From: Benson I. Margulies

To: MTB Distribution

Subject: Unified Process Management .5: Cleaning Up Process Initialization

Date: January 2, 1981

This MTB describes some changes to process initialization that clean up the existing mechanisms and improve their efficiency.

All comments are solicited. The author can be reached by Multics mail as Margulies @ MIT-Multics, or Margulies.Multics on System M.

This MTB is the second in a series that began with MTB-468, and uses terminology defined in that document.

This MTB is about the pathway from cpg_ in the Initializer's process to act_proc and init_proc in ring zero to the init_admin_'s and real_init_admins_ in the user ring. These programs are old. They have code that was put in for efficiency when they were originally coded, but which now is useless or worse than useless. The separation of function amongst the various programs is not very good, and a better modularization would make the implementation of other facilities, like process management, much easier.

The current scheme for process creation is as follows: the answering service calls cpg_ in the Initializer's process. Cpg_ calls act_proc with a template PIT, and act_proc creates the process directory, creates a PIT, fills it in, and creates the ring zero address space segments. Act_proc then calls pxss to start the process. When the process starts, it runs in init_proc which determines the initial user rign procedure and calls out to it.

Cpg_'s contract is to create a process given all of the information that will eventually be in the user's PIT. It must fill in all the user-ring data bases and call hardcore to create the address space and start the process. Its primary concern, then, is the process directory of the new process. Today it is hardcore that creates the process directory. This is because the process dir entry name is developed from the process id, and the process id includes the offset of the APTE in tc_data. Act_proc, the responsible hardcore program, generates its half of the process id after allocating the APTE. The other half of the process id is a number provided by cpg_, from a pseudo-clock in the answer table.

It is proposed to make the generation of process id's a supervisor function. In the current implementation the supervisor trusts the uniqueness of a process id partially supplied by the user ring for all its locking, but generates a guaranteed unique lock id for the user ring's locking! Act_proc will use a pseudo-clock in internal static for the unique portion of the process id.

It is important that act_proc not create the process directory. Only some of its contents are managed by hardcore process management. Others contain information passed from the initiator to the new process. The PIT is the case in point. The data in the PIT is of no interest to act_proc. Yet, in the current scheme, any change to the PIT format potentially requires changing cpg_, act_proc, init_proc, and then the init_admins_ and user_info_. Allowing cpg_ to create the process dir would allow it to create the PIT, and changes to its contents could be made without having to change hardcore programs.

There are two ways that cpg_ could create the process directory. One is to create it with a name other than a representation of the process id. An alternative is to call hardcore process management to get a process id, then create the directory, and then call hardcore to create the address space and start the process.

The first alternative is cleaner. The directory could be named following the process creation time, which is available in the user tables to programs that want to find process directories. This would break private tools that find process directories from process id's. A new interface will be provided that converts a group id to a process id, which will isolate these tools from the details of the implementation of process id's.

Init_proc is the other ring zero program in the direct path of process initialization. Today it looks in the PIT to find the initial procedure. It has an entire reimplementation of expand_pathname_ in lire to expand the string from the PMF. It would be simpler to have cpg_ expand the initproc keyword in the PDT. This data could be passed to act_proc, and left in the pds. This would be consistent with the existing policy that ring zero looks at the pds, and the user ring looks at the PIT.

Once act_proc and init_proc's interests in the PIT is removed, the data structure in the PIT can be cleaned up. This structure is currently a rat's nest. It has aligned character strings that should be unaligned, single aligned bits lying all over the place, and is not at all extensible. A reorganization would make changes, like new PMF keywords, much easier.

The next step in the chain are the init_admin_'s and real_init_admins_. Today, there are three apparently distinct alm programs (user, absentee, and daemon init_admin_), each of which calls a pl1 program (the corresponding real_init_admin_). This was done for efficiency, as it avoids keeping the stack frame of the pl1 code that does the work around for the life of the process. The same effect is now available with non_quick internal procedures in pl1.

It is proposed to combine the init_admin_'s and real_init_admins_ into one program. Right now they are nearly identical. User_real_init_admin_ and daemon_real_init_admin_ are identical. Absentee_real_init_admin_ establishes a cu_$cl hook, and find the arguments. The other difference in absentee is the cpu and spending limit control established by absentee_real_init_admin_. Current plans call for this responsibility to be taken over by the absentee_user_manager_. Even if it remains, it is a trivial deviation from the actions of interactive processes. It seems likely that the different environment types offered by the standard system are likely to get more alike as time goes on, rather than more different. Thus from a maintenance point of view it is reasonable to collapse the existing three nearly identical programs into one. This is also a performance enhancement; the major performance problem of Multics is paging, and the current situation has three nearly identical sets of pages that have to come in to start the three different types of processes. I measured the cpu and page faults for a version of user_init_admin_ using a non_quick internal procedure instead of the alm/pl1 design, and found significant savings.

Another programming fossil in the real_init_admins_ is the initialization of the three basic syn_ attachments. Currently the program ios_$ios_quick_init is called. It calls syn_attach_ directly instead of iox_$attach. I have metered a version of user_init_admin_, and found that this is not significantly cheaper than doing attachments with iox_ in line. So there seems to be no point in preserving this mechanism. We should remove it, if only to get reasonable error messages when the attachments cannot be made.

All the work described above is perhaps three person-days of programming. There would be another two days to run development and debug. An interface to fetch the creation time from the user table would be another day. The whole project could be budgeted at a weeks's work. Documentation would consist of updating the SWG description of process creation, which could be done from this MTB, or the developer could take another day and rewrite the SWG text.