Published: 01/10/68
(Supersedes: BF.2.26: 08/14/67)

Identification

The I/O Assignment Module
R. C. Daley, S. I. Feldman

Purpose

This section describes the design of the I/O Assignment Module (IOAM) and its prinicpal data bases, the I/O Assignment Tables (IOATs). The IOAM is the module of the hardcore supervisor through which any request to assign an I/O device or detachable medium (e.g., tape drive, tape reel, GIOC channel) must ultimately be directed. Requests for assignment are accepted only from procedures of the hardcore supervisor, principally the Resource Assignment Module (see BT.1.01). The main function of the IOAM is to maintain a record of all current I/O assignments and to make information concerning these assignments available to certain modules of the hardcore and administrative rings.

The Registry Files and the I/O Assignment Tables

Associated with every I/O device or detachable medium known to a system is a file containing information about the device and its connections with other devices. These files are used by the I/O system and are accessed by the I/O Registry File Maintainer (see BF.2.22) and the Attachment Module (see BF.2.23). These files are organized into type directories which are in turn immediately inferior to the Registry file directory (path name >rfd). If a user is permitted to allocate a particular device to himself, he has read access from the administrative ring for the corresponding Registry File (RF). If he is the present assigned user or control user for the device, he also has write access to the file.

In each Registry File type directory, there is a special file called the I/O Assignment Table (IOAT). This segment is normally accessible only from the hardcore ring. The type directories themselves are writable in the hardcore ring for all users who may allocate a device of the type associated with the directory. (Some special system users have write access to the type directories and the IOATs from the administrative ring.) When a user becomes the assigned or control user for a device, the IOAM gives him the proper access attributes for the RF. When he loses his status as assigned or control user for a device, this access

permission is removed.

The RFs are accessed using the type and name (entry name in the type directory) of the file. For user convenience, these RFs will probably have several names. The RF contains, among other things, an array of resource names associated with the device. (It is possible for a group of "devices" to be considered as a single device. For example, a full-duplex typewriter connection requires two GIOC channels connected to a single data set. The pair of channels will be considered to be a single device, although there will be two resource names, one for each channel.)

The I/O Assignment Table segments are declared as follows:

```
dcl 1 ioat based(p),              /*I/O Assignment Table, one per
        "                           Registry File type directory,
        "                           accessible from hardcore ring only
        "                           in most process groups. The IOAT
        "                           is the data base of the I/O
        "                           Assignment Module (see BF.2.26)*/
  2 lock bit(108),                /*standard lock*/
  2 type char(32),                /*name of type directory in
        "                           which file resides*/
  2 normal fixed bin,             /*index of first element of resource
        "                           array of devices not in the free pool*/
  2 free fixed bin,               /*index of first element of resources
        "                           array of resource in free pool*/
  2 vacant fixed bin,             /*index in resources array of
        "                           first element of vacant list.
        "                           Initially zero.*/
  2 highest fixed bin,            /*index of last element of resources.
        "                           Can be extended up to max*/
  2 max fixed bin,                /*largest possible subscript of element
        "                           in resources array*/
  2 auto_free bit(1),             /*if 1, a resource that is unassigned
        "                           is to be automatically freed (put
        "                           on free list)*/
  2 resources(MAX),
    3 resource_name char(32),     /*name of resource to be allocated*/
    3 assigned_user char(50),     /*user to whom resource is presently
        "                           assigned. If unassigned, set to
        "                           blanks*/
    3 control_user char(50),      /*user who is assigned to control
        "                           device. May be changed by assigned
        "                           user. Initially equal to
        "                           assigned_user*/
    3 rf_name char(32),           /*entry name of Registry File in
        "                           this directory associated with this
        "                           resource*/
    3 temp bit(1),                /*if 1, destroy this entry and associated
        "                           files when resource is deallocated*/
    3 next fixed bin;             /*index of next element of resources
```

```
"                                   array.  Zero for last entry*/
```

The "assigned" user of a device is the user to whom the device is presently assigned. The "control" user of the device is the user which has permission to physically control the device. When a device is assigned, the control user is set equal to the assigned user. The assigned user may change the control user of the device at any time. For example, the assigned user of a device may give a Universal Device Manager Process permission to control a device.

The elements of the resource array are chained together using indices (stored in the "next" item in the element of ioat.resources) of elements of the array. There is also a vacant list of items not presently being used. When a resource is to be added to the table (by a call to ioam$create_resource) and the vacant list is empty, ioat.highest is increased by 1 (unless ioat.highest is already greater than or equal to ioat.max, in which case the segment has overflowed and there probably is a system bug.)

Calls to the I/O Assignment Module

There are eight calls to the IOAM. The following is a list of these calls and a declaration of the arguments. A detailed description of these calls follows.

```
call ioam$assign(type,resource_name,cstatus);

call ioam$unassign(type,resource_name,force,cstatus);

call ioam$allocate(type,resource_name,cstatus);

call ioam$free(type,resource_name,cstatus);

call ioam$set_control(type,resource_name,user_id,cstatus);

call ioam$get_assignment(type,resource_name,user_id,cstatus);

call ioam$check_assignment(type,resource_name,cstatus);

call ioam$delete_resource(type,resource_name,cstatus);

dcl type char(*),
    resource_name char(*),
    user_id char(*),
    force bit(1),
    cstatus bit(18);
```

ioam$assign

Whenever a device or medium is assigned by the Resource Assignment Module, the IOAM is informed by means of the following call, accessible only from the hardcore ring:

```
call ioam$assign(type,resource_name,cstatus);
```

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

1. Set cstatus equal to zero.

2. If the present user does not have write access from the hardcore ring in the type directory with name type, set bit 6 of cstatus and return.

3. If the present user does not have write permission from the hardcore ring in the IOAT in type directory type, set bit 5 of cstatus and return.

4. Lock the IOAT by means of a call to the ILOCK routine (see BG.15).

5. Search the normal and free lists of the IOAT for the resource with name resource_name. If none is found, set bit 1 of cstatus and go to (11).

6. If the present user does not have read permission from the caller's ring for the file with name rf_name in the type directory, set bit 7 of cstatus and go to (11).

7. If the present assigned user of the resource is the present user, set bit 4 of cstatus and go to (11).

8. If the present assigned user is neither blank nor the current user, set bit 11 of cstatus and go to (11).

9. Store the present user id in both assigned_user and control_user in the IOAT entry.

10. Give the present user permission to write from the administrative ring in the RF with name rf_name in directory type. If the element of ioat.resource was found on the free list, remove it from that list and place it at the head of the normal list.

11. Unlock the IOAT and return.

ioam$unassign

The following call is supplied to change the status of a device from assigned to unassigned. There is an argument that tells the IOAM whether it should unassign the device regardless of present assignment or only if the present user is the assigned user of the resource. The call may be made only from the hardcore ring except in certain privileged process groups, which may call this entry

point from the administrative ring.

```
call ioam$unassign(type,resource,force,cstatus);
```

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

1. Set cstatus equal to zero.

2. If the present user does not have write access from the hardcore ring in the type directory with name type, set bit 6 of cstatus and return.

3. If the present user does not have write permission from the hardcore ring in the IOAT in type directory type, set bit 5 of cstatus and return.

4. Lock the IOAT by means of a call to ILOCK.

5. Search the normal list of the IOAT for the resource with name resource_name. If no such name is found,set bit 1 of cstatus and go to 12.

6. If the present assigned_user is blank, the resource is already unassigned, so set bit 2 of cstatus and go to 12.

7. If the present assigned_user is neither blnak nor the present user and the force switch is OFF, set bit 11 of cstatus and go to 12.

8. Remove the present assigned user's write permission to the RF associated with the resource.

9. If the present control user is not the same as the present assigned user, remove the control user's write permission to the RF associated with the resource.

10. Store blanks in the assigned_user and control_user of the block of the IOAT.

11. If ioat.auto_free is ON, remove the block from the normal list and thread it onto the free list.

12. Unlock the IOAT and return.

ioam$allocate

Whenever a user wishes to allocate a device of a given type from the free pool, the following call is made by the Resource Assignment Module:

```
call ioam$allocate(type,resource_name,cstatus);
```

This call may be made only from the hardcore ring. In response to this call, the following steps are taken:

1. Set cstatus equal to zero.

2. If the present user does not have write access from the hardcore ring in the type directory with name type, set bit 6 of cstatus and return.

3. If the present user does not have write permission from the hardcore ring for the IOAT in directory type, set bit 5 of cstatus and return.

4. Lock the IOAT by means of a call to ILOCK.

5. If ioat.free is zero (free pool empty), set bit 8 of cstatus and go to (11).

6. If the present user does not have read permission from the caller's ring for the RF associated with the first resource in the free pool, set bit 7 of cstatus and go to (11).

7. Remove the first element of ioat.resources threaded on the free list and thread it at the head of the normal list.

8. Store the id of the present user in both assigned_user and control_user in the IOAT entry handled above.

9. Give the present user permission to write from the administrative ring in the RF associated with the resource discussed above.

10. Return the name of the resource allocated in resource_name.

11. Unlock the IOAT and return.

ioam$free

When it is necessary to force a resource onto the free list, the following call is made:

```
call ioam$free(type,resource_name,cstatus);
```

This entry point is normally accessible only from the hardcore ring. Certain privileged users will be able to make this call from the administrative ring. For example, the tape librarian will be able to put a tape back in the free pool.

The following steps are taken in response to this call:

1. Set cstatus equal to zero.

2. If the present user does not have write access from the hardcore ring for the type directory with name type, set bit 5 of cstatus and return.

3. If the present user does not have write permission from the hardcore ring for the IOAT in directory type, set bit 6 of cstatus and return.

4. Lock the IOAT by a call to ILOCK.

5. Search the free list for the given resource. If found, go to (9).

6. Search the normal list for the given resource. If not found, set bit 1 of cstatus and go to (9).

7. If the resource is found but the assigned user is not blank, set bit 9 of cstatus and go to (9).

8. Otherwise, remove the resource block from the normal list and thread it on the free list.

9. Unlock the IOAT and return.

ioam$set_control

If the assigned user of a device wishes to give a user permission to control a resource, the following call is made:

```
call ioam$set_control(type,resource_name,user_id,cstatus);
```

This call may be made from the hardcore and administrative rings. In response to the call, the following steps are taken:

1. Set cstatus equal to zero.

2. If the present user does not have write access from the hardcore ring for the type directory with name type, set bit 6 of cstatus and return.

3. If the present user does not have write permission from the hardcore ring for the IOAT in directory type, set bit 5 of cstatus and return.

4. Lock the IOAT by a call to ILOCK.

5. Search for the resource with name resource name in the normal list. If none is found, set bit 1 of cstatus and go to (10).

6. If the present user is not the assigned user of the resource, set bit 2 of cstatus and go to (10).

7. If the present control_user is neither blank nor the assigned user, remove that user's write permission to the associated RF.

8. If user_id is neither blank nor the present assigned_user, give that user write permission for the associated RF.

9. Store user_id in control_user.

10. Unlock the IOAT and return.

ioam$get_assignment

The following call is supplied to allow the control user to find out the assigned user of a resource. (This call is intended for use by the Dispatcher, see BF.2.25). The call may be made in the hardcore and administrative rings.

```
call ioam$get_assignment(type,resource_name,user_id,cstatus);
```

If cstatus is zero upon return, this user is the control user of the resource; otherwise, he is not. In response to the call, the following steps are taken:

1. Set cstatus equal to zero.

2. If the present user does not have write permission in the IOAT in directory type, set bit 5 of cstatus and return.

3. Lock the IOAT.

4. Search the normal list of resources in the IOAT for resource_name. If not found, set bit 1 of cstatus and go to (7).

5. If the present user is not the control user of the resource, set bit 3 of cstatus and go to (7).

6. Otherwise, set user_id equal to the assigned_user for the resource.

7. Unlock the IOAT and return.

ioam$check_assignment

The following call is supplied to enable a user to find out whether he is the assigned user of a device. The call may be made from the hardcore and administrative rings.

```
call ioam$check_assignment(type,resource_name,cstatus);
```

If cstatus is zero upon return, the user is the assigned user of the device; otherwise, he is not. The following steps are taken in response to this call:

1. Set cstatus equal to zero.

2. If the user does not have write permission in the IOAT from the hardcore ring, set bit 5 of cstatus and return.

3. Lock the IOAT.

4. Search the normal list in the IOAT for the given resource. If no such entry is found, set bit 1 of cstatus and go to (6).

5. If the present user is not the assigned user of the resource, set bit 2 of cstatus.

6. Unlock the IOAT and return.

ioam$create_resource

The following call is made to create a Registry File and to add a resource to the appropriate IOAT:

```
call ioam$create_resource(type,resource_name,name,
     temp,cstatus);

dcl temp bit(1),
    name char(*);
```

The call may only be made by certain priveleged users, such as the tape librarian and the typewriter universal device manager process group. In response to this call, the IOAM takes the following steps:

1. Set cstatus equal to zero.

2. If the user does not have write permission for the type directory with name type, set bit 6 of cstatus and return.

3. If the user does not have write permission from the hardcore ring in the IOAT in the type directory, set bit 5 of cstatus and return.

4. Lock the IOAT by a call to ILOCK.

5. Search the IOAT for a resourcw with name resource name. If one is found, set bit 12 of cstatus and go to (11).

6. Search the type directory for a file with name name. If one is found, set bit 12 of cstatus and go to (11).

7. If there is no file with name "prototype" in the type directory, set bit 13 of cstatus and return.

8. Otherwise, add an entry to the normal list of the IOAT for the new resource, and store resource name in that entry. Set the temp

bit in that entry equal to temp. Store name as rf_name, and set both the assigned and control user of the resource equal to blanks.

9. Create a file in the type directory with name name, and initialize it by copying the prototype file into the new RF. Store resource name as rf.devices(1).resource_name.

10. If there is a file with name "prototype_ro" in the directory, add the name resource name || "_ro" to the list of names for that file.

11. Unlock the IOAT and return.

ioam$delete_resource

The following call is proved to delete a resource from the system. This deletion involves removing the entry for the resource in the IOAT, deleting a Registry File, and removing a name from a file in the type directory. This call is made by the Attachment Module whenever the UNLOAD disposal is specified on a detach call. The resource will be deleted only if the user has write access to the directory and to the IOAT from the caller's ring, or if the temp switch is ON for the resource in the IOAT entry. An example of a device with a temporary Registry File is an IBM 1050, which does not have sufficient hardware identification to associate it definitely with a particular machine. Therefore, a dummy file is created when the machine dials in by the I/O Registry File Maintainer. The call is:

```
call ioam$delete_resource(type,resource_name,cstatus);
```

The following steps are taken in response to the call:

1. Set cstatus equal to zero.

2. If the user does not have write access to the IOAT from the hardcore ring, set bit 5 of cstatus and return.

3. Lock the IOAT.

4. Search the normal and free lists for resource name. If no such entry is found, set bit 1 of cstatus and go to (10).

5. If the resource is currently assigned to a user other than the present user, set bit 10 of cstatus and go to (10).

6. If the current user has write access in the type directory from the caller's ring or if the current user is the assigned user and the temp bit is ON in the IOAT entry, go to (7). Otherwise, set bit 6 of cstatus and go to (10).

7. Delete the file in the type directory with name equal to the rf_name in the element of ioat.resources.

8. If there is a file with name equal to resource name || "_ro" in directory type, delete that name from the file. If that is the the only name of the file, delete the file.

9. Remove the current entry in the IOAT from the thread in which it resides and add the block to the vacant list.

10. Unlock the IOAT and return.

Summary of Cstatus Bits

1 resource_name not found in IOAT
2 resource not assigned to this user
3 user not control user of device
4 resource already assigned to this user
5 user does not have write access to IOAT
or IOAT does not exist
6 user does not have write access in type directory
or type directory does not exist
7 user not permitted to assign device
8 no available resource in free pool
9 attempt to free an assigned device
10 attempt to delete resource assigned
to a user other than the present user
11 resource assigned to other user
12 resource already exists
13 no prototype file
18 system bug