Ada – select statement

Select - termination

- tasks often contain infinite loops to allow them to serve an arbitrary number of calls in succession
- control cannot leave a task's parent until its tasks terminate, need a way to know when it should terminate
- it is done by a *terminate alternative* in a selective wait

The task terminates when:

- at least one terminate alternative is open, and
- there are no pending calls to its entries, and
- all other tasks of the same parent are in the same state (or already terminated), and
- the task's parent has completed (i.e. has run out of statements to execute).

- First two conditions ensure that the task can stop, it is ready for it.
- The other two conditions ensure that the task has no negative effects on other parts as there are no further calls that might change its state

Select - terminate

```
task type Terminating_Buffer_Task_Type is
    entry Insert (An_Item : in  Item);
    entry Remove (An_Item : out Item);
  end Terminating_Buffer_Task_Type;
  ...
  task body Terminating_Buffer_Task_Type
is
    Datum : Item;
  begin
    loop
      select
        accept Insert (An_Item : in  Item) do
          Datum := An_Item;
        end Insert;
      or
        terminate;
      end select;
      select
        accept Remove (An_Item : out Item)
        do
          An_Item := Datum;
        end Remove;
      or
        terminate;
      end select;
    end loop;
  end Terminating_Buffer_Task_Type;
```

Select - delay

- Relative delay: delay Duration_Expression;
- Absolute delay: delay until Time_Expression;
- Duration is a special Float type, Time is provided by Ada.Calendar
- delay alternative in a selective wait statement allows a task to no more accept calls after a maximum delay in achieving rendezvous with any client

```
task Resource_Lender is
    entry Get_Loan (Period : in Duration);
    entry Give_Back;
 end Resource_Lender;
 ...
 task body Resource_Lender is
    Period_Of_Loan : Duration;
 begin
    loop
   ......
   end loop;
 end Resource_Lender;
```

Select - delay

```
select
    accept Get_Loan (Period : in Duration) do
           Period_Of_Loan := Period;
     end Get_Loan;
        select
              accept Give_Back;
              or
              delay Period_Of_Loan;
              Log_Error ("Borrower did not give up loan soon enough.");
        end select;
  or
     terminate;
end select;
```

Select - until

```
task body Current_Position is
begin  ...
  loop
    select
      accept Request_New_Coordinates (X :   out Integer;  Y :   out Integer) do
        -- copy coordinates to parameters   ...
      end Request_New_Coordinates;
    or
      delay until Time_To_Execute;
    end select;

    Time_To_Execute := Time_To_Execute + Period;
    -- Read Sensors and execute coordinate transformations
  end loop; ...
end Current_Position;
```

Time_To_Execute is of type Ada.Calendar.Time and can get value by Ada.Calendar.Clock   which provides system time

Select - else

- else alternative in a selective wait statement allows a task to not accept calls where the rendezvous can not be achieved immediately
- the receiver task wants immediate rendezvous, which means the caller has to be executing the entry point call statement in the very same moment, or the call has to be already in the called entry's queue
- the task is not waiting for communication in this case, continues with the else part
- if any delay might appear, then the **or delay** version has to be used

Select - else

```
task body Buffer_Messages is   …
begin …
  loop
    delay until Time_To_Execute;
    select
      accept Get_New_Message (Message : in     String) do
        -- copy message to parameters …
      end Get_New_Message;
    else  -- don't wait for rendezvous
      -- perform built in test Functions  …
    end select;
    Time_To_Execute := Time_To_Execute + Period;
  end loop;   ..
end Buffer_Messages;
```

Select – timeout

- A *timed entry call* lets a client specify a maximum delay before achieving rendezvous, failing which the attempted entry call is withdrawn and an alternative sequence of statements is executed.
- To time out the *functionality* provided by a task, two distinct entries are needed: one to pass in arguments, and one to collect the result. Timing out on rendezvous with the latter achieves the desired effect.

Select – timed call or delay

- Timed entry calls
- An entry call can be made waiting a well defined time interval, so that it is withdrawn if the rendezvous is not achieved after the defined time. This uses the select statement notation with an **or delay** part, the construct is:

```
select
   Callee.Rendezvous;
 or
   delay 1.0;
 end select;
```

Select – timed call or delay

```
task Password_Server is
   entry Check (User, Pass : in String; Valid : out Boolean);
   entry Set (User, Pass : in  String);
 end Password_Server;
 ...
 User_Name, Password : String (1 .. 8);
 ...
 Put ("Please give your new password:");
 Get_Line (Password);
 select
   Password_Server.Set (User_Name, Password);
   Put_Line ("Done");
 or
   delay 10.0;
   Put_Line ("The system is busy now, please try again later.");
 end select;
```

Select - else

- Conditional entry calls
- An entry call can be made conditional, so that it is withdrawn if the rendezvous is not immediately achieved. This uses the select statement notation with an **else** part, the construct is:

```
select
   Callee.Rendezvous;
 else
   Do_something_else;
 end select;
```

Select – or delay 0.0

- Conditional entry calls
- the attempt to start the rendezvous may take some time, especially if the callee is on another processor, so the *delay 0.0;* may expire although the callee would be able to accept the rendezvous, whereas the *else* construct is safe.
- Seems equivalent to or delay 0.0 (busy waiting)

```
select
   Callee.Rendezvous;
 or
   delay 0.0;
   Do_something_else;
 end select;
```