To: Jeryy Saltzer
Bob Daley
Corby
Ted Glaser
Carla Marceau

From: Art Evans

Date: June 8, 1967

A basic concept in the Mark II scheduler is the idea of an interaction. As it happens, it is not [redacted] immediately obvious just what an interaction is in Multics, or how to tell when one has taken place. The attached document attempts to raise some of the relevant issues and to suggest a [redacted] possible solution. Comments are requested.

The second Multics scheduler uses [redacted] a multi-queue, CTSS-like algorithm. [redacted] Basic to [redacted] this algorithm is the idea of an interaction. Intuitively, it seems that an interaction takes place when [redacted] a user, seated at a console, has completed a "think time" and has [redacted] given the system something to do. A basic idea in Multics is that such a user should be favored, at least to the extent of giving him extra priority the next time he is scheduled. [redacted] The mechanism of favoring him is not relevent here -- instead, we are concerned with just what [redacted] an interaction is and how we can tell one has taken place.

This problem has no real counterpart in CTSS. There, [redacted] an interaction is defined to take place whenever a program goes into "input wait" status. (Output wait also produces an interaction -- [redacted] an aspect of CTSS that many find curious.) In the nature of things, a CTSS program can be in input wait only if it is waiting for console input, and it is [redacted] reasonable to regard the arrival of that input as motivation for high priority scheduling. (One can "beat" the system with judicious use of interconsole messages.)

Because of the read-ahead feature in Multics, it is harder to tell just [redacted] when an interaction has taken place. We want to be sure that an interaction is reported only when [redacted]

(a) a process must block itself for lack of console input, [redacted] and

(b) the input [redacted] actually arrives from the device.

The problem has to do with the diffuseness of the code that processes console input. Let [redacted] us consider the [redacted] case of input from a console with type-ahead. At some stage the working process asks for input. In due course, the [redacted] Device Strategy Module (DSM) in the process' Ring 1 considers the request. If there is available text that has already been typed [redacted] it is merely returned to the working process. If not, [redacted] however, the DSM must wait for text, by calling the Wait Coordinator. It is at this time that [redacted] the DSM knows that there is the possibility of an interaction. (As we will see, events may transpire that result in no interaction's taking place.) [redacted]

The DSM actually gets input from a Device Control Module (DCM) [redacted] in a different process with which it communicates via suitable Event Channels and shared data bases. To simplify a [redacted] complicated situation, the DSM leaves its request in coded form in a place available to the DCM [redacted] and then informs the DCM [redacted] of the existence of the request. Now we return to our case. [redacted] The DSM in its request for the next line includes a bit indicating that the line may well produce an interaction. Normally, when the DCM [redacted] gets [redacted] data for a process, it makes it available in the [redacted] shared data base and calls wakeup on behalf of the [redacted] working process. In the case when the interaction bit [redacted] is on, however, the DCM [redacted] calls a special entry of wakeup (or, perhaps, supplies a [redacted] special parameter) to indicate that the working process has apparently experienced an interaction. Wakeup will then set an interaction bit in the working process' Active Process Table (APT) entry before calling ready_him. The working process' scheduler, observing this bit, will take appropriate scheduling action and then reset the bit. The cycle is complete. To recoup: The DSM, asked for input when there is none, sets a switch indicating that an [redacted] interaction is possible. The DCM observes the switch and sets a bit in the APT indicating that the interaction has taken place. The scheduler then gives the user special action as desired and resets the bit so that the user will not be given special consideration twice for one interaction.

Although it should be clear that the scheme described will work, the perceptive reader should have the feeling that it is overly complicated. However, all of the complexity is necessary, as we now show. [redacted] We first show why both the DSM and the DCM must take part in the decision, and we then present some more subtle problems.

It would not be possible for the DCM alone to detect an interaction. Consider the situation in which the DCM indicated an interaction whenever it called wakeup on behalf of a process. The clever user will, in his working process, start the I/O system in read-ahead input mode, and then go [redacted] about a long computation. The man at the console can then type carriage return every few seconds, secure in the knowledge that he is [redacted] thereby moving his working process to the top of the queue each time. (The working process [redacted] never asks for input.) What this solution misses is the ability to know when the working process [redacted] cannot proceed without input.

[redacted]

There are also problems in trying to detect interaction in the DSM alone. One might propose the following solution: On realizing that input is requested and not available, the DSM before going blocked would set an interaction bit in the process' APT entry. This would then entitle the process to high priority on its next scheduling. Unfortunately, this solution also [redacted] can be beaten. Setting the interaction bit this way has the effect that the process gets priority on its next scheduling, no matter why the scheduler is called. The clever user arranges to have some other [redacted] friendly process send his working process [redacted] an event periodically [redacted] over an [redacted] Event Call Channel. Then the working process would ask for[redacted] input every two minutes. The person would carefully keep his hands off the keyboard so that [redacted] these input requests would all produce calls to block with the interaction bit set, and then the friendly event would result in scheduling with priority.

It should now be clear that both the DSM and the [redacted] DCM must contribute to the decision that an interaction has transpired. The DSM knows that work cannot proceed without input, [redacted] the [redacted] DCM knows that input has arrived, and both [redacted] are needed. There is [redacted] ome more problem: Consider the interactive user who, in typing, "gets ahead" of his process. That is, he supplies data faster than the process, considering the share of the processor available[redacted] to it, can eat it up. Stated differently, in the quantum available to it the process does not use up all [redacted] available input. Then on succeeding executions things are worse, since each is at lower priority. (This problem exists in CTSS.) In some sense it seems intuitive[redacted] that this user is interacting and [redacted] is entitled to [redacted] preferential treatment. Unfortunately, however, there [redacted] seems to be no way to give him [redacted] priority without opening an immense loophole to beat the system. We must stick to our decision that an interaction has taken place only if the process cannot proceed without input and if the input then becomes available.

Firm [redacted] adherence to this principle produces one more change to the algorithm as described. We said that wakeup, when called at its priority entry, would set the interaction bit in the working process' APT entry before calling ready_him. We now add the proviso that it do so only if the working process is currently blocked. [redacted] Consider [redacted] a process which blocks waiting for input, and suppose that while waiting an event arrives for it on an Event Call Channel, producing a wakeup. The [redacted] input comes from the console while the process is either ready or running as a result of this wakeup. We do not in this case give priority because [redacted] one critical requirement for an interaction is missing: that the process be unable to proceed without input. (Clearly, the process was proceeding.) This [redacted] part of the algorithm does not close any loopholes, but it [redacted] is consistent with annouмced principles.

A few problems [redacted] remain:

1. For there to be an interaction, must the input come from the command console [redacted] or is any attached console good enough?

2. Presumably the Multics equivalents of QUIT and RSTART should produce an interaction. Also, QUIT should be processed with high priority. We have yet to see how.

3. We do not yet [redacted] know how to call wakeup so as to indicate that an [redacted] interaction has taken place.