To: MTB Distribution

From: Larry Johnson

Date: May 1, 1979

Subject: Improvements to FNP input handling.

This MTB proposes changes to the way input is handled in the FNP on HSLA lines. The goals of these changes are 1) to increase the reliability and reduce the overhead of the tty mode 'breakall' which is required for emacs, and 2) to reduce the amount of FNP buffer space required to run an HSLA channel so that more channels may be conected to an FNP.

The current implementation of breakall mode on hsla's is not adequate for supporting a large number of lines in breakall mode, or even to support a small number of lines on a busy FNP. The reason for this is that we cannot expect to be able to set up a new input buffer for each character typed in real time and keep up with the load. This is similiar to the situation that existed on pre-MR6.5 systems where we tried to run high-speed synchronous lines using small input buffers. When any kind of load is placed on the system, we could not keep up.

The same sort problems exist today in trying to run emacs on a heavily loaded FNP. It has been demonstrated that emacs is not usable on a FNP that is also running a few synchronous lines becuase hsla_man is unable to setup input buffers fast enough. The same problem has also been seen recently using emacs over TYMNET into MIT. TYMNET, at its discretion, may block several input characters together which are then sent at line speed into the FNP, which, regardless of the load, is unable to provide new buffers fast enough.

The solution to this problem is to change the way hsla_man handles input on asynchronous lines. Each hsla channel will have (while in receive mode) two permanent input buffers of 8 words each, enough room for 15 (1) input characters. (Terminals in 'blk_xfer' mode would use larger buffers based on 0.5 seconds of

(1) An 8 word buffer can hold only 15, rather than 16, characters because of the way the HSLA hardware works. Once an input buffer is exhausted, the HSLA stores additional input characters in the next character after the buffer; the 16th character must be reserved for this.

data at their baud rate, as today). Hsla_man would not switch input buffers on break characters, as is done now. Buffers would only be switched on the pre-tally-runout status which occurs when the current buffer is full. The CCT's (character control tables) would be setup to generate marker status on every interesting character (characters which require software response, such as new-line, tab while in tabecho mode, or every character while in echoplex or breakall modes). The hsla_man marker status and pre-tally-runout handlers would scan the current input buffer and perform whatever action each input character requires. This would normally involve copying the character into a termorary buffer which would eventually be passed over the dia, and to setup any echoing required by this character.

The major advantage of this approach is that it would reduce tremendously the pressure on hsla_man to respond to interrupts quickly. Instead of being forced to setup each new buffer before the user can type two more characters, at least 15 characters can now accumulate before hsla_man would lose any input because it could not run fast enough. In addition to having more time, hsla_man would also have less work to do. Since the input buffers are permanant, a new buffer does not have to be allocated; only the icw needs be reset.

Several other benefits accrue from handling hsla's in this manner.

1. Less memory would be required to support hsla lines, which is very important on the 18X. Instead of two 32 word input buffers active all the time, each line would only require two 8 word buffers. Since the format of these buffers would be completely internal to hsla_man, our current buffer mechanism would not have to be changed.

2. This new method of handling hsla's makes them look very much like lsla's. The only difference is that all the characters in the input "frame" are all for one terminal, rather than for several. An obvious way of implementing this new feature would be to provide a common input processing subroutine which is shared by both lsla_man and hsla_man. This would remove a great deal of duplicated code from the system, as every input mode that exists is currently implemented twice, once in hsla_man and once in lsla_man. These include crecho, lfecho, tabecho, echoplex, breakall, block_xfer, column counting for delay calculations, etc. In addition, each new mode that is proposed must be implemented twice in the current system, once each in hsla_man and lsla_man. If the input processing were shared, implmentation of new modes would be simplified. New modes that have already been proposed or discussed that this would affect are x-on, x-off processing, real time erase and kill processing, emacs "echo-negotiated" input, replay on demand, etc.

There are other related improvements that could be combined with the above changes.

1. Hsla_man interupt processing and scheduling could be reduced. To the software, marker status would mean process all input that has come in since the last marker. The hsla_man interrupt handler could make use of this by not storing the additional marker status if one were already queued.

2. This could also be done in the hardware (with greater efficency) by using a special CCT to reduce the number of hsla interrupts generated in echoplex and breakall mode. The idea here is to make the base CCT entries which generate marker status also switch to a second CCT which would not generate any status. The result is that once a marker status were stored, the channel would never generate anymore interrupts until the software that handles marker status had been scheduled and run. It would pick up all the characters that had arrived, and reset the CCT back to the standard base CCT which would then generate marker status on the next important character. Implementing this requires resolving some critical timing problems in the way an hsla fetches the CCT and updates the ICW, but this should be solvable.

3. Breakall mode on both hsla's and lsla's could be sped up by having the input processor call dia_man directly to store the character(s) in the channels dia queue, rather than in buffers. If the last thing in the queue were already one of these input requests, the two requests could be combined. When a mailbox became available, the characters would be moved directly into the mailbox, using the existing "input in mailbox facility". All this would happen without an input buffer ever being created, or the interpreter ever being called. Because some tty modes (specifcally polite and replay) and implemented in the control_tables using the interpreter it may be necessary to inhibit this optimazation if the channel is using any of these modes. Emacs already turns them off, so would not be effected. We should attempt to preserve the meanings of the existing modes where ever possible.