Published: 01/23/68

Identification

Login and Quit Responders for Operators
K. J. Martin

Purpose

Operators face special problems which make it necessary that they have different procedures invoked immediately after login and following each quit than those invoked for other users. The normal login and quit responders are combined in the listener procedure described in BX.2.02. The operators' login and quit responders are also combined in a single procedure, op_listener, which closely resembles the listener. This section explains the need for and describes op_listener.

References

Section BX.15.00 presents an overview of the operator commands. Sections BX.15.02 and upward describe individual operator commands. BX.2.02 describes the listener. BQ.2.03 explains the concept of login and quit responders and details their manipulation.

Discussion

Operators of a Multics installation must cope with at least two problems not encountered by other users. These two problems arise because of unsolicited requests which are associated with certain operator functions. The media management function, for example, must deal with requests originated by other users to handle magnetic tapes and other appropriate media.

The first problem arises because processes belonging to other users wish to send inter-process events (see BQ.6) to the operator in charge of an unsolicited-request function. The other users expect to be able to find out the process id of the appropriate operator's working process. Section BT.2. on the media management module illustrates how the module associated with the media operator function expects to work.

When an operator becomes responsible for media handling (or any other unsolicited request function), the op_here command (BX.15.02) informs media management (and any other modules corresponding to functions assigned to him) of his presence and the process id of his working process. If at some later time that operator quits his computation, he will receive a new working process. The media management module and other modules must be informed of the process id of the new working process. It is preferable not to depend on the operator to issue the op_here command either initially or following each quit; a single slip-up could cause unsolicited requests to be lost.

The solution proposed here is to employ login and quit responders cognizant of the operator's responsibilities. The login responder would invoke the op_report procedure (BX.15.02) immediately following an operator's login. The quit responder would automatically invoke op_here on the operator's behalf following each quit. A few changes to the listener are sufficient. Details are given in implementation.

The second problem arises because unsolicited-request functions need, ideally, some operator constantly waiting for them to demand his attention. The operator in charge of an unsolicited-request function cannot be placed in such a position forever since he may be in charge of other functions. On the other hand, expecting the operator to faithfully check on each of his responsibilities with no prompting is simply courting trouble.

Consider a middle solution in which following each command sequence invoked by the operator, a special procedure, op_checker described in BX.15.03, is invoked to service unsolicited-request functions. The operator's login responder (also a slight revision of the Listener - BX.2.02) cooperates in this scheme. After reading a command sequence and before calling the Shell with that command sequence, it appends the string: "; op_checker". This causes the Shell to call op_checker after all other commands in the sequence. Op_checker determines if there are any unsolicited-request functions to be serviced by this operator (by checking the op_function data base described in BX.15.03). If there are none, op_checker returns and the operator is back at command level.

If, on the other hand, the operator is responsible for any unsolicited-request functions, op_checker calls in turn the service procedure associated with each. A service procedure typically takes care of all unsolicited requests which may be pending, then returns to op_checker. After explicitly checking each function, op_function waits on the event channel associated with each. Whenever an event is received on one of the event channels, op_checker calls the appropriate procedure to service the function.

The operator who <u>is</u> responsible for any unsolicited-request functions can get out of op_checker only by quitting. Hopefully this will increase the amount of time that the operator is servicing these functions, but also allow him to service other functions.

Note that the operator may explicitly invoke op_checker as well as any of the service procedures for unsolicited-request functions. The media command (BX.15.09) is such a service procedure.

<u>Implementation of op listener</u>

The reader is referred to BX.2.02 where detailed implementation of the listener procedure is given. The two steps given below are additions to be made to the implementation of the listener to produce the procedure op_listener.

The login responder portion of op_listener requires the addition of a step between steps 4 and 5 and a second step between steps 7 and 8 of the listener portion of the listener procedure

4A) Call op_here~~report~~, causing the operator to effectively report for duty to System Control.

7A) Concatenate the character string "; op_checker" to the completed command sequence. From this point the command sequence appears to have been typed with op_checker as the last command in the sequence.

The quit responder portion of op_listener requires the addition of a step between steps 7 and 8 of the quit handler portion of the listener procedure:

7A) Call op_here~~[illegible]~~ in order to inform interested modules that the operator's process id has changed.