Copyright Notice

A post-scriptum to EWD501.

Dear Tony!

Monday morning I went to XEROX to have a few copies made of EWD501 and from there via the THE home. At the THE I showed Wim Feijen, what I had written during the weekend, and I discussed with him what I intended to write in the afternoon.

In the afternoon I wrote EWD502 --"On a gauntlet thrown by David Gries"--; when that was completed, Wim came along. He had studied EWD501, and his first remark was, that the procedure cons on page EWD501 - 5 (13 - 7 lines from below) can be simplified, thanks to the initial emptyness of the train temp:

```
proc cons(c: integer):
     do n < c → con:= (con, me); me:= head(temp) od;
     n:= n - c; me:= head(temp)
corp cons;
```

Another observation he made was that when, for instance in the procedure acquire on tup of page EWD501, the second line

nonbusy:= (nonbusy, me); me:= head(nonbusy)

is omitted, it is still correct, but now implements a last-in-first-out strategy. I had these remarks in the back of my mind when I designed the readers and writers monitor and the dischead monitor (see following pages).

Monday evening I was tired --Ria and I went away on the tandem--, Tuesday was my day at the THE. In the morning I had some examinations, in the afternoon we studied EWD501 with the little group and made a first solution to the readers and writers. Tuesday evening I embellished it, and thought about a few linguistic alternatives. This morning I had to write a referee's report, this afternoon I designed the diskhead monitor, and typed both monitors.

It is now early in the evening. Let me describe to you the linguistic alternative I have been thinking about. Up till now we have done as if the monitor only exists after the initialization has been completed. But we could regard the monitor "existing" as soon as the initialization starts, and regard the initialization as performed by an (anonymous) process in monitor state. The one consequence would be that all initializations in the monitors I have written these last days, should end with an additional "me:= nil". That obligati is hardly a recommendation, in contrast, perhaps, to the now created possibility that after initialization the monitor process can place "me" on the shunting yard, thereby remaining available for activities that would be hard to place otherwise.

In the dischead monitor you will see that the sortprocess, that should insert the new requester --placed in qu1-- in the correct position into the train upsweep will fail to do so, when the new requester should be placed at the rear end of upsweep --this "appending is no insertion--. As a result, requests and releases have to begin with

"upsweep:= (upsweep, qu1)"

just to be on the safe side. (When qu1 = nil , the above shunting has no effect.) This could be regarded as ugly. If the monitor itself could sleep on the shunting yard as well, I think that this could be remedied by attaching the monitor at the rear end of upsweep , before the new requester is placed in the correct position. It gives us the possibility to have some activity inserted after the last one, and that, in general, seems a sound and useful facility.

```
readers and writers: monitor:
begin ar, aw: integer;
      readers, writers: train;
      proc startread:
           readers:= (head(writers), readers, me); me:= head(readers);
           do aw ≠ 0 → readers:= (me, readers); me:= nil od;
           ar:= ar + 1; me:= head(readers)
      corp startread;
      proc endread:
           if ar > 0 → ar:= ar - 1; me:= head(writers) fi
      corp endread;
      proc startwrite:
           writers:= (writers, me); me:= head(writers);
           do ar ≠ 0 or aw ≠ 0 → writers:= (me, writers); me:= nil od;
           aw:= 1; me:= nil
      corp startwrite;
      proc endwrite:
           if aw = 1 → aw:= 0; readers:= (readers, head(writers));
                       me:= head(readers)
           fi
      corp endwrite;
      ar:= 0; aw:= 0
end readers and writers
```

This is my version of the readers and the writers, according to your specifications of page 556. (Although I wrote it on Tuesday evening, I should say "our", as the problem was discussed on Tuesday afternoon at the THE with the usual group; particularly the contribution of Wim Feijen should be acknowledg

It has, I think, some charming features. The invariance of

$$ar \geq 0 \text{ and } aw = 0 \text{ or } ar = 0 \text{ and } aw = 1$$

is beautifully maintained, when we remember that the repetitive construct can only terminate with its guard(s) false. (The alternative constructs in endwrite and endread, which may cause abortion, are only there for safety.) The nice thing is that these two guards derived from the invariant relation occur only once! The whole choice of strategy is reflected in the shunting and switching! Isn't that nice?

The way in which in "startread" the presence of a waiting writer prevents new readers to get access, also pleases me. At first it may strike you as a coding trick, but after having played with these trains for a while, it comes quite natural. The way in which "endwrite" gives priority to the readers is also quite nice; at least, I think so.

In programming style, the above is very much different from your approach, in which the continuation after a "wait" can do no harm on account of what has been checked by the other process, that caused the "signal". In such a way one can also get one's programs right, but in principle I think the approach a wrong one: your procedures are logically more intertwined --at least so it seems to me-- and it is therefore a stronger invitation to make logical spaghetti.

The convincing beauty of the above contrast with the program on the next page, where I did the dischead monitor without the scheduled wait, and without the "condname.queue". That was not easy!

```
dischead: monitor
begin headpos, newdest: cylinder;
      direction: (up, down);
      busy: boolean;
      upsweep, downsweep, qu1, qu2: train;
      proc request(dest: cylinder);
           upsweep:= (upsweep, qu1); downsweep:= (downsweep, qu2);
           newdest:= dest;
           if dest > headpos or dest = headpos and direction = up →
                qu2:= (upsweep); qu1:= (head(qu2), me); me:= head(qu1);
                do busy → if newdest ≥ dest → upsweep:= (upsweep, me)
                                               me:= head(qu2)
                          [] newdest < dest → upsweep:= (upsweep, qu1, me, qu2);
                                               me:= nil
                          fi
                od
            [] dest < headpos or dest = headpos and direction = down →
                qu1:= (downsweep); qu2:= (head(qu1), me); me:= head(qu2);
                do busy → if newdest ≤ dest →
                                  downsweep := (downsweep, me); me:= head(qu1)
                          [] newdest > dest →
                                  downsweep:= (downsweep, qu2, me, qu1); me:= nil
                          fi
                od
           fi;
           if headpos < dest → direction:= up
            [] headpos > dest → direction:= down
            [] headpos = dest → skip
           fi;
           headpos:= dest; busy:= true; me:= nil
      corp request;
      proc release; busy:= false;
           if busy → upsweep:= (upsweep, qu1); downsweep:= (downsweep, qu2);
                     if direction = up →
                          downsweep:= (head(upsweep), downsweep);
                          me:= head(downsweep)
                      [] direction = down →
                          upsweep:= (head(downsweep), upsweep);
                          me:= head(upsweep)
                     fi
           fi
      corp release;
      headpos:= 0; direction:= up; busy:= false
end dischead
```

Salvo errore et omissione, the above is a replacement for your dischead monitor on page 555-556. It could be argued that the above could only be programmed on a very warm day with thunderstorms --for you information: it is such a day!--. But it has not the danger of individual starvation when all requests are for the same cylinder! Your "scheduled wait" does not talk about this. Agreed? On account of the above I understand that you yielded to the temptation to introduce the scheduled wait. Note, how in the "release", some shunting avoids the need for "condname.queue". That part of the construction I think quite neat!

Greetings and best wishes! Yours ever

Edsger