Published: 05/06/68

Identification

A procedure which provides a description of I/O status
check_io_status
W. R. Strickler

Purpose

The last argument of every call to the I/O system is status, a bit string containing on/off indicators of all conditions of interest which can occur during an I/O transaction (see BF.1.21). A procedure which has called an I/O procedure can, by subsequent call to check_io_status, obtain an abbreviated code for and a description of the status of the terminated I/O transaction. The procedure may then call seterr (BY.11.01) for inclusion of this code and description in a message placed in the error segment. In some instances the procedure should also use status as the extra_bit_info argument of seterr.

Status Formats

The format of the 144-bit status string returned by a call to the I/O system is given in BF.1.21 and BF.2.24. Briefly, the string is divided into the substrings:

Bits	Purpose
1-18	primary bits
19-54	serious or fatal errors
55-90	advisory status and nonfatal errors
91-108	call-oriented status (dependent on type of call)
109-126	hardware status
127-144	call identification (transaction block index)

The primary status bits 1-18 are summarized as follows:

Bit	Meaning when set to value = 1
1	successful logical initiation (see Section BF.1.04).
2	successful logical completion (see Section BF.1.04).
3	successful physical initiation (see Section BF.1.04).
4	successful physical completion (see Section BF.1.04).
5	transaction terminated (no more status change).
6	serious or fatal error (nonzero bits in 19-54).
7	advisory status or nonfatal error (nonzero bits in 55-90).
8	call-oriented status (nonzero bits in 91-108).
9	hardware status (nonzero bits 109-126).
10	new status bits set (used during status exchange).
11	unassigned.
12	unassigned.
13	unassigned.
14	transaction aborted.
15	abort was due to quit condition.
16	reset condition (transaction not to be restarted).
17	device absent from channel.
18	sync control; return condition (see Section BF.2.02).

Codes and Descriptions

The procedure check_io_status returns a six-character code. The first two characters are either "ok" (no serious or fatal error) or "er" (serious or fatal error). If any of bits 7-9 is set, the third character is ":", and characters 4-6 are permutations of "a" (advisory status exists; bit 7 is set), "c" (call-oriented status exists; bit 8 is set), and "h" (hardware status exists; bit 9 is set). If fewer than six characters are needed in the code, the remaining right-most characters are blanks.

The description returned by check_io_status is built from a table of I/O error descriptions, which allows for a character string description for each of the status bits 1-126. Individual descriptions are separated by appropriate punctuation characters - ":", ";", ",", "." - to yield a (hopefully) literate overall description of the terminated transaction.

If, for example, a transaction is terminated after the occurrence of a fatal error, secondary bits 30 and 44 are set, and hardware status exists with secondary bits 112 and 121 set:

Then code is "er:h ".

The description has the form,

```
"FATAL ERROR:  "||description for bit 30||",
               "||description for bit 44||";"
               ||" HARDWARE STATUS: "||description for bit 112||",
               "||description for bit 121||"."
```

Usage

```
call check_io_status (status, iocall, ioname, code, info);

dcl                status bit (144),

                   iocall char (N1),

                   ioname char (N2),

                   code char (6),

                   info char (N3) var;
```

status	the 144-bit string in which I/O status for the transaction is recorded
iocall	the type of I/O call, used to determine the orientation of bits 91-108
ioname	the framename attached to the I/O device involved in the transaction (use of indirect framename depends on implementation of upstate)
code	returned by check_io_status
info	description of I/O status returned by check_io_status

The call to check_io_status is made after the call to an I/O procedure. If code returned to the calling procedure indicates to that procedure that it has been sufficiently "offended", it should then, before signalling (see BY.11.00), issue the following call to seterr:

```
call seterr (error_loc, code, info, " ", status);
```

where code and info are the arguments returned by check_io_status. The seriousness of an "offense" is decided by the calling procedure. It may, for example, decide to ignore calling seterr and signalling whenever the first two characters of code are "ok".

Both check_io_status and seterr have been designed with procedures such as the listener, read_in, and write_out in mind. The listener (BX.2.02) calls read_in (BY.4.02), which calls the I/O procedure, read (BF.1.12). Thus read_in is the offended procedure when the status string is so interpreted after the call by read_in to check_io_status. In this case read_in calls seterr. Figure 1 shows the relationship among calling procedures, an I/O procedure, check_io_status, and seterr. Procedures are indicated by rectangles, arguments in circles between calling and called procedures.

Numbers indicate sequencing of calls.

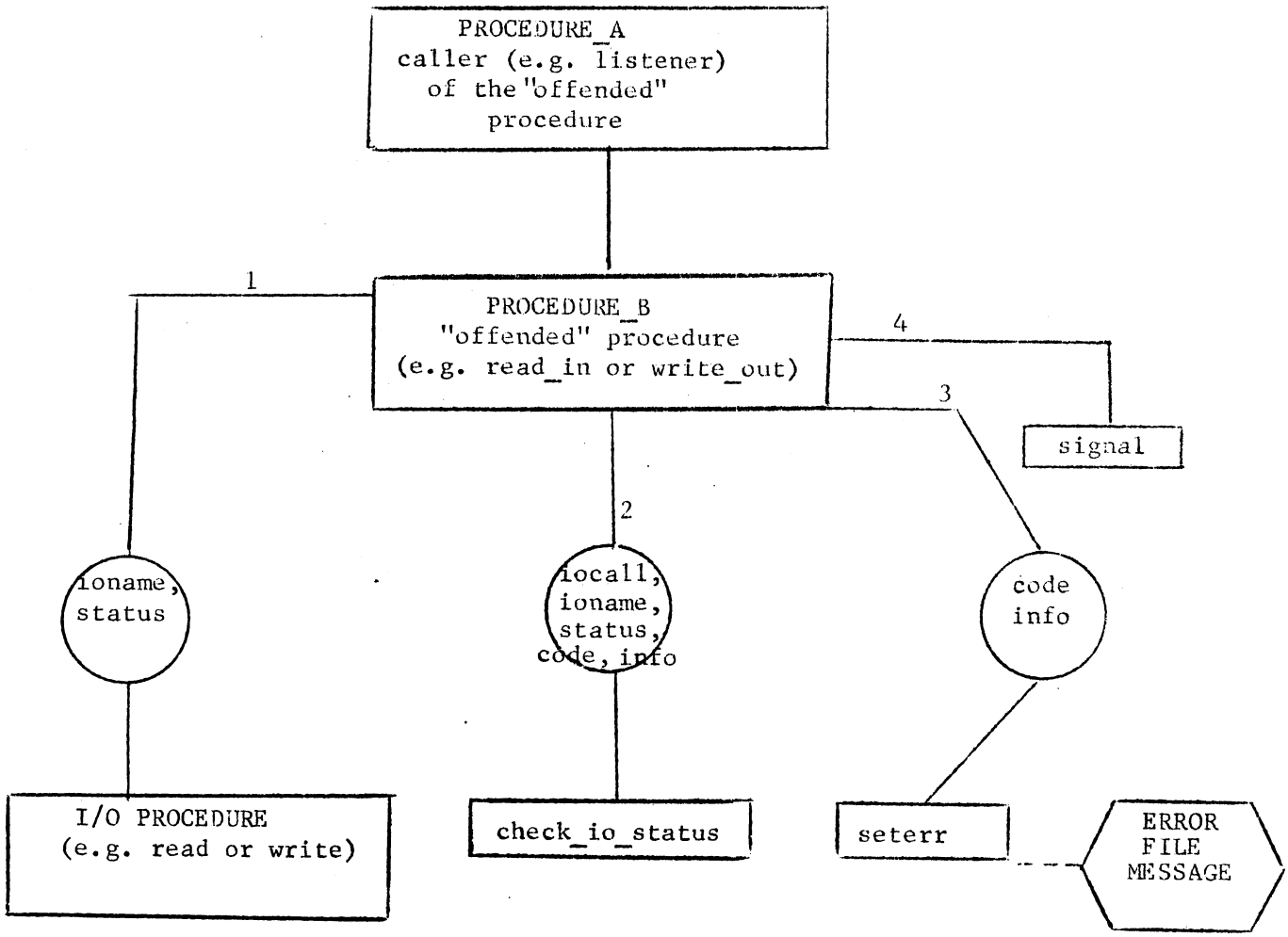

Figure 1

Implementation

A table of descriptions of I/O error and status conditions appears in a data segment containing the structure:

```
dcl 1 dsc based (p),          /*table of error descriptions*/
    2 pr (18),                /*primary errors*/
      3 tell bit (1),         /*is a description for this
                                primary condition allowed
                                in the general description?*/
      3 n fixed bin (17),     /*significant characters in d*/
      3 d char (100),         /*error description*/
    2 sf (36),                /*serious or fatal errors*/
      3 n fixed bin (17),
      3 d char (100),
    2 ad (36),                /*advisory status*/
      3 n fixed bin (17),
      3 d char (100),
    2 hd (18),                /*hardware status*/
      3 n fixed bin (17),
      3 d char (100),
    2 n_call fixed bin (17),  /*number of iocalls*/
    2 call (p→dsc.n_call),    /*array of call-oriented status*/
      3 n_name fixed bin (17),/*significant characters in name*/
      3 name char (32),       /*iocall, e.g. read or write -
                                see BF.1.00 for a list of
                                iocalls*/
      3 des (18),             /*error descriptions for iocall*/
        4 n fixed bin (17),
        4 d char (100);
```

Initially, no description of status indicated by secondary bits will appear in the table. That is, for all i and j:

```
p->dsc.sf(i).n = 0

p->dsc.ad(i).n = 0

p->dsc.hd(i).n = 0

p->dsc.call(i).des(j).n = 0
```

Consequently no secondary descriptions will be placed in the string, info. This deficiency will be corrected by the placing of secondary descriptions in the table when secondary bits are defined.

The procedure, check_io_status, does the following:

1. Check for completion of status string (termination of transaction): if bit 5 is set, go to step 3; otherwise:

2. Call upstate (BF.1.11) and go back to step 1.

3. Initialize: xcode to "ok", info to the null string. (xcode is a varying character string.)

4. Put a general description, if any, of status into info: Scan the substructure dsc.pr; for each i ($1 \leq i \leq 18$) for which p->dsc.pr(i).tell = "1"b and the i-th bit of status is set, concatenate info with the p->dsc.pr(i).n significant characters of p->dsc.pr(i).d. Prior to this concatenation, if info is not the null string it is concatenated with ",".

   p->dsc.pr(i).tell = "0"b for $6 \leq i \leq 9$, since these descriptions appear as prefixes of descriptions for secondary status conditions, as shown in the steps which follow.

5. Check for a serious or fatal error: If bit 6 of status is not set, go to step 6; otherwise: Set xcode = "er".

   If info is not null, concatenate it with ";". Concatenate info with p->dsc.pr(6).n characters of p->dsc.pr(6).d, and then with ":".

Scan the substructure dsc.sf. For each i ($1 \leq i \leq 36$) for which the (18+i)-th bit of status is set, concatenate info with "," if a previous bit in the secondary substring was set and then with the n significant characters of p->dsc.sf(i).d.

6. Check for advisory status: If bit 7 of status is not set, go to step 7; otherwise:

   Concatenate xcode with ":a".

   Use the same scheme described in step 5 to concatenate info with p->dsc.pr(7).d and then with those p->dsc.ad(i).d for which the (54+1)-th bit of status is set.

7. Check for call-oriented status: If bit 8 of status is not set, go to step 8; otherwise:

   If length (xcode) is 2, concatenate xcode with ":".

   Concatenate xcode with "c".

   Scan dsc.call for that j for which p->dsc.call(j).name = iocall. Then use the scheme described in step 5 to concatenate info with p->dsc.pr(8)d and those p->dsc.call(j).des(i).d for which the (90+i)-th bit of status is set.

8. Check for hardware status: If bit 9 of status is not set, go to step 9; otherwise, use the same scheme described in step 7 to concatenate xcode with "h" and info with p->dsc.pr(9).d and appropriate p->dsc.hd(i).d.

9. Set code = xcode. If info is not null, concatenate it with ".". Return.