Published: 01/10/68

Identification

Interaction between the I/O System and the Quit/Start, Save/Resume, and Logout Mechanisms
S. I. Feldman

Purpose

This section describes the io_control procedure, which is the interface between the Overseer and the I/O System. Certain entry points are called by the Overseer to stop, start, and reset various iopaths. The other entries are call by the I/O System to update the data base needed by the other entries of io_control, the Overseer ioname List (OIL).

The Overseer ioname List (OIL)

The OIL is the data base of the io_control procedure. This table contains the list of ionames of devices known to the process group, and the set of event channel names associated with that ioname. The OIL also includes a lock list, certain indices into the ionames array of the OIL, and the process id of the Overseer. There is also the name of an event channel to be signaled which will cause io_control$event to be called in the Overseer process.

```
dcl 1 oil based(p),              /*Overseer ioname List*/
  2 overseer_id bit(36),         /*process id*/
  2 quit_report_event bit(70),    /*if io_control$set_com_source
        "                          has been called, this event
        "                          will be signaled if a quit is
        "                          subsequently detected on the device*/
  2 command_hangup_event bit(70), /*event to be signaled
        "                          when command source hangs up*/
  2 io_control_event bit(70), /*event to be signaled to cause
        "                      io_control$event to be called in the
        "                      Overseer process*/
  2 response_event bit(70),    /*event to be signaled when finished
        "                      handling io_ctl$event*/
  2 response_proc_id bit(36), /*process to receive above signal*/
  2 command_source fixed bin,  /*index in the ionames array
        "                      of the present command source*/
  2 create bit(1),             /*if 1 when io_control$event is called,
        "                      create an event.  If 0, destroy
        "                      an event*/
  2 current fixed bin,         /*index in ionames array of device
```

```
          "                          for which event is to be created
          "                          or destroyed when io_control$event is
          "                          called*/
  2 maxionames fixed bin,          /*=N2*/
  2 last_used fixed bin,           /*index in ionames array of last
          "                          element in use (in free
          "                          or active ioname lists*/
  2 first_ioname fixed bin,        /*index of first ioname
          "                          block in thread containing
          "                  presently-used ionames*/
  2 first_free fixed bin,          /*index of first ioname block in
          "                          free thread*/
  2 recursion_count fixed bin,     /*0 if unlocked, increased by
          "                          1 each time OIL is locked in
          "                          a given process, and decremented
          "                          each time a routine returns*/
  2 attach_ringno fixed bin,       /*attach ring number ioname for
          "                          command source had before being changed
          "                          to the command source*/
  2 use_ringno fixed bin,          /*use ring number ioname for command source
          "                          had before being changed to the
          "                          command source*/
  2 oil_lock_list bit(144),        /*standard lock*/
  2 ionames(N2),
    3 next fixed bin,              /*next block in present thread
          "                          (active or free).  If zero, no
          "                          more blocks*/
    3 ioname char(32),             /*ioname to be used for device*/
    3 type char(32),               /*type to be used in attach call
          "                          for device*/
    3 description char(32),        /*description to be used in attach
          "                          call for device*/
    3 dmp_proc_id bit(36),         /*process id of the DMP*/
    3 quit_event bit(70),          /*signaled by Overseer to stop the
          "                          device*/
    3 restart_event bit(70),       /*signaled by Overseer to restart
          "                          the device*/
    3 hangup_report_event bit(70), /*if device can hang up, this
          "                          event is signaled if that happens*/
    3 quit_state fixed bin;        /*0 if device neither quit nor held
          "                          1 if device quit
          "                          2 if device held*/
```

The ionames array of the OIL contains the information on all of the devices attached by this group. The elements of the ionames array with indices less than or equal to oil.last_used are threaded onto two lists. The active list contains all of the blocks that represent presently-attached devices. The free list contains blocks freed by a detachment and available for use when another device is attached. When a new ioname is to be added to the active list, a block is removed from the free list if there are any. Otherwise, oil.last_used is incremented by one unless it is already greater than or equal to oil.maxionames. If this

condition holds, the OIL has overflowed. Otherwise, the block with index equal to oil.last_used is used as the new block and threaded at the head of the active list.

Initialization

The following call is made by the overseer procedure in the Overseer process before any other call to the I/O system in the process group:

```
call io_control$init(quit_report_event,hangup_report_event);
dcl quit_report_event bit(70),
    hangup_report_event bit(70);
```

In response to this call, the following steps are taken:

1. If an OIL segment already exists in the group directory, set bit 3 of cstatus and return.

2. Create the OIL segment as a branch of the group directory with.entry name oil_seg_.

3. Store the process id of the Overseer in oil.overseer_id.

4. Create an event channel, declare it to be an event call channel, and store the name of the channel in oil.io_control_event. Whenever that event is signaled, the Wait Coordinator calls io control$event.

5. Initialize the switching complex by making the following call:

```
call atm$group_init;
```

6. Initialize the Transaction Block Maintainer by making the following call:

```
call tbm$init("0"b);
```

7. Store quit report event in oil.quit_report_event and store hangup report event in oil.command_hangup_event. When the first event is signaled, the quit procedure in the Overseer is called. The second event is signaled by io control when the command device hangs up.

8. Zero oil.recursion_count and zero the lock list.

9. Set oil.last_used, oil.first_ioname and oil.first_free equal to zero, and set oil.maxionames equal to some appropriately large number.

10. Return.

Locking

All calls to io_control other than io_control$event and io_control$init call the Locker to lock the OIL and not to return until it is locked. The OIL must be locked since Attachment Modules executing in the various processes call certain entry points of io_control. In certain cases, one entry of this procedure will call another entry in the same process. This recursive calling happens only in certain special cases. Because of the possibility of recursion, a recursion count is kept in the OIL. Each call that locks the OIL increments the recursion count by 1 when it is entered and decrements it by 1 when it returns (with one exception). When the recursion count goes to zero, the OIL is unlocked. Note that in most cases, a recursive call does not occur.

There are several exceptions to the above rule. First, io_control$init neither locks nor unlocks the OIL since, at the time it is called, no other process is capable of locking the OIL. Another special case is the entry point io_control$lock. This procedure is called by the Attachment Module when it must rename some nodes in the Attach Table and the note the change in the OIL. The OIL is locked by the call to io_control$lock in order to prevent another process from using an inconsistent OIL. The OIL remains locked throughout recursion count or unlock the OIL upon return. Therefore, the OIL stays locked until the last call to io_control has been completed.

The other exception to the rules is io_control$event. This call can only be made in the Overseer process. When an entry is added or deleted from the OIL, an event channel that belongs to the Overseer may have to be created or destroyed. By means of an event call channel, io_control$event is invoked, although the caller may be in a different process. The caller is expected to lock and unlock as necessary.

Calls for Use by the Overseer

Five calls are made by the Overseer to handle quit and start:

Stop

When the Overseer is signaled that a quit has been detected on the command device, it quits all of the working processes in the group and then makes the following call:

```
call io_control$stop(cstatus);
```

In reponse to this call, the following steps are taken:

1. Lock the OIL and increment oil.recursion_count by 1.

2. Signal the quit_event for each ioname in the active list in the OIL.

3. If there is no command source (oil.command_source equals zero), go to (5).

4. Otherwise, divert the command source:

```
call divert(oil.ionames(oil.command_source).ioname,
     oil.ionames(oil.command_source).ioname,"",status);
```

Scan all of the blocks on the active list of the OIL. Whenever an ioname on that list has a quit_state equal to zero, change that state to one (from normal to quit).

6. Decrement oil.recursion_count by 1. If this is zero, unlock the OIL.

7. Return.

Start

When the Overseer wishes to restart the quitted processes in the group, it wakes up the working processes and makes the following call:

```
call io_control$start(cstatus);
```

The following steps are taken in reponse to this call:

1. Lock the OIL and increment oil.recursion_count by 1.

2. If oil.command_source is zero (no command source), go to (3). Otherwise, make the following call:

```
call revert(oil.ionames(oil.command_source).ioname,
     "",status);
```

3. Signal the restart_event associated with each ioname on the active list, and then call iosw$queue_restart for each of those ionames.

4. For each element of ionames with quit_state equal to 1, change quit_state to 0 (from quit to normal).

5. Decrement oil.recursion_count by 1. If it is zero, unlock the OIL.

6. Return.

Reset

When the Overseer wishes to destroy the present set of working processes and the present iopaths for the devices (other than the command source), it makes the following call:

```
call io_control$reset(cstatus);
```

The following steps are taken to handle this call:

1. Lock the OIL and increment oil.recursion_count.

2. For each element of oil.ionames on the active list other than the command source with quit_state equal to 1, set the quit_state equal to zero and make the following two calls:

```
call divert(ioname,ioname,"",status);

call invert(ioname,status);
```

The first call is guaranteed to pass through any I/O System locks and creates a new iopath. The second call destroys all paths other than the newly created (by the divert) one for the device.

3. If the quit_state of the command source is one, set it equal to zero and make the following call:

```
call invert(oil.ionames(oil.command_source),status);
```

4. Decrement oil.recursion_count by 1. If it is zero, unlock the OIL.

5. Return.

Hold

When the user wishes to put his quitted processes in the "hold" state, the following call is made:

```
call io_control$hold(cstatus);
```

For each ioname on the active list in the OIL with quit_state equal to 1, the quit_state is changed to 2 (from quit to hold).

Release Hold

When the user wishes to release his processes from the hold state and place them in the quitted state, the following call is made:

```
call io_control$release_hold;
```

For each ioname on the active list of the OIL with quit_state equal to 2, quit_state is changed to 1 (from hold to quit).

Set Command Source

When the user changes command sources or when the Overseer initially assigns the command source, the following call is made:

```
call io_control$set_com_source(ioname,cstatus);
     dcl cstatus bit(18);
```

This call may only be made in the Overseer.

1. Lock the OIL and increment oil.recursion_count by 1.

2. Search the active list of the OIL for ioname. If no such entry is found, set bit 1 of cstatus and go to (7). Otherwise, remember the index of the entry for use below.

3. If oil.command_source is non-zero, do the following:

   a. Make the following call:

```
call order(oil.ionames(oil.command_source).ioname,
     "trap_quits",argptr,null,status);

dcl argptr ptr,
    1 arg based(argptr),
      2 proc_id bit(36),
      2 event_name bit(70);
```

   Both proc_id and event_name are zero. This call will stop the related Device Manager Process from signaling the Overseer whenever a quit is detected.

   b. Make the following calls to restore the ioname to its old accessibility:

```
call atm$set_attach_ringno(oil.ionames(oil.command_source),
     oil.attach_ringno,cstatus);

call atm$set_use_ringno(oil.ionames(oil.command_source),
     oil.use_ringno,cstatus);
```

4. If ioname is null, zero oil.command_source and go to (7).

5. Store the index found in step 2 in oil.command_source and then make the following call:

```
call order(oil.ionames(oil.command_source),"trap_quits",
     addr(oil.overseer_id),null,status);
```

Whenever a quit is detected for that device, the report event will be signaled.

6. Make the following calls to save the access information for the ioname and then to make the new command source accessible for

use from the user's ring but detachable only from the administrative ring:

```
call atm$get_attach_ringno(oil.ionames(oil.command_source),
     oil.attach_ringno,cstatus);

call atm$get_use_ringno(oil.ionames(oil.command_source),
     oil.use_ringno,cstatus);

call atm$set_attach_ringno(oil.ionames(oil.command_source),
     administrative_ring_number,cstatus);

call atm$set_use_ringno(oil.ionames(oil.command_source),
     user_ring_number,cstatus);
```

7. Decrement oil.recursion_count by 1. If it is now zero, unlock the OIL.

8. Return.

Logout

When the user logs out, the Overseer makes the following call:

```
call io_control$logout(cstatus);
```

In response to this call, the following two calls are made for ioname on the active list of the OIL:

```
call divert(ioname,ioname,"",status);

call detach(ioname,"","",status);
```

After the calls have been completed, io control returns.

Save and Resume

Two calls are supplied to handle save and resume. The functions of these calls are not specified at present:

```
call io_control$save;

call io_control$restore;
```

Calls for Use by the I/O System

The Attachment Module (see BF.2.23) makes use of six entry points of io control. One has already been discussed: io control$lock. Other entry points are for maintaining the OIL after handling attach, divert, revert, and detach outer calls. Three of these entry points run in the process in which they are invoked. The

other is called by signaling an event since it must run in the Overseer (it creates and destroys certain event channels for which the Overseer is the receiving process). Finally, there is an entry that is called in the Overseer process whenever a device assigned to the group hangs up.

Attach

When a new device is attached, the following call is made by the Attachment Module:

```
call io_control$attach(ioname,type,description,overseer_id,
     dmp_proc_id,quit_event,restart_event,hangupable,
     overseer_hangup_report_event,cstatus);

dcl ioname char(*),
    type char(*),
    description char(*),
    overseer_id bit(36), /*return argument*/
    dmp_proc_id bit(36), /*forward argument*/
    quit_event bit(70), /*forward argument*/
    restart_event bit(70),  /*forward argument*/
    hangupable bit(1),  /*forward argument*/
    overseer_hangup_report_event bit(70),
                        /*return argument*/
    cstatus bit(18);    /*return argument*/
```

The following steps are taken in response to this call:

1. Lock the OIL and increment oil.recursion_count by 1.

2. If oil.first_free is non-zero, remove the first block from the free list and put it at the head of the active list. Otherwise, if oil.last_used is greater than or equal to oil.maxionames, set bit 2 of cstatus and go to (9). Otherwise, increment oil.last_used by one and thread the element of oil.ionames with that index at the head of the active list.

3. Store ioname in in oil.ionames(oil.first_ioname).ioname.

4. Store oil.overseer_id in overseer_id.

5. Store quit_event and restart_event in the corresponding entries in the element of oil.ionames.

6. Create the quit_response and hangup_report event channels by the following steps:

   a. Store the index of the element of oil.ionames being handled in oil.current.

   b. Set oil.create ON.

c. If hangupable is OFF (equal to zero), store zero in oil.ionames(oil.current).hangup_report_event and in overseer_hangup_report_event and go to (7).

d. Create an event channel, store its name in oil.response_event and store the present process id in oil.response_proc_id.

e. Signal the event channel with name oil.io_control_event for for the receiving process with id equal to oil.overseer_id and wait for the response event to be signaled.

f. Upon return from wait, destroy the response event channel.

g. Store the name of the event channel stored in the hangup_report_event entry of the element of the OIL in overseer_hangup_report_event.

It is necessary to use this roundabout method of creating the event channel because only the receiving process is permitted to create or destroy an event channel.

7. Store dmp_proc_id in the corresponding entry in the OIL.

8. Set quit_state equal to zero for the new ioname.

9. Decrease the recursion count by 1. If it it zero, unlock the OIL.

10. Return.

Rename

When revert and divert calls are being handled, the ioname of the device may change. If this is the case, the following call is made:

```
call io_control$rename(newioname,oldioname,cstatus);
dcl newioname char(*),
    oldioname char(*),
    cstatus bit(18);
```

In response to this call, the following steps are taken:

1. Lock the OIL. Increment oil.recursion_count by 1.

2. Search the active list of the OIL for ioname oldioname. If no such entry is found, set bit 1 of cstatus and go to (4).

3. Otherwise, replace the present ioname with newioname.

4. Decrement oil.recursion_count by 1. If it is now zero, unlock the OIL.

5. Return.

Detach

If a device is detached, the following call is made:

```
call io_control$detach(ioname,cstatus);
dcl ioname char(*),
    cstatus bit(18);
```

The following steps are taken in response to this call:

1. Lock the OIL and increment oil.recursion_count.

2. Search for ioname ioname in the active list of the OIL. If it is not found, set bit 1 of cstatus and go to (5).

3. Remove the element of oil.ionames found above from the active list and thread it at the head of the free list.

4. If the hangup_report_event in the OIL is zero, go to (5). Otherwise, do the following to destroy that event channel:

   a. Set oil.create OFF.

   b. Store the index of the present element of oil.ionames in oil.current.

   c. Create an event channel, store its name in oil.response_event and store the present process id in oil.response_proc_id.

   d. Signal the event with name oil.io_control_event and wait for the response event.

   e. Destroy the response event channel.

Upon return from wait, the event channels will have been destroyed. Again, this roundabout method is necessary because only the receiving process can destroy an event channel.

5. Decrement oil.recursion_count by 1. If it equals zero, unlock the OIL.

6. Return.

Hangup

When a hangup event is signaled, the Wait Coordinator makes the following call:

```
call io_control$hangup(index,event_indicator);
dcl index fixed bin,  /*index in oil.ionames of ioname for
     "                      this device*/
    event_indicator(3) bit(70);
```

In response to this call, the following steps are taken:

1. Call the Locker to lock the OIL. Upon return, increment oil.recursion_count by 1.

2. If index equals oil.command_source, signal the event channel with name oil.command_hangup_event and go to (4).

3. Make the following calls:

```
call divert(oil.ionames(index).ioname,"",status);

call detach(oil.ionames(index).ioname,"","",status);

dcl status bit(144);
```

4. Decrease oil.recursion_count by 1. If it is now 0, call the Locker to unlock the OIL.

5. Return.

Event

The following call is issued when the io_control_event is signaled in the Overseer, and is used to create or destroy event channels, as necessary.

```
call io_control$event(null,event_indicator);
```

The arguments are ignored. The OIL is neither locked nor unlocked in response to this call.

The following steps are taken:

1. If oil.create is OFF, go to (2). Otherwise, create an event channel and store its name in oil.ionames(oil.current).hangup_report_event and go to (3). This event will be signaled if the device hangs up, and io_control$hangup will be called by the Wait Coordinator in response to the signal.

2. If oil.create is OFF, destroy the event channel with name oil.ionames(oil.current).hangup_report_event.

3. Signal the event channel with name oil.response_event for process oil.response_proc_id.

4. Return to the Wait Coordinator.

Summary of Cstatus Bits

1 ioname not found in search of oil.ionames
2 OIL overflow
3 Attempt to initiate io_control twice
4 OIL not found
5 appendb error
6 ATM error
7 ECM error
8 Outer call error
9 Unimplemented call
10 Locker error
11 TBM error