To: Distribution

From: Paul A. Green

Subject: Restructuring the Way the Answering Service Handles Terminals

Date: March 5, 1975

This memo proposes several changes to the way Multics handles terminal devices. The restructuring is prompted by the inability to add new features for the Access Isolation Mechanism under the current structure, and by the long-range goal of better terminal handling.

I. Overview

Currently in Multics two data bases specify which terminals are attached to the system. (The word "terminals" is a misnomer; channels would be more accurate, as the Mohawk is included here). For hardcore programs such as the ttydim and the Datanet 6600 Front End Processor, the LSLA and HSLA cards specify every hardware channel attached through the 6600. These channels are available for attachment by any process, once permission has been granted by the answering service. The answering service has its own list of channels, called the lines file, which specifies the symbolic name of every channel it should "listen" to for dialups. Channels specified in the config deck, but missing from the lines file, may be attached by the first process which calls the supervisor. (The Initializer terminal and the Mohawk channel are such channels). At system initialization, the Initializer assigns all channels in the lines file to itself, and waits for them to dial up. Each channel has a unique event call channel, and its own entry in the answer_table. When a user logs in, the answer_table entry for that terminal completely describes both the terminal and the process.

Since terminal information and process information are stored together in the answer_table (and since many programs reference the answer_table directly) there are inherent difficulties in the present multi-terminal per process support, and in changing over to a better method.

For the Access Isolation Mechanism we would like to associate several new fields with each terminal. These include the terminal access class, expected answerback, an audit selectivity

---

flag, and a hardwired flag. The access class will be used to control the maximum authorization of a process logged-in over that terminal. (The authorization of the process must be less than or equal to the access class of the terminal). The expected answerback will be used (for hardwired terminals) to verify that the correct terminal is connected to the line, and the audit select flag will be used to provide a "physical security breach" message if the authorization of the person logging-in is less than the access class of the terminal.

Unfortunately, there is not enough room in the present answer table entry to add these variables cleanly. The next section proposes a new data base devoted exclusively to channels.

II. The Channel Table

The present lines file will be extended into a Channel Master File that will specify the attributes of each channel. This ASCII segment, similar to a Project Master File, will be compiled into a binary segment called a Channel Definition Table. This binary segment will be used by the answering service to hold all information about every channel on the system. Variables now kept in the answer_table will be moved to the channel table; the answer_table will hold only per-process information.

Appendix A gives a sample Channel Master File and Appendix B gives the proposed declaration for the Channel Definition Table (CDT).

Fourteen programs currently reference the twelve variables which will be moved to the CDT. A substantial amount of editing and testing will be required to move these variables, therefore an interim solution is also being proposed.

Interim Channel Facility

The present answer table entry (ATE) has about six (6) words free per entry. This may be enough space to squeeze in the new fields, if so, we will implement the interim facility as an extended lines file which is parsed, and the new variables squeezed into the ATE for that channel. This will save us from having to modify the existing programs which reference the channel variables in the ATE. At some future point we can come back and split off the channel info from the ATE.

III. The Dial Facility

In MTB-013, Tom Van Vleck proposed some improvements to the present Multics dial facility. We need a quick means of solving one of the problems he mentions; the inability to dial to a

specific instance of a process which has several login instances. Since we don't have time to implement the full dial facility (allowing registered dial ids, multiple servers per dial id, multiple dial ids per server, etc.), we are proposing only an extension to the present user interface which will allow us to specify a 3rd argument to the dial command. Since the person name and project name must still be given, the 3rd argument (called the dial qualifier) need only be unique within one Person.Project combination, and no access control is needed on the name.

Under the extended dial mechanism, a process will pass both an event channel and a dial qualifier to the answering service. Messages from the answering service will continue to be limited to 72 bits; this is adequate for now.

We also need the ability to dial to a daemon process; this will also be implemented at this time.

IV. Request Dispatcher Proposal

In order to pass this dial qualifier to the answering service a wider path than the present 72 bits is needed. Other answering service facilities could also make use of a wider path, therefore we propose that a message segment be created in >system_control_1 for the purpose of passing requests to the Initializer. An event-call-driven program will dispatch these messages (all having a common header) to the proper module (absentee, dial, etc). Initially, only the dial facility will use this path. See Appendix C for a declaration of the request header, and see Appendix D for a declaration of the dial request.

V. Summary

We will implement the request dispatching mechanism in full, but restrict its use to communicating with the new dial facility.

We will extend the dial facility to allow a process to specify a dial qualifying name. The dial command will be modified to search for a match on Person_id, Project_id, and dial qualifier. The dial facility will also be extended to allow dialing to daemon processes.

We will add new variables to the lines file and answer_table for the Access Isolation Mechanism. In a second stage we will split the terminal-control information now in the answer_table into a separate channel table.

APPENDIX A

```
/* A Sample Channel Master File */

name:                   tty001;
  access_class:         system_high;
  comment:              "Bldg A, Room 432, Communications Office.";
  charge:               tty;
  service:              dialup;
  answerback:           "GT1";
  attributes:           audit, hardwired, set_modes;

name:                   tty002;
  access_class:         system_low;
  comment:              "555-1234, Modem bay A, shelf 3, LSLA B";

name:                   net001;
  comment:              "Network TELNET line 1.";

name:                   net002;
  comment:              "Network FTP line 1.";
  charge:               ftp;
  service:              ftp;
  attributes:           ~set_modes;

end;
```

In this sample CMF, channel "tty001" is a hardwired terminal cleared to system_high. It's answerback must be "GT1", it is a normal "dialup" line (i.e., logins or dials are allowed; this is the default), it is charged at the "tty" rate (this is the default), access class errors will be audited, and initial modes will be set by the answering service (this is the default...it includes setting tabs for Terminets).

Channel "tty002" is a "normal" dialup line; it is not hardwired, instead it has a phone number one can call to reach it. It's access class is system_low, as would be expected for a telephone line (system_low is the default). The comment gives some identifying information about the location of the modem which services this particular number.

Channel "net001" is a "normal" network dialup line; except for the fact that it is connect to the network instead of the telephone system, it is identical to "tty002".

Channel "net002" is a network file transfer line; logins and dials are <u>not</u> permitted on it, instead a special program handles all traffic over it in the format required by the network file transfer protocol. Initial modes will not be set, and usage will be charged at the "ftp" rate.

```
/* BEGIN INCLUDE FILE ... cdt.incl.pl1 */

/* Channel Definition Table.
   This table lists all of the hardware channels (ports)
   connected to the system, and maintains the attributes
   of each one.

   PG 741230
*/

dcl  CDT_version fixed bin internal static initial (1);

dcl (cdtp, cdtep) ptr;

dcl 1 cdt based (cdtp) aligned,    /* all system channels */
    2 author like author_dcl.author, /* standard header */
    2 max_size fixed bin,          /* # of cdte's in 255K */
    2 current_size fixed bin,      /* number of last cdte. */
    2 version fixed bin,
    2 freep fixed bin,             /* chain of free cdte's */
    2 n_cdtes fixed bin,           /* number of used cdte's */
    2 pad (40) fixed bin,          /* pad header to 64 words */
    2 cdt_entry dim (4079) like cdte; /* # of cdte's in 255K */

dcl 1 cdte based (cdtep) aligned, /* a channel */
    2 state fixed bin,             /* 1=used, 2=free */

/* The following variables represent constant data
   which should not be reset */

    2 name char (12),              /* ASCII name of channel */
    2 access_class bit (72),       /* access class of channel */
    2 comment char (48),           /* info about channel */
    2 charge_type fixed bin,       /* billing group */
    2 service_type fixed bin,      /* service group */
    2 answerback char (8),         /* answerback expected */
    2 duplex char (4),             /* half, full */
    2 flags,
      3 ck_answerback bit (1),     /* ON means check answerback */
      3 audit_access_error bit (1), /* ON means do auditing */
      3 hardwired bit (1),         /* ON means it is */
      3 set_modes bit (1),         /* ON means to do it */
      3 execute_initial_command bit (1), /* ON means to do it */
      3 pad bit (31)) unaligned,
    2 initial_command char (64),   /* pseudo first input line */

/* The following variables represent dynamic control info,
   and may be used as necessary. */

    2 event fixed bin (71),        /* event call channel */
    2 twx fixed bin,               /* channel device index */
    2 state fixed bin,             /* channel state */
```

```
    2 tty_type fixed bin,          /* channel type code */
    2 tty_id_code char (4),        /* channel id (answerback) */
    2 baud_rate fixed bin,         /* 110, 133, 150, etc. */
    2 process ptr unal,            /* ptr to ATE of process */
    2 next_channel fixed bin,      /* (this process only) */

/* The following variables are kept for metering purposes. */

    2 n_dialups fixed bin,         /* # times channel dialed up */
    2 n_logins fixed bin,          /* # logins on this channel */
    2 dialed_up_time fixed bin,    /* total time dialed (secs) */
    2 dialup_time fixed bin (71), /* time of present dialup */
    2 pad (9) fixed bin;           /* pad to 64 words per entry */

/* Values for cdte.service_type field */

dcl (dialup_service init (1),      /* login or dial */
     ftp_service init (2)          /* file transfer service */
     ) fixed bin internal static;

/* END INCLUDE FILE ... cdt.incl.pl1 */
```

```
/* BEGIN INCLUDE FILE ... as_request_header.incl.pl1 */

/* This include file declares the standard header
   for requests to the answering service/user control
   request dispatcher.

   Written 750304 by PG
*/

dcl  as_request_version fixed bin internal static initial (1);

dcl  dial_server_type fixed bin internal static initial (1);

dcl 1 as_request_header based aligned,
    2 version fixed bin,                  /* version number */
    2 request_type fixed bin,             /* what to do */
    2 reply_channel fixed bin (71);       /* who to tell */

/* END INCLUDE FILE ... as_request_header.incl.pl1 */
```

APPENDIX D

```
/* BEGIN INCLUDE FILE ... dial_server_request.incl.pl1 */

/* This include file declares the data structure to be
   passed to the dial facility to allow a process
   to accept dials.

   Written 750304 by PG
*/

dcl  dial_server_request_version fixed bin int static init (1);

dcl 1 dial_server_request aligned based,
    2 header like as_request_header,     /* std header */
    2 version fixed bin,                 /* version number */
    2 dial_control_channel fixed bin (71),
                                         /* AS sends msgs here */
    2 dial_qualifier char (22);          /* used by dial cmd */

/* END INCLUDE FILE ... dial_server_request.incl.pl1 */
```