To: Distribution

From: John J. Bongiovanni

Date: 01/20/81

Subject: Measuring Response Time on Multics

1. INTRODUCTION

This document contains a description of a method to measure response time on Multics in a manner which is independent of user-ring and which approximates the response seen by a user at an interactive terminal (excluding delays external to the mainframe, such as communications delays). In the following discussion, some definitions critical to understanding the design will be presented. Following this, the conceptual model of user interaction will be discussed. Finally, the (rather trivial) implementation of this model into Multics will be given.

This MTB addresses only the measurement of response time and the accumulation of response time data in ring-0. In order for this data to be of much use, tools to summarize and present this data must be developed. Such tools are not within the scope of this document.

Send comments by one of the following means:

By Extended Mail Facility, on MIT or System M:
Bongiovanni.Multics

By Telephone:
HVN 261-9314 or (617)-492-9314

2. OBJECTIVES

It is curious that Multics, arguably the earliest commercial interactive computer system, has no facility to measure end-user response time, the single most important performance measure for an interactive system. The traffic controller maintains an average "response time", but this is merely an eligibility queue time. The theory here is that highly interactive users tend to use interactions with arbitrarily small resource demands, and so the queue time for memory (eligibility) is an adequate approximation of computer response time (a term which will be made more precise later). Observation of this "response time" for any length of time refutes this theory convincingly. There are two problems. First, many interactive users call relatively long-running commands. Second, a "rule of thumb" from queueing theory is that at high but not excessive loads, the average queue time for a resource approximates the average service time for that resource. So the service time for interactions cannot be discarded in measurement of response time, if accurate measurement is desired.

This MTB describes a design which meets the following objectives:

- o measures response time for each interaction which approximates the response time seen by the user (except for delays external to the mainframe)
- o accumulates response time by interaction types, where an interaction type is based on the load-independent resources consumed during the processing of that interaction
- o measures response time in a manner independent of user-ring

3. DEFINITIONS

Interaction - the processing caused by the input of a unit of data by a user at a remote terminal. The definition of a unit of data is application dependent. For command-level and for typical user program interaction, the unit of data is a line of text. For the Emacs editor and for similar exotic applications, the unit of data is less precise. It is the amount of text handed to user-ring in response to one call to ring-0 tty modules.

Response Time - for an interaction, the clock time between the arrival of an interaction at the mainframe and completion of processing for that interaction by the serving process. Specifically excluded from response time for an interaction is response time being accumulated for an earlier interaction being served by the same process (several interactions for the same process may be in the mainframe at the same time due to type-ahead).

Queue Time - for an interaction, the clock time between the arrival of an interaction at the mainframe and the award of eligibility to the serving process (zero if the process is eligible), eliminating overlapped time due to type-ahead as above. This corresponds to the existing measure "response time".

4. MODEL

The following diagram represents a finite-state automaton which models the interaction of a single user with Multics:

```
                  +------------------+
                  |                  |
                  |    Processing    |
                  |                  |
                  +------------------+
                      |          ^
                      |          |
processing for        |          |   interaction
interaction           |          |   received at
completed             |          |   mainframe
                      |          |
                      V          |
                  +------------------+
                  |                  |
                  |    Thinking      |
                  |                  |
                  +------------------+
```

This is a relatively simple, but nonetheless useful model. A user is either processing (that is, the user's process is demanding resources in response to a previously entered interaction) or thinking (idle between interactions). Of course, all of this is with the perspective of the mainframe, so that delays external to the mainframe (e.g., communications delays) are represented as thinking. Although type-ahead does not appear explicitly, it can be incorporated fairly easily, as hinted above. Specifically, an interaction which has been "typed ahead" and is queued at the mainframe is viewed as not having arrived. When the current interaction is completed, it is immediately available (i.e., the "think" time between the two interactions is zero).

The importance of this model in the present context is that it allows a fairly simple interpretation of response time for an interaction. It is simply the amount of time the interaction spends in the "Processing" state of the finite-state automaton. If this model could be implemented into Multics, and the instantaneous state of each process tracked according to the model, then response time could be derived easily. Unfortunately, the model is a bit too simple to be useful in terms of the objectives defined above. However, some relatively minor extensions to the model make it useful in this context. The revised model is represented as follows:

```
                           +-------------------+
                           |                   |
                           |  Processing       |
                           |                   |
                           +-------------------+
             call ring-0 tty    ||         ^
     for next interaction       |          |  return from ring-0 tty with
                                V          |     interaction
    award eligibility      +-------------------+
         +--------------->  |                   |
         |                 |     Other         |
+-------------------+      |                   |
|                   |      +-------------------+
|     Queued        |         |        ^
|                   |         |block   |
+-------------------+         |        | non-tty wakeup
         ^                    V        |
         |          +-------------------+
         |          |                   |
         +----------|     Blocked       |
     tty wakeup     |                   |
                    +-------------------+
```

The "Other" state was introduced to handle interactions which block themselves for other than terminal input. For such interactions, it is not appropriate to include the delay time (i.e., the time spent blocked) in the response time. It is also likely that such interactions are not very "interactive" in an intuitive sense (a common reason for blocking is to await completion of a tape I/O).

There is an added complication because of the implementation of blocking in the traffic controller. Specifically, a process does not block itself for anything in particular (at least, not from the traffic controller's point of view). It merely blocks itself, which causes it to remain inactive (ineligible) until some external event occurs (typically a wakeup). There is no necessary relationship between the reason the process blocked itself and the event which caused it to be awakened. A further complication is introduced by the protocols for receiving tty input. When a user-ring process calls the ring-0 tty routines for input, protocol is for the process to block itself in user ring if no input was returned.

Because of all of this, it is not possible to know in ring-0 at the time a process blocks itself whether it is awaiting terminal input. This determination can be made when the process is awakened, as the source of the wakeup request is known. If the wakeup was sent by ring-0 tty routines because of input available, it can be assumed safely that the process is awaiting terminal input. This assumption is safe, since the wakeup will be sent only if the process had previously called ring-0 tty for terminal input, and none was available. The point of this discussion is that the model in the last diagram is not strictly a finite-state automaton, since state changes do not depend only on the current state and the transition. Other than for theoretical tidiness, this concern is not especially important.

The utility of the model pictured above is that all of the transitions depicted correspond to well-defined events in ring-0. These events are as follows:

award eligibility - the event of the same name in pxss

call ring-0 tty for next interaction - call tty_read through hcs_

return from ring-0 tty with next interaction - return from tty_read with a non-zero count of characters returned

block - routine in pxss with the same name

tty wakeup - call to pxss from the ring-0 tty routines upon receipt of input for a process for which the last call to ring-0 tty returned no characters of input other wakeup - call to pxss to wakeup a process, other than for a tty wakeup

The basic implementation of the model is quite simple. The state of each process is maintained in some convenient per-process location. This state is maintained according to the model by some central routine. Subroutine calls to this routine are placed in ring-0 modules at locations corresponding to the above transitions. Upon change of state, the routine computes the time spent in the previous state, and meters appropriately. According to the above definitions, queue time is the time spent in the "Queued" state. "Think" time is the time spent in the "Blocked" state, provided that the transition which causes the process to leave that state is "tty wakeup". "Processing" time is the time spent in the "Processing" state. Finally, "Response" time is the sum of "Queue" time and "Processing" time.

5. DETAILED IMPLEMENTATION

5.1. New Data Cells

The following cells will be added to the apte:

current_response_state - a number indicating the current state of the finite-state machine described above which represents the process.

last_response_state_time - the value of the real-time clock the last time the state of the finite-state machine changed.

number_processing - the number of times the process has entered the "Processing" state

total_processing_time - the total clock time spent in the "Processing" state

The following cells will be added to the wcte:

number_thinks - the number of times any process in this work class entered the pseudo-state "Thinking"

number_queues - the number of times any process in this work class entered the "Queued" state total_think_time - the total amount of time any process in this work class spent "Thinking"

total_queue_time - the total amount of time any process in this work class spent in the "Queued" state

number_processing - an array of number of times a process in this work class entered the "Processing" state. This array corresponds to the array "vcpu_response_bounds" in tc_data (described below).

total_processing_time - an array of total time spent by processes in this work class in the "Processing" state. This array corresponds to the array "vcpu_response_bounds" in tc_data (described below).

total_vcpu_time - an array of virtual cpu time spent by processes in this work class in the "Processing" state. This array corresponds to the array "vcpu_response_bounds" in tc_data (described below).

The following cells will be added to tc_data:

vcpu_response_bounds - an array of virtual cpu times, increasing in value, which define the boundaries of virtual cpu times for various response classes. By means of this array and the corresponding arrays in the wcte, response time will be accumulated for various response classes. Intuitively, interactions with virtual cpu times below that in the first element of the array can be considered "trivial"; interactions with cirtual cpu times larger than that in the last element of the array can be considered "heavy".

vcpu_response_bounds_size - the number of elements in the vcpu_response_bounds array (needed since the program which uses this array is in ALM).

meter_response_time_calls - the total number of calls to the routine meter_response_time

meter_response_time_overhead - the total cpu time spent in the routine meter_response_time

meter_response_time_invalid - the number of calls to meter_response_time with invalid state/transition requests

5.2. meter_response_time

meter_response_time is an ALM program, bound into the same wired segment as pxss, which maintains the finite-state machine described above, and accumulates the metering information into apte's and wcte's. This program is in ALM since it must run inhibited (so that the clock values it uses make sense), and since it must be called during critical pxss processing.

meter_response_time is called with two arguments at the time of a state transition. The first argument is a number identifying the transition (valid transitions will be described in an include file). The second is the process-id. Using an internal table, the state of the target process is updated. If appropriate, the counts and times described above in the apte and wcte are updated. If the processid is invalid, no action is accomplished, but no error message is returned (since this is a highly-non-critical function of no particular interest to the calling program). A "side-door" is provided for pxss wherein the arguments are passed in registers, and the normal "entry/getlp/return" sequence is bypassed. Again, the point in pxss when meter_response_time is called is in a critical processing section.

The internal table used by meter_response_time is pictured as follows:

TABLE DESCRIBING RESPONSE TIME TRANSITIONS

	Transitions					
State	1	2	3	4	5	6
Initial (I)	I	I	P	I	I	Q
Blocked (B)	I	I	I	I	O	Q
Queued (Q)	O	I	I	I	Q	I
Other (O)	O	O	P	B	O	O
Processing (P)	P	O	I	B	P	I

Transitions:

1 - Award eligibility

2 - Call ring-0 tty for next interaction

3 - Return from ring-0 tty with next interaction

4 - Block

5 - Non-tty wakeup

6 - Tty wakeup.

The Initial state is the state of the process at process creation and after any invalid state/transition request. Any transition which causes the state of the finite-state machine to change to Initial from anything other than Initial is metered as an error.