To: MTB Distribution

From: Benson I. Margulies

Date: November 11, 1980

Subject: Message Coordinator Considerations part 0:
New Considerations for the Message Coordinator

This MTB is the first of a series on the Message Coordinator. It surveys the existing implementation, and discusses some new needs that might be met by a new one. Future MTB's will discuss the details of such an implementation and its applications.

Any and all comments are solicited, and may be directed to the author by Multics mail as:

Margulies @ MIT-Multics
or
Margulies.Multics <@ System M>

or by conventional communication:

Benson I. Margulies
Cambridge Information Systems Lab
HED MA29
575 Tech Square
Cambridge MA o2139

This MTB is about the Multics Message Coordinator, often referred to as the "mc" or "the cordinator". The coordinator was originally devised to deal with the output produced by the various system processes. While all of the functions of Daemon printing, tape backup, and the like were clearly system functions, they were given their own processes to avoid cluttering the Initializer's process. The result of this was a row of terminals in the machine room, each connected to its own system process. This was clumsy and required too many terminals.

The obvious solution to this was a facility for controling multiple processes from the system console. All it had to do was take some format of input that gave the destination process, and collect all the output and print it on the console. This was called the message coordinator.

Soon, however, other things were asked of the coordinator. It was neccessary to reduce the traffic across the BOS console, which was (is!) slow and at times unreliable. So support was added for the use of ordinary terminals as additional system consoles. A routing scheme was devised for distributing the output to the various possible destinations. The coordinator was not mandatory in its initial release. As a result, the old means of handling the BOS console had to be preserved, and the message coordinator was to be grafted on top.

All of these functions, except running the system without the coordinator, are still needed. Some new functions are needed, and some of the programming techniques that made the original implementation efficient have become obsolete.

The current implementation can be thought of a consisting of an input routing mechanism, an output routing mechanism, a stream data utility, an operators' interface, and a terminal manager.

The input routing mechanism has two features: the ability to deliver a message to a specified Daemon process, and the ability to "send a quit" to a Daemon. All terminal input is logged in the global answering service log rather than any log associated with the recipient process. Various restrictions can be imposed on terminal input: a terminal can be unrestricted, restricted to only deal with daemon replies, or prevented from supplying any input at all.

The output routing mechanism is designed around the idea of "Virtual Consoles." These are the virtualized equivalents of that original row of terminals in the machine room. The output generated by any particular daemon is directed to one of these "vconses," and the vconses are in turn connected to physical terminals. The table that connects daemon output to vconses is called the Message Routing Table, or MRT, and the table that connects vconses to terminals is called the vcons_tab. A vcons can be connected to a log instead of a terminal; this causes the information sent there to be put in a standard format system log in >system_control_1. The code also provides for "sink" vconses that just discard whatever they receive. The net result of all this is that output of daemons is directed to logical destinations (the vconses), and then the logical destinations are multiplexed onto the available physical devices. Thus an administrator can redirect all the output logically grouped by

a vcons without having to change all the individual routings. She is isolated from much of the difficulty of dealing with the small number of output devices.

The stream utility is just an I/O Module called mr_ that reads and writes messages via the input and output routing schemes. It replaces tty_ as the primary attachment in daemon processes.

The operator's interface is the most complex part of the coordinator. coordinator. In some sense there are two parts. The first consists of the operator commands that actually control the coordinator. The second part is the general control of all the other operator commands. If the coordinator was only concerned with the delivery of messages, then the system would have an input destination to which operator commands (like abs, or word) were sent, in the same way that commands are sent to daemons. But having to prefix all commands to the system with the "reply" command for the coordinator to tell it where to deliver them would be an unpleasant interface, to say the least. Instead, most of the things typed on the consoles are directly delivered to the program execute_ec_command_, which executes them. Thus in order to implement the restriction of a terminal to talking only to daemons, or only to a particular daemon, each input line must be scanned to determine whether it is one of the special subset allowed to restricted consoles. Thus the coordinator must have knowledge of the general format of operator commands.

All this must be rearranged for admin mode. In admin mode the console or terminal on which the admin command is given is connected to a standard listen_ in the Initializer's process. Admin mode is entered when an operator gives the "admin" command and supplies the correct password. In this mode, all input lines are directly interpreted as Multics commands lines until the "admin_mode_exit" command is given. To do this, the BOS console or coordinator terminal on which the admin command is given is connected up to the switch user_i/o in the Initializer's process, giving a nearly normal Multics environment with the full power (and danger) of the Initializer's process. The program borrow_tty_from_mc_ temporarily patches the terminal entry in the coordinator terminal manager's database (mc_anstbl) to prevent routed output from being sent to it, connecting the switch user_i/o to it over an appropriate IO module, and calling listen_.

The terminal management function is an event-driven facility based on as_tty_. The terminals belonging to the coordinator are found in two ways: MC service type, and "dial system." MC service type is a special service type in the cdt that causes the terminal to be available to the coordinator rather than for login. MC service is functionally identical to slave service. The only difference is that processes other than the Initializer may not priv_attach them, even if they have access to the neccessary ACS. The Initializer serves the registered dial-id "system" to allow any login service terminal to be connected to the coordinator.

MC service type terminals are automatically attached when the operator "defines" a vcons to send output to them, using the operator define or redefine command. Terminals dialed to "system" are attached when the operator accepts them with the "accept" command. Terminals are detached if and only if the operator uses the "drop" command on them. Coordinator terminals are not

attached with tty_. Instead, they are entered in an table called the mc_anstbl, and managed with direct calls to the hcs_ tty entrypoints. Input from the terminals is scanned in order to implement the input command restrictions.

There are many reasons for replacing this implementation. The first is that it is old code. The technology used dates from expensive internal procedures, restrictions on the efficiency of calls involving certain data types, and the like. It has many outstanding problems, and is difficult to maintain. This is itself would not be a reason to change the design. The problem is that the current implementation depends on a series of very unmodular techniques. In particular, the coordinator depends on knowledge of data in the cdt and other Initializer databases. The conceptual basis of the design does not provide the facilities to use a cleaner method. Admin mode is the best example. To do admin mode without patching the cdt, it is neccessary to have a way of sending messages to and from the Initializer in a symetric way. The current distinction between input, which can only be delivered to daemons, and output, which is routed, prevents this.

A second reason is that the coordinator appears to be unneccessarily expensive. Any message sent sets off a chain of wakeups in the Initializer's process provoking the delivery of the message. It should be possible to design a mechanism in which the only wakeups needed are to inform each destination of a message that it is available. It should not be necessary to have a protocol chain in which all the involved processes receive wakeups.

An other reason for a new implementation is to provide new functionality. There are several examples. One of the original purposes of the coordinator was to allow a reduction of traffic across the BOS console. Unfortunately, the coordinator is entirely in the user ring. This requires all participating processes to be trusted system processes. As a result, any inner ring subsystems that need to send messages to logs or operators must pass through ring 0 syserr. The configurations in the field are getting larger and larger, and more and more subsystems need to log information or print it where the operator can see it. So it is important to reduce the syserr traffic. A good example is RCP. Ring 1 RCP could easily send many of its messages via the message coordinator if any process could talk to the coordinator.

DSE has a concept called NOI in DSAC, which allows operators anywhere in the network to send control messages to any of the machines. In the current coordinator, only the Initializer process can generate input to daemons. Thus to do NOI the Initializer would have to have the DSA connections neccessary to serve NOI. Worse, the coordinator does not extend well to the idea of sending commands to other machines (eg, UNCP). A more general facility for routing messages could provide most of the needed functionality for NOI.

Of late it has become clear that the concentration of functions in the Initializer's process has resulted in performance problems. Thus it is important to be able to remove things from the Initializer's process. The current strategies of message routings and terminal management, as outlined above, are not suited to removal from the Initializer's process, because they work by directly manipulating Answering Service data. A new implementation of

the coordinator could provide the same functions without these dependencies, and could be removed from the Initializer's process.

It seems likely that all of the coordinator except the management of terminals could be made passive. That is, no process other than the sender of the message should need to run to delivery it. All that is left is the i/o to system control terminals. Unfortunately, to move that task out of the Initializer's process requires a way to get the information (the operator commands) to execute_sc_command_. This program must run in the Initializer's process because many of the operator commands deal with subsystems like volume management which really have to be in that process. The current design provides no scheme for sending input from a non-Initializer process back to the Initializer in a way usuable for this function. Thus to decentralize the terminal management we must have a more general message delivery system.

Finally, there is considerable interest in an upgraded facility to replace send_admin_command. The current facility just allows administrators to "shoot their commands into the air." The result of a sac'd command do not return to the sending process. In fact, they do not even always make it into a log. A better coordinator could allow a facility along the lines of user_telnet where administrators could temporarily "connect their terminals to the Initializer."

We could try to design a coordinator that does what it does, these new functions, and nothing else. We could also try to design a general facility for secure, routed message delivery that provides the current functionality, allows the implementation of the facilities descussed above, and leaves the door open to other development in the future. It seems clear that the second alternative is better so long as the "general facility" really is general, and not merely restricted in different ways. The form of such a design will be the subject of the next MTB.