EWD657.html

On leaves and nodes: a simplification of EWD653.

I distributed EWD653 aware of "its imperfect and incomplete state"; on the one hand apologizing for problems its unclarity or unconclusive arguments may have caused some of my readers, I am on the other hand glad that I did distribute it, as its exposure to a wider public --in particular to the Tuesday Afternoon Club-- enabled me to reach a uniforming synthesis that, unaided, I had been unable to accomplish.

I refer to the discussion about leaf segments and node segments that starts with the dilemma introduced on the middle of EWD653 - 6 and ends at the top of EWD653 - 8 with: "The question is still open, I think that I have a mild, intuitive preference for the second choice; the reason is probably that then we can come away with one type of shift operations only." The mild preference has disappeared, because a different coding of the information in the descriptors makes a singly memory shift operation suffice.

The discussion in EWD653 is unsatisfactory for several reasons. It rejected two alternatives (EWD653 - 6, bottom) because in the one the global index chains were one element longer than necessary, and in the other the block height increased one more than necessary. It was justly remarked that it was hard to decide how much weight should be given to those (otherwise) correct observations, as neither the "price" of longer global index chains nor the "price" of more deeply nested inner block was very clear. In trying to make the different cost aspects of in particular the deeply nested inner blocks more clear I encountered an until that moment tacit assumption on my part that, thanks to the assumed availability of the memory shift operation, seemed no longer justified.

Consider a procedure with the following declaration structure:

```
proc P: begin var x: int; var u: array of int; .........
            begin var y: int; var v: array of int; ......
            end; .....
        end
corp
```

When I implemented ALGOL 60 in 1960 I had to accommodate all variables in a single stack. At the call of P the stack was extended (in order) with: the variable x , a descriptor du for the array u , and the elements of u itself; upon the inner block entry the stack would be similarly extended with the variable y , a descriptor dv , and the elements of v itself. Because in general the memory requirements for u were unknown at compile time --and so the "distance" in the stack between x and y -- it was decided to give y and v a block height one higher than x and u , i.e. to implement the inner block as an anonymous procedure, hardly exploring that this anonymous procedure would only be called with "x, u" --i.e. the outer block of P -- as top activation in the stack. If in the object code the variables x and u were referenced by the address couples

x: 3, 0

u: 3, 1

then y and v would be referenced by

y: 4, 0

v: 4, 1 .

(We did explore the special nature of the inner block at run time by shortcutting the there superfluous part of the operation UPDATE DISPLAY upon its entry and exit.) When EWD653 was written it was tacitly assumed that in the stack segment a new block segment would be created upon inner block entry in exactly the same way as would occur upon procedure call.

* * *

In the new arrangement only at procedure call a new block segment needs to be introduced, upon inner block entry its uninterpreted information (for the local scalars) and/or its header (with a descriptor for each local array) will be extended. In the example given above, we would have as address couples --ignoring for the time being such minor details as possible size differences between fixed length integers and descriptors-- in P's outer block

x: 3, -1

u: 3, +0

and in its inner block in addition to the above

y: 3, -2

v: 3, +1

The layout of the block segment is still --like any other segment--

uninterpreted information --- header --- subsegment(s) .

During the activation of P's outer block the corresponding layout of its block segment would be

x --- du --- u ;

during the activation of its inner block the corresponding layout of its block segment would be

y x --- du dv --- u v .

Observe that both the header and the uninterpreted information are allocated (or "used") in stack fashion, but that the stacks are in opposite direction. The two stacks meet with their bottoms. Thanks to this convention of opposite directions of the two stacks it is now possible to code the relevant information in the descriptor in such a way that one memory shift operation will do the job. Again, as in EWD653, the descriptor contains four scalars but now a different linear combination seems more appropriate, with only one of them fixing the position of the segment by fixing the position where its two stacks meet: for instance (I use different identifiers than I did on EWD653 - 6)

sp (= segment position)
dl (= downward length)
hl (= header length)
ul (= upward length)

Suppose that we have a supersegment the header of which starts at a location with address b ; for one of its subsegments described by the above quadruple the uninterpreted information would comprise locations M[i] with

$$b + sp - dl \leq i < b + sp$$

the header would comprise the locations M[i] with

$$b + sp \leq i < b + sp + hl$$

and for its subsegments (subsubsegments of the supersegment the position of which is fixed by b) the location M[i] are available with

$$b + sp + hl \leq i < b + sp + ul \quad .$$

Note that (dl + ul) equals the length of the subsegment in question.

Note. In the above it is suggested twice that the possibility of coming away with one shift operation only is the result of the different linear combination of the position and size data in the descriptor. This is a wrong suggestion: the possibility is the consequence of choosing different orientations for the header stack and the stack of the uninterpreted information. (End of note.)

* * *

I am pleased with the above simplification, because it simplifies and --more important!-- removes a dilemma that posed a question that I had to leave open. It is the same old story again: whenever a design decision presents itself as a sort of conflict and you are tempted to make a compromise, back up and rethink, because it is much nicer if the conflict can be avoided. It is a pity that this note only simplifies the discussion of the single sequential process and not the second half of EWD653, which is much harder.

Plataanstraat 5
5671 AL Nuenen
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow