NWG RFC 103
NIC 5764

IMPLEMENTATION OF INTERRUPT KEYS

R B Kalin
MIT Lincoln Laboratory
24 Feb 1971

The current protocol specifications contain a serious logical error in the implementation of the program interrupt function. This paper discusses the problem and offers a solution that is simple to implement.

THE PROBLEM

As found on most time-sharing systems the program interrupt key, elsewhere known as the break key, or help request button, has two functions. It suspends temporarily the user process being run, and it switches the keyboard input stream to a dormant supervisory process. Unaccepted input typed prior to the interrupt request remains buffered for the suspended user process. Subsequent typing is sent to a supervisory routine.

The current NCP protocol implements only half this function. It pprovides, through use of INS and INR control messages, for the suspension of a remote process, but it offers no mechanism for notifying the remote host at what time the data stream should be switched. INR and INS messages are sent via the control link and because messages on this link travel concurrently with those on the user's keyboard input link, the receiving host can not rely on relative arrival times as a source of synchronizing information. Without such information the remote NCP can not know which input characters are meant for the user process and which are meant for the supervisory routine.

A solution found on some systems to this problem is that of mapping the interrupt signal into some code from the character set -- typically an ASCII control-C. Unfortunately, this is not general enough to be used within the ARPA network. Some systems, eg. MULTICS, make use of all available ASCII codes for other purposes, none are available for such an assignment. Even if such an assignment could be made, there is the problem of getting the interrupt character to be recognized by the remote host. Buffers on that user link may be full and the sending host may be unable to transmit the message containing the interrupt code. If the remote user process loops without accepting data, there is the possibility that its input buffers will never become free and that the message will never get through.

A partial answer is that of providing at the serving end a teletype scanner process that is always hungry for input. Because all input messages are immediately consumed, buffers remain available and interrupt codes can get through. Unfortunately, this implies that at times characters must be thrown away. After being scanned there may be no buffer space available for them. While not critical during console interactions -- users can type only when the program demands input -- this defect prevents the scanner from being driven from a text file.

A SOLUTION

The following defines a solution to this problem for the case of ASCII data streams.

1) Character messages should use eight bit fields for each character code.

2) For all of the defined ASCII character codes the left most bit in the eight bit field shall be zero.

3) An interrupt sync character (arbitrarilly given the code octal 200) should be placed in the data stream at the correct point in the typing sequence.

4) All codes from octal 201 to octal 377 are officially to be ignored by a receiving host. Their use is reserved for additional control information, should it become necessary. Attempts to use them as additional character codes will meet with resistance from PDP-10 systems that internally pack characters into seven bit fields. Note that this objection can not be made against the interrupt sync character because it is filtered out by the system and never appears in a user’s input buffer.

5) Because of the possibility that there may be an insufficient allocation to allow the user message containing the interrupt sync character to be sent, the INR/INS mechanism currently defined must be kept. An INS control message should be sent at the time an interrupt sync character is entered into a text stream. Upon its reception by the foreign host, the attached process should be immediately suspended and the associated input stream should be scanned. If possible, all input up to the interrupt sync character should be buffered for the suspended process. Once the sync character is found, the stream

should be switched to the newly activated supervisory process. If it is not possible to buffer all of the user process's input, it can be thrown away, and a error message returned to the user by the supervisory process. In either event it must be guaranteed that outstanding input will be consumed and message buffers will be freed so that pending character messages can be sent.

6) In the event that an interrupt sync character is received before the matching INS, the user process should be suspended and the NCP should wait for the INS before proceeding.

7) The function of the NCP is the above discussion can, of course, be delegated a separate modulo, eg. a TELNET process. If this is done, the NCP can be transparent to message content.

COMMENTARY

The proposed change to the second level protocol described herein is not meant as a general solution, but rather as a specific patch to the current NCP design with the intent of correcting a critical error. Its more obvious deficiencies are...

1) It only works with seven bit code character streams. No extensions are allowed for EBCDIC, ASCII-8, or other large character sets. No provision is made for interrupting a process to which there is no character stream, although the author knows of no case in which the concept means more than closing the connection.

2) It requires the system to scan all data coming over an interruptable connection. Presumably this means that at the time the connection is created, the receiving host must be told that this scan is to be done. Various techniques, both implicit and explicit, could be used.

3) The technique is not immune to loss character boundaries within a message nor can it tolerate INS control messages that do not have matching sync characters, or vis versa.

4) It may not possible to get either the INS or the text message containing the interrupt sync character to a remote host. Possible reasons include user console failure, local host failure, network failure, blocked control link, insufficient allocation etc. Under such circumstances the remote process may loop indefinitely.

The only comprehensive solution known to the interrupt synchronization problem, those that avoid the above difficulties, require more than minor changes to the current NCP protocol. Unless simpler answers are suggested, their implementation must be postponed until the next major design revision.