ERCIM Workshop on Dependable Software Intensive Embedded systems
In cooperation with EUROMICRO 2005
Porto, Portrugal

From message queue to ready queue

Case study of a small, dependable synchronous blocking channels API
"*Ship & forget* rather than *send & forget*"

Øyvind Teig
Autronica Fire and Security, Trondheim
(A UTC Fire and Security company)
http:\\home.no.net\oyvteig

18 pages

Abstract

- CSP style synchronous interprocess communication
- on top of a run-time system supporting SDL asynchronous messaging
- Unidirectional, blocking channels are supplied
- Benefits are
  - no runtime system message buffer overflow
  - "Access control" architectural design
- A pattern to avoid deadlocks is provided
- The message buffer is obsoleted, and a ready-queue-only could be asked for.
- May be formally verified with the CSP process algebra.

1. Introduction

2. SDL and CSP

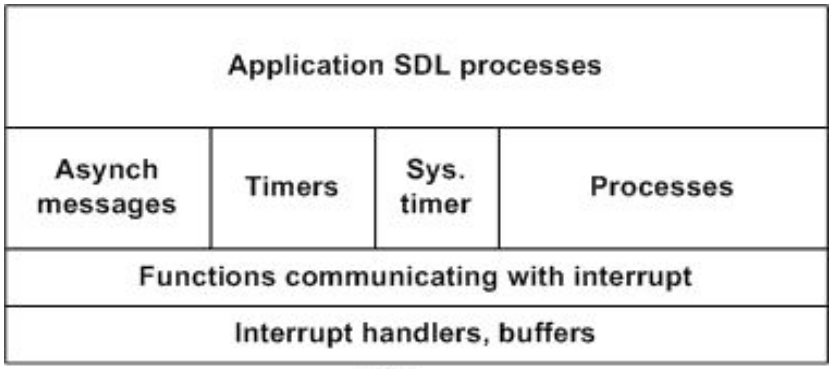
3. Blocking

4. Access control of other processes

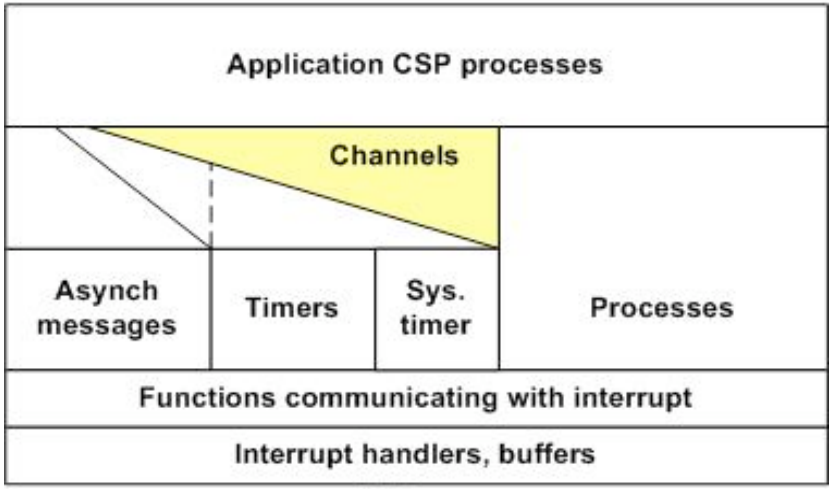
….

All these points will be implicitly covered in the next pages

5. The layered architecture



SDL

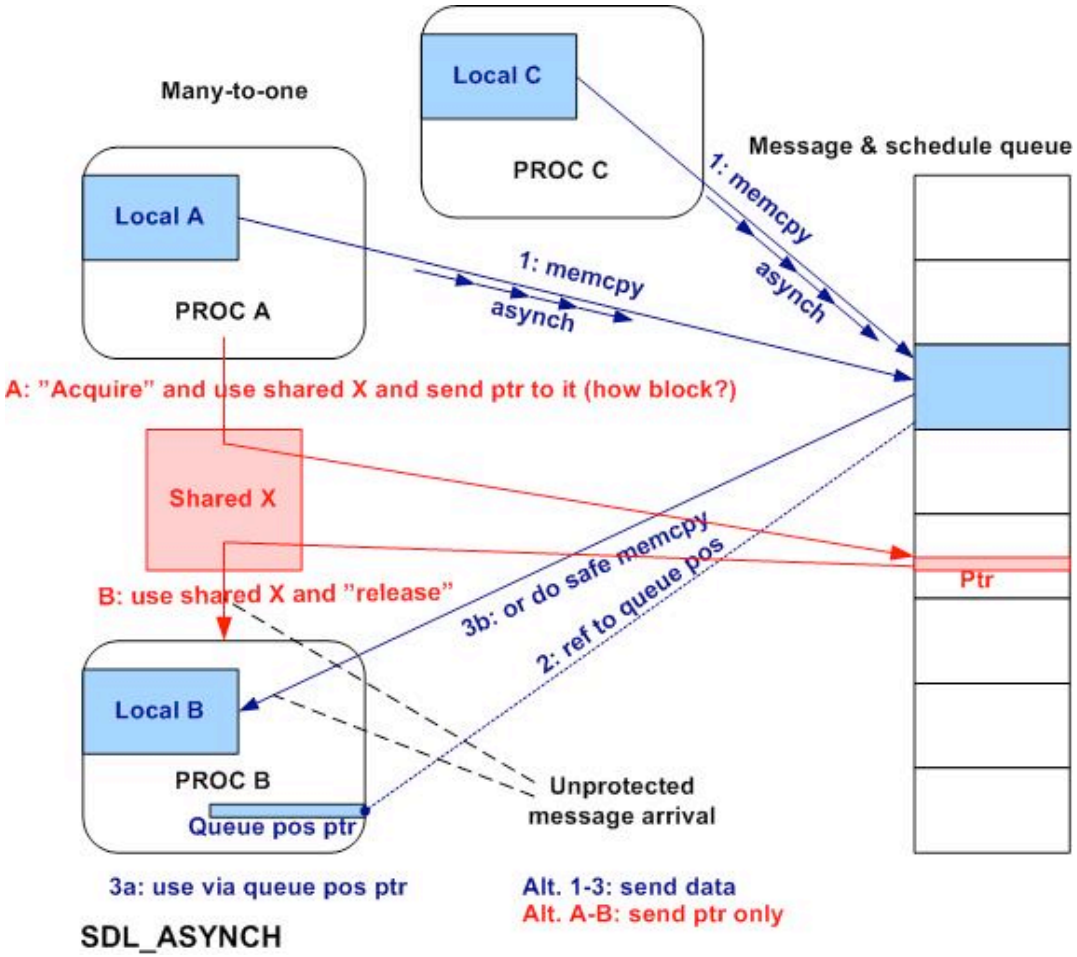


CSP

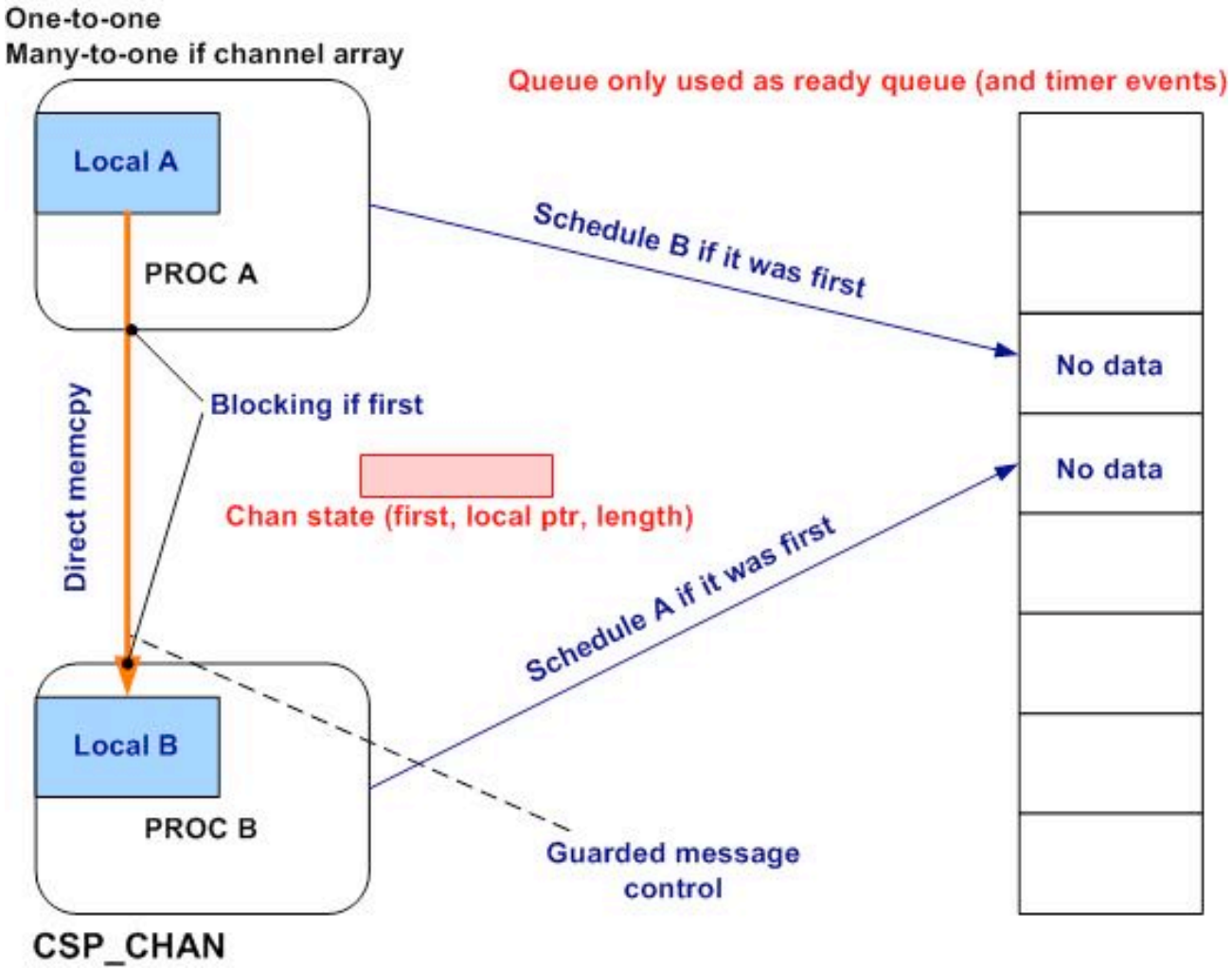
6. The channels C API abstraction

```
#define CHAN_INIT_F
        (CHAN,SENDER,RECEIVER,ALTTAKEN)
#define CHAN_IN_F
        (CHAN,DATA,EVENT)
#define CHAN_IN_VARLEN_F
        (CHAN,LEN,DATA,EVENT)
#define ALT_CHAN_IN_F
        (GUARD,CHAN,DATA,EVENT,ALTTAKEN)
#define ALT_CHAN_IN_VARLEN_F
        (GUARD,CHAN,LEN,DATA,EVENT,ALTTAKEN
#define ALT_CHAN_IN_ASYNC_SIGNAL_F
        (GUARD,CHAN,EVENT,ALTTAKEN)
#define CHAN_OUT_F
        (CHAN,DATA,EVENT)
#define CHAN_OUT_VARLEN_F
        (CHAN,LEN,DATA,EVENT)
#define CHAN_OUT_ASYNC_SIGNAL_F
        (CHAN,RESCHEDULEME)
#define CHAN_IN_ASYNC_SIGNAL_F
        (CHAN,EVENT)
#define ALT_TIMER_IN_F
        (GUARD,TIME,UNIT,EVENT,ALTSTATE,ALTTAKEN)
#define FSM_RESCHEDULE_F
        (EVENT)
```

7. Semantics of asynchronous messages



8. Semantics of synchronous channels

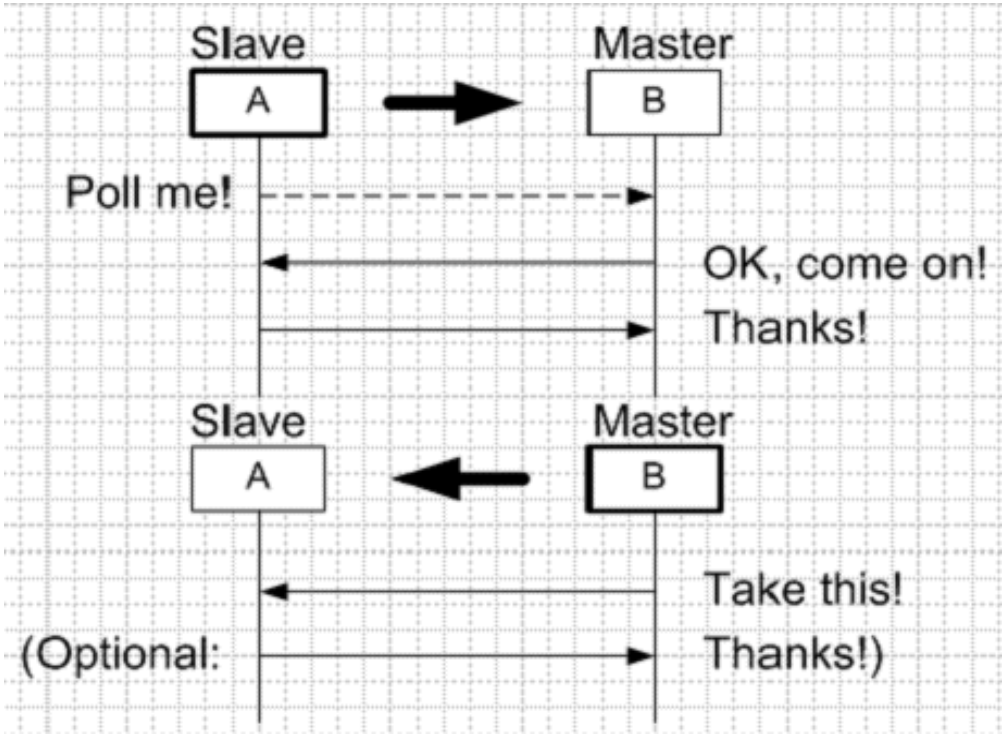


9. “From message queue to ready queue”

- Explained with figures above, i.e.:
- Message queue(s) not needed any more
- Ready/scheduling queue(s) all we need
- But we stuck to what we had!
- Of course, timer queue(s) needed

10. Deadlock avoidance

- Observe that we cannot deadlock on erroneous use of semaphores, we have none
- Only on mutual waiting for each other in a cycle
- The pattern above *requires* Master to treat Slave-only after "Poll me!", no other process



- (An asynch channel equals an overwritebuffer composite process with size 1 communicating over synchronous channels)

11. Coding examples (1)

```
P_BS100_Sim (void)
{
    switch (ContextPtr->State)
    {
        ...  ST_INITIALIZING_A (Initialize)
        ...  ST_STATE_IN_ALT_030_032_104_A
        ...  --> ST_STATE_IN_ALT_030_032_104_END_A
        ...  ST_STATE_OUT_031_A
        ...  ST_STATE_OUT_110_A
        ...  ST_STATE_OUT_END_A
        ...  ST_STATE_STOPPED_A, (crash)
        ...  default, (crash)
    }
    ...  Common action: get more to do from workpool
}
```

- Non-preemptive "return" to scheduler gives *one common stack*

4. Access control of other processes (more)

- In addition to the "channel switch" mentioned we have:

- A process needs to obey the protocol semantics, it does not need to know the semantics of the other processes' internal behaviour
- Only have to look at *this* process to understand what service it offers its environment. That service is independent of its environment - its behaviour is the same anywhere
- This is WYSIWYG semantics
- Processes become dependable software components: no unpublished interactions (side-effects) between CSP parallel processes
- It also shows the compositional semantics of CSP
- Erroneous use of semaphores - not WYSIWYG!

11. Coding examples (2)

```
case ST_STATE_IN_ALT_033_093_A:
{
    bool_a alt_taken = FALSE;
    g_ALT_AL_ComP_Driver = CHAN_ALT_ENABLED_ON_A;

    ALT_CHAN_IN_F  (ContextPtr->Guard_033_093, g_chan_033,
                    ContextPtr->ALCP_033i_093i,
                    S_EVENT_ALT_033_093_A, &alt_taken);
    ALT_CHAN_IN_F  (ContextPtr->Guard_033_093, g_chan_093,
                    ContextPtr->ALCP_033i_093i,
                    S_EVENT_ALT_033_093_A, &alt_taken);
    ALT_TIMER_IN_F (ContextPtr->Guard_Timer_ENTX, MS_TIMEOUT_A,
                    TU_MS_A, S_EVENT_TIMEOUT_A,
                    &g_ALT_AL_ComP_Driver, &alt_taken);

    ContextPtr->State = ST_STATE_IN_ALT_033_093_120_END_A;
    break;
}
```

12. Formal basis of the architecture

- The channel layer API discussed here was modelled on macros used by the code generator of the *SPoC* occam-to-C compiler
- Based on occam, which is a running subset of CSP
- The CSP process algebra "discovered" by this used through occam
- However, the CSP may be used to model and verify any system
- This would be out of reach (expensive), and not very interesting for us (small system and use of known software patterns).
- However, other process algebras, like FSP, analysed with the free LTSA tool, may also be used.
- Modelling asynchronous systems (albeit with finite size buffers, which makes them synchronous when buffers are empty/full) is also possible with Promela and the free SPIN tool

13. Discussion (1)

- "CHAN_CSP" adds about 2 KB of program memory with some IN, ALT and OUT macros used.
- Execution time overhead and *memcpy*, discussion
- Processor cycles for ALT, discussion
- SDL runtime system is about 20 KB
- But even with 128 KB of code space (or, soon 256 KB – nice for an 8 bit machine) and ~~16~~(11) MHz clock, the added well-being of knowing that the system never overflows the message queue or sends unwanted messages into a processes, outweighs the overhead.

13. Discussion (2)

- This author used occam, SPoC and a C CSP library for15 years,
- "Ship & forget" rather than "send and forget"
- Communication states need to be learned
- Complexity and engineers' preferences and background.
- If OO has had its way, the CSP (or the like) also has a way to go.
- Grasping the communicating state machines is individual.
- A channel most probably seems as belonging to OSI *network* (3) or *transport* (4) layer, and certainly not the application layer (7).
- Some programmers learn this methodology easily.
- However, when the communication infrastructure code once has been set up, it tends to stay stable and work.

13. Discussion (3)

- Size of the present "ready queue" is a matter of finding the maximum scheduling incidence volume.
- When maximum has been found, there is no room for further surpises, since the value is a function of the number of channels and processes, not the communication pattern.
- A subset of?) Ada available for microcontrollers of this type
- Java (where CSP libraries are available).
- Or hope that result of ongoing occam research will hit industry some day.
- In the meantime, we could use solutions as the one discussed here, which really is quite dependable, even if it is based on hand-written C.

14. References (1)

- The edit-by-anyone internet based *Wikipedia* dictionary has been used for some essential computer science terms.
- Wikipedia articles often point to more academic sources.
- The last reference (to www.wotug.org) has been added since it is a good starting point for both theory and practice of this field of computer science.
- The reference list (of this "industrial" paper - as opposed to "academic") should be used as hands-on *and* academic starting points, not especially for direct referencing of origins.

14. References (2)

Added references, not in paper:

- http://rmox.net/prelude - Raw Metal occam - an OS for embedded applications
- http://www.transpreter.org - occam on anything