Published: 04/10/68

Identification

The Generic Device Strategy Module (DSM)
and The Device Control Module (DCM)

S. I. Feldman

Purpose

In the I/O System, Device Strategy Modules (DSMs) are the outer modules responsible for device attachment, synchronization management and interfacing with the Device Control Modules. This section describes features common to all DSMs, and gives a brief summary of the standard IOS procedures and inner modules available to DSM writers. This section also describes certain disciplines which must be followed by DCM writers.

Introduction

In the I/O System, devices are controlled by Device Control Modules (DCMs), which execute in special processes called Device Managers (DMPs). Normally, the DCM runs in a system process group in a universal DMP. Device Strategy Modules (DSMs) can run in all of the processes in the process group assigned to a device. The interface between the DSM and the DCM therefore has special properties. (See BF.2.01.) DSMs handle data buffering and are responsible for synchronization (read-ahead, write-behind) and for handling the calls relating to synchronization (readsync, writesync, worksync, resetread, resetwrite, and iowait). There are several inner modules and standard procedures for use especially by DSMs. The use of these modules will be discussed and some of the data bases of these modules will be mentioned. The restrictions the use of these modules place on the DSM writer will then be discussed, and finally the handling of certain calls will be described.

Inner Modules

Since DSMs and DCMs are outer modules, they use the Mode Handler (see BF.2.27) and the Transaction Block Maintainer (see BF.2.20). Also, the DSM and DCM are expected to use the standard data bases found in the per-ioname segment (IS) and the standard conventions for use of auxiliary data bases in the IS (see BF.2.20).

In addition to the above standard modules, the DSM uses the Attachment Module and the Request Queuer.

Code conversion is usually done by the DCM. However, if the DSM must do any code conversion and needs a driving table of some sort, it should use the second driving table pointer to access the data base. The Switching Complex will set up this pointer if a segment name is given in the Type Table for this driving table. The Attachment Module will set up this pointer if this is indicated by the Registry Files it examines. As an example of the use of this driving table, the typewriter DSM uses a procedure that canonicalizes ASCII text.

The Attachment Module

The Attachment Module (see BF.2.23) is called to handle the following outer calls:

attach
detach
divert
revert
invert

The DSM calls the Attachment Module immediately to handle the last three calls, and does no other processing. In response to an attach call, the Attachment Module calls the Mode Handler, traces through the Registry Files implied by the arguments of the attach call, allocates devices associated with the files, creates a private Device Manager, (if requested and permitted), and attaches the DCM. It splices in the Sectional Formatting Module (SFM) if the SECTional mode is specified. In response to a detach call, the resources are deallocated, the DCM is detached, and the DMP is destroyed if it was created above.

The Attachment Module also handles three order calls:

trap_quits
trap_hangup
get_rf

These calls are handled entirely by the Attachment Module. The last call may be of interest to the DSM itself (see Device Profiles, below). The names of the entry points of the Attachment Module are the above eight names of calls handled.

The Request Queuer

The Request Queuer (see BF.2.24) is called to pass calls to the DCM. Basically, the Request Queuer stores a representation of an outer call in a TBE associated with a transaction block allocated in the DSM's auxiliary chain by the Request Queuer, and then signals an event that causes the DMP to wake up. In the DMP, the Dispatcher calls the Driver which reconstitutes the call and passes it to the DCM. The Driver is also responsible for

updating status and signaling events.

For each queuable outer call (for a list, see BF.2.24), there is an entry point of the Request Queuer. This call includes all of the normal arguments other than the ioname and the status arguments. There is only one DCM per DSM, so no ioname needs to be passed to the Queuer. The DSM gets return status by making a call to rq$get_chain. Among the other arguments required by the Request Queuer are a transaction block index, an 18-bit mask, and two event channel names. The transaction block index returned by the Queuer is the index of the block allocated by the Request Queuer and threaded onto the auxiliary chain; the DSM can include this block in a down chain from a buffer or call transaction block chain. The status mask defines the conditions under which one of the event channels will be signaled. It is possible at a later time to change an event or mask.

The use of the Request Queuer puts restrictions on certain types of outer call arguments. Specifically, delayed use arguments (workspace pointers and nelemt arguments of read and write-type calls, see below) must reside in the DSM's per-ioname segment. This implies that the DSM must have its own intermediate buffers for data, since the DCM cannot transmit directly into the user's workspace. The location of workspaces in Request Queuer calls is restricted to reduce the number of segments the DMP must initiate and to simplify inter-process communication.

DSM Data Bases

The Attachment Module and the Request Queuer make use of a special data base in the per-ioname segment, the Inter-process Communication Block. This data base is accessed via a relative pointer in the segment header. The main body of the DSM is not interested in the ICB.

The primary data base of the DSM is the Per-Ioname Base (PIB). PIBEs (per-ioname base extensions) are chained together using relative pointers in the first word of each block. If there are more than two PIBEs, the first one should contain an array of relative pointers to the later PIBEs to increase speed of accessing.

The first driving table pointer in the PIB points to the mode control structure used by the Mode Handler.

The Request Queuer uses the auxiliary transaction block chain for communication with the Driver.

Registry Files and Device Profiles

DSMs need to get at device profiles from the Registry Files for the devices they handle. The device profiles are used to hold

relatively constant information. A Registry File may hold permanent information, such as the association between a particular tape drive and a particular tape controller. A Registry File may contain temporary information such as the tab settings on a typewriter, but a Registry File may not be used to contain information as transient as the present position of a typewriter carriage. Typically, the DSM examines the profile at attach and restart time, or when a status return from the DCM indicates a change in the profile. Data is stored into the profile by the DCM and DSM as necessary.

In order to get at the profile, the DSM uses the Registry File Maintainer (see BF.2.22). A call to attm$get_rf returns the information needed to find the Registry File implied by the type and description arguments of the attach call. Registry Files are identified by two 32-character strings called type and name. These strings are returned by the above call to the Attachment Module and may be used in calls to the Registry File Maintainer (RFM). A call to rfm$get_devices will return a list of resource names (for use in calling the GIM, for example), and a list of device types. A call may then be made to rfm$get_profile to get the desired information. If the given RF is not the desired one, the type and name of the next one in the chain of connected devices may be ascertained via a call to rfm$get_down.

As an example of the use of Registry Files, consider magnetic tape. The call to attm$get_rf will return the type and name of the tape reel. The call to rfm$get_devices will therefore return a code indicating that the device is a reel of magnetic tape. The call to rfm$get_profile uses the type and name of the file, and index of the device (found by checking the device types if more than one is possible), a pointer and the number of bits desired. These data in the profile will be passed back for use by the DSM. The device profile of a magnetic tape would presumably contain such information as the amount of data on the tape, number of tracks and density at which the tape was recorded, and possible the Registry File names of other tape reels if the given reel is part of a multi-reel file. If it is necessary to store information in the profile, the DSM may call rfm$set_profile, which overwrites the entire profile. A call to rfm$get_down will return the type and name of the device on which the reel is mounted, the tape drive. A further call to rfm$get_down using the name of the tape drive would get the name of the Registry File for the tape controller to which the drive is connected.

Attachment

In response to an attach call, the DSM first stores the ioname, type, and description arguments in the appropriate parts of the PIB. Then, the DSM calls attm$attach, as described above. If no errors are detected by the Attachment Module, pib.bmode contains a valid mode string, the DCM has been attached, and the SFM (if

any) has been spliced in. The DSM then allocates the first PIBE and does any other processing needed to initialize itself.

Specifically, the DSM may need to allocate extra resources for its own use. For example, the tape DSM will have to allocate all reels other than the first of a multi-reel file.

Diversion

The divert outer call creates a new iopath for a device. To handle the diversion, the DSM calls attm$divert. Because divert outer calls pass through I/O segment locks, care must be taken not to disturb the contents of the IS. No transaction block is allocated by the Switch for this call.

The new DSM is created in two steps by the Attachment Module: First, the switching complex is called to establish a new node. All of the processing for handling an attach is done except that no call is passed to the DSM. The Attachment Module then initializes parts of the new per-ioname segment (the ICB and the parts of the PIB containing the three arguments of the attach call that would have been passed). After the DCM has been attached, an "attach" order call is made for the new DSM. This special order call is supposed to cause the DSM to do all of the attach call processing other than the initial storing of arguments the items in the PIB and the calling of the Attachment Module.

If any manipulation of media is required, the DSM must call the Media Request Module (see BT.2.02). The Attachment Module does not make any calls to the Media Management Module.

Detachment

When the DSM gets a detach call, it is supposed to force out any remaining I/O and then detach the device. First, the DSM should call the Mode Handler. If the modes are invalid, it should return immediately. Otherwise, the DSM should complete all pending transactions. If the UNLOAD mode is specified, the DSM should call the Media Request Module to unload any media it has loaded. If the RELEASE disposal mode is specified, the DSM should deallocate any devices it explicitly allocated. After the DSM has done all of its internal cleanup, it should call attm$detach. If there are no errors in the performance of that call, the DCM will have been detached upon return. The DSM should then call atm$delete_ioname with the delayed bit ON and then return. Upon return, the ATM will free any transaction blocks held for the DSM and will then destroy the DSM's per-ioname segment and the DSM's information in the Attach Table.

Process-Dependent Information

Since the DSM is expected to operate in several different processes for several different ionames, it must be very careful about process-dependent data. Specifically, care must be taken with pointers and with event channel names. Certain pointers are handled automatically by the Switch (the driving table pointers). However, the DSM will have to store a workspace pointer in a read call with asynchronous workspace. It is necessary to store the process id for which the pointer is valid along with the pointer; the pointer cannot be used unless the DSM is operating in the proper process. Event channels are a somewhat different problem. The DSM will need at least one event channel for use in calls to the Request Queuer. It is suggested that the DSM keep a table with process ids and event channel names to avoid creation and destruction of channels. Furthermore, the DSM must call rq$give_access before the first use of an event channel name in a Queuer call. (The DMP may be a universal DMP in a different process group, and must be given permission to signal on the event channel.)

Peculiarities of the DSM-DCM Interface

The DSM calls the Wait Coordinator (see BQ.6.06) to wait for an event to be signaled if it must synchronize itself with the actions of the DCM. Such a need will arise if the synchronization modes require the DSM to return only after a transaction is complete in some sense. If the DSM knows in advance that it will have to wait for a particular condition to hold, it will set the status_mask and create an event channel before calling the Request Queuer.

Normally, the DSM creates event channels and passes their names to the Request Queuer. However, it is possible for the user to take over synchronization management. He informs the DSM of his intention by adding an extra pointer argument when he makes an outer call. (see BF.2.02) The I/O Switch will assume that the extra argument points to a structure containing two event channel names. These channel names will be copied into the DSM's PIB (pib.sync_event and pib.error_event). It is the DSM's responsibility to use these event channel names, if non-zero, instead of the event channels it would normally have used. Thus, the error event should be used in all calls to the Request Queuer relating to the outer call, and the completion (sync) event should be used for the final Queuer call.

It is sometimes necessary for the DSM to synchronize itself with a transaction in progress. This is the case if the DSM has requested advancd input from the DCM, and then receives a read call with synchronous workspace. In this case, the DSM must wait until a certain number of characters have been read in (or until a read delimiter is reached).

The following technique should be used in such a case: Whenever the DCM is called and observes an interesting status change, it inverts the "special happening" bit in the call-oriented status field of the outer call(s) affected by the interrupt. Normally, the interesting event will be a change in the number of elements transmitted, but the DSM may inform the DCM of a different criterion to use by means of order calls. When the TBM is called to getstatus for that call, it will set bit 10 to one in order to indicate that there has been a status change since the last getstatus. Therefore, if a status mask with only bit 10 equal to 1 is used, the Driver will signal an event when the DCM returns. If the DSM is still not satisfied by the status of the call, it can make a call to rq$new event and then wait for another signal.

Whenever the DSM returns before logically completing (see BF.1.04) a read or whenever the DSM returns before completing a write where the contents of the workspace were not copied into a buffer in the DSM's IS, it must save a pointer to the user's workspace. Because pointers are not valid except in their process of origin, the DSM must associate a process id with each such pointer. Later calls will be able to move data to or from the user's workspace only if made in the same process as the original call.

Status Updating and Error Handling

In order to get status for calls made to the DCM, the DSM calls rq$get chain. If the ERRFIX mode is specified, the DSM is supposed to try to correct errors if they occur. For example, on output, the DSM usually restarts the transaction at a reasonable point (such as the beginning of the last page on a line printer or the beginning of the last line on a typewriter or the beginning of the last physical record on tape or cards). This error correction is done when a device error is detected or when a transaction is marked "aborted due to quit". If the ERRRET mode is specified, the DSM just marks its status for the call indicating the error and return. This mode would be used by a RUNOFF type program that could not tolerate extra lines on a typewriter, or by a system tape generator that could not tolerate error records on tape.

If a transaction is marked "aborted and reset", the DSM marks all of its outstanding transactions with the same status, aborts any other work it has planned, and returns. (Transactions will be reset when a revert call with the RESET mode specified is made after a diversion).

Reading and Writing

As discussed above, the DSM is responsible for handling read-ahead and write-behind.

If the DSM handles a read call with asynchronous workspace, it must keep track of the read and break delimiters at the time of the call since subsequent setdelim calls may change these delimiters. However, the DSM is required to make it appear as if the calls were handled, in their entirety, in the proper time sequence.

As mentioned above, workspaces for DCM calls must reside in the DSM's per-ioname segment. This is also true of nelemt arguments. (nelemt is the number of elements actually transmitted by a read or write call; arrays of such arguments are passed in readrec and writerec calls.) Because all DCMs operate in workspace asynchronous mode, nelemt would always be zero upon return if the normal definition were used. Therefore, DCMs interpret nelemt differently than do other outer modules. nelemt and is updated whenever the DCM updates its own status. It equals the number of elements physically transmitted at any given time; nelemt attains its final value when bit 5 of status (no more status change) becomes 1. Therefore, it is recommended that nelemt be kept in a TBE associated with the call, since disaster will ensue if the storage is freed too soon.

Device Control Modules

The following is a brief discussion of some points relevant to Device Control Modules (DCMs).

First, there is only a single DCM attached at a given time for a device. When the DSM is diverted and a new iopath is created, all pending transactions with the DCM are aborted. The newly-attached DSM then makes calls to the old DCM. When the path is reverted, it is necessary for the DSM to remind the DCM of the modes active at the time of the divert. This can be accomplished by use of an order call that passes a bit string equal to the DSM's "bmode". The use of this call implies that the mode control structures of the DSM and DCM are essentially equivalent.

When the DCM is first attached, it should set up communication with the GIM. In order to do this, it makes the following GIM call (see BF.20):

```
call hcs_$assign(resource_name,devx,event,type,rcode);
```

The resource_name to be passed to the GIM is stored in the related Registry File, and can be extracted by a call to rfm$get devices (see BF.2.22). Alternately, the resource name might be passed to the DCM as the description arguments of its localattach call. The event argument is the name of the event to be signaled whenever there is a hardware interrupt. The name of this event can be gotten via the use of the following statement:

```
dcl disp$hardware_event entry returns (bit(70));
event = disp$hardware_event;
```

DCM writers should read the beginning of BF.2.22, which gives a basic description of the use of Registry Files. DCM writers will, in particular, be interested in the device profile (see above).