Published: 06/05/67

Identification

Validate_arg
R.M. Graham, M.A. Padlipsky

Purpose

When an inner-ring procedure is called by an outer-ring procedure, the called procedure has, by definition, greater access privileges than does the calling procedure. Some means must be furnished within the protection mechanism to enable the inner-ring procedure to determine that the arguments passed to it will not cause it to exercise those privileges unwisely. Procedure validate_arg, discussed in this section, performs such argument-checking. It may be called by any inner-ring procedure which desires validation of arguments from the viewpoint of the arguments' having been accessible to the outer-ring procedure which called the inner-ring one. Also, validate_arg is called by the Gatekeeper's arg_pull and arg_push subroutines (BD.9.03), to assure that these ring-0 routines copy only arguments which are accessible to the procedures they are operating in behalf of.

"Accessibility", in this context, means only that the segment containing the argument has a protection mechanism "access bracket" such that the ring of the procedure in question is not forbidden access to the segment. The mode of a particular segment is not taken into account here. See BD.9.00, BG.9.00.

In addition to checking access considerations, there is a second task performed by validate_arg. As elsewhere in the protection mechanism, argument validation also raises the problem of possible alteration of other-ring data which reside in a segment which is shared by several processes. Therefore, steps must be taken to assure that the pointers to arguments being validated cannot be changed after they have been validated. To this end the argument list and associated specifiers being validated are copied by validate_arg into an area specified by the inner-ring procedure which called validate_arg. The area will be within the caller's ring (rather than within <u>its</u> caller's ring, which is where the originals are), so that the copies can be trusted when subsequently referenced through. Note that it is not necessary to copy data and dope, for if the outer-ring segment which contains them is subsequently altered, the only harm done is to the outer-ring caller of the inner-ring procedure which called validate_arg.

Usage

The calling sequence is

```
     call validate_arg (ap, types, count, copies);
```

with declarations

```
     dcl (ap, types, copies) ptr, count fixed bin (17);
```

where

ap is a pointer to the argument list to be validated (i.e., the argument list which validate_arg's caller was called with).

types is a pointer to a consecutive block of integers (fixed bin 35) which contains the data types of the arguments in the list pointed to by *ap*. "Data types" are per BD.1.00, and may be any of the "standard symbol types" defined there. The value in types(1) is the type of the first argument, the value in types(2) is the type of the second argument, and so on.

count is the length of the array pointed to by *types*.

copies is a pointer to a storage area into which validate_arg will copy argument pointers and specifiers, to guard against the possibility of their being changed out from under the inner-ring procedure after validation. Note that validate_arg's caller must use *copies* as its argument pointer after return from validate_arg. If *copies* is null, validate_arg will not copy; this is provided for use of arg_pull and arg_push, which must do their own copying, of dope and data as well as of argument pointers and specifiers.

Note that the ring number which will be validated against is the "validation level" (see BD.9.00, BD.9.01), which is by convention located at sb|3.

Error Handling

Validate_arg reflects errors by means of the standard Multics error-handling mechanism (see BY.11): If an inaccessible argument is detected, validate_arg places an appropriate comment in the user's error segment and calls *signal* for "validate_arg_err". "Inaccessible" is taken here to mean either "*ring* is outside access bracket" or "segment does not exist".

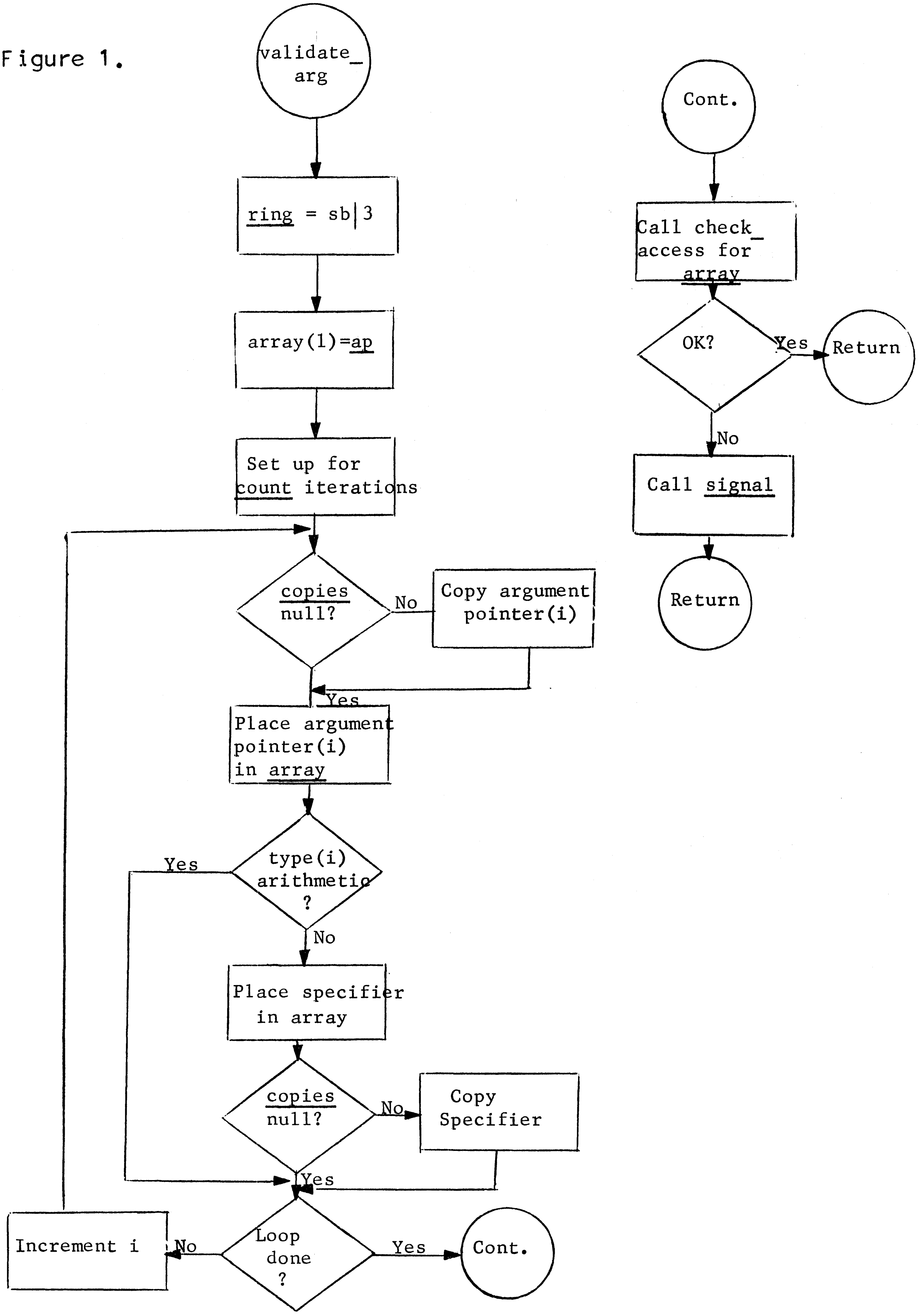
Method

Validate_arg is a slave procedure which operates in whatever ring it was called from, without a ring-crossing. That is, like condition, reversion, and signal, it has a protection list of 0, 63, execute only.

Figure 1 presents a block diagram of validate_arg. The logic is as follows: Get the ring number to be validated against from sb|3; call it ring. "Validation" will subsequently be accomplished by passing an array of segment pointers (ITS pairs) to the Basic File System for checking that the segments pointed to are accessible from ring; the basic task of validate_arg, then, is placing appropriate information into this array. (Call the array array.) The first pointer to go into array is validate_arg's argument, ap; that is, the segment in which the argument list itself resides must be validated; therefore, for each of the argument pointers in the argument list: If types(i) is that of an arithmetic scalar, only the corresponding argument pointer need by dealt with; it is placed in array and copied into the area pointed to by the copies pointer, provided that pointer is not null.

For all other data types, not only must the argument pointer be placed in array, but so must the pointers which comprise the specifier (pointed to by the argument pointer); depending on the value of type(1), either two or three additional pointers are involved here: data and dope pointers always, and free storage area pointer if relevant. The specifier is copied into copies, if that pointer is non-null. This processing of the types array continues for count iterations. Then a call is made to check_access (BG.3.02) for the array of segment pointers accumulated in array. (In all likelihood, most of the segment pointers will involve the same segment number; check_access will take care of eliminating superfluous checks.) If check_access returns with an indication that the segments involved are all accessible from ring, validate_arg returns to its caller. In case of an inaccessible argument, validate_arg calls signal (BD.9.04) for "validate_arg_err". (See Error Handling, above.)

Figure 1.


validate_ arg
ring = sb|3
array(1)=ap
Set up for count iterations
copies null?
No
Copy argument pointer(i)
Yes
Place argument pointer(i) in array
type(i) arithmetic ?
Yes
No
Place specifier in array
copies null?
No
Copy Specifier
Yes
Increment i
No
Loop done ?
Yes
Cont.
Cont.
Call check_ access for array
OK?
Yes
Return
No
Call signal
Return