To: Distribution

From: Robert S. Coren

Date: 12/03/81

Subject: Measurement of FNP Performance

ABSTRACT

Currently available measurements of FNP performance and throughput are sketchy. Plans for extensive modification of FNP space allocation increase the desirability of better performance measurement in order to determine the impact of the changes on performance. Proposals for improved performance measurement include the following:

- metering of FNP throughput;
- improvement of the FNP "idle time" measurement to facilitate the calculation of a moving average and to account for bursts of activity;
- improvement of the "instruction counter histogram" facility provided by debug_fnp to identify control tables modules;
- metering of time spent in certain interesting interrupt-inhibited subroutines, especially those used for buffer pool management;
- configuring pairs of back-to-back channels in order to test performance with data being pumped through continuously.

THE NEED FOR BETTER PERFORMANCE MEASUREMENT

In MTB 536, a proposal was described for making more effective use of extended memory in the FNP. The MTB included a rather vague prediction that the proposed change would carry a performance penalty. The fact of the matter is that presently available meters make it very difficult to determine the exact size or nature of this presumed penalty. It was therefore

---

decided that better instrumentation of the FNP should be implemented before the changes proposed in MTB 536 so that the effect of those changes could be observed. This decision was bolstered by a feeling that FNP performance could be improved, but that without better measurements it would be difficult to determine where it needed improvement and to what extent any attempts to improve it had succeeded.

FNP THROUGHPUT

As a result of a technical oversight, the FNP meters included in MR9.0 did not include an overall throughput meter on a per-FNP basis. This oversight has been corrected, and the number of characters per second input and output through a given FNP is now included in the output of the channel_comm_meters command, and is also available separately by means of the fnp_throughput command.

IDLE TIME MEASUREMENT

The method currently used to measure the percentage of time that the FNP is idle is as follows: the elapsed timer is set to a specified value (which defaults to 50 milliseconds); when it runs out, an interrupt is generated, and the interrupt handler compares the value of the instruction counter at the time of the interrupt with the (known) address of the DIS instruction at which the FNP sits when it has nothing to do. The handler then increments one or the other of two cells, depending on whether or not the interrupt occurred at the idle DIS; comparison of the values of these two cells is used to determine the percent of time that the FNP is idle (or busy).

While this method has the virtue of simplicity, it has some disadvantages as well.(1) It does not actually account for all idle time, since it only records any data when the timer actually goes off. It is also not well suited to the maintenance of moving averages or the observation of bursts of activity.

The alternative method proposed here is based on a suggestion from Ricki Vick of FSO. The dispatcher, instead of sitting at a DIS when idle, loops incrementing a counter; the elapsed timer is

---

(1) One apparent disadvantage, namely the possibility that a timer runout when interrupts were inhibited might incorrectly be recorded as idle, does not actually obtain; the idle DIS is preceded by two NOP instructions during which the timer runout can be handled.

set for one second rather than 50 milliseconds. When the timer runs out, the interrupt handler does the following:

- records the value of the counter and resets it to zero;
- adds the value of the counter to a running total, and increments a count of the number of times the counter has been recorded;
- tests the value to see if it is either a new minimum or a new maximum, and records it as such if it is either.

The running total is used to derive an average. The maximum is used to determine an idle percentage; that is, any interval during which the count matches the maximum is considered 100% idle. This approach is based on the assumption that the theoretical maximum idle time is experienced fairly early in any given bootload (this could be ensured by forcing 1 second of idle time near the beginning of each bootload). The minimum is used to identify the busiest second of the bootload.

A user-ring metering program copies all the stored values described above on request, and reports an average idle time over the current metering interval, as well as the percentages reflected by the last recorded count and the lowest recorded count (if desired). The usual resetting mechanism can be used to start a fresh metering interval (although the maximum count is only reset when a new maximum is reported by the FNP). An absentee job that reported and reset the idle meters at regular intervals could thus provide a moving average of FNP idle time.

Depending on configuration and activity, it might be desirable to use a timer interval other than one second. The length of this interval could be made a parameter optionally supplied in the FNP bindfile. Varying it dynamically, on the other hand, would introduce severe difficulties in interpreting the data, since values accumulated using one interval size would not be comparable to those accumulated with another, and the recorded maximum count might become completely meaningless.

INSTRUCTION COUNTER HISTOGRAM

The timer interrupt handler described above can also be made to keep track of the values of the instruction counter when it is invoked. Essentially, it divides FNP memory up into "buckets" of some specified size, and whenever the instruction counter does not reflect an idle state, it increments the bucket corresponding to its current value. Various requests exist in the debug_fnp command to turn this feature on and off, and to use the contents

of the buckets to determine in which modules the FNP is spending the bulk of its time.

This approach has two problems. The first is that, if the interrupt occurred while running in the interpreter, this information is not very interesting; the control tables module in use at the time would be more useful information. This can be solved fairly simply by having the recording program use the contents of index register 2 rather than the instruction counter if the latter is within the interpreter, since index register 2 normally points to the current op block when running in the interpreter.

The second problem is that, for the purposes of monitoring the instruction counter, the failure to record anything while interrupts are inhibited constitutes a serious loss of information. Unfortunately, there does not appear to be any clean or reliable way to solve this general problem. The best approach seems to be to accept the fact that the percentages accumulated by the instruction counter sampling method are percentages of interrupt-enabled time only, and to use a different mechanism to record the time spent in certain selected interrupt-inhibited routines.

The simplest and probably most effective method of doing this is to make each such subroutine read the elapsed timer at entry and exit, and to keep a running counter (at exit) of the number of calls and the total time spent. This could be done either by explicit subroutine calls in the modules being metered, or by an additional parameter to the "subr" and "return" macros used to generate entry and exit sequences. The recording of the actual data would presumably use the existing metering mechanism. Note that if instruction counter sampling is being used, the timer will obviously have to go off more often than once a second. The idle counter sampling described above would not be done at every timer interrupt in this case, but still only once every second.

A complete list of routines to be monitored in this fashion has not been determined, but the most obvious candidates are the various subroutines used for allocating and freeing buffer space. Some parts of the scheduler are also possibilities, as are the subroutines in dia_man dealing with the management of the per-channel request queues and the construction of inbound mailboxes. Some care has to be taken both in the placement of the metering calls/macros and in analyzing the results of this metering, since some of these routines call each other. In addition, the calls themselves will take time and space, so we probably do not want to be overly prodigal with their use. It would probably be wise to make the metering described here be an option (separate from the option of whether to meter the FNP at all) settable at bind time, and possibly at assembly time as well.

EXPERIMENTS

The mechanisms described above can be used to gather statistics on our various service systems, of course, but they can also be used to observe FNP performance under more or less controlled conditions by configuring pairs of channels connected back-to-back. Various types of channels will be run in various combinations, driven by user-ring programs that simply generate output more or less continuously and discard all input. For example, we could run from one to 48 pairs of asynchronous channels at whatever combination of speeds we chose; or, say, 4 pairs of bisync channels and 2 pairs of X.25 channels. The purpose in wiring the channels back-to-back is to avoid uncontrollable variables introduced by the use of external hardware, or the complications resulting from the use of an external driver such as CUESTA.

HARDWARE REQUIREMENTS

In order to perform the experiments described above, the following hardware would have to be available (preferably at CISL):

- o -- a full complement of 12 HMLC "mother boards" for a full FNP
- o -- 48 dual asynchronous subchannel boards
- o -- at least 4 HDLC subchannel boards
- o -- at least 4 bisync subchannel boards

TIME ESTIMATES

The following is a summary of the tasks required to implement the proposals in this document, along with very rough time estiemates for each.

Task	Time (weeks)
Add idle loop and modify interrupt handler to record counter values	1
Modify timer interrupt handler to record x2	0.5
Add timing meters to inhibited routines	2
Modifications to and/or addition of metering commands for idle meters	2
Command to analyze time measurements for inhibited routines	1
Documentation of new/changed commands	1
Loopback experiments	3
Total	10.5