From: Benson I. Margulies

To: MTB Distribution

Subject: Towards Unified Process Management

Date: October 31, 1980

This MTB discusses some terminology that may be useful in considering the design of Multics' process management, and suggests the broad outlines of a unified process management facility.

Process management is the creation, maintainance, and eventual destruction of processes. At this time, all process management takes place in the Initializer's process, in several subsystems. As a result there is often a certain amount of imprecision in discussions of just what is happening, and what programs are responsible. The following terms can be used to describe the existing state of affairs, and provide an unambiguous medium for future discussion.

An initiator is a process that creates one or more other processes. In the current system, there is only one initiator process, the Initializer. It in turn has three logical subsystems that act as initiators: dialup_, absentee_user_manager_, and daemon_user_manager_.

Hardcore process management consists of those interfaces to the supervisor that will create, monitor, and destroy processes. It is important to note that that is all that hardcore process management does, for reasons to appear.

A process access type, or just access type, is a classification of processes for access control. It is reflected in the existing process type tags. The "a" processes have a password given at their login, the "m" processes are requested by a process that already have given such a password, the "p" processes are created by a third party with privileged access, and the "z" processes are created by direct operator intervention.

Other typologies of processes are possible and useful. An origin type describes the means by which the process's creation was requested. In the existing system we have three: interactive, absentee, and daemon. Origin types are distinct from access types because we might well have several different mechanisms for creating processes that are equivalent for access control. The origin type is not a user visible issue. It specifies how the process was created, not how trustworthy it is or how it initializes its environment.

enviroment types classify processes according to their initial environment. In the current system we really only have two of these: interactive and absentee. The daemon processes are identical to the interactive processes from this point of view. The fact that the process attaches mr_ to a virtual terminal instead of tty_ to a physical one

Internal Multics Project working document. Not to be reproduced or distributed outside of the Multics Project.

makes no difference to it. Absentee, on the other hand, requires the retrieval of the arguments from the pit, the establishment of a special cu_$cl handler, and the like.

The final term to be defined is Unified Process Management. Unified Process Management would be a set of interfaces that would allow processes other than the Initializer to become initiators in an orderly, controlled fashion. There is good reason for the emphasis on orderly and controlled. Today, all load and accounting control takes place in the Initializer's process. Any scheme for allowing other processes to act as initiators would need to apply those limits to them.

The first question to be answered about Unified Process Management is "Why do we want it?" The answer to this question is, by and large, modularity. There are two different modularity problems that could resolved by this facility. The first is in the system itself. There are three subsystems that act as initiators in the Initializer's process. There has been some study in moving them to their own processes, for performance and other reasons. In theory, at least, a subsystem like the absentee facility can be moved out to another process so long as it has access to read and write the cdt and some other things.

If, however, we believes that new initiator subsystems or major reimplementations of any of the existing are likely, then the considerations change. The debugging (and maintainance) of these interlocked user ring subsystems is already infamous for its difficulty. The seperation of function and creation of a secure common interface could improve that situation immensely. This is a classic argument of simplicity: the fewer processes or programs writing into a database, the easier it is to isolate mistakes. The cdt is an example in the existing system. All of dialup_ keeps its per-process/channel information there, as do the various multiplexor management functions, as does (from time to time) the message coordinator. If any of them fail, they can (and do) leave trash in the others data.

This argument that the system may have new needs in the area of process creation is related to the second major reason for Unified Process Management: providing process creation as a service to ordinary user processes. It has been argued in the past that Multics process creation is too expensive for such a thing to be useful. Certainly process usage along the lines of UNIX is impractical. However, I claim that there are applications in which the user must create processes, and is willing to pay the price. Consider a TP system, where the worker and i/o process have to be created. The current practice of using absentee for such things is clumsy. Such processes are kept around long enough that the cost of their creation is insignificant beside the cost of whatever they do for a living. Unified Process Management would allow a user to specify the attachment of the standard i/o streams, the initial procedure, in short, everything that is not found in the pit. This would be a major enhancement to Multics functionality.

All this is not meant to say that we should begin an immediate crash program to design and implement Unified Process Management. All that I am proposing is that it should be adopted as an eventual goal, and that any development done to related parts of the system be done with it in mind. Future MTB's will discuss the actual design and implementation of this facility.