`CHAN`nels to deliver memory? `MOBILE` structures and `ALT`ing over memory?

Øyvind TEIG
Kongsberg Maritime Ship Systems, Ship Control (KMSS-SC)
7005 Trondheim, Norway
oyvind.teig@kmss.no

Abstract. Memory objects are assigned to processes over a `CHAN`nel like construct. This way one can wait for an object indefinitely, or with timeout in an `ALT` construct - coexisting with `CHAN`nel inputs. The run-time `SYSTEM` will handle requests. Alternatively, a user memory handler process may use the underlying `SYSTEM` and serve other clients. Occam 2 is used as catalyst language.

1 - Introduction

This note was directly inspired by Barnes and Welch, "Mobile Data Types for Communicating Processes" [1], where the concept of copy by ownership moving between concurrent processes is bound into the occam 2 language.

The paper introduces "the basic idea to make dynamic memory allocation not break scheduling independence for parallel (occam-like) systems" (to quote one of the referees). It introduces a way to handle memory allocation failures in these systems. In f.ex. limited memory embedded devices, the idea suggested opens a possibility to dynamically allocate memory, not by a go/no-go *malloc*, but by treating memory allocation as a blocking synchronisation primitive, which may coexist with channel inputs in ALT statements. The memory "delivered" by the run-time system may be of any type, specified by a system module.

This note tries to "think aloud". It is not part of any ongoing research, contains ideas only, collected by a software engineer working in industry.

Because this paper consists of ideas only, as input from industry to academia or toolmakers, it has not been a goal to suggest usable examples. Also, how the underlying run-time system would function is not discussed. This also goes for the liveliness properties of the run-time system, as well as liveliness properties of occam programs using the concept. So, whether f.ex. processes can deadlock blocking for memory, is not handled. The individual elements in the list below have on purpose been presented in a rather terse form. The good thing is that page count is low!

2 - Notes

Let us just jump on the ideas by playing with imaginative occam examples (this is *not* occam 2):

a)
```
MEM MyIntrinsicMem
  CASE
    STACK
    HEAP
    PLACED; address; accessRights
    ROM; address
    FLASH; address; accessRights
    VIRTUAL
    EXCEPTION; type -- exceptionally handled
:
```

Just like **CHAN OF aPROTOCOL**, we introduce a similar **SYSTEM OF aMEM**. Above, *MyIntrinsicMem* is a name defined by the user. In the example MEM, STACK, HEAP etc. are new keywords which are understood by a configurer and run-time system. Maybe their properties, like address and size could be defined above, or maybe a separate configuration language is needed. This probably depends on how often a MEM is used in the program, one time or scattered throughout.

b)
```
SYSTEM OF MyIntrinsicMem aMEM:

MOBILE THINGa buffera: -- A data type with SIZE=512
MOBILE THINGb bufferb: -- A data type with SIZE=64

THINGc        bufferc: -- Static
```

The SYSTEM keyword is inspired from Modula-2, where features of the "system" were defined within the SYSTEM module, like the size of an integer and how *coroutines* should be started and synchronized. In Modula-2, this module was built into the compiler "because some of the objects it defines cannot be expressed in the Modula-2 language" [2]. The MOBILE keyword is taken from [1] and informs us that the data may be dynamically allocated in some way, and ownership of data defined may be passed around.

c)
```
aMEM[HEAP]             ? buffera     -- blocks for object from HEAP
aMEM[HEAP,STACK]       ? buffera     -- blocks for object from HEAP or STACK
aMEM[PLACED(#1000,RW)] ? dualPortMem -- blocks for object from dual-port memory
aMEM[]                 ? buffera     -- blocks for object from any segment(?)
```

The examples above show how different kinds of memory may be assigned for memory objects. In effect, we have a parameterised *new* operator.

d)
```
aMEM[HEAP] ? bufferc -- blocks for static usage(?)
```

It may be a good idea also to let static data come into presence by this mechanism, an example is shown above. (This may pre-empt the sub-title of this note..) The compiler would be able to see how an object is supposed to come into being.

e)
```
aMEM[HEAP] ? buffera AFTER time -- ILLEGAL
```

Above, it would give up after *time* if memory did not become available. This is an exception from CHANnel syntax - it cannot not be legal, since we do not have a mechanism to handle timeout failure.

```
f) ALT
     (NOT needsBuffer) & aCHAN ? someData
       ... Process, set needsBuffer
     (needsBuffer) & aMEM[HEAP] ? buffera
       SEQ
         ... Use buffera
         ... Send off if appropriate
     (needsBuffer) & clock ? AFTER timeout
       ...  Handle timeout
```

However, we can always time out in an ALT. This server receives *someData* on *aCHAN*, processes it, but needs *buffera* in order to do some more interesting things. When, or if, *buffera* has been received, it processes it and sends it on to another process, in which case a new reclaim later on will be recognised by the runtime SYSTEM as a proper request. If it decides *not* to send off, the runtime SYSTEM will, on next reclaim, see that it already has a buffer, and pass on the privileges it already has.

```
g) ALT
     aMEM[] ? CASE
       STACK; buffera
         ...
       HEAP; buffera
         ...
```

Above, we just want space for *buffera*, and we do not care from where. If neither STACK nor HEAP is available, we have a STOP situation.

```
h) ALT
    aMEM[PLACED(#2000,RW)] ? CASE
       PLACED; dualPortMem
         ... Do this
       EXCEPTION; type
         .. Handle it
```

Above, we want to serve a dual port memory which resides on a plug-in card. If the card is not present, we receive an EXCEPTION instead. If we decided that we wanted to be signalled whenever a card was inserted, we could just drop the EXCEPTION handling, and the runtime SYSTEM would signal us in due course.

```
i) ALT
     [2]ab IS [buffera,bufferb]: -- or RETYPES?
     PRI ALT i = 0 FOR SIZE ab
       aMEM[HEAP] ? ab[i]
         SEQ
           ... Now we have the largest buffer available of the two
           ... Inform client which buffer size I have
```

The server above gets the largest of the two buffers, since *buffera*, which is largest and has been assigned highest ALT priority, is indexed as [0].

```
j) PROC MEMHandler ([]SYSTEM OF MyIntrinsicMem aMEMS)
     WHILE TRUE
       ALT i=0 FOR SIZE aMEMS
         aMEMS[i][] ? CASE
           .. Process request and send out access right
   :
```

Above, we have inserted a local *MEMHandler* between our processes and SYSTEM. *MEMHandler* itself is able to communicate with SYSTEM directly. It would perhaps be most natural to let *MEMHandler* reply over the same `SYSTEM` "channel" on which the request arrived. By studying the examples above, we understand this has to be so. Our servers request some type of memory (out) and receive some kind of response (in) without specifically using uni-directional mechanisms. An implicit bi-directional scheme is instead suggested.

k) How this dynamic scheme "plugs into" the occam OO-like suggestion described in [3] , and into the fuller MOBILE concept of [1] remains to be seen. All three concepts should break no occam (or extended occam) laws. (Late addition: [3] and an ancestor of [1] are actually present in this very proceeding.)

Acknowledgements

I must thank Peter Welch for sending me a draft version of [1] half a year ago, and quietly listening on some of the thoughts above at that stage. When I recently asked him how it went with [1], he sent it to me just prior to the PDPTA conference. This gave me time to write this note for the CPA 2001 deadline.

References

[1] F.R.M. Barnes and P.H.Welch, "Mobile Data Types for Communicating Processes", CSREA Press, June 2001. The 2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001)
[2] Logitech M2, Modula-2 Language Reference, Multiscope Inc, Dec.1991, page 273
[3] Øyvind Teig, "From safe concurrent processes to process-classes? PLUSSING new code by ROLLING out and compile?", submitted to CPA 2001.

Øyvind Teig is Senior Development Engineer at *Kongsberg Maritime Ship Systems, Ship Control*. He has worked with embedded systems for 25 years, and is especially interested in real-time language issues. See http://home.no.net/oyvteig/ for publications.