TO: Multics Distribution
FROM: C. Marceau
SUBJECT: Answering Service
DATE: 06/10/68

The attached revision of BQ.2.01 is intended to document the implementation of the answering service in more detail than appeared in the original. Other changes:

1) There is only one user control process rather than one per user; a reflection of the rethinking of user control initiated by Stu Feldman.

2) The communication line interface module has disappeared.

Published: 06/10/68
(Supersedes: BQ.2.01, 06/15/67)

Identification

The Answering Service
H. M. Deitel, J. H. Saltzer, H. J. Hebert

Purpose

The Answering Service is a system process which monitors the communication lines from which a user may log in to Multics. This section describes the functions and implementation of the Answering Service.

Summary of Answering Service Functions

Following system initialization, the Answering Service is created by the System Control Procedure. It initializes itself by attaching to itself all of the communication lines from which users may attempt to log in. Next it places all of these lines except one in the offhook state. The exception is a line with the registry file name, "op_channel". This line is placed in the onhook state so the system operator can log in. The event channels which are necessary for a dialin to occur are created for this line only. Then event channels are created so the system operator can signal the get_line and set_line commands. A shutdown channel is created, and system control is notified that initialization is complete. Then Answering Service waits for the shutdown signal from system control.

Meanwhile the system operator logs in as an ordinary user, identifies himself to system control, then presumably issues a set-line command, specifying the initial configuration of active lines, i.e., the particular subset of all the communication lines from which Answering Service is to accept "dialup" signals and ultimately allow users to attempt to log in to Multics. Once the initial configuration of lines is set the Answering Service is prepared to handle incoming interrupt signals from these lines.

In attempting to gain access to Multics, a potential user dials up on a DATAPHONE data set (or turns on some device attached to a dedicated line with data sets). The resulting system interrupt causes the Answering Service to wake up and determine from which line the interrupt came. The Answering Service then signals the User Control Process, creates an event channel to receive a return signal from this process, and tells the User Control Process over which channel the user is trying to log in. Upon successfully

logging in, the user proceeds to utilize the system. When he is finished using the system, he types the "logout" command which eventually causes the user control process to send a return signal to the Answering Service. The Answering Service reattaches the communication line to itself and calls wait to await further wakeups.

While Multics is running, the system operator may want to reset the configuration of active communication lines, and at system shutdown time he specifies that no lines should be active. To set the configuration, the system operator types a command specifying either the number of lines of each hunt group* which should be active, or the Registry File names of particular lines and whether these lines should be active or not active. The Answering Service sends calls to the I/O system to "activate" or "deactivate" particular lines until the desired configuration of lines is achieved, and then calls wait to await further wakeups. If a registry file name was specified and the line was in an inuse state then the make_busy_switch in the Line Table is set to the desired state so that when the line comes out of the inuse state it is placed in the desired state. If a hunt group was specified and the desired configuration could not be achieved then an entry is made in the make_busy_table. When a line goes from either inuse to onhook state or disabled to offhook or no_answer to offhook then the make_busy_table is checked. If its hunt group has an entry then that line is placed in the desired state and one is subtracted from the number given in the make_busy_table. At system shutdown time, the Answering Service deactivates all lines and then signals to the System Control Procedure that shutdown has been completed.

Status of a Communications Line

Before the System Control Process makes its initial call to the Answering Service, the communications lines have not as yet been allocated to the Answering Service Process (or any other process, for that matter) and hence a user cannot successfully dial up and log in to Multics. Once the communications lines have been attached to it, the Answering Service Process maintains each line in one of five possible types of status. These status types reflect

---

*A "Hunt Group" is a group of telephone lines with the property that a caller dialing one number will be connected to any non-busy line in the group. Thus if there are 15 lines from the telephone exchange to the computer, of which 13 are busy, a caller does not need to dial several times to find one of the two non-busy lines.

whether the line is currently in use, ready to be used, or cannot be used until made ready, and are indicated by the status switches for the line in the Answering Service data base. These status types are:

1. "in-use" - The line is in use by someone and is currently allocated to that user's user-process-group. No other user may successfully dial up on this line until this user has relinquished control over it by logging out.

2. "on-hook" - The line is not currently in use. It is allocated to the Answering Service Process and the data-set is in automatic answer state. If a dial up occurs the Answering Service will wake up and set up the user so that he may log in.

3. "off-hook" - The line is not currently in use. It is allocated to the Answering Service Process which has forced the data set into the busy state.

4. "no answer" - The line is not currently in use. The data-set is not in the automatic answer state, so if a user dials up he will hear ringing but the computer will not answer. Unless directed by the system operator, the Answering Service will avoid placing switched lines in this state, using the "off-hook" state instead.

5. "disabled" - The state of this line is uncertain. The Answering Service is not able to maintain this line in proper operational status. When the system operator notes that a line is disabled, he sees that it will be serviced.

We may now define more precisely some previously used terms. Lines in the "in-use" or "on-hook" state are termed active, while lines which are "off-hook", "no_answer", or "disabled" are considered inactive.

When the Answering Service Process receives a call from the system operator to set (or reset) the configuration of active lines, it will first check the status switches in the Answering Service data base to see which of the above status types is applicable for each line. From this information it is able to avoid attempting to put a line "on-hook" if it is already "on-hook" and similarly for the other status types. If a line is currently in the "in-use" status and the System Operator has requested this line to be made "inactive" (i.e., placed in the "off-hook" or "no-answer" state), then the Answering Service notes the request and waits until a return signal is received from the user-process-group in control of this line before it proceeds to place the line "off-hook".

Types of Communication Lines

The Answering Service monitors those GIOC channels from which a user may log into the system. Each of these falls into one of the following categories:

1. - dedicated line with data sets - This consists of a communication line running from the GIOC to a terminal device with a data set on each end of the line. At the GIOC end there is a data set adapter between the data set and the GIOC. Over this type of line, the user causes an interrupt to occur by turning on the power switch at his terminal. This interrupt causes the Answering Service to wake up if the line is in onhook state.

2. - lines with data sets connected to a Private Branch Exchange (PBX) - On the GIOC side of the PBX, there is a fixed number of lines to the GIOC. On the opposite side of the PBX, there can be any number of lines running into the PBX from remote terminals. A user wishing to gain access to the system dials a special telephone number which is unique to his terminal device and communication line types. There may be many terminals of this same type on the terminal side of the PBX and on the GIOC side there may be many lines to the GIOC which are prepared to handle this type of device and line combination. When the user dials his particular telephone number, the PBX will hunt through the GIOC lines to find an unbusy line corresponding to this particular telephone number. If it finds such a line it connects the line to the terminal with the line to the GIOC and sends a ring signal, as a result of which a dialup interrupt occurs, eventually waking up the Answering Service. The user may dial a number which is inappropriate for this type of device. To discover such special cases the I/O system runs tests to determine if a proper connection exists.

The Answering Service Interface with I/O

When the Answering Service attaches a communication line it specifies in its call to the I/O system that the line is a typewriter channel. The attachment is made with a standard typewriter device interface module, except that the I/O System tolerates the absence of a device from the channel. The device interface module has the ability to control data set control leads (e.g., to hang up the data set) and to verify the presence of the proper typewriter type for the channel when there is a dialup. Further information on the I/O interface may be found in BF.3.05.

Answering Service Initialization

Following system initialization, system control issues a call to create_proc (see BJ.8.01) to create a process called the Answering Service Process. The new process' first job is to find an ascii segment in the answering service directory (pathname: >system_process_reserved_storage>ans_service) and to convert the information into a PL/I structure in the segment asplt. The ascii segment is named cltext and contains information on all the communication lines available to the system. The segment is a sequence of lines, each line with the following two items separated by semicolons:

(1) registry file name of the line;

(2) hunt group of the line.

An extra semicolon at the end of the last line signifies the end of the table.

The segment, asplt, is created by answering service in the answering service directory and contains the following PL/I structure.

```
declare   1 aslt based (p);          /*Answering Service Line
                                       Table*/

            2 lines-avail fixed      /*number of lines onhook
              bin (17),                or inuse*/

            2 n_busy_entries fixed   /*number of entries in
              bin (17),                make busy table*/

            2 entries (20),          /*make busy table*/

              3 ht_grp char (32),    /*hunt group of lines to
                                       be changed when available*/
```

```
      3 n fixed bin (17),       /*number of lines to
                                  change*/
      3 state bit (3),          /*state these lines are
                                  to assume*/
    2 shut_ec bit (70),         /*a private shutdown event
                                  channel*/
    2 n_lines fixed bin (17),   /*number of entries in
                                  line table*/
    2 lines (1000),             /*line table*/
      3 rf_name char (32),      /*registry file name of
                                  the line*/
      3 ht_grp char (32),       /*hunt group for the line*/
      3 ioname char (32),       /*io name of the line*/
      3 state bit (3),          /*state of the line*/
      3 make_busy_sw bit (3),   /*state line is to assume
                                  when it comes out of the
                                  inuse state*/
      3 d_r_ec bit (70),        /*event channel for dialups
                                  and returns from
                                  user_control*/
      3 err_ec bit (70),        /*event channel for dialup
                                  errors*/
      3 attch_rfn char (32),    /*registry file name of
                                  attached terminal*/
      3 iostatus bit (144);     /*status of io operations
                                  with line*/
```

The hunt group entries of the make busy table are initially set to "xxxxxxxx" indicating that these are empty slots. The hunt group and registry file entries of the line table are filled from the cltext segment. The state entry is set to offhook except for the line with registry file name of "op_channel." This line is placed in onhook state, its dialup and error event channels created so the operator can log in. Each line is given an ioname of line_i, where i is its index in the "lines" array.

Every other entry in the line table is initially set to zero or blank depending on whether it is a binary, bit, or character type variable. During the process of converting the ascii segment and initializing the answering service line table, each line is attached to the answering service. However no dialups and error event channels are created for a line until it is placed in the onhook state by the system operator, this guarantees that only "onhook" lines can cause dialup signals.

Answering Service Event Dispatching

Following the command by the system operator to set the initial configuration of "on-hook" lines, the Answering Service Procedure is ready to receive wakeups from user dialups and system operator commands. Whenever an event of interest to the Answering Service occurs (i.e., whenever an event occurs for which the Answering Service has created an event channel), the process noting the event will send a wakeup to the Answering Service Process via the Interprocess Communication Facility. There are many different types of events which are of interest to the Answering Service and the Answering Service contains a subroutine to process each type of event. Each event channel is of the event call type, with an associated subroutine to be called by the Wait Coordinator. (See BQ.6). Hence, there exists no event dispatching mechanism within the Answering Service module. Instead, whenever the Answering Service wakes up, the appropriate procedure within the Answering Service is called automatically by the Interprocess Communication Facility.

Processing of User Dialups

When a user wishes to gain access to Multics, he dials up on a data set (such as the Bell System DATAPHONE data set 103A) or turns on some device attached to a dedicated line with data sets. The resulting interrupt causes the Answering Service to become active if the line is in an onhook state.

The event channel associated with the wakeup uniquely identifies which line caused the wakeup. Therefore by searching the answering service line table for the name of that event channel answer$dialup can determine which line caused the dialup. Answer$dialup then associates that line's event call channel (d_r_ec in the as_lt, see above) with answer$logout so user control can signal a logout of the user over the same event call channel. Answer$dialup then fills in the appropriate slot in the lines array of the following structure.

```
declare  1 uc_data based (pointer),     /*Structure found in
                                          segment ucp_comm, used
                                          for communication among
                                          Answering Service,
                                          User_Control and Overseer*/

         2 ucp_id bit (36),             /*process id of user_control
                                          process*/

         2 signal_ucp_chn bit (70),     /*channel used by Answering
                                          Service to signal a login
                                          to User_Control*/

         2 overseer_pid bit (36),       /*process id of overseer
                                          process*/

         2 signal_overseer bit (20),    /*channel used by
                                          User_Control to signal
                                          Overseer*/

         2 answer_pid bit (36),         /*process id of Answering
                                          Service*/

         2 n_lines fixed bin (17),      /*number of lines being
                                          used in the lines array*/

         2 lines (1000),

           3 description char (32),     /*description to be used by
                                          user_control to attach
                                          console and line*/

           3 rf_name char (32),         /*registry file name of
                                          line*/

           3 logout_chn bit (70);       /*channel to be used by
                                          user_control to signal
                                          logout of user*/
```

This structure is located in the segment ucp_comm, created by system control in the system control process-group directory. After filling in the appropriate slot in the lines array, answer$dialup signals user_control over the signal_ucp_chn event channel. An event indicator sent with the event identifies the index of the slot in the lines array that was just filled in. Answer$dialup then places the line in the inuse state and returns to the wait coordinator awaiting another dialup.

Processing of User-Process-Group Returns

When a user has finished using the system he types the "logout" command to his working process which eventually results in a wakeup to the Answering Service Process. The answer$logout entry is activated and proceeds to call the I/O system to attach the line to the Answering Service Process. Following this the make_busy_switch of the line is checked. If it is non-zero then the state of the line is set as indicated by the switch and appropriate actions for active or inactive lines taken. If the switch is not set then a call is made to ans_subr$btm to determine if the line should go into an active or inactive state. If the line is to be onhook then the d_r_ec event call channel is caused to point to answer$dialup to await a dialup on that channel. Then the state is set to onhook and answer$logout returns to the wait coordinator to await another logout. If the line is to be placed in an offhook state then the d_r_ec event channel and the error event channel are deleted and the state of the line is set to offhook. Then answer$logout returns to the wait coordinator to await another logout. A line can go into the disabled or no answer state only if fatal errors are detected on the line or on a specific request from the system operator. Hence the job of ans_subs$btm is merely to decide between onhook and offhook states for a line.

Processing Operator Requests

During the operation of the system it is desirable for the operator to be able to specify the configuration of communication lines over which the Answering Service should accept user dialups. For example, at system shutdown time the operator specifies that all of the communication lines should be placed into the "off-hook" state so that no user may dial up, and at system initialization time the operator specifies the initial configuration (perhaps "all") of the lines which should accept dialups. During the operation of the system, the operator may want to ready specific lines and busy others so that only users of a certain type of line may have access to the system. Each of these configuration requests is typed by the operator to his Working Process. This results in an interprocess signal which wakes up the Answering Service Process and activates the ans_commands$set_line entry. The configuration information is found in the set_line segment found in the request directory (pathname:>system_process_reserved_storage>request_dir). The segment has the following PL/I structure imposed on it.

```
declare  1 line_set based (ptr),    /*segment used by the set_line
                                      and get_line commands*/

         2 request bit (1),         /*"1"b implies set_line
                                      command, "0"b implies get_line
                                      command*/

         2 n_args fixed bin(17),    /*number of arguments*/

         2 args (90),

           3 switch bit (1),        /*"1"b implies line_name is
                                      a registry file name, "0"b
                                      implies all other choices
                                      see BX.15.08*/

           3 line_name char(32),    /*registry file name, type of
                                      line, "all", or "none"*/

           3 n fixed bin(17),       /*if line name is a type then
                                      this gives number to set*/

           3 state bit(3);          /*state the line is to be
                                      placed in*/
```

The set_line entry then tries to satisfy the operator's requests. If the line_name is a registry file name or "all" and the line is in an inuse state then the make_busy_switch is set to the desired state. If the line_name is a type and the request could not be satisfied then an entry is made in the make_busy_table. If an entry already existed in the table for that line type then it is updated. If the particular request was accomplished then its entry in the line_set structure is zeroed out. If ans_commands$set_line was unable to fully satisfy the request then the entry ptr->line_set.args(i).n gives the number of lines which it could not immediately set to the desired state. Ans_commands$set_line then signals system control over its request_serviced event channel and returns to the wait coordinator awaiting further configuration commands. System control then reflects this request_service event to the operator.

The second operator request accepted by the Answering Service is the get_line command. This request also uses the set_line segment. The system operator places in the PL/I structure either the particular registry file name or line type for which he desires status. The entry, ans_commands$get_line, then returns the state of those lines. If the line name given was a line type and all lines of that type are in the same state then only one

entry is returned in the set_line segment. However if they are not, then an entry is returned for each line of that type. After placing the state of all the requested lines in the set_line segment, ans_commands$get_line signals over the request_serviced channel and returns to the wait coordinator to await further get_line commands.

Summary of Entries in the Answering Service Module

The first entry called in the Answering Service Process is ans_init$init. This entry is called when system control creates the process. Its job is to attach all the communicatin lines and to create all the event channels needed by the process.

The second entry called is answer$dialup. This entry is called by the wait coordinator when the system operator logs in. The job of this entry is to handle all user dialups.

Another entry is answer$logout. The job of this entry is to handle all user logouts. This entry also is called by the wait coordinator.

A fourth entry is ans_commands$set_line. This entry handles all the system operator's configuration commands.

A fifth entry is ans_commands$get_line. This entry furnishes the state of all the lines requested by the system operator.

The segment ans_subr contains utility procedures called by the answering service. The function of ans_subr$delete_chns is to delete the event channels associated with a particular line. This is done when the line goes from an active state to an inactive state.

Ans_subr$create_chns is another entry. Its job is to create the event channels associated with a particular line. This is done when the line goes from an inactive state to an active state.