Published: 01/10/68;
(Supersedes: BF.2.25, 08/14/67)

Identification

The Dispatcher
R. C. Daley and S. I. Feldman

Purpose

The Dispatcher is the module in each Device Manager Process that is the interface between the Wait Coordinator and the Driver. The Dispatcher is called when certain events are signaled by Device Strategy Modules in other processes. The Dispatcher handles six types of event channel: reassign, reenable, locall, quit, restart, and hardware. The basic data base for the Dispatcher is the Process Dispatching Table, which contains information on each of the devices that may be controlled by that process.

Introduction

In order to permit quick response to hardware interrupts, I/O devices are controlled by special processes called Device Manager Processes (DMPs). There are two classes of DMP, the universal DMP and the private DMP. A universal DMP can handle many devices for many different users; a private DMP can handle a single device for a single user. A private DMP is a member of the user group for which the device is to be run. The Dispatcher is the module in each DMP that handles event signals for any number of devices for any number of users. The basic data base of the Dispatcher is the Process Dispatching Table (PDT). This table is created before the DMP is initialized and contains an entry for each of the devices that the DMP may control.

It is assumed that the reader is familiar with the Wait Coordinator and the concept of an "event channel" (see Section BQ.6). The Dispatcher is responsible for creating certain event channels and for handling signals on those channels. The Dispatcher operates in conjunction with the Attachment Module (Section BF.2.23), with the Request Queuer, and with the Driver (Section BF.2.24).

In the following descriptions, an event call channel is "disabled" by declaring it to be an event wait channel and is "enabled" by declaring it to be an event call channel associated with the proper procedure.

The following is a summary of the event channels that are of interest to the Dispatcher:

There is one "reassign" event channel (relative priority 6) per DMP. It is signaled whenever the status of this process as the control user of a device is changed.

There is one "quit" event channel (relative priority 2) per device assigned to a DMP. When that event channel is signaled, the Dispatcher calls the quit entry of the Driver to make the device to stop.

There is one "restart" event channel (relative priority 3) per device assigned to a DMP. When that event channel is signaled, the Dispatcher calls the restart entry of the Driver to restart the path. This channel is enabled only when the route is in external quit condition.

There is one "hardware" event channel (relative priority 4) per device assigned to a DMP. This channel is signaled whenever a hardware interrupt is received for the device. In response, the Dispatcher calls the hardware entry of the Driver.

There is one "iocall" event channel (relative priority 5) per iopath per channel assigned to a DMP. This channel is used to inform the Driver that there is a new I/O call in the Request Queue, to force the Dispatcher to create a new iopath,to cause pushed-down paths to be deleted, and to restart a path that has been quit.

There is one "reenable" event channel (priority 1) per iopath per channel. Normally, this event channel is disabled. When certain data bases are found to be locked, the Dispatcher disables the other three per-device channels and temporarily enables this channel. When it is signaled, the event channel is disabled and the other event channels are re-enabled.

The Driver gets the list of Transaction Block Extensions containing outer call requests by calling the get chain entry of the Transaction Block Maintainer (see BF.2.20). However, those transaction blocks are located in the user's TBS, and not necessarily in the DMP's TBS. Therefore, before each call to the Driver, a call is made to tbm$tbs to temporarily switch TBS segments and to change the locking strategy. Also, the auxiliary transaction block chain is locked using the first lock list in the ICB.

Throughout this paper, "cstatus" is a bit string of length 18 which contains status information on a particular call.

The Process Dispatching Table

The Process Dispatching Table has an entry for each device that may be controlled by a given DMP. The PDT is created and initialized with the names of the devices to be controlled before the DMP itself is initialized. The following is a declaration of the PDT:

```
dcl 1 pdt based(p),               /*Process Dispatching Table*/
  2 init_proc char(32),           /*name of procedure to be
        "                           called for initialization.
        "                           Equal to "disp$init"*/
  2 dmp_proc_id bit(36),          /*id of this  Device Manager
        "                           Process*/
  2 reassign_event bit(70),       /*event channel to be signaled
        "                           when device is assigned or
        "                           unassigned to this process*/
  2 creator_id bit(36),           /*id of process that created this
        "                           Device Manager*/
  2 init_done_event bit(70),      /*event channel to be signaled when
        "                           initialization of this process is
        "                           complete.*/
  2 current ptr,                  /*pointer to element of routes
        "                           for device for which work
        "                           is being done at present*/
  2 pdt_name char(32),            /*name used by other processes to
        "                           find PDT*/
  2 dtabp ptr,                    /*pointer to Driver"s driving
        "                           table*/
  2 disp_ptr,                     /*pointers to entry points of
        "                           the Dispatcher*/
    3 reassign ptr,
    3 locall ptr,
    3 reenable ptr,
    3 restart ptr,
    3 quit ptr,
    3 hardware ptr,
  2 nroutes fixed bin(17),        /*number of entries in routes array*/
  2 routes(n),                    /*an entry for each device which
        "                           may be assigned to this process.
        "                           n = pdt.nroutes*/
    3 type char(32),              /*type of resource*/
    3 resource_name char(32),     /*resource_name for this device*/
    3 user_id char(50),           /*user to whom device is assigned*/
    3 ioname char(15),            /*DCM ioname, a unique character string*/
    3 pibp ptr,                   /*pointer to PIB for this DSM*/
    3 icbp ptr,                   /*pointer to ICB for DSM*/
    3 tbsp ptr,                   /*pointer to Transaction Block
        "                           segment in user"s group
        "                           directory*/
    3 att_stack ptr,              /*pointer to entry in attach_stack
        "                           area for pushed-down DCM*/
    3 locall_event bit(70),       /*event to be signaled by DSM
        "                           for localling, resetting,
        "                           inverting, and diverting*/
    3 restart_event bit(70),      /*signaled to restart a path
        "                           in external quit condition*/
```

```
     3 hardware_event bit(70), /*event channel signaled when
            "                    interrupt received from device*/
     3 quit_event bit(70),     /*event to be signaled to stop
            "                    device and prepare for a divert*/
     3 reenable_event bit(70), /*signaled when auxiliary
            "                    chain or TBS is unlocked*/
     3 device_absent bit(1),   /*1 if device not present*/
     3 assigned bit(1),        /*1 if device assigned to this
            "                    process*/
     3 attached bit(1),        /*1 if attach call has been
            "                    issued*/
     3 ext_quit bit(1),        /*1 if device in external quit
            "                    condition*/
     3 int_quit bit(1),        /*1 if device in internal (hardware)
            "                    quit condition*/
   2 attach_stack area((10000));/*area into which blocks are
            "                    allocated for diverted paths*/
/*


*/
dcl 1 att_thread based(p),     /*declaration of block to be
          "                      allocated into att_stack
          "                      area for pushing down of
          "                      DCMs*/
   2 ioname char(15),          /*DCM ioname*/
   2 iocall_event bit(70),     /*event channel name*/
   2 reenable_event bit(70),   /*event channel name*/
   2 pibp ptr,
   2 icbp ptr,
   2 status,
     3 attached bit(1),
     3 ext_quit bit(1),
   2 next ptr;                 /*points to next block in thread
          "                      of pushed-down DCMs*/
```

The creating process allocates the PDT, stores its process id in pdt.creator_id, stores the name of an event channel in pdt.init_done_event, stores the value of N in pdt.nroutes, and stores the name of one of the devices associated with each route in resource_name. The character string "dmp$init" is stored in pdt.init_proc.

When the Wait Coordinator makes a call to the Dispatcher in response to an event signal, it calls with a pointer argument. This pointer points at an element of the routes array. The Dispatcher uses an auxiliary structure "route" with a declaration equal to the declaration of an element of "routes" in conjunction with this pointer to access one of the relevant entries in the PDT.

Device Manager Initialization

After the PDT is created, the creating process makes a call to create_proc with the path name of the PDT as argument. This causes a process to be created, it causes the PDT to appear in the new process's process directory, and it causes a call to the procedure whose name equals the first 32 characters of the new segment. Therefore, the first 32 characters of the PDT contain the string "disp$init". The following call is made:

```
call disp$init(pdtptr);
dcl pdtptr ptr;
```

The pdtptr (a pointer to the PDT) is stored in internal static storage, and then the following call is made:

```
call dmp$init(pdtptr);
```

The following steps are taken in response to that call:

1. The process id of the DMP is stored in pdt.dmp_proc_id.

2. The attach_stack area of the PDT is initialized.

3. The assigned bit for each route is set OFF.

4. When an event channel is declared to be an event call channel (see BQ.6.02), a pointer to the procedure to be called when the channel is signaled must be provided. Call generate_ptr (see BY.13.02) to get pointers to the six entry points of the Dispatcher called by the Wait Coordinator. Coordinator and store them in the corresponding entries of pdt.disp_ptrs.

5. The reassign event channel is created and declared to be an event call channel. Whenever that channel is signaled, the Wait Coordinator will make the following call:

```
call disp$reassign(null,event_indicator);
```

The event_indicator is an array of three 70-bit strings containing the event channel name, the event id, and the sending process id. The event_indicator is passed as the second argument of all calls to the Dispatcher, but will be ignored.

6. Signal the init_done_event for process creator_id.

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

```
call tbm$init("1"b,cstatus);
```

8. Return.

The Inter-process Communication Block

The Interprocess Communication Block (ICB) is a part of the DSM's per-ioname segment (IS) and contains information used by the Attachment Module, Request Queuer, and Dispatcher. For completeness, a declaration of the ICB is included here. For a discussion of how this data base is allocated and initialized, see BF.2.23.

The following is a declaration of the ICB.

```
dcl 1 icb based (p),               /*inter-process communication block*/
  2 queue_lock_list bit(144),      /*standard lock for request queuing*/
  2 locall_event bit(70),          /*event channel name*/
  2 dmp_proc_id bit (36),          /*device manager process id*/
  2 dmp_user_id char(50),          /*user id of dmp if not private*/
  2 private_dmp bit(1),            /*1 if a private DMP was created*/
  2 quit_event bit(70),            /*event name*/
  2 restart_event bit(70),         /*name of event channel to be signaled
        "                            to restart path in DMP without
        "                            passing an outer call*/
  2 reset bit(1),                  /*set to 1 to cause a reset
        "                            of all calls in request queue
        "                            when next restart is done*/
  2 invert bit(1),                 /*set to 1 to cause diverted paths
        "                            in DMP to be detached*/
  2 invert_event bit(70),          /*name of event channel to be
        "                            signaled when inversion complete*/
  2 divert bit(1),                 /*set to 1 to cause present iopath
        "                            to be quit*/
  2 divert_event bit(70),          /*name of event channel to be
        "                            signaled when diversion complete*/
  2 trap_quits bit (1),            /*if 1,signal if quit occurs
        "                            on device*/
  2 overseer_trap_hangup bit(1),   /*if 1, signal overseer if
        "                            hangup occurs on device*/
  2 trap_hangup bit(1),            /*if 1, signal if hangup occurs on
        "                            device*/
  2 quit_id bit (36),              /*id of process to be signaled on quits*/
  2 overseer_id bit(36),           /*process id of overseer*/
  2 hangup_id bit(36),             /*id for process to be signaled
        "                            when device hangs up*/
  2 quit_report_event bit(70),     /*event signaled if device quit*/
  2 overseer_hangup_report_event bit(70),  /*event to be
        "                            signaled if hangup occurs*/
  2 hangup_report_event bit(70),   /*event to be signaled if
        "                            device hangs up*/
  2 diverted bit(1),               /*1 if this ioname has been
        "                            diverted*/
  2 divert_type bit(1),            /*when diverting, set to 1 if
        "                            the two ioname arguments are
```

```
        "                      equal*/
  2 alloc_down bit(1),       /*how this registry file was reached.
        "                      If ON, device of given type was
        "                      allocated and name returned.
        "                      Otherwise, name came from description
        "                      argument of call.*/
  2 dsm_rf_type char(32),    /*type of first RF (highest level)*/
  2 dsm_rf_name char(32),    /*name of first RF*/
  2 dcm_type char(32),       /*type to be used in attach
        "                      calls to the DCM*/
  2 dcm_description char(32), /*description to be used in attach
        "                      calls to the DCM*/
  2 nchar_dcm_mode fixed bin(17), /*number of characters
        "                      in dcm_mode*/
  2 dcm_mode_relp bit(18),   /*relp to character string
        "                      equal to mode of DCM*/
  2 old_dsm_ioname char (32), /*previous dsm ioname*/
  2 new_is_name char(32),    /*for use when diverting.
        "                      Name of new
        "                      per-ioname segment*/
  2 dcm_ioname char(32),     /*for possible future use in
        "                      handling NODMP mode*/
  2 old_dcm_ioname char(32), /*same as above*/
  2 icb_lock_list bit(144),  /*standard lock*/
  2 invert_proc_id bit(36),  /*response event for invert*/
  2 divert_proc_id bit(36);  /*response event for divert*/
```

Device Reassignment

Whenever the Attachment Module assigns a device to a user group, it signals the reassign event for the appropriate DMP. This causes, as described above, a call to disp$reassign. In response to this call, the following is done for each element of the routes array in the PDT:

1. Make the following call to the Device Assignment Module (see Section BF.2.26):

```
call ioam$get_assignment(type,resource_name,user_id,cstatus);
dcl resource_name char(32),  /*from the PDT*/
    user_id char(50),     /*return argument:  user to whom
                               device is assigned*/
    cstatus bit(3);       /*status for this call*/
```

If cstatus is zero, then this DMP is the control user for this device. If bit 1 is ON, then there is an error in the PDT: the device does not exist. Otherwise, this DMP is not the present control user for this device.

2. If this DMP is not the control user for this device, then

   a. If the assigned bit in the route is OFF, then go on to the next route in the PDT.

   b. If the assigned bit is ON, then call the internal detach procedure and then go on to the next route in the PDT.

3. If this DMP is the control user for this device, then

   a. If the assigned bit in the PDT entry is OFF, go to step 4.

   b. If the assigned bit in the PDT entry is ON and if the user_id in the PDT entry is equal to the assigned user of the device, go on to the next route in the PDT.

   c. If the assigned bit in the PDT entry is ON but the user ids do not match, call the internal detach procedure and then go on to the next step.

4. Store the user_id returned by the IOAM in the user_id entry in the PDT.

5. Create a unique name (by a call to unique_chars) and store that name in the ioname entry for the route.

6. Initiate the per-ioname segment (IS), which can be found by a link with name equal to the resource_name in the user's group directory. (Use the name created in step 5 above as the call name.) Using this pointer and relative pointers in the DSM ioname segment header, get pointers to the PIB and to the ICB.

Store these pointers in pibp and icbp, respectively.

7. Set the assigned bit ON, set the attached, ext_quit, and int_quit bits OFF.

8. Create the reenable, locall, quit, restart, and hardware event channels and declare them to be event call channels. If the assigned user of the device is not the same as the present user, give the assigned user access to these event channels. When these channels are signaled, the Wait Coordinator will call the corresponding entries of the Dispatcher with a pointer to the appropriate element of the routes array as an argument. Store the event channel names in the corresponding entries in the PDT. Disable the hardware, quit, restart, and reenable event channels.

9. Find the Transaction Block Segment (TBS) in the user's directory and store a pointer to it in the tbsp entry in the PDT.

10. Go on to the next entry in the PDT.

When all entries in the PDT have been checked, return to the Wait Coordinator.

Quit Conditions

A route is in one of three "quit conditions": no quit, internal quit, and external quit. The normal condition is no quit. If the trap_quits bit is OFF in the ICB when a hardware quit is detected, the path is restarted and the route remains in no quit condition. If the trap_quits bit is ON when a hardware quit is detected, the path is placed in internal quit condition, the event channel set by the last trap quits call is signaled and the locall and hardware channels are disabled.

When the quit event channel is signaled, the path goes into external quit condition. If the path had been in internal quit condition, then the locall is is re-enabled. Otherwise, the hardware event channel is disabled and driver$quit is called to abort outstanding transactions.

When the locall or restart event is signaled, the hardware event is enabled, the restart event is disabled, and the path is restarted and removed from external quit condition.

The Hardware Event

When an interrupt is received from the device, the hardware event is signaled. In response to this signal, the Wait Coordinator makes the following call:

```
call disp$hardware(p,event_indicator;
```

```
dcl p ptr;  /*p points at the routes entry in the
              PDT for the device that
              caused the interrupt*/
```

The following steps are taken in response to this call:

1. Call the Locker to lock the DSM's auxiliary transaction block chain. If the lock attempt succeeds, go to step 2. Otherwise, enable the reenable event channel, signal the hardware event, disable the locall, restart, hardware, and quit events, and return.

2. Make the following call to the Transaction Block Maintainer (see BF.2.20):

```
call tbm$tbs(p->route.tbsp,p->route.reenable_event,cstatus);
```

3. Make the following call the the Driver:

```
call driver$hardware(p->route.ioname,p->route.pibp,cstatus);
```

4. If the return status indicates that the Driver made an unsuccessful attempt to lock the user's TBS, enable the reenable event channel, signal the hardware event, disable the locall, restart, hardware, and quit event channels, and return.

5. Go to check_status.

The Quit Event

When the quit event is signaled, the device is stopped and placed in external quit condition. The Wait Coordinator makes the following call when the quit event for a device is signaled:

```
call disp$quit(p,event_indicator);
dcl p ptr;   /*p points to the element of
               of the route array corresponding
               to the device that interrupted*/
```

The following steps are taken in response to this call:

1. Call the Locker to lock the DSM's auxiliary chain using the name of the reenable event channel as argument. If the attempt to lock succeeds, go to step 2. Otherwise, signal the quit event, enable the reenable event channel, disable the locall, restart, hardware, and quit events, and return.

2. Call tbm$tbs.

3. Make the following call to the Driver:

```
call driver$quit(p->route.ioname,p->route.pibp,
```

p->route.int_quit,cstatus);

This call stops the device and aborts all pending transactions.

4. If the returned status indicates that the Driver made an unsuccessful attempt to lock the TBS, signal the quit event, disable the quit, restart, locall, and hardware events, enable the reenable event channel, and return.

5. Enable the restart event for the route.

6. If the route is not in internal quit condition, disable the hardware event and go to step 8.

7. If p->route.int_quit is ON, re-enable the locall event and turn that bit OFF.

8. Set the ext_quit bit ON for the route and reset any pending locall events.

9. Return to the Wait Coordinator.

The iocall Event

The iocall event is used for causing an I/O call to be performed. It is also used to force the Dispatcher to examine certain bits in the ICB and PDT and, in response to these bits, to divert a path, to invert a path, or to restart a path. When the iocall event is signaled, the following call is made by the Wait Coordinator:

```
call disp$iocall(p,event_indicator);
dcl p ptr;
```

The following steps are taken in response to this call:

1. Call the Locker to lock the DSM's auxiliary chain. If the lock attempt succeeds, go to step 2. Otherwise, signal the iocall event, disable the iocall, restart, hardware, and quit events, declare the reenable event to be a call event channel, and return.

2. Call tbm$tbs.

3. If the ext_quit switch is ON, re-enable the hardware event.

4. If the divert bit in the ICB pointed to by icbp is ON, then do the following:

   a. Reset all waiting iocall events and disable the iocall event channel.

   b. Turn off the divert switch in the ICB.

   c. If the ext_quit switch in the PDT is OFF, turn it ON and make the following call:

   ```
   call driver$quit(p->route.ioname,p->route.pibp,cstatus);
   ```

   d. If the returned status from the call indicates that the Driver made an unsuccessful attempt to lock the TBS, signal the iocall event, enable reenable event, disable the hardware, quit, iocall and restart event channels, and return.

   e. Allocate an att_thread block. Store the present ioname, pibp, icbp, iocall event channel name, reenable event channel name, ext_quit status bit, and attached status bit in the block. Thread the block on the head of the att_stack chain.

   f. Compute a unique ioname and store it in the PDT entry.

   g. Set the ext_quit bit OFF in the route.

h. Set the attached bit OFF in the route.

i. Initiate the new per-ioname segment, which can be found with name equal to icb.new_is_name in the user's group directory. Using the header of the new IS, get pointers to the new PIB and ICB. Store these pointers in pibp and icbp.

j. Create a new locall event channel, declare it to be an event call channel, and store the event channel name in the locall entry of the PDT route. If the assigned user is not the present user, give the assigned user access to the new channel.

k. Create a new reenable event channel and store it in the reenable_event entry in the PDT.

l. Signal the event channel with name equal to icb.divert_event for the process with id icb.divert_proc_id.

m. Go to check_status.

5. If the invert bit in the ICB is ON, then do the following. For each att_thread block chained to this PIB entry,

a. Make the following call:

```
call driver$detach(att_thread.ioname,att_thread.pibp,cstatus);
```

b. Destroy the event channel with name att_thread.locall_event.

c. Destroy the event channel with name att_thread.reenable_event.

d. Terminate the segment pointed to by att_thread.pibp.

e. Free the att_thread block.

After all of the att_thread blocks have been freed, do the following:

a. Set p->route.att_stack equal to the null pointer.

b. Turn off the icb.invert bit.

c. Signal the event channel with name equal to icb.invert_event for the process with id equal to icb.invert_proc_id.

d. Go to check_status.

6. If the route is in external quit condition, do the following:

a. Turn the ext_quit bit OFF.

b. Make the following call:

```
call driver$restart(p->route.pibp,icb.reset,cstatus);
```

c. Disable the restart event.

d. If the reset bit in the ICB is ON, turn it OFF and go to check_status.

7. If the divert and invert bits are both OFF in the ICB, then this is a call to perform an I/O call. If the device is in either internal or external quit condition, this call is an error. If the attached bit in the PDT is ON, go to step 8. Otherwise, make the following call:

```
call driver$init(p->route.ioname,p->route.pibp,cstatus);
```

Store the "device absent" return status bit in p->route.device_absent. If the return status indicates that the device is now _attached_, set the attached bit in the PDT ON and enable the hardware and quit events for the route. Go to check_status.

8. If the device has already been _attached_, call _driver$iocall_ using the same arguments as in the above call. If, upon return, the device has not been _detached_, return. Otherwise, do the following:

a. Destroy the iocall event.

b. Terminate the segment pointed to by pibp.

c. Set pdt.current equal to the null pointer.

d. Destroy the present reenable and iocall event channels.

e. If the att_stack pointer for the route is null, set the attached bit in the PDT entry OFF and destroy the quit and hardware event channels, set the assigned bit OFF, and go to check_status.

f. If the att_stack pointer in the PDT entry is not null, pop up the pushed_down path by copying the pibp, icbp, ioname, iocall event channel name, reenable event channel name, and status bits from the top att_thread block, free that block, and update the att_stack pointer. Re-enable the iocall event. If the ext_quit switch is now ON, disable the hardware event. Go to check_status.

The Restart Event

When the restart event is called, the Wait Coordinator makes the following call:

```
call disp$restart(p,event_indicator);
```

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

1. Call the Locker to lock the DSM's auxiliary transaction block chain. If the lock attempt is successful, go to step 2. Otherwise,signal the restart event, disable the locall, restart, quit, and hardware event channels, enable the reenable event channel, and return.

2. Call tbm$tbs.

3. Make the following call to restart the path:

```
call driver$restart(p->route.pibp,icb.reset,cstatus);
```

4. Turn off the reset bit in the ICB.

5. Turn the ext_quit bit OFF.

6. Disable the restart event channel.

7. Return.

The Reenable Event

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

```
call disp$reenable(p,event_indicator);
```

In response to the call, the Dispatcher takes the following steps:

1. If the route is not in internal quit condition, enable the hardware event channel.

2. If the route is in neither internal nor external quit condition, enable the locall event channel.

3. If the route is in external quit condition, enable the restart event channel.

4. Enable the quit event channel.

5. Disable the reenable event channel

6. Return to the Wait Coordinator.

Check status

After each call to the Driver, the following is done to check for quit signals and hangups:

1. If the status returned by the Driver indicates that there has been a quit and if the trap_quits bitin the ICB is OFF, make the following call:

```
call driver$restart(p->route.pibp,"0"b,cstatus);
```

Go to step 3.

2. If the returned status indicates that there has been a hardware quit and if the trap_quits bit is ON, then do the following:

   a. Set the int_quit bit in the route ON.

   b. Signal the quit_report_event in the ICB for process quit_id (in the ICB).

   c. Disable the locall and hardware events.

3. If the returned status indicates that the device is absent and if the device_absent bit in the route is OFF, then do the following:

   a. If icb.overseer_trap_hangup is ON, then signal the overseer_hangup_report_event for process overseer_id.

   b. If the trap_hangup bit is ON, signal the hangup_response_event for process hangup_id.

   c. Go to step 5.

4. If the device is not absent, then set the device_absent bit in the route OFF.

5. Return to the Wait Coordinator.

Internal Detach Procedure

Whenever it is necessary for the Dispatcher to detach a route, the following call is made:

```
call detach(p);
dcl p ptr;      /*pointer to the appropriate
                  route in the PDT*/
```

The following steps are taken in response to this call:

1. Call the Driver to detach the present path:

```
call driver$detach(p->route.ioname,
      p->route.pibp,cstatus);
```

2. Call the Segment Management Module to terminate the segment pointed to by p->route.pibp.

3. Destroy the reenable, locall, hardware, restart, and quit event channels.

4. For each att_thread block for this route, do the following:

   a. Terminate the segment pointed to by att_thread.pibp

   b. Destroy the reenable and locall event channels.

   c. Free the att_thread block.

5. Set the assigned bit in the route OFF.

6. Set the att_stack pointer for the route equal to null.

7. Return.

Special Call for DCM Usage

The following call is provided in order to permit a DCM to discover the name of the hardware event channel for its device:

```
call disp$get_hardware(hardware_event,alone);
dcl hardware_event bit(70),
    alone bit(1);         /*equal to one if the present path
     "                      is the only one*/
```

This call sets _hardware event_ equal to pdt.current->route.hardware_event, sets _alone_ ON if and only if the present path is the only one for this device (i.e., the att_stack pointer is null), and returns.

Summary of Dispatcher Calls and Arguments

```
call disp$init(pdtptr);
call disp$reassign(anyptr,event_indicator);
call disp$hardware(p,event_indicator);
call disp$quit(p,event_indicator);
call disp$locall(p,event_indicator);
call disp$restart(p,event_indicator);
call disp$reenable(p,event_indicator);
call disp$get_hardware(hardware_event,alone);


dcl pdtptr ptr,
    anyptr ptr,                  /*ignored*/
    p ptr,                       /*point to a route entry*/
    event_indicator(3) bit(70),/*standard event indicator*/
    hardware_event bit(70),      /*name of hardware event*/
    alone bit(1);                /*equal to one if the present
                                   path is the only one*/
```