  when <condition> =>
    accept ... do ...
  end ...
or
  when <condition> =>
    accept ... do ...
  end ...
...
end select;
```

- Analogue to Dijkstra's guarded commands
- All accepts closed will raise a Program_Error
 - set of conditions need to be complete

©2007 Dept Computer Science, Australian National University

Ada95: Guarded Select-or-else

```
select
  [ when <condition> => ]
    accept ... do ...
    end ...
  or
  [ when <condition> => ]
    accept ... do ...
    end ...
  or
  [ when <condition> => ]
    accept ... do ...
    end ...
  else
    <statements>
  ...
end select;
```

- If none of the open entries can be accepted immediately, the else alternative is selected.
- There can be only one else alternative and it cannot be guarded.

©2007 Dept Computer Science, Australian National University

Ada95: Guarded Select-or-delay

```
select
  [ when <condition> => ]
    accept ... do ...
    end ...
  or
  [ when <condition> => ]
    delay ...
    <statements>
  or
  [ when <condition> => ]
    delay ...
    <statements>
  ...
end select;
```

- If none of the open entries has been called before the amount of time specified in the earliest open delay alternative, this delay alternative is selected.
- There can be multiple delay alternatives if more than one delay alternative expires simultaneously, either one may be chosen.
- delay and delay until can be employed.

©2007 Dept Computer Science, Australian National University

Ada95: Guarded Select-or-terminate

```

select
  [ when <condition> => ]
    accept ... do ...
    end ...
or
  [ when <condition> => ]
    accept ... do ...
    end ...
or
  [ when <condition> => ]
    terminate;
...
end select;

```

- The terminate alternative is chosen if none of the entries can ever be called again, i.e.:
 - all tasks which can possibly call any of the named entries are terminated.
- or**
 - all remaining active tasks which can possibly call any of the named entries are waiting on selective terminate statements and none of their open entries can be called any longer.
- This task and all its dependent waiting-for-termination tasks are terminated together

©2007 Dept Computer Science, Australian National University

Ada95: Guarded Select-or-else, Select-or-delay, Select-or-terminate

```

select
  [ when <condition> => ]
    accept ... do ...
    end ...
or
  [ when <condition> => ]
    accept ... do ...
    end ...
else
  <statements>
...
end select;

select
  [ when <condition> => ]
    accept ... do ...
    end ...
or
  [ when <condition> => ]
    delay ...
    <statements>
...
end select;

select
  [ when <condition> => ]
    accept ... do ...
    end ...
or
  [ when <condition> => ]
    terminate;
...
end select;

select
  [ when <condition> => ]
    accept ... do ...
    end ...
or
  [ when <condition> => ]
    delay ...
    <statements>
...
end select;

select
  [ when <condition> => ]
    accept ... do ...
    end ...
or
  [ when <condition> => ]
    terminate;
...
end select;

```

©2007 Dept Computer Science, Australian National University

Ada95: Guarded Select-or-else, Select-or-delay, Select-or-terminate

```

select
  [ when <condition> => ] or
    accept ... do ...
    end ...
or
  [ when <condition> => ]
    delay ...
    <statements>
else
  <statements>
  [ when <condition> => ]
  accept ... do ...
  end ...
end select;

select
  [ when <condition> => ]
  accept ... do ...
  or
  [ when <condition> => ]
  terminate;
  ...
end select;

```

else - delay – terminate
 alternatives
 cannot be mixed!

©2007 Dept Computer Science, Australian National University

Conditional & timed entry-calls

```

conditional_entry_call ::=
  select
    entry_call_statement
    [sequence_of_statements]
  else
    sequence_of_statements
  end select;

timed_entry_call ::=
  select
    entry_call_statement
    [sequence_of_statements]
  or
    delay_alternative
  end select;

select
  Light_Monitor.Wait_Light;
  Lux := True;
else
  Lux := False;
end;

select
  Controller.Request(Medium)
  (Some_Item);
  -- process data
or
  delay 45.0;
  -- try something else
end select;

```

©2007 Dept Computer Science, Australian National University

Conditional & timed entry-calls

```
conditional_entry_call ::=
select
  entry_call_statement or
  [sequence_of_statements]
else
  sequence_of_statements
end select

timed_entry_call ::=
select
  entry_call_statement
  [sequence_of_statements]
or
  delay 45.0;
  -- try something else
end select;
```

There is only
one entry call
and either
one 'else'
or
one 'or delay'

©2007 Dept Computer Science, Australian National University

Conditional & timed entry-calls

```
conditional_entry_call ::=
select
  entry_call_statement
  [sequence_of_statements]
or
  entry_call_statement
  [sequence_of_statements]
end select

timed_entry_call ::=
select
  entry_call_statement
  [sequence_of_statements]
or
  delay 45.0;
  -- try something else
end select;
```

There idea in both cases is to **withdraw a synchronization request** and **not to implement polling or busy-waiting.**

```
select
  Light_Monitor.Wait_Light;
  Lux := True;
  Controller.Request(Medium)
  (Some_Item);
  -- process data
else
  Lux := False;
  delay 45.0;
  -- try something else
end;
```

©2007 Dept Computer Science, Australian National University

Non-determinism in selective synchronizations

- If equal alternatives are given, then the program correctness (incl. the timing specifications) must not be affected by the actual selection.
- If alternatives have different priorities, this can be expressed e.g. by means of the Ada real-time annex.
- Non-determinism in concurrent systems is or can be also introduced by:
 - non-ordered monitor or other queues
 - buffering / routing message passing systems
 - non-deterministic schedulers
 - timer quantization
 - clock drifts
 - network congestions
 - ... any other form of asynchronism

©2007 Dept Computer Science, Australian National University

The concurrent programming abstraction

- *Remember our introduction:*
 - *Models and Terminology*
- *Correctness of concurrent non-real-time systems [logical correctness]:*
 - *does not depend on speeds / execution times / delays*
 - *does not depend on actual interleaving of concurrent processes*
- **Does *depend* on all possible sequences of interaction points**

©2007 Dept Computer Science, Australian National University

The concurrent programming abstraction

- Extended concepts of correctness in concurrent systems:
 - Termination is often not intended or even considered a failure

- Safety properties:

$$(P(I) \wedge \text{Processors}(I, S)) \Rightarrow \Box Q(I, S)$$

where $\Box Q$ means that Q does *always* hold

- Liveness properties:

$$(P(I) \wedge \text{Processors}(I, S)) \Rightarrow \Diamond Q(I, S)$$

where $\Diamond Q$ means that Q does *eventually* hold (and will then stay true) and S is the current state of the concurrent system

©2007 Dept Computer Science, Australian National University

The concurrent programming abstraction

- *Correctness of concurrent non-real-time systems [logical correctness]:*
 - **does depend on all possible sequences of interaction points**
- Isn't there an actual unique sequence of interaction points which is determined by the system and can be calculated?
- **In general – NO**
 - *due to common intrinsically non-deterministic effects*

©2007 Dept Computer Science, Australian National University

Selective waiting

- Dijkstra's guarded commands:

```
if x <= y -> m := x
□ x >= y -> m := y
fi
```

selection is
non-deterministic!

- the programmer needs to design the alternatives as 'parallel' options: all cases need to be covered and overlapping conditions need to lead to the same result
- *Systems based on non-deterministic alternatives extent canonically to concurrent systems*

©2007 Dept Computer Science, Australian National University

Ada95: Guarded Select-or

```
select
  when <condition> =>
    accept ... do ...
  end ...
or
  when <condition> =>
    accept ... do ...
  end ...
or
  when <condition> =>
    accept ... do ...
  end ...
...
end select;
```

- Considering all alternatives leads to many different interleavings!
- How to keep it understandable / verifiable?
 - avoid combinatorial explosions!
 - reunite different paths as soon as possible
 - specify unique system-wide synchronization-(check)-points

©2007 Dept Computer Science, Australian National University

Summary: Non-Determinism

- **Selective synchronization**
 - Selective accepts
 - Selective calls
 - Indeterminism in message based synchronization
- **General Non-Determinism in Concurrent Systems**