February 15, 1968

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

To: Prof. F. J. Corbató
Mr. R.C. Daley
Prof. J.H. Saltzer

From: Prof. Arthur Evans, Jr.

Subject: Requirements on Multics for the Mark-II Scheduler

Although the design of the Mark-II scheduler is by no means complete, it is clear that its successful implementation will put certain requirements on Multics. In particular, the detection of an interaction will place requirements on the I/O system and the traffic controller. Attached is a somewhat discursive document explaining what these requirements are and why they are necessary. Briefly, however, the following is required:

1. When a working process requires input from the command console, and when no such input is available in the data base shared with the device manager process, then the working process in its request for more data must include a bit indicating that an interaction may be about to take place.

2. When a device managing process supplies data in response to a request such as the above, it must call "wakeup" on behalf of the working process with a special parameter indicating that an interaction has taken place.

3. Wakeup must be able to pass on to the scheduler of the working process the fact that an interaction has taken place. This can be done via a parameter or any other way that seems suitable. It does so only if the process is blocked.

The attached paper details the reasons for these requirements.

Detection of an Interaction in Multics

Arthur Evans, Jr.

February, 1968

Introduction

The second Multics scheduler will use a multi-queue, CTSS-like algorithm. Basic to this algorithm is the idea of an interaction. Intuitively, it seems that an interaction takes place when a user, seated at a console, has completed a "think time" and has 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. The mechanism of doing the favoring is not relevant here--instead, we are concerned with just what an interaction is and how we can tell that one has taken place.

This problem has no real counterpart in CTSS. There, an interaction is defined to take place whenever a program goes into "input wait" status. (Output wait also produces an interaction -- 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 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.)

The Strategy to be Used

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

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

(b) the input actually arrives from the device.

The problem has to do with the diffuseness of the code that processes console input. Let us consider the case of input from a console with type-ahead. At some stage the working process asks for input. In due course, the Device Strategy Module (DSM) in the process' Ring 1 con-

siders the request. If there is enough available text that has already been typed, the text is merely returned to the caller. If not, the DSM must wait for text by calling the Wait Coordinator. It is at this tine that the DSM knows that there is the possibility of an interaction. (As we will see, events may transpire that result in no interaction taking place.)

The DSM actually gets input from a Device Manager Process (DMP) with which it communicates via Event Channels and shared data bases. To simplify a complicated situation, the DSM leaves its request in coded form in a place available to the DMP and then informs the DMP of the existence of the request. In the situation described in the previous paragraph, the DSM in its request for the next line includes a bit indicating that the line may well produce an interaction. Normally, when the DMP gets data for a process, it makes it available in the shared data base and calls wakeup on behalf of the working process. In the case when the interaction bit is on, however, the DCM calls a special entry of wakeup (or, perhaps supplies a 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 interaction is possible. The DMP 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.

Justification of the Strategy

Although it should be clear that the scheme described will work, the perceptive reader may well have the feeling that it is overly complicated. However, all of the complexity is necessary, as we now show. We first show why both the DSM and the DMP must take

part in the decision, and we then present some more subtle problems.

It would not be possible for the DMP alone to detect an interaction. Consider the situation in which the DMP indicated an interaction whenever it called wakeup on behalf of a process. The shrewd user will, in his working process, start the I/O system in read-ahead input mode, and then go about a long computation. The man at the console can then type carriage return every few seconds, secure in the knowledge that he is thereby moving his working process to the top of the queue each tine. (The working process never asks for input.) What this solution misses is the ability to know when the working process requires input.

But it only moves the process up.

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 the interaction bit in the process' APT entry. This would then entitle the process to high priority on its next scheduling. Unfortunately, this solution also 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 shrewd user arranges to have some other friendly process send his working process an event periodically over an Event Call Channel. Then the working process would ask for input every few minutes. The user would carefully keep his hands off the keyboard so that 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 be clear that both the DSM and the DMP must contribute to the decision that an interaction has transpired. The DSM knows that work cannot proceed without input, the DMP knows that input has arrived, and both are needed.

There is one 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 to it, can eat it up. Stated differently, in the quantum available to it the process does not use up all 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 intuitively clear that this user is interacting and is entitled to preferential treatment. Unfortunately, however, there seems to be no way to give him priority without opening a loophole to beat the system. We must stick to our decision that an interaction has taken place only if (a) the process cannot proceed without input and (b) the input then becomes available.

Firm 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. Consider 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 desired input then 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 one critical requirement for an interaction is missing: that the process be unable to proceed without input. (Clearly, the process was proceeding.) This part of the algorithm does not close any loopholes, but it is consistent with announced principles.

A few problems remain:

(1) For there to be an interaction, must the input come from the command console 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) Output wait produces an interaction in CTSS. Should the corresponding Multics effect produce an interaction?