To: Distribution

From: T. H. Van Vleck

Date: 11/01/73

Subject: The dial facility

This memorandum describes a proposed improvement to the answering service which will allow support of multiple terminals attached to a single process.

There are several justifications for this facility:

1. Several users have access to special devices, such as graphic terminals, which they wish to use as slave devices attached to their user processes, in addition to a regular typewriter terminal.

2. Other users are comtemplating true multi-terminal support programs, in which a single process may respond to information-retrieval requests or message transmission requests from several typewriters. The overhead involved in creating a process for each terminal is prohibitive in some cases.

3. The system itself can make use of such a facility for the support of remote printers attached to an IO daemon process, for connection of multiple terminals to the initializer process, and for testing of new versions of the initializer message coordinator and answering service.

A limited version of this facility has existed within the answering service for several years, and is used to support some of the functions in point 3 above. Similar facilities were implemented in both CTSS and CP/67, and proved valuable.

With a modest amount of effort, the rough edges in the current implementation can be eliminated, and the facility improved to the point where we can make it available to users to support the objectives listed above.

User Interface

The dial facility is invoked to make a dial connection by calling the computer from a terminal in the usual fashion. After the system has typed its greeting message, the user types

dial DIAL_ID

instead of issuing a "login", "enter", or "enterp" request. Currently, the DIAL_ID may be either "system", to give the channel to the message coordinator, or "Person_id Project_id" to dial a user process.

The system validates the request, and if the connection is permitted, responds with the message

connected

and then places the channel under control of the master process. The master process then performs whatever input-output operations it wishes on the channel. If the master process logs out or terminates, a message is typed on the channel and the answering service waits for another dial request, or a login. If a dialed channel hangs up, the answering service informs the master process.

To become a master process, a user process must request this ability from the answering service, via a call to "attcon_". If the answering service permits the process to become a master process, it remembers an event channel sent from the master process which will be signalled whenever a dialed channel connects to the master process or hangs up on the master process.

When a channel issues a dial request, the answering service checks that the named master process exists and is ready for dial service. It then attaches the terminal to the master process (via a call to hcs_$tty_new_proc) and signals the master process to inform it of the dialup. The master process must then attach the terminal channel by calls to the ring-0 TTY DIM.

The master process cannot use an attach call through the I/O switch which specifies the "tw_" outer module if it intends to have outstanding read requests on more than one terminal at a time, because the ring-4 TTY DIM will call block and wait for input if nothing is ready to read on any channel. In other words, the master process would "hang" waiting for input from one channel, even if input were available from another channel. Special code, resembling "dialup_" or the message coordinator, must be run in the master process to allow the terminal-input handling to be done asynchronously, as a result of an event-call. While this code is not difficult to write, it does depend on the internal interfaces used by the TTY DIM, and programs which use these interfaces will be affected by any changes to the TTY DIM

(such as the proposed rewrite to move many functions into ring 4). It is not difficult, though, to provide users with two or three subroutine packages which can be used to drive multi-terminal environments, or to support output-only dialed terminals, which do not involve the user in the system's internal interfaces.

Permission to have dialed consoles is controlled by a user control attribute, "dialok", which may be given by the system administrator to any or all projects, and by the project administrator of a project with "dialok" to any or all users on the project.

Dialed channels are charged to the master process's user ID in the same way that ARDS surcharges are added. The charge is per terminal hour.

Deficiencies in the Current Implementation

In the current implementation of the dial facility, the DIAL_ID identifier used in a dial command may be either "system" or a person ID and project ID, separated by a space. A "dial system" request is taken to be a request to make the terminal available to the message coordinator; there is no difficulty with this variation. However, specifying person and project has some undesirable attributes:

1. What should be done if more than one instance of Person.Project is logged in? Currently, we take the first process we find with that name and project as the intended target. If that process has not enabled dial communications, no connection will be made.

2. Suppose that we wish to tell the user to dial a generic name, such as "dial basic" or "dial GRTS"? The person and project identifier is not really the user's concern. It would be possible to have the one-argument form default to assume that the person name and project name were the same, but this would require us to set up whole projects for a single user.

3. Consider the proposed use of dial for the remote I/O daemon. Here, we wish to have multiple instances of IO.SysDaemon logged in and waiting for a channel to dial to them. We want a "hunt group" effect, so that each waiting server process will snag one dial request, and then appear "busy" so that the request will be reflected to the next process.

These needs can be satisfied neatly by creating a new system data base, called "dial_table". This table is a list of dialable names. Each entry in the table contains

identifier	The DIAL_ID specified in a dial request
access list	The list of legal users who may be the serving process
max lines	Maximum number of dialed channels
current lines	current number of dialed channels
active flag	ON if a serving process exists
process id	process ID of the serving process
event channel	master event channel, which the dial facility uses to signal wakeups and hangups from attached terminals to the serving process.
hunt indicator	ON if multiple serving processes exist
hunt pointer	points to next server in the dial table for this hunt group.
password	if nonblank, a password will be requested before the user is permitted to make the dial connection. (If a random value for this password is generated at system startup time for the "system" entry in the dial table, the operator "accept" command can be eliminated.)

This table should be handled in much the same way that other system-administration tables are managed. That is, the system administrator should be able to install new copies of the table while the system is running; utilities should be available for listing the table, etc.

The other major deficiency in the current dial facility is that the communication path from the serving process to the dial control program in the answering service is too narrow-band. All that a dial server can send now is a single 72-bit wakeup message, which is interpreted as either an event channel ID, or, if all zero, as a request to terminate dial service. There is currently no room to specify the dial channel identifier, the maximum lines wanted, the hunt flag, and other desirable parameters.

This problem should be solved by introducing a per-system message segment to be used like the absentee queues: the serving process calls (a new entry of) "attcon_", specifying all the necessary information in its calling sequence, and a message is placed in the message segment. A wakeup is then sent to the answering

service, with the 72-bit ID of the message in the message segment. The return channel can continue to be a simple wakeup from the answering service to the serving process, since there are very few possible messages going in that direction.

Once the dial table and the message segment are introduced, all requirements of the system itself can be satisfied. Applications software must then be created to meet whatever user requirements seem most important. The case of output-only slaves is the easiest; this case may, in fact, be handled under the current I/O switch. A demonstration program for the request-processing case was written some time ago, and could be brought up to system standard with moderate effort: this program would establish dial service and then call a user program with any input lines typed by slave consoles; the user program would return the "response" to be typed back to the slave. Another possibility is to adapt the current message coordinator and to provice a suitable user wrapper so that full operation of multiple consoles could be provided as a package. Finally, it might be possible to adapt existing code or write new code to simulate GRTS or other message-processing systems, in a special process provided by the system, which GCOS or Multics processes could then negotiate with to perform multi-terminal I/O.