Broadcast Experiment using Ada

This programming experiment provides the ability for a single producer task to broadcast messages to a multiplicity of consumer tasks.

This experiment appears to succeed but needs some review.

The experiment consists of a single package and a test procedure for that package.

The package name is Broadcast. Following is the package spec:

--

-- Broadcast objects

--

-- This package creates an vector of pointers to a generic protected object

-- As each consumer registers to receive the message type used to

-- instantiate the generic protected object the a new protected

-- object is created and the access value of that object is passed

-- back to the consumer. The access value is also appended to a vector

-- of access values so that the producer will be able to send data to

-- each protected object accessed in the vector.

--

generic

type Element\_Type is private;

package Broadcast is

-- Consumer calls the Read entry for its registered buffer object.

-- Producer calls the Broadcast procedure to send the data to all

-- registered consumers.

--

-- Buffer is a single element buffer. The consumer will always read

-- the most recent value read to Buffer but will not read

-- the same message instance more than once.

protected type Buffer is

procedure Write (Item : in Element\_Type);

entry Read (Item : out Element\_Type);

private

Message : Element\_Type;

Is\_New : Boolean := False;

end Buffer;

type Buffer\_Access is access Buffer;

-- Register function dynamically allocates a Buffer object,

-- adds the object access to an internal vector and returns

-- the object access to the consumer registering to receive

-- the data.

function register return Buffer\_Access;

-- Broadcast procedure iterates through the internal vector of

-- buffer\_access calling the write procedure for each element in

-- the vector.

procedure Broadcast ( Item : in Element\_Type);

end Broadcast;

The package spec only exposes the minimum details about the workings of broadcast to the producer and consumer tasks using this package. The buffer used in this experiment is a single element buffer. A consumer cannot read a message before any message has been written to its instance of the buffer. Neither can a consumer read the same data from the buffer more than once. It is possible for a slow consumer to miss a message that is overwritten by the producer before the consumer gets around to reading its buffer. The most recent data written by the producer will always be available to every registered consumer.

The Broadcast package body is:

pragma Ada\_2012;

with Ada.Containers.Vectors;

package body Broadcast is

package Buffer\_Vector is new Ada.Containers.Vectors(Index\_Type => Positive,

Element\_Type => Buffer\_Access );

use Buffer\_Vector;

Channel : Vector := Empty\_Vector;

------------

-- Buffer --

------------

protected body Buffer is

-----------

-- Write --

-----------

procedure Write (Item : in Element\_Type) is

begin

Message := Item;

Is\_New := True;

end Write;

----------

-- Read --

----------

entry Read (Item : out Element\_Type) when Is\_New is

begin

Item := Message;

Is\_New := False;

end Read;

end Buffer;

--------------

-- register --

--------------

function register return Buffer\_Access is

Temp : Buffer\_Access := new Buffer;

begin

Channel.Append(Temp);

return Temp;

end register;

---------------

-- Broadcast --

---------------

procedure Broadcast (Item : in Element\_Type) is

begin

for receiver of Channel loop

receiver.write(Item);

end loop;

end Broadcast;

end Broadcast;

The package body instantiates the package Ada.Containers.Vectors for Element\_Type. A single instance of that Vector type is created in the package and is invisible to calling tasks. That instance of Vector is named Channel. Channel is a Vector of Buffer\_Access. Buffer\_Access is an access type declared in the package spec that references instances of Buffer.

The register function registers a consumer with the producer. The producer is ignorant of which tasks register to consume its data. The register function dynamically allocates a new Buffer object, appends its Buffer\_Access reference to the vector Channel and returns the Buffer\_Access to the task calling register.

The Broadcast procedure writes the value passed to it as the parameter Item to each element of the Vector Channel.

The test procedure for the package Broadcast is:

with Ada.Text\_IO; use Ada.Text\_IO;

with Broadcast;

procedure Main is

package Int\_Msgs is new Broadcast(Integer);

use Int\_Msgs;

task type consumer(Id : Positive);

task body consumer is

My\_Buffer : constant Buffer\_Access := Register;

Value : Integer;

begin

Put\_Line ("Consumer" & Id'Image & " starting.");

loop

My\_Buffer.Read(Value);

exit when Value = Integer'First;

Put\_Line("Consumer" & Id'Image & " read:" & Value'Image);

end loop;

Put\_Line ("Consumer" & Id'Image & " terminating.");

end consumer;

C1 : consumer(1);

C2 : consumer(2);

C3 : consumer(3);

begin

-- main acts as producer in this test

for I in 1 .. 10 loop

Put\_Line("Producer writing value" & I'Image);

Int\_msgs.Broadcast(I);

delay 0.01;

end loop;

Int\_msgs.Broadcast(Integer'First);

Put\_Line ("Producer terminating.");

end Main;

This test communicates simple integer values from the producer to some number of consumers. The package Broadcast is therefore instantiated with the type Integer and that instantiation is called Int\_Msgs.

A task type named consumer is created.

The task body for consumer creates the local constant named My\_Buffer which is initialized by calling the register function and a local integer variable named Value. The consumer outputs a message saying the consumer instance has started, reads values from the instance of Buffer referenced by My\_Buffer, outputs the value read and terminates when the value read is Integer’First.

Three instances of consumer are created. Each instance automatically registers to read messages written by the producer.

The executable portion of the main procedure acts as the producer task. It iterates through integers in the range 1 through 10, broadcasting each value and then delaying for 0.01 seconds to ensure the consumers have time to read their values. After broadcasting all 10 values the producer broadcasts the value Integer’First, indicating the end of data and signaling the consumer instances to terminate.