Published: 07/18/67

Identification

Indicate the occurrence of a condition
signal
M. A. Padlipsky

Purpose

When a programmer wishes to have the System invoke the currently-active handler for a given "condition" (if such a handler exists), he calls subroutine signal with the name of the condition as argument. If there is no handler established for the condition (i.e., no procedure previously specified, in the current process, to be invoked when a signal is made of the condition at hand), a default action will be taken as discussed below; it should be noted, however, that circumstances can arise where signal is called irrespective of whether condition has been (cf. BY.11, the Multics error-handling mechanism). Section BD.9.04 contains a detailed description of the Multics condition-handling mechanism. Section BY.12.04 contains a description of subroutine condition, which is used for establishing the existence of, and handlers for, conditions.

Restriction

The condition name "cleanup" is reserved, and will be rejected by signal; see BB.5.08 and BY.12.06.

Usage

The calling sequence is

```
call signal (condname, rtn_flag, ptr);
```

with declarations

```
dcl condname char(*), rtn_flag fixed bin (17), ptr ptr;
```

where

condname is the name of the condition being signalled; it may be declared in the user's procedure as varying or non-varying and the "*" should, of course, be replaced by an appropriate number.

rtn_flag indicates whether the user's procedure will accept a return from the condition-handler which signal is to invoke; if rtn_flag is 0 a return will not

be accepted from the handler, and if one is attempted signal will cause a terminate-process fault; if rtn_flag is 1 a return will be accepted.

ptr will be passed to the invoked handler as an argument; it may, of course, be a pointer to an argument list; it may also, for that matter, be null.

Defaults

If there is no handler active for the signalled condition, default action will be taken in one of two ways, depending upon the type of the condition. Conditions may be either system-defined or programmer-defined. System-defined conditions are those listed in BB.5.08, and in general are characterized by having special default handlers. That is, if the occurrence of a system-defined condition is signalled and there is no handler active for it, a predetermined procedure (the "default handler") appropriate to the condition at hand will be invoked by signal. Programmer-defined conditions, on the other hand, all receive the same default handling: if there is no handler active for a programmer-defined condition (i.e., one which is not listed in BB.5.08), signal will signal the occurrence of a condition named "unclaimed_signal". In normal Multics use, the Shell will be in the process and will have a handler active for condition "unclaimed_signal"; see BY.11.05. Note that "unclaimed_signal" is itself a system-defined condition; the action that will be taken in the event of there being no handler active for the "unclaimed_signal" condition is the causing of a terminate-process fault (BB.3.05). Sub-systems other than the Shell will most likely have their own handlers active for "unclaimed_signal"; should any sub-system not have a handler active, the process-terminating default case is reached by signal.

Implementation

Details of the implementation of signal are specified in BD.9.04. Briefly, signal inspects the "signal vector" (a software analogue of the 645's hardware fault vector) for the protection ring it is invoked in, searching for the active handler for the condition signalled. If the active handler is not in the current ring, the search continues back along the process at hand's path of ring-crossings until either the active handler is found or the record of ring-crossings is exhausted. See also BD.9.00, the overview of the Multics protection mechanism, for definitions of terms.