Avoiding real-time requirements: a case study

We consider a distributed system consisting of two components connected by means of a medium of communication. The one component, called Source, sends so-called messages to the other component, which is a computer. This computer is equipped with a 1-place buffer —that is, capable of holding 1 message— in which arriving messages are stored. The computer runs, among other programs, a dedicated program called Sink. Process Sink removes messages from the buffer and "consumes" them. We assume that Sink is appropriately synchronised with the buffer —for example, by means of interrupts— to avoid attempts to remove messages from an empty buffer. If, on the other hand, the next message arrives in the buffer before the previous one has been removed, one of the two messages is lost and this is considered as an error.

This error is avoided if we impose upon process Sink a real-time requirement, stating that Sink is so fast that each message is removed from the buffer before the next one arrives. This requirement can be formulated more precisely as follows.

We assume arrival of a message in the buffer to be an atomic event requiring no time and we assume the communication medium to be such that between arrival of any two successive messages at least

T units of time elapse. Now, the real-time requirement imposed upon Sink can be formulated as follows:

R0 : Every message must be removed from the buffer within T units of time after its arrival.

<u>aside</u>: Mind you that this requirement not only involves process Sink proper but also affects the scheduling of processor time in the computer.

□

In this note we study several arrangements designed to relax or avoid requirement R0. Notice, however, that <u>if</u> Source sends messages at a rate of 1 per T units of time then Sink <u>must</u> consume messages with an average rate of 1 per T units of time as well. This is a long-term real-time requirement that can never be avoided.

Whether or not requirement R0 is to be taken seriously depends upon the actual value of T and the time needed by Sink to consume a message. As an extreme example, if consuming 1 message takes 1 second and T = 1 day then R0 will affect the scheduling of processor time only marginally.

<u>Solution 0</u>

The only way to avoid real-time requirements altogether is to adapt the transmission rate of Source to the processing rate of Sink. This requires a form of synchronisation that can be implemented as follows. We assume that the communication medium between Source and the computer allows communication in both directions. After each removal of a message from the buffer, Sink sends a dedicated message, a so-called acknowledgement, to Source. The "meaning" of the arrival at Source of an acknowledgement is that the buffer is empty, so that Source may send its next message without risk of error.

More formally, correct operation is guaranteed if the system maintains the following invariant:

$n \leq 1$, with n = "the number of messages in the buffer" .

Because the system is distributed Source has no access to n ; therefore, we must derive a <u>local</u> invariant for Source that implies $n \leq 1$. With:

ms	=	"the number of messages sent by Source"
ma	=	"the number of messages arrived in the buffer"
mr	=	"the number of messages removed from the buffer"
as	=	"the number of acknowledgements sent by Sink"
ar	=	"the number of acknowledgements received by Source" ,

we have:

$$\begin{array}{ll}
 & n \\
= & \quad \{ \text{ property of buffering } \} \\
 & ma - mr \\
\leq & \quad \{ \text{ property of the communication medium } \} \\
 & ms - mr \\
\leq & \quad \{ \; as \leq mr : \text{ behaviour of Sink } \} \\
 & ms - as \\
\leq & \quad \{ \; ar \leq as : \text{ property of the communication medium } \} \\
 & ms - ar \; .
\end{array}$$

Hence, $n \leq 1$ follows from the local invariance of

P0 : $ms \leq 1 + ar$.

As a matter of fact, the one and only motive for the introduction of the acknowledgements is our need of invariant P0 : variable ar serves as a local approximation of mr. Notice that the term 1 in P0 accounts for the (assumed) initial emptiness of the 1-place buffer.

Although correct, the above protocol is not very efficient. Suppose that, due to geographical separation of Source and Sink, every message (in either direction) arrives D units of time later than the moment at which it was sent. (So, the message "travels" for D time units through the communication medium.) Then, the acknowledgement for a message sent by Source arrives at Source at least $2*D$ time units after sending that message: the maximal transmission rate is thus reduced —assuming $T \leq 2*D$— from $\frac{1}{T}$ to $\frac{1}{2*D}$; if $T \ll D$ this is unacceptable. (For example, for $T = 10^{-3}$ s and $D = 1$ s we get a slow-down by a factor of 2000.)

<u>Solution 1</u>

Ignoring solution 0 we choose a different approach. A simple way to relax requirement R0 is to replace the 1-place buffer by an N-place buffer for some fixed N, $1 \leq N$. Under the assumption of a FIFO discipline for the buffer, the real-time requirement imposed upon Sink now becomes:

R1: Every message must be removed from the buffer within $N*T$ units of time after its arrival.

The larger N is, the weaker R1 will be. Of course, the long-term real-time requirement is not affected, but the short-term requirement can be relaxed as much as desired by choosing N sufficiently large.

The N-place buffer can be implemented entirely by means of dedicated hardware (, such as a so-called DMA channel), which is the preferable solution. Alternatively, the buffer can be implemented by means of the old 1-place buffer and an additional <u>buffering</u>

process in the computer; this process repeatedly removes messages from the 1-place buffer and places them in an (N-1)-place buffer in the computer's store. The old real-time requirement R0 now pertains to the new process; nevertheless, this is an improvement in those cases where the act of moving a message from one buffer to the other is (much) less time-consuming than consumption of a message by Sink.

Solution 2

In this solution we combine the two previous solutions. We use the synchronisation protocol from solution 0 and we replace the 1-place buffer by an N-place buffer. The effect of the latter is that the synchronisation invariant —P0— for Source may be weakened to:

P1: $ms \leq N + ar$,

where we have again assumed the initial emptiness of the buffer. This weaker invariant enables Source to send as many as N messages before the acknowledgement of the first message arrives. Sending these N messages requires $N*T$ units of time, whereas the first acknowledgement arrives (in the best case) after $2*D$ time units. Hence, a necessary condition for message transmission at maximal speed is:

$$N*T \geq 2*D, \text{ i.e.: } N \geq \frac{2*D}{T} .$$

This condition also is sufficient, in the following way: if the condition is satisfied message transmissions are not delayed by the synchronisation protocol; the rate of transmission is now determined by the speeds of Source and Sink, in particular by the slower of the two. Thus, the above formula provides a lower bound for the buffer size such that maximal-speed message transmission is possible. (For example, for $T = 10^{-3}$ s and $D = 1$ s we get $N \geq 2000$.)

This solution is nice for its combination of safety and efficiency. When process Sink is too slow the synchronisation protocol prevents buffer overflow; when Sink is fast enough the buffer prevents slow-down of the transmission due to the protocol. As a result, whether Sink meets its real-time requirements now only affects the performance and not the correctness of the system.

Obviously, this solution cannot be used if

— Source may not be delayed by some additional synchronisation, or

— the communication medium disallows two-way communication.

The moral of this story is that these cases are better avoided: if this is impossible then only solution 1 can be applied.

Eindhoven, 20 may 1993

Rob R. Hoogerwoord
department of mathematics and computing science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven