MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Machine Structures Group
Memo No. 8

Memorandum MAC-M-189
September 25, 1964

An Example of Intersphere Communication and Asynchronous Parallel Processing--Typewriter Console Message Handling by Protected Service Routines

J. B. Dennis

The principal purpose of this note is to study an example of communication between processes operating in distinct spheres of protection.[1] We choose the example of a protected service routine[2] that handles message traffic between users' processes and typewriter terminals through a common input/output device and a multi-line controller. This illustration is also a good exercise in programming asynchronous parallel processes[3], and points out some functions that any basic language for system programming should provide. Further, although admittedly oversimplified, the principles outline here could form the model of implementing the typewriter communication function in the second phase MAC system. However, the latter would probably not be feasible unless a method of automatic process assignment were implemented.[4] (Otherwise switching of processes might be too inefficient to be practical at this level.)

Context

We suppose that each user's process (or a process operating on behalf of a user) calls the message handling protected service routine (MHR) by one format of procedure step if it wishes to accept a character from the console assigned to it and a second format if it wishes a character to be printed on the associated console. In either case, operation of a users' process is continued as soon as the MHR has processed the call and is prepared to accept another call from the same process.

The multi-line controller (MLC) is presumed to be commanded by the following six i/o functions.

input

listen $() \rightarrow (n,d)$
enable lsn (n)
disable lsn (n)

output

print $(d) \rightarrow n$
enable prt (n)
disable prt (n)

Here d takes on a range of values that are the character codes of the system plus - in the case of $\langle$print $(d) \rightarrow n\rangle$ - the value ∅ signifying the null character. The value of n is an index that ranges over all lines attached to the MLC. The operation of the MLC is presumed to be completely independent for input and output. First the operation for input is described.

Each line is at any time either enabled or disabled for input. This status is established for line n by the $\langle$enable lsn (n)$\rangle$ and $\langle$disable lsn (n)$\rangle$ i/o functions. These procedure steps require minimal time for completion. The i/o function $\langle$listen $() \rightarrow (n,d)\rangle$ is completed as soon as there is a character waiting to be read from the MLC for any line enabled for input. Then n is the number of a line and d is the character code just received from line n, where n is selected from all enabled lines with received characters by the MLC through a suitable priority algorithm.

Each line may also be *enabled* or *disabled* for *output* by the <enable prt (n)> and <disable prt (n)> i/o functions. The function <print (d)→ r> is completed as soon as *any* line enabled for output is prepared to accept a new character for transmission. The value *n* of <print> is the line number of one such line selected by a priority discipline. The argument (*d*) of print is the next character code to be transmitted on the line selected by the *previous* <print> function.

Named Objects and Attachment

Following the ideas outlined in our previous work, the following entities may be referenced by name.

1) segments of memory
2) i/o functions
3) spheres of protection

By the process of *attachment* we mean the association of a name with an *attachment tag* a0, a1 ..., so that effective reference to the entity is permitted by a particular process. Sucessful attachment of a name implies that the name is valid in the sphere of protection of the process and associates with the attachment tag supplementary indicators that specify the use to which the name may be put, and additional data descriptive of the named entity. Unsuccessful attachment results in a sphere violation fault.

In the sphere of protection of the MHR there will be the following named entities.

1) a procedure segment containing the coding of the MHR.
2) a data segment containing status and assignment information.
3) the i/o functions

    <print>
    <listen>
    <enable lsn> , <disable lsn>
    <enable prt> , <disable prt>

4) The names of the spheres of protection of processes calling the MHR (for the duration of the calls - at least for purposes of this note).

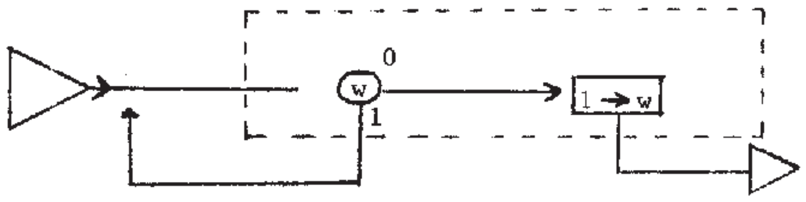
Special Procedure Steps

To implement the MHR and define its calling sequences we must define a number of procedure steps not familiar in well known programming languages. These are concerned with interaction of parallel processes and with communication across sphere of protection boundaries.

1) fork l, a — Initiate a parallel process within the same sphere of protection starting at location with address word a/l. The state word of the new process is identical to the state word of its parent at the time of the fork (except for the procedure address, of course).

2) quit — Terminate the current process.

3) lock (w) — The lock indicator w takes on values 0 and 1. <lock> performs the following operations



where the steps in the box are accomplished within a single memory cycle to prevent race conditions among distinct processors.

4) unlock (w)

is equivalent to

$\langle 0 \rightarrow w \rangle$

5) attach S, a

attach name S to current process with attachment tag a.

6) enter sphere a,l,b

Attachment tag a is associated with a segment name S and a descriptor that includes the name P of a sphere of protection. This procedure step causes the following action:

1) the state word of the current process is entered in the <u>dead process</u> list and a pointer ρ to this process entry is created.

2) The sphere name of the current process is associated with attachment tag b as a <u>sphere return name</u> and the pointer ρ is entered in its descriptor.

3) The descriptor of S is modified to describe S as a procedure segment and include information to make reference effective.

4) A process is initiated in sphere P at word address a/1.

7) <u>set</u> <u>X</u> (a,w)	These steps are used to examine and modify the contents of the old state word of a process that corssed
8) <u>get</u> <u>X</u> (a)→w	a sphere boundary. X designates the intended component of the state word and <u>a</u> is an attachment tag that is associated with a sphere return name.

In our example we use A in place of X to specify the accumulator of the process state word.

9) sphere return (a)	<u>a</u> is an attachment tag associated with a sphere return name Q. The current process is continued as the process whose state word is designated by the descriptor of Q.

The <u>coding</u>

The calling sequences for the MHR are as follows where MHR is the name of the procedure segment continuing the MHR coding.

<u>to listen</u>:

```
    attach MHR, a1
    enter sphere a1, lsn, a2
```

```
to print
     attach MHR, a1
     enter sphere a1, prt, a2
```

The coding of the MHR for input and output is presented in Figure 1 and Figure 2, respectively. The starred quantities i*, b* and S* are to be treated as private to processes in this coding, that is they must be represented in processor registers and must never be stored in memory. The alternative requires either private data segments for distinct callers, a pushdown list, or other means of distinguishing the different appearances of these quantities. The result would be a more confusing representation of the MHR. The function map() is used to map the caller identifiers (which are sphere names) into line numbers, and hence embodies the assignment of consoles to users. The mechanism for establishing map() has been omitted. A table located in the MHR data segment contains an entry for each line that is used to synchronize the processes that interact with the MLC, with the users' processes communicating with MHR. This entry for the ith line contains the following items:

1)	lsn wait [i]	a binary indicator set to one when a user's process is waiting for a character to be typed in.
2)	lsn done [i]	a binary indicator set to one when an input character is available to a user's process.
3)	lsn ch [i]	the input character being transferred.
4)	lsn lk [i]	the lock indicator for controlling access to the above items by asynchronous processes.
5)	prt wait [i]	serve the analogous functions for output.
6)	prt done [i]	
7)	prt ch [i]	
8)	prt lk [i]	

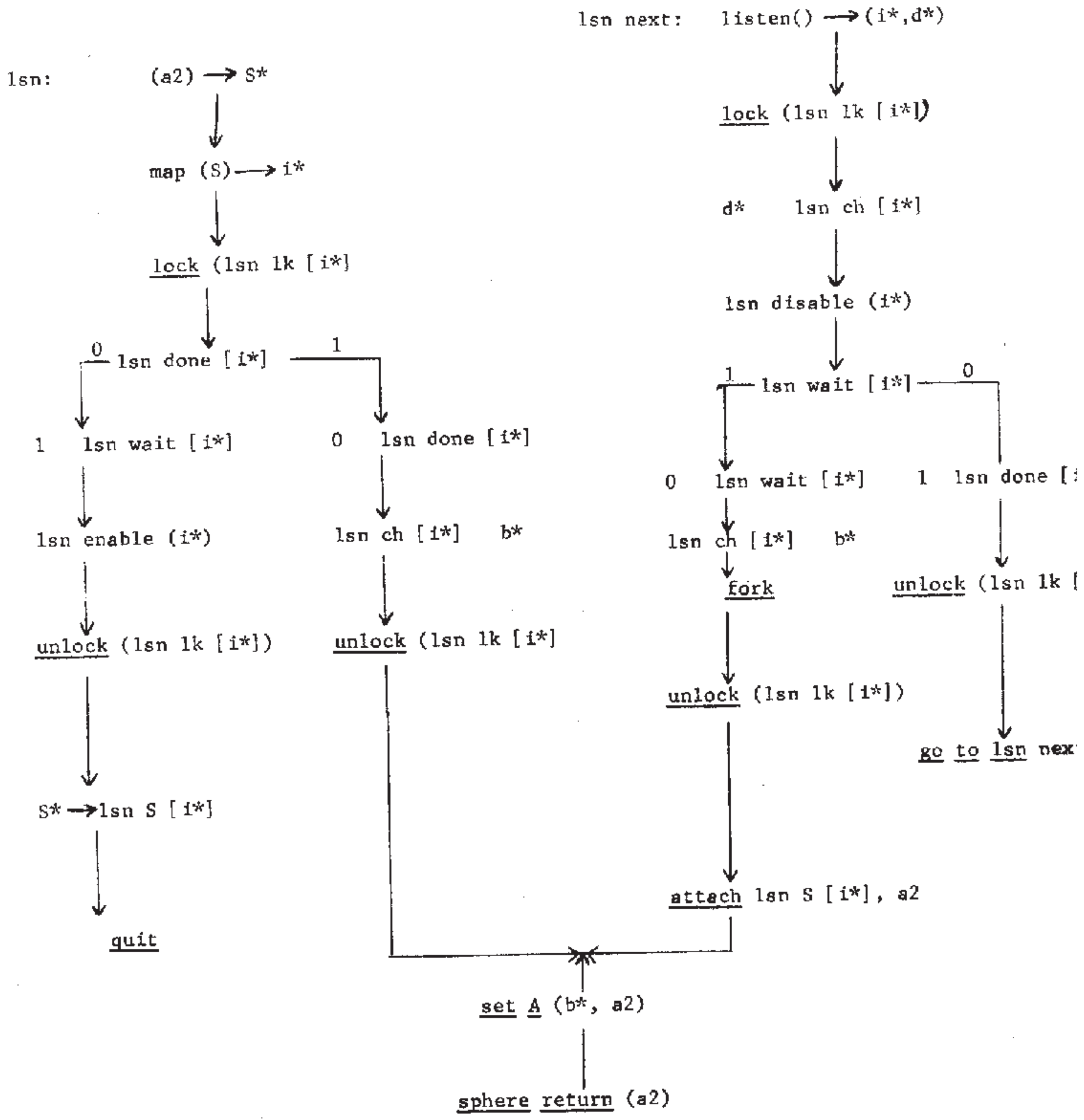


Figure 2. Protected Service Routine for Accepting Console Messages.

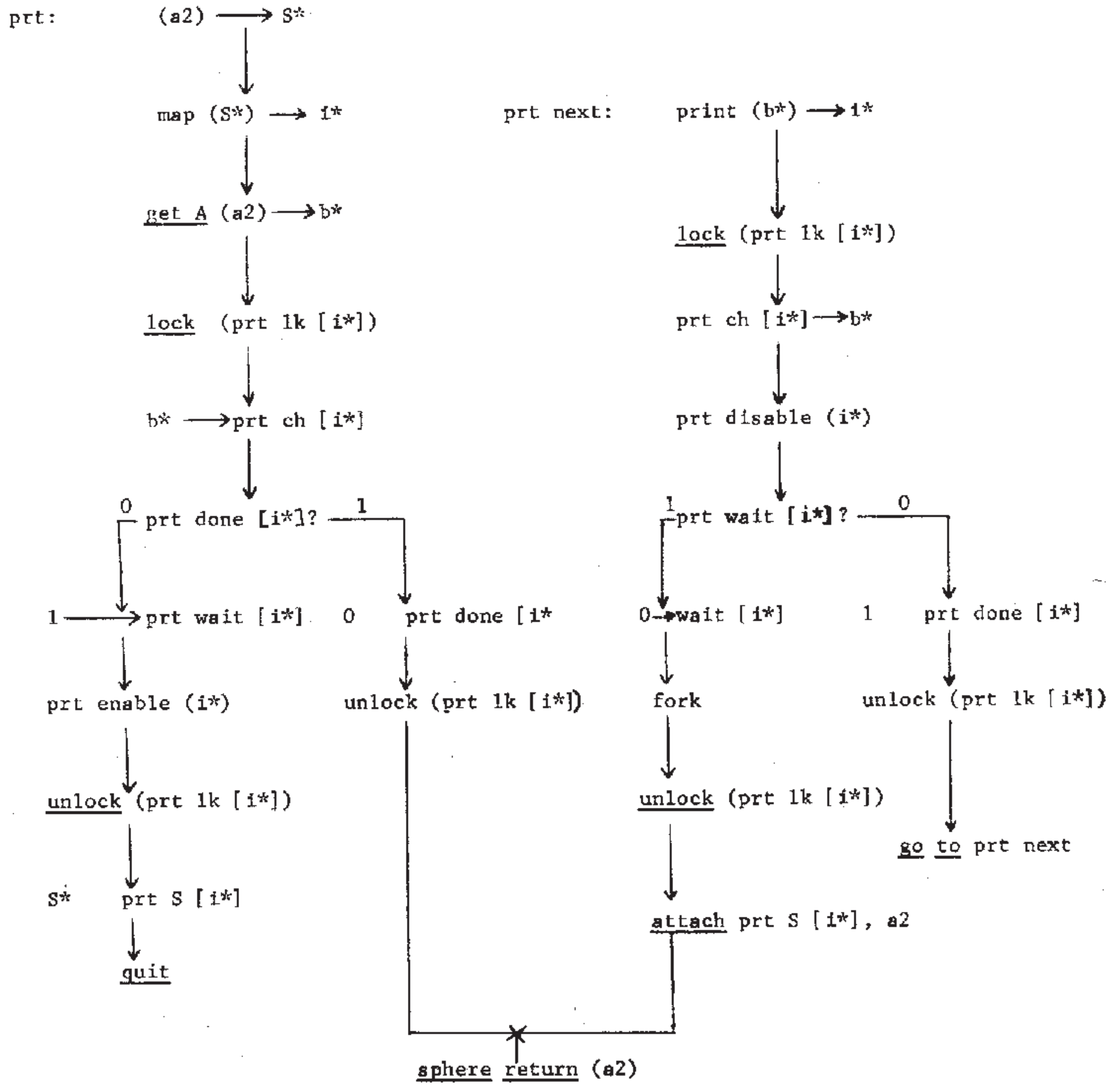


Figure 1. Protected Service Routine for Transmitting Console Messages.

REFERENCES

1. J.B. Dennis, "Program structure in a multi-access computer", Project MAC Technical Report MAC-TR-11, (M.I.T. October 1964).

2. J.B. Dennis, and E.L. Glaser, "The structure of on-line information processing systems", Project MAC Memo MAC-M-181, (M.I.T. August 1964). (A paper submitted to the Second Congress on the Information System Sciences.)

3. H. Witzenhausen, "A note on assynchronous parallel processing", Project MAC Memo-M-187, (M.I.T. July 1964).

4. J.B. Dennis, "Automatic scheduling of priority processes", Project MAC Memo MAC-M-188, (M.I.T. October 1964).