Published: 07/19/68
(Supersedes: BY.11.05, 08/07/67)

Identification

Unclaimed_signal
R. J. Sobecki

Purpose

"Unclaimed_signal" is the name of the condition which is signaled by the Multics condition handling procedures when it is discovered that there is no handler active for a user-defined condition. In the Shell (BX.2.00) the handler for the unclaimed_signal condition calls the procedure unclaimed_signal. Unclaimed_signal prints the name of the condition that was signaled and gives the user the capabilities to proceed in one of the following ways:

1) Continue from where the unestablished, user-defined condition was signaled.

2) Give up and return to command level in the listener. (In which case all Stack frames beyond the first generation of the listener are lost.)

3) Preserve the Stack frames preceding unclaimed_signal's Stack frame and permit the user to issue commands such as debug to inspect the situation.

Introduction

The listener (BX.2.02) enables for all user-defined conditions which are not otherwise explicitly established as follows:

```
dcl unclaimed_signal ext entry;
call condition ("unclaimed_signal", unclaimed_signal);
```

When an unestablished user-defined condition is signaled, the signal procedure (BD.9.04) notes that it is undefined, then signals the condition "unclaimed_signal". Provided, of course, that the user himself has not established a handler for the condition in the meantime, the handler established in the Shell is invoked; that is, the unclaimed_signal procedure is called. The "unclaimed_signal" condition is itself a Multics system-defined condition; its default condition handler executes a terminate process fault (see BB.5.03). Unclaimed_signal (the procedure) operates in the ring from which it is called.

Usage

The signal procedure invokes unclaimed_signal by issuing the following call:

```
call signal ("unclaimed_signal", "1"b);
```

where the "1"b indicates that a return from the unclaimed_signal condition handler is acceptable. The signal procedure must call seterr (BY.11.01) with the name of the unestablished user-defined condition as error_info before calling itself to invoke unclaimed_signal. Unclaimed_signal can then obtain the name of the condition that was signaled from the error segment.

Implementation

Unclaimed_signal begins by calling geterr with arguments indicating that it wants the contents of the error_info field from the last error description in the error_out segment (see BY.11.02). After the call to geterr, unclaimed_signal writes the error_info returned (the name of the unestablished user-defined condition) in the user_output stream. Unclaimed_signal then calls delerr to delete the error descriptor placed in the error_out segment by the signal procedure.

Unclaimed_signal proceeds to write out a message telling the user that he can now issue any command, reminding the user (only if the brief option is off) that the following commands are meaningful at this time:

1) issue the continue command (BX.3.10) to continue from where the user-defined condition was signaled.

2) issue the abort command (BX.3.11) to return to command level.

3) issue the printerr command to print the last error description deposited by the user in the error file.

4) issue other commands such as the debugging commands to inspect the current status of the process.

Loosely speaking, unclaimed_signal now calls the listener (BX.2.02) which reads a command sequence and passes it to the Shell. When the Shell has processed the command sequence it returns to the listener which (depending on

the setting of the housekeeping option) either returns or reads another command sequence. If the listener returns to unclaimed_signal, unclaimed_signal simply turns around and calls the listener again. Thus, it is obvious that under these circumstances unclaimed_signal never returns to signal. (A return to signal is necessary for processing to continue from where the user-defined condition was signaled.)

In order to make the continue and cancel commands work, unclaimed_signal provides label variables through which continue and cancel may execute a non-local go to. The label variable must be associated with this generation of unclaimed_signal since it is possible for unclaimed_signal to be invoked while another (previous) generation of unclaimed_signal still exists in the Stack. Unclaimed_signal stores the label variables in internal static. It also sets a bit on to indicate that the label is valid. Any previously-stored values of the bit and the label variables in external static are first saved in unclaimed_signal's automatic storage. (Before returning (or transfering to the listener in the case of cancel), unclaimed_signal restores the internal static bit and label variable to the previously-stored values.) Unclaimed_signal then calls the listener.

In order for unclaimed_signal, the continue command, and the cancel command to utilize internal static label variables, all three are contained in the same procedure segment. The listener calls unclaimed_signal$init to set into that internal static, the location to which a cancel command should transfer.