SOFTWARE COMPONENT SPECIFICATION

SYSTEM:	LEVEL 6 MOD400 OPERATING SYSTEM
SUBSYSTEM:	LOCAL AREA NETWORK
COMPONENT:	LACS DRIVER INTERFACE SERVICES
PLANNED RELEASE:	MOD400 4.0
SPECIFICATION REVISION NUMBER:	B
DATE:	JULY 19,1985
AUTHOR:	PETER STOPERA

This specification describes the current definition of the subject software component, and may be revised in order to incorporate design improvements.

TABLE OF CONTENTS

REFERENCES

[1] CLM User Extensions, Richard Taufman, May 14,1979.

[2] Engineering Product Specification (H/W), Local Area Controller Subsystem (LACS), Rev F, A. C. Hirtle, Oct 4, 1984.

[3] Engineering Product Specification, LAN Software, R. Dhondy, Aug. 16, 1985.

[4] LAN S/W Component Specification, System Management, D. O'Shaughnessy
Aug. 16, 1985.

[5] LAN S/W Component Specification, LACS Driver 802 Logical Link Control Layer Server,
P. Stopera, Aug. 16, 1985.

[6] LAN S/W Component Specification, LACS Driver Megabus Services,
P. Stopera, Aug. 16, 1985.

[7] Lan S/W Component Specification, Configuration Requirments
L. Vivaldi, Aug. 16, 1985.

[8] LAN S/W Component Specification, LACS Link Layer Protocol, H. King
Aug. 16, 1985.

[9] L6 LAN Data Structures, P. Stopera - Aug. 16, 1985

1 INTRODUCTION AND OVERVIEW

1.1 BACKGROUND

The lacs driver interface services (ldis) is a component of the lacs driver in the lan subsystem. The ldis is the lacs driver's interface to a user application, and the lacs driver's layer servers (ls).

1.2 BASIC PURPOSE

The ldis has several purposes, they are: to receive requests from applications using the lan subsystem, to determine which layer server to invoke from requests issued to the lan subsystem, to provide common routines used by the layer servers, to provide initialization services routines, to provide termination services routines, to provide power failure routines to handle power failure in the 16.

1.3 BASIC STRUCTURE

Figure 1 shows the relation of the ldis to the other components of the lacs driver. Figure 2 shows the subcomponents of the ldis. The following is a brief description of the functions of each subcomponent of the ldis:

lorb processing routines - The lacs driver must have routines in place to interface with the user. The user will use the $rqio interface. The lorb processing routines in the ldis are routines which interface with the user. These routines accept the $rqio from the user and release the $rqio when the lan subsystem is finished processing the request. The lorb processing routines are defined below:

interface services pre lorb processing - This routine is responsible for receiving the user request, performing some validation of the request, placing the request on the appropriate rct queue (the rct is know from the lrn in the lorb), then invoking either a layer server, or either an initialization service routine or a termination service routine.

interface services post lorb processing - This routine is responsible for posting all requests back to the user, after the layer server or one of the initialization or termination services has completed processing the request.

common layer server routines - The layer servers are structured such that they contain common functions to perform in processing requests. The common layer server routines in the ldis are the common functions used by the modules of the layer servers. The common routine are defined below:

LAN L6 SOFTWARE ARCHITECTURE

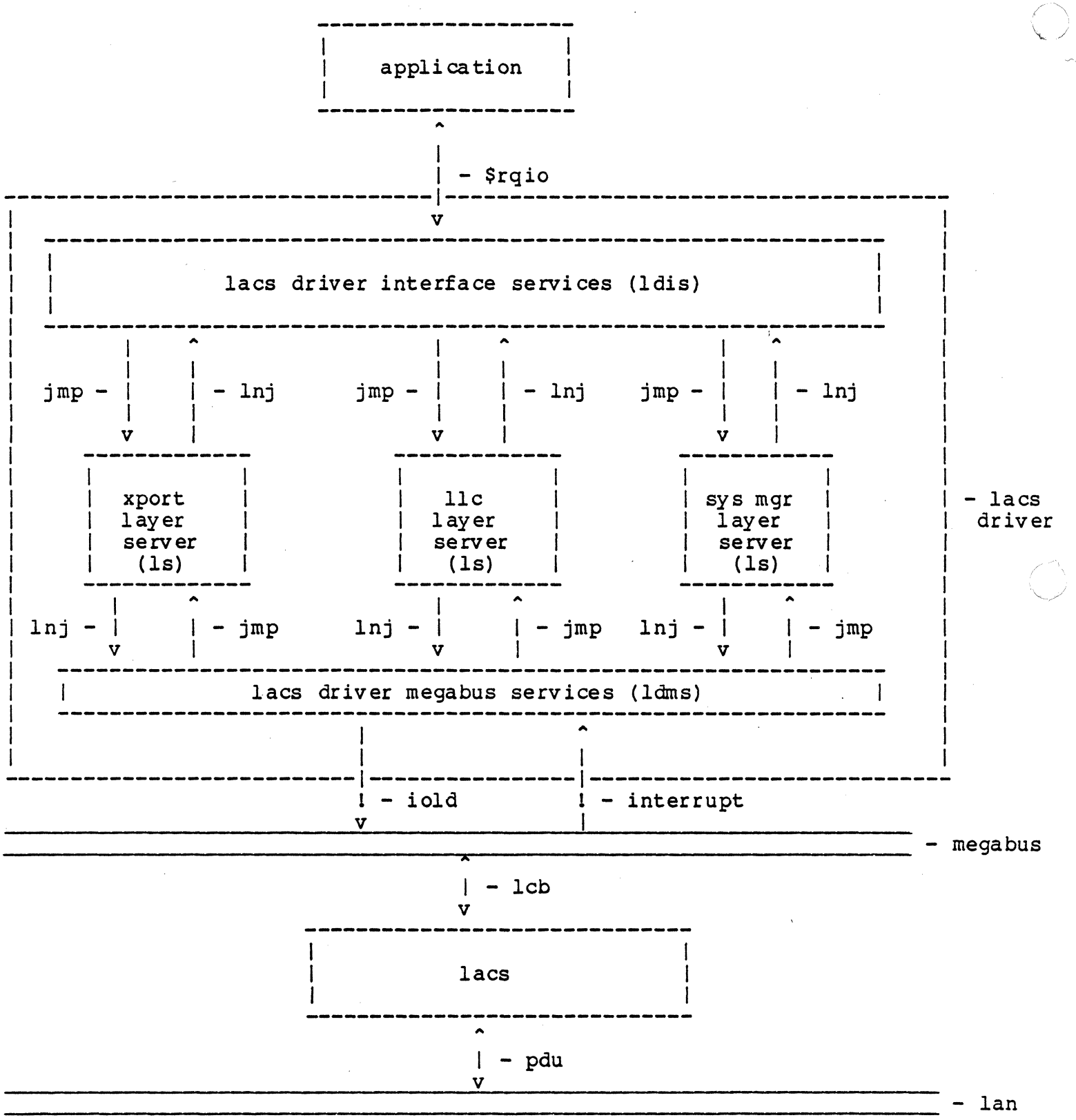


figure 1

LACS DRIVER INTERFACE SERVICES SUBCOMPONENTS

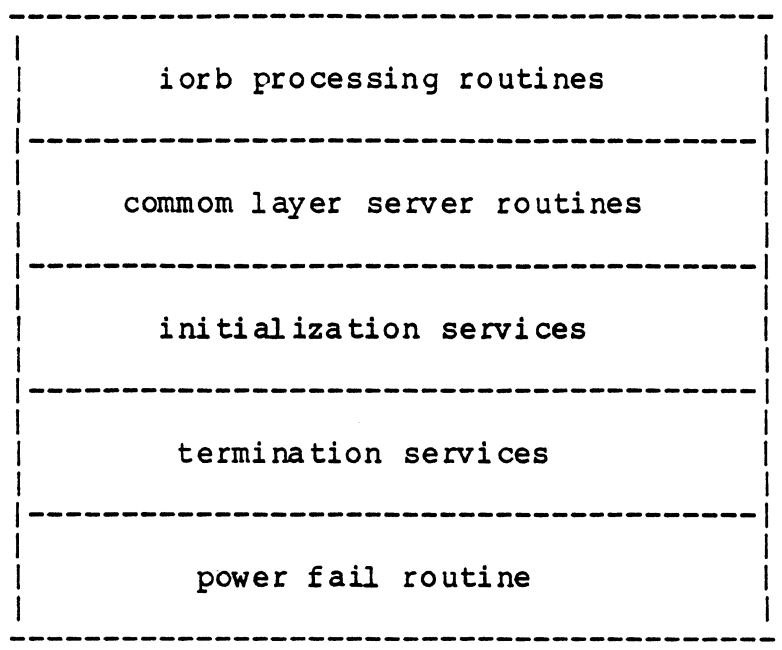


figure 2

iorb validation routine - This routine will validate fields that are common to all layer servers.

build a icb routine - This routine will fill in fields that are common to all layer servers in the icb.

terminate task - This routine will terminate a task level when called.

event routine - This routine will perform some event request processing for a layer server.

assign segement visiblity - This routine will call the executive to gain visiblity to a user's segement so a layer server can access the iorb's buffer.

buffer address absolutizing routine - This routine will call the executive to convert a virtual address into a physical address.

More common function routines will be added as development continues.

sap initialization services - A user of the lan subsystem will need to retrieve information from the lacs driver in order to use the lan subsystem. The initialization services routines in the idis allows a user to retrieve information to use in it's $rqio interface. The initialization services aslo provides service to perform initialization for a sap (remote and local). The routines are defined below:

associate local user - This routine will return a lrn to a user, from a supplied symbolic name.

activate local sap - This routine activates the local sap in the controller and the level 6.

activate remote sap - This routine return a logical remote sap address to the user, from a supplied symbolic name.

sap termination services - The user of the lan subsystem may, at some point in it's processing, wish to end it's association with the lan subsystem. The termination services routines in the idis are used to terminate a user's association with the lan subsystem and to terminate all connections associated with the sap. These routines process all termination services whether issued by the user or the executive (disconnect with queue abort when a task group is abnormally aborted). The routines are defined below:

deactivate local sap - This routine process deactivate with queue abort iorbs (disconnect with queue abort in l6 jargon). It clears all queue associated with the sap, then deactivates the local sap (places in a inactive state) in the l6 and in the controller. In a c o n n e c t i o n

oriented enviroment, this routine causes any connection associated with the sap to be disconnected, all other queues to be cleared and the sap to be placed in a inactive state.

deactivate remote sap - This routine will deactivate remote saps, the routine is needed to clear up resources held by remote saps.

power fail routine - The lan subsystem must have routines in place to handle power failure (pf) by the 16. The pf restart routine in the idis are required because the goal of the pf restart routines is to reinitialize the lacs and make it's services available again. The routines will place the lan subsystem into a state where it was when the 16 was booted, (i.e. no user had interfaced to the lan subsystem). The routines are defined below:

1st power failure routine - This routine is invoked by the executive at level 2 during it's power failure restart operation. The routine simulates an interrupt to the lowest lan subsystem interrupt level (highest priority), then changes the p-counter in the lowest level's tcb to the start of the 2nd power fail routine.

2nd power failure routine - This routine is invoked as a result of the 1st pf routine simulating an interrupt and changing the p-counter for it's level. The routine will post all current request outstanding to the lan subsystem back to the user, cleans up all structures, and if any levels are active, the routine deactivates the levels.

1.4 BASIC OPERATION

1.4.1 BASIC FLOW OF THE IORB PROCESSING ROUTINES

1.4.1.1 INTERFACE SERVICES IORB PREPROCESSING ROUTINE

The interface services iorb processing routine will be invoked as a result of any request made to the lan subsystem (execpt associate local user). The routine will fetch the lrn from the iorb, then index into the lrt by the lrn to obtain the pointer to the rct. The routine will test the active bit in the rct, if the bit is not set and if the request is not an activate, the routine will call the exec dequeue and post routine to post the request with an error. Otherwise, the routine will dequeue the lrb off the tcb queue, and enqueue the lrb on the tail of the lrb queue on the rct. The routine will then fetch the iorb function code, and if the request is not one of the initialization or termination routine functions the routine will then fetch the start address of the layer server from the rct, and jump to the address. Otherwise, the routine will jump to the appropriate initialization or termination service routine.

1.4.1.1.1 INTERFACE SERVICES IORB PROCESSING ROUTINE ERRORS

Errors which are reported in processing the request are:

1. Rct is not active.
2. Invalid iorb function code.
3. Invalid iorb parameter.

1.4.1.2 INTERFACE SERVICES IORB POST PROCESSING ROUTINE

The interface services iorb post processing routine will be invoked as a result of a call from a lacs driver module (layer server) wishing to dequeue and post a request back to a user. Three parameters must be passed when calling the routine, the pointer to the rct, the pointer to the irb, and the return status. The routine will dequeue the irb off the rct queue of irbs, then call the executive to post the request back to the user. The routine will then return to it's caller.

1.4.1.2.1 INTERFACE SERVICES IORB POST PROCESSING ROUTINE ERRORS

Errors which are reported in posting the request are:

1. Invalid parameters from a caller.
2. Executive errors in posting the request.

1.4.1.3 SUMMARY OF THE IORB PROCESSING ROUTINES

Regardless of the request, the interface services iorb preprocessor routine is invoked when each request is issued, the routine selects the appropriate layer server. The layer server process the request, and upon completion call the interface services post iorb routine. The interface services post iorb routine posts the request back to the user, then returns to the is.

1.4.2 BASIC FLOW OF THE INITIALIZATION SERVICES ROUTINES

1.4.2.1 ASSOCIATE USER MCL

The user of the lan subsystem is required to use a lrn in the iorb. This is obtained by an associate user mcl. The user must supply a symbolic name as an input parameter. The routine will search the user directory until a matching symbolic name is found. The routine then call the executive's "get physical device" routine to associate the lrn with the user's task group. The executive returns to the routine. The routine then returns the lrn to the user with a successful completion status.

1.4.2.1.1 ASSOCIATE USER MCL ERRORS

Errors which are reported by the mcl routine are:

1. A matching symbolic name is not found.

2. The lrn is already in use by another task group.

1.4.2.2 ACTIVATE LOCAL SAP ROUTINE

The user is required to perform the activate local sap request after performing the associate local user mcl. The activate local sap request is required by the lan subsystem becase the sap must be activated. The user must supply the lrn and proper function code in the lan type lorb. The routine will validate the lorb, call the sys mgr ls. Upon return from the sys mgr ls the routine will build an activate local sap lcb including the symbolic name of the sap (the symbolic name is retrieved from the local sap table). The lcb is then issued to the controller. The controller will place a 16 bit logical address in the lcb and return it to the l6. The activate routine will place the logical address into the local sap table, then will post the request successfully back to the user.

1.4.2.2.1 ACTIVATE LOCAL SAP ROUTINE ERRORS

Errors which are reported by the activate routine are:

1. Invalid lorb parameter.

2. Sap is already active.

3. Sys mgr is returns with an error.

4. Controller returns with an error.

1.4.2.3 ACTIVATE REMOTE SAP ROUTINE

The user must perform an activate remote sap request after the activate local sap request. The user must supply a lrn (representing the local sap) and a symbolic name (representing the remote point) in the lan type lorb. The routine will validate the lorb. The routine will then build a activate remote sap lcb including the symbolic name of the remote sap. The lcb is then issued to the controller. The controller will place the 16 bit logical remote address into the lcb and return to the l6. The activate remote sap routine will place the logical remote address into the lorb. The routine will then successfully return to the caller.

1.4.2.3.1 ACTIVATE REMOTE SAP ROUTINE ERRORS

Errors which are reported by the activate routine are:

1. Invalid lorb parameter.

2. A matching symbolic name was not found in the l6.

3. A matching symbolic name was not found in the lacs.

4. Controller return with an error.

1.4.3 BASIC FLOW OF THE TERMINATION SERVICES ROUTINES

1.4.3.1 DEACTIVATE LOCAL SAP ROUTINE

The executive will issue a deactivate local sap request when the task group using the sap aborts abnormally, or a user can issue a deactivate local sap request at anytime. The request is destructive to all current operations. The iorb will be validated once the request is received by the routine. The routine will then retrieve the rct from the lrn in the iorb, and queue the lrb on the tail of the rct lrb queue. The routine will mark the rct as deactivating, this is done to prevent subsequent requests from being processed. The routine will then build a lcb, issue the request to the lacs . The lacs will abort all outstanding orders it currently has active for the local sap, the lacs will also desolve any connection associated with the sap, and post the orders back to the l6, finally the lacs will post the deactivate local sap lcb back to the l6. The layer server will post each request received from the lacs back to the user with the appropriate status. The deactivate local sap routine will be invoked when the deactivate lcb is completed, when this happens the routine will mark the rct as not active and post the deactivate local sap request back to the issuer. In a connection oriented enviroment all connections will be disconnected by the routine.

1.4.3.1.1 DEACTIVATE LOCAL SAP ROUTINE ERRORS

Errors which are reported by the deactivate sap routine are:

1. Invalid iorb parameters.

2. Invalid logical local sap.

1.4.3.2 DEACTIVATE REMOTE SAP ROUTINE

The deactivate remote sap routine, is used by an application to free up resources taken up by the remote sap. The routine will be used in future releases, currently the routine will require a lan type iorb with a lrn and the logical remote sap the user wishes to deactivate. The routine will validate the iorb. The routine will build the deactivate remote sap lcb and issue it to the lacs. The lacs will deactivate the remote sap, then return the lcb to the l6. The routine will clean up data structures associated with the remote sap, then return successfully to the caller.

1.4.3.2.1 DEACTIVATE REMOTE SAP ROUTINE ERRORS

Errors which are reported by the remote sap deactivation routine are:

1. Invalid iorb parameter.

2. Invalid logical remote sap.

1.4.4 BASIC FLOW OF THE COMMON LAYER SERVER ROUTINES

1.4.4.1 TERMINATE TASK ROUTINE

This routine will be called by a ls or an initialization or termination routine when it would like to terminate it's task level. The routine will call the executive terminate task routine which will terminate the current task level.

1.4.4.1.1 TERMINATE TASK ROUTINE ERRORS

No errors.

1.4.4.2 BUILD LCB ROUTINE

The build lcb routine requires a lcb as a input parameter. The routine will fill in fields in the lcb from information in data structures, which were also passed as input parameters. The routine will return to it's caller after it fills in common lcb fields.

1.4.4.2.1 BUILD LCB ROUTINE ERRORS

Errors which are reported by the build lcb routine are:

1. Invalid input parameters.

1.4.4.3 VALIDATE IORB ROUTINE

The validate lan type iorb routine requires a iorb as a input parameter. The routine will validate that the iorb is a lan type request. The routine will return to it's caller after it has validated the iorb is of lan type.

1.4.4.3.1 VALIDATE IORB ROUTINE ERRORS

Errors which are reported by the validate iorb routine are:

1. Iorb is not a lan type iorb.

1.4.5 BASIC FLOW OF THE POWER FAIL ROUTINE

1.4.5.1 FIRST POWER FAIL RESTART ROUTINE

The first power fail restart routine (pfrs) is invoked after the level 6 experiences a power failure. The routine is invoked via a call from the exec pfrs routine. The routine is responsible for invoking the second pfrs routine, then returnig to the exec pfrs

routine.

1.4.5.2 SECOND POWER FAIL RESTART ROUTINE

The second pfrs routine is invoked by the first pfrs routine. The routine will clear all queues of active lcbs, post all active iorbs back to the users, clean up all data structures, and terminate any active lan subsystem levels. The routine will place the lan subsystem into the state it was in before the first associate. The routine will supend itself when processing is complete.

1.4.5.3 POWER FAIL ROUTINE ERRORS

The power fail routines do no report any errors.

1.5 MAJOR DEPENDENCIES

1.5.1 EXECUTIVE RESOURCES

The mod400 executive software supplies routines which are used by the ldis. These routines deal with irb and iorb processing and tasking.

1.5.2 SYSTEM MANAGER LAYER SERVER RESOURCES

The sys mgr is provides additional processing for the activate local and remote user requests, and the deactivate and deactivate with queue abort requests by supplying routines which are called by the appropriate ldis routine.

2 EXTERNAL SPECIFICATION

2.1 OWNED DATA STRUCTURES

The 16 lan subsystem data structures are defined in the data structures document.

2.2 EXTERNAL INTERFACES

2.2.1 MOD400 EXECUTIVE SOFTWARE ROUTINES

2.2.1.1 ZXREQ - Request task

```
entry:      lnj  $b5,zxreq

input:      $b4 = address of task request block
            $b5 = return address

output:     $r1 = 0 - task request was queued successfully
            $r1 > 0 - task request was not queued
            $b4 = address of task request block

modifies:   $r1,$r2,$r3,$b1,$b2,$b3
```

function: Request a normal task with á supplied request block pointer.

2.2.1.2 ZHCOMM - Null address

function: Will load the null address when referenced i.e. ldb $b5,<zhcomm will load $b5 with the null address.

2.2.1.3 ZXD_PR - Dequeue and post IRB

entry: inj $b5,zxd_pr

input: $r2 = completion status for request
$b5 = return address

output: $r1 = 0 - request was dequeued and posted
$r1 > 0 - no request on queue exist

modifies: any register may be modified

function: Post completion status to the first IRB on the request queue. Awaken any waiters queued on the IRB, record completion status in the RB, return the IRB to the system pool.

2.2.1.4 ZXD_TR - Internal terminate

entry: inj $b5,zxd_tr

input: $r2 = completion status for request
$b4 = new default start address or null
$b5 = return

output: $r1 = 0 - no error on internal terminate
$b1 = address of IRB for next request or is null if hardware interrupt
$b4 = address of RB for next request

modifies: any register may be modified

function: Dequeue and post currently dispatched request. Get next request in queue of task requests. Delete task if queue empty and delete bit on. Suspend task until next request or hardware interrupt at issuing task's level if queue empty and delete bit off.

2.2.1.5 ZXPOST - Post IRB

entry: inj $b5,zxpost

input: $r1 = contains status to be returned to waiter
$b1 = points to the IRB
$b5 = return address

output: $b4 = points to the RB

modifies $r1,$r2,$r3,$b1,$b4

function: Post completion status to the referenced IRB. Awaken any waiters queued on the IRB, record completion status in the RB if still attached, and return the IRB to the system pool.

2.2.1.6 ZXDQ - Dequeue IRB

entry: lnj $b5,zxdq

input: $b5 = return address

output: $r1 is status:
0 = dequeue accomplished
0814 = no IRB to dequeue or it is not dispatched
$b1 = points to the IRB

modifies: nothing

function: Dequeue the IRB at the head of the request queue of the issuing task.

2.2.2 MOD400 EXTERNAL DATA STRUCTURES IMPLEMENTED

The following system owned data structures are referenced by the ldis:

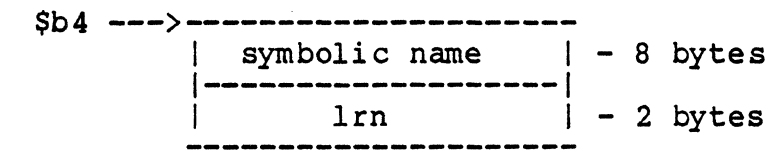
Task Control Block (TRB)
System Control Block (SCB)
Logical Resource Table (LRT)
Group Control Block (GCB)
Resource Control Table (RCT)
Intermediate Request Block (IRB)

2.2.3 USER INTERFACES

2.2.3.1 ASSOCIATE LOCAL USER MCL

entry: mcl with function code = x'2a01'

input: $b4 = a(parameter block)



| symbolic name | - 8 bytes
| lrn | - 2 bytes

Where the symbolic name is supplied by the user and the lrn is supplied by the system.

output: $b4 = a(parameter block) - described above
$r1 = status

0000 - association successful
0781 - matching symbolic name not found
0782 - lrn reserved by another task group

modifies: all registers are perserved

function: return a lrn to the user from a supplied lrn, lrn is to be used in lan subsystem requests

2.2.3.2 STANDARD IORB FORMAT

rb_lrx

input - bit 0 - rb_adr points to a bd when set
bit 1-3 - na
bit 4-f - lrn when rb_ct2 bits 0-7 = x'fd'
output - na

rb_rrb

input: na
output: na

rb_ct1

input: bit 0-7 - mbz
bit 8-e - na
bit f - must be set
output: bit 0-7 - status
60 - event successful
64 - invalid iorb parmaeter
6b - event aborted, reference rb_ fss field for reason
6c - inconsistent request, reference rb_ fss field for reason
bit 8-f - same as input

rb_ct2

input: bit 0-7 - lrn
bit 8-a - na
bit b - must be set
bit c-f - function code
bit c-f = b - deactivate iorb
bit c-f = a - activate iorb
output: same as input

rb_adr

input: pointer to buffer, or if rb_lrx bit 0 is set this field contains a pointer to a buffer discriptor, or this field may be null if no data is being passed (i.e. event iorb)
output: same as input

rb_rng

input: range of the buffer, or total range of all buffers in the buffer discriptor block if bit 0 in rb_irx is set, or mbz if no data is being passed
output: same as input

rb_dvs

input: bit 0-7 - class of service
bit 8-f - mbz
(note: bit e being set and the iorb function code = b means this is an exec deactivate)
output: same as input

rb_rsr

input: mbz
output: residual range of the buffer for read operations only, or same as input for all other operations

rb_st1

input: mbz
output: bit 0-7 - same as input
bit 8 - invalid function code when set
bit 9 - ram memory exausted when set
bit a - ram location non-existent when set
bit b - ram parity error when set
bit c - level 6 memory yellow when set
bit d - level 6 memory non-existent when set
bit e - level 6 bus parity error when set
bit f - level 6 memeory red when set

rb_ext

input: bit 0-7 - xx
bit 8-f - xx
output: same as input

rb_fsf

input: function specific function code
iorb function code = b
0010 - deactivate local
0020 - deactivate remote
iorb function code = a
0010 - activate local
0020 - activate remote
output: same as input

rb_fss

input: mbz
output: function specific status
0001 - sap already active
0002 - lack of resources
0004 - controller down
0008 - sm is error
0010 - lrn already in use
0020 - sap already active
0040 - sap already disconnected
0080 - receive buffer too small

rb_abs

input: mbz
output: accual buffer size when rb_fss = 0080, otherwise same as input

rb_lra

input: logical remote address for deactivate remote, otherwise mbz
output: logical remote address for activate remote, otherwise same as input

2.2.3.2.1 ACTIVATE IORB EXTENSION

rb_sym

input: 8 byte symbolic name representing sap
output: same as input

rb_pms

input: proposed max sdu size
output: same as input

rb_pmr

input: proposed max read credit
output: same as input

rb_typ

input: mbz
output: type of sap

rb_mss

input: proposed maximum sdu size for activate local and ci user
mbz for activate remote sap or co user
output: maximun sdu size for activate local and ci user
same as input for activate remote sap or co user

rb_iss

input: mbz
output: ideal sdu size for activate local and ci user
same as input for activate remote or co user

rb_mpr

input: proposed ci data read credit for activate local and ci user
mbz for activate remote or co user
output: maximum number of pending ci read calls for activate local and ci user
same as input for activate remote or co user

rb_wcc

input: mbz
output: ci write credit for activate local and ci user
same as input for activate remote or co user

rb_mcc

input: mbz
output: maximum number of connections supported for a activate local and co user
same as input for activate remote or ci user

rb_acb

input: mbz
output: trash

2.2.3.2.2 DEACTIVATE IORB EXTENSION

rb_dcb

input: mbz
output: trash

2.2.3.4 STANDARD LCB FORMATS

cb_pri

input: mbz
output: na

cb_ncb

input: mbz
output: na

cb_rct

```
     input:    address of caller's rct
     output:   same as input
```

cb_lit

```
     input:    address of the lit in which this lcb will be
               queued
     output:   same as input
```

cb_frw

```
     input:    bit 0-3 - 9
               bit 4-7 - mbz (iorb major function code ?)
               bit 8-f - xx
     output:   same as input
```

cb_itp

```
     input:    address of the post processing routine, or trb,
               on null
     output:   same as input
```

cb_ind

```
     input:    indicators
                 bit 7 - cb_itp points to a trb when set
                 bit 6 - sm lcb when set
                 all other bits mbz
     output:   same as input
```

cb_icw

```
     input:    bit 0-5 - mbz
               bit 6-9 - cpu number to interrupt
               bit a-f - level to interrupt the cpu
     output:   same as input
```

cb_fsf

```
     input:    function specific function code
                 function codes for activate lcbs are:
                         001a - activate local sap
                         002a - activate remote sap
                 function codes for deactivate lcbs are:
                         001b - deactivate local sap
                         002b - deactivate remote sap
     output:   same as input
```

cb_cts

```
input:    mbz
output:   bit 0-7 - rfu and mbz
          bit 8 -   invalid function code when set
          bit 9 -   ram memory exausted when set
          bit a -   ram location non-existent when set
          bit b -   ram parity error when set
          bit c -   level 6 memory yellow when set
          bit d -   level 6 memory non-existent when set
          bit e -   level 6 bus parity error when set
          bit f -   level 6 memeory red when set
```

cb_fss

```
input:    mbz
output:   function specific status
            0001 -  sap not active
            0002 -  lack of resources
            0004 -  controller unavailable
            0008 -  sm layer instance error
            0040 -  sap already disconnected
            0080 -  recieve buffer too small
            0100 -  illegal logical address
            0200 -  invalid icb
            0400 -  write credit violations
            0800 -  read credit violations
```

cb_cbs

```
input:    mbz
output:   bit 0 -   icb is complete when set
          bit 1 -   icb not processed when set
          bit 2-f - rfu and mbz
```

cb_abs

```
input:    mbz
output:   actual buffer size if cb_fss = 0080, otherwise
          same as input
```

cb_lsa

```
input:    logical local address for cl operations
output:   same as input
```

cb_lra

```
input:    logical remote address for cl write operation, mbz
          for event and read operations
output:   logical remote address for cl read operations,
          otherwise same as input
```

```
cb_trg

     input:    total byte range
     output:   same as input

cb_bct

     input:    number of buffers
     output:   same as input

cb_ad1

     input:    buffer #1 address
     output:   same as input

cb_rg1

     input:    buffer #1 range
     output:   same as input

cb_rs1

     input:    mbz
     output:   buffer #1 residual range

cb_ad2

     input:    buffer #2 address
     output:   same as input

cb_rg2

     input:    buffer #2 range
     output:   same as input

cb_rs2

     input:    mbz
     output:   buffer #2 residual range

cb_ad3

     input:    buffer #3 address
     output:   same as input

cb_rg3

     input:    buffer #3 range
     output:   same as input

cb_rs3

     input:    mbz
     output:   buffer #3 residual range
```

cb_ad4

input: buffer #4 address
output: same as input

cb_rg4

input: buffer #4 range
output: same as input

cb_rs4

input: mbz
output: buffer #4 residual range

cb_ad5

input: buffer #5 address
output: same as input

cb_rg5

input: buffer #5 range
output: same as input

cb_rs5

input: mbz
output: buffer #5 residual range

cb_ad6

input: buffer #6 address
output: same as input

cb_rg6

input: buffer #6 range
output: same as input

cb_rs6

input: mbz
output: buffer #6 residual range

cb_ad7

input: buffer #7 address
output: same as input

cb_rg7

input: buffer #7 range
output: same as input

cb_rs7

input: mbz
output: buffer #7 residual range

cb_ad8

input: buffer #8 address
output: same as input

cb_rg8

input: buffer #8 range
output: same as input

cb_rs8

input: mbz
output: buffer #8 residual range

2.2.3.4.1 ACTIVATE LCB EXTENSION

cb_sym

input: symbolic name of the sap
output: same as input

cb_pms

input: proposed max sdu size
output: same as input

cb_prc

input: proposed max read credit
output: same as input

cb_mss

input: proposed maximum sdu size for activate local and cl user
mbz for activate remote sap or co user
output: maximun sdu size for activate local and cl user
na for activate remote sap or co user

cb_iss

input: mbz
output: ideal sdu size for activate local and cl user
na for activate remote or co user

cb_mpr

input: proposed ci data read credit for activate local and ci user
mbz for activate remote or co user
output: maximum number of pending ci read calls for activate local and ci user
na for activate remote or co user

cb_wcc

input: mbz
output: ci write credit for activate local and ci user
na for activate remote or co user

cb_mcc

input: mbz
output: maximum number of connections supported for a activate local and co user
na for activate remote or ci user

2.2.3.4.2 DEACTIVATE LCB EXTENSION

The deactivate lcb uses only the standard portion of the lcb.

2.2.4 COMMON ROUTINES

2.2.4.1 TERMINATE TASK ROUTINE (ISTMTK)

call: lnj $b5,istmtk

input: $b2 = a(rct)

output: none - task is terminated

modifies: all registers are perserved

2.2.4.2 BUILD LCB ROUTINE (ISBLCB)

call: lnj $b5,isblcb

input: $b1 = a(lcb)
$b2 = a(rct)
$b3 = a(tt)
$b4 = a(iorb)
$b5 = a(return)

output: $b1 = a(lcb)
$b2 = a(rct)
$b3 = a(tt)
$b4 = a(iorb)
$b5 = a(return)
$r1 = status
0000 - lcb was built
0001 - error occured in the build

modifies: $r1

2.2.4.3 VALIDATE IORB ROUTINE (ISVIOB)

call: inj $b5,isviob

input: $b4 = a(iorb)
$b5 = a(return)

output: $b4 = a(iorb)
$r1 = status
0000 - iorb validated
0164 - invalid iorb

modifies: $r1

2.2.4.4 EVENT ROUTINES (ISEVNT)

call: inj $b5,isevnt

input:

output:

modifies:

2.2.4.5 ASSIGN SEGEMENT VISIBLITY (ISASVB)

call: inj $b5,isasvb

input:

output:

modifies:

2.2.4.6 ABSOLUTIZE BUFFER (ISABSL)

call: inj $b5,isasvb

input:

output:

modifies:

2.3 INITIALIZATION REQUIREMENTS

The clm process will load the ldis into system memory, and configure at least on e task level for the ldis to run under. When the ldisis loaded into memory, it will perform a initialization subroutine. This routine will:

1. Place the scb, gcb, and lrt pointers into vectors for use in subsequent processing.

2. Set up the lan monitor call major function handling code by placing a pointer into the executive monitor call vector table at slot x'2a'.

3. Set up the associate user monitor vector, this is done by placing the address of the mcl routine into the lan monitor call vector table at slot x'00'.

4. Reclaim patch space.

No initialization processing is done when the ldis is activated initially by a request.

The ldis must be loaded into memory before any other components of the lacs driver are loaded into main memory.

2.4 TERMINATION REQUIREMENTS

The ldis will be active as long as mod400 is active, therefore there are no termination requirements.

2.5 ENVIRONMENT

The following items are required by the ldis for it to perform it's task:

1. Mod400 operating system.

2. Any l6 computer model except 6/10 and 6/20.

3. A lacs attached to the l6 megabus.

4. System manager layer server.

5. A user of the lan subsystem to drive the ldis.

2.6 TIMING AND SIZE REQUIREMENTS

Currently memory usage and timing requirements are not an issue. However, the code should be as efficient as possible.

2.7 ASSEMBLY AND LINKING

The software will be written in Series 6 Assembly Language using a subset of the instruction set that is present on all Series 6 systems. The ldis will be linked with the lacs driver megabus services module by the gcos6 mod400 linker to produce one of the lacs driver's bound units.

2.8 TESTING CONSIDERATIONS

Since the product is new, all functions will be tested by the developer, and software test.

2.9 DOCUMENTATION CONSIDERATIONS

The ldis source listing will include a program design language used by the developer to aid in the maintenance by future developers and also to aid in the development by the developer.

2.10 ERROR MESSAGES

2.10.1 APPLICATION ERROR MESSAGES

2.10.1.1 ASSOCIATE USER MCL ERROR MESSAGES

1. 0781 - A matching symbolic name was not found.
2. 0882 - Another task group has already reserved the lrn.

2.10.1.2 IORB ERROR MESSAGES

1. 0164 - Invalid iorb parameters.
2. 016b - Request was not processed.
3. 016c - Inconsistent request.

2.10.2 INTERNAL ERROR MESSAGES

10.2.1 IST ERROR MESSAGES

1. tbd

3 INTERNAL SPECIFICATION

3.1 OVERVIEW

The ldis can be invoked three different ways, they are:

1. A user performing a associate user mcl, the ldis runs at the user task level.
2. A user issues a request to the lan subsystem, the ldis runs at the task level configured in clm.
3. A layer server executing one of the common routines, the ldis can run at the task level configured in clm or an interrupt level configured in clm.

3.2 SUBCOMPONENT DESCRIPTION

3.2.1 IORB PROCESSING ROUTINES

The iorb preprocessing routine will be the start address of all request processing done by the lan subsystem. Every request issued to the lan subsystem will first be processed by this routine. Using information in the iorb the routine will either branch to a layer server or branch to another ldis routine.

The user will build an iorb and issue it to the lan subsystem. The system preprocessor (or lan preprocessor) will be invoked as a result of the request io. The system preprocessor will build an irb, and copy data buffers and the iorb into system memory if required to do so. The system preprocessor will fetch the irn from the iorb and index into the irt to find the pointer to the rct. From the rct the tcb pointer is obtained, and in the tcb is the start address and level of the task the driver will operate under. The system preprocessor queues the irb (which points to the iorb on the tail of the active queue of irbs) off the tcb. The system preprocessor now schedules the task represented by the tcb is which the request was queued. The task is invoked at the iorb preprocessing routine.

3.2.1.1 IORB PREPROCESSING ROUTINE

The routine requires the following requires the following input parameters:

$b1 = a(irb)

The routine supplies the following output parameters:

$b2 = a(rct)
$b1 = a(irb)

The iorb preprocessing routine performs the following:

1. Obtains the irn from the iorb.

2. Indexes into the irt by the irn to obtain the pointer to the rct.

3. Fetches the major function code from the iorb.

4. If the rct is not active and the request is not an activate local user, call the dequeue and post executive routine to post the iorb with an inconsistent request error or if the power fail bit is set post the request with a power fail occurred error.

5. If the rct is active and the power fail bit is not on, then the routine dequeues the head irb off the tcb queue of active irb's and enqueues the irb on the tail of the active irb queue of on the rct.

6. If the function code is a read, write, system management, flow control, or event indicate request, the iorb preprocessor will fetch the start address from the rct and branch to the address.

7. If the function code is a activate, or deactivate, the routine will branch to the appropriate routine.

8. If the function code is anything else the routine will call the post iorb routine with an invalid iorb error status, then call the terminate task routine.

3.2.1.1 IORB POST PROCESSING ROUTINE

The iorb post processing routine can be called by any module in the lacs driver. The routine requires the following input parameters:

```
$b1 = a(irb)
$b2 = a(rct)
$r1 = completion status
      0160 - request successful
      0164 - invalid iorb parameter
      016b - request aborted
      016c - inconsistent request
$b5 = a(return)
```

The routine supplies the following output parameters:

```
$b2 = a(rct)
```

The routine performs the following function:

1. Dequeue the irb off the rct queue.
2. Call the executive post irb routine (zxpost).
3. Return to the caller.

3.2.2 INITIALIZATION SERVICES ROUTINES

The activate local and activate remote initialization service routines are invoked by the iorb preprocessor. The iorb preprocessor will branch to the appropriate initialization routine depending on the iorb major function code. The associate user mcl is invoked by the lan mcl handling routine in the ldms.

3.2.2.1 ASSOCIATE USER MCL

The mcl will transform a symbolic name into a lrn. The user will issue the mcl which has a function code = x'2a00'. The routine requires the following input parameters.

```
$b5 = a(return)
$b6 = a(scb)
$r1 = minor function code (=00)
$b4 = a(parameter list)
```

where the parameter list is defined as follows:

```
symbolic name = 8 bytes
lrn           = 2 bytes
```

The routine supplies the following output parameters:

```
$b4 = a(parameter list)
$b6 = a(scb)
$r1 = status
      0000 -      symbolic name associated
      0781 -      no matching symbolic name found
      0782 -      symbolic name already associated with
                  another task group
      note:       if $r1 is non zero the lrn is not placed
                  into the parameter list, if $r1 is zero
                  the lrn is placed into the parameter
                  list.
```

The user supplies the symbolic name, and the routine supplies the lrn. The routine performs the following function:

1. Retrieve the pointer to the controller directory from the scb.

2. Retrieve the pointer to the user directory from the cd.

3. Search the ud until a matching symbolic name is found.

4. If a match is not found, return to the caller with an error status.

5. If a match is found, call the executive get physical device routine to associate the lrn with the user.

6. If the executive get physical device routine return with an error, return to the user with an error status and no lrn (error would be that the lrn is already reserved by another task group).

7. If no error resulted in the executive call, place the lrn into the parameter structure and return to the caller.

3.2.2.2 ACTIVATE LOCAL SAP REQUEST

The activate local sap request is required by an application so the lan subsystem can activate the local sap in the lacs and in the 16. The user must supply a lan type iorb with the following fields:

```
Iorb function code = c.
Lrn from associate mcl.
Device specific word bit 0 set specifying this is an
activate local sap request.
Event mask.
```

The routine is invoked by the iorb preprocessing routine, the routine requires the following input parameters:

$b1 = a(irb)
$b2 = a(rct)

The routine will supply the following output parameters:

none - task level is exited after posting the request.

The routine will perform the following:

1. Validate the request, if the request is invalid call the post iorb routine with an invalid iorb parameter error, call the terminate task routine.

2. Call sm ls.

3. Retrieve the pointer to the lst from the rct.

4. Build the lcb, place in the symbolic name from the lst, issue the lcb via a call to the ldms.

5. Set the sap (rct) into activating mode, call the terminate task routine.

6. When the lacs completes the request, the routine wakes up on interrupt level, via a call from the ldms. If an error resulted in the request, dequeue the iorb associated with the lcb and call the post iorb routine with an error status, return to the ldms.

7. Place the logical name into the lst. Dequeue the iorb associated with the lcb and call the post iorb routine with a successful status, issue an event lcb to the controller the local sap has access to, return to the ldms.

3.2.2.3 ACTIVATE REMOTE SAP REQUEST

The activate remote sap request is used by an application to obtain a logical remote address from a supplied symbolic name. The user must supply a lan type iorb with the following fields set:

iorb function code = c.
Device specific word bit 1 set specifying this is an activate remote sap request.
Symbolic name representing the remote sap in the rb_oas field.

The routine is invoked by the iorb preprocessing routine, the routine requires the following input parameters:

$b1 = a(irb)
$b2 = a(rct)

The routine supplies the following output parameters:

task level is exited after posting the request back to the user with the logical remote sap address.

The routine will perform the following:

1. Validate the request, if the iorb is invalid call the post iorb routine to post the request with an invalid iorb parameter error, call the terminate routine.

2. Obtain the pointer to the cd from the scb.

3. Retrieve the pointer to the appropriate remote sap directory from the cd, this is determined by the local sap (layer).

4. Search the remote sap directory until a matching symbolic name is found, if a matching sysmboic name is not found call the post iorb routine with a invalid iorb error status, call the terminate task routine.

5. If a match is found, determine from the controller mask if the local sap can access the remote sap, if the remote sap cannot access the local, call the post iorb routine with an invalid iorb error status, call the terminate task routine.

6. Determine form the adapter mask if the local sap can access the remote sap, if the remote sap cannot access the local, call the post iorb routine with an invalid iorb error status, call the terminate task routine.

7. Build the activate remote lcb, place in the symbolic name from the iorb, call the ldms to issue the lcb.

8. Call the terminate task routine.

9. When the lacs completes the request, the routine wakes up on interrupt level, via a call from the ldms. If an error resulted in the request, dequeue the iorb associated with the lcb and call the post iorb routine with an error status, return to the ldms.

10. Place the logical number into the iorb rb_dad field, place the logical number into the rst, mask in the controller bits.

11. Dequeue the iorb, and call the post iorb routine with a successful status.

12. Return to the ldms.

3.2.3 TERMINATION SERVICE ROUTINES

The termination service routines are invoked by the iorb preprocessor. The iorb preprocessor will branch to the deactivate local sap routine when the iorb function code is = b. The deactivate local sap routine will check the device specific word in the iorb, and if bit f is set, the routine will invoke the deactivate remote sap routine. Otherwise, if bit e is set the routine will process the deactivate local sap request. If bit e or f is not set the routine will call the post iorb routine to post the iorb with an invalid iorb parameter error, then call the terminate task routine.

3.2.3.1 DEACTIVATE LOCAL SAP ROUTINE

The deactivate local sap routine will clear all active requests outstanding on the sap specified, and place the sap into a non active state. Note: a lan type iorb is not required for this request. The routine requires the following input parameters:

$b1 = a(irb)
$b2 = a(rct)
iorb with major function code = b
device specific word bit e set

The routine supplies the following output parameters:

none - level is exited via a call to the terminate task routine

The routine performs the following function:

1. If bit f of the iorb device specific word is set branch to the terminate remote sap routine.

2. If bit e of the iorb device specific word is not set, call the post iorb routine with an invalid iorb parameter error, call the terminate task routine.

3. Mark the rct into deactivate mode.

4. Call the deactivate flow control routine (to purdge all requests queued because of flow control and reset flow control parameters).

5. Retrieve a lcb from memory, fill in the deactivate local sap lcb fields, issue the lcb via a all to the ldms.

6. Call the terminate task routine.

7. The lacs will recieve the deactivate lcb and post all orders associated with the sap back to the l6. When the requests are posted back, the layer server (invoked because of a call from the ldms running at interrupt level) will determine that the sap is in deactivate mode and call the deactivate mode routine to handle the processing.

8. If the completed lcb is not the deactivate request, the routine will call the post iorb routine to post the request with the specified status. If the rct is marked as the deactivate was received before other requests have completed mode and if there is only one request left on the rct queue (it would be the deactivate) and the request is finished, the routine will call the post request routine to post the iorb with a successful status then mark the rct as not active and clear the deactivate mode bit and the received bit, the routine will return to the ldms.

9. If the completed lcb is the deactivate request, the routine will test if only one request is on the rct queue (this would be the deactivate request) and if so will call the post iorb routine to post the request with a successful status, then mark the rct as not active and take the rct out of deactivate mode, then return to the ldms. If there are more requests on the queue, the routine will update the deactivate iorb, mark the rct into deactivate received before other request have completed mode and return to the ldms.

3.2.3.2 DEACTIVATE REMOTE SAP ROUTINE

The deactivate remote sap routine is invoked via a branch from the deactivate local sap routine. The routine requires the following input parameters:

```
$b1 = a(irb)
$b2 = a(rct)
iorb major function code = b
device specific bit e set in the iorb
```

Ther routine supplies the following output parameters:

none

The routine performs the following function:

tbd

3.2.4 COMMON LAYER SERVER ROUTINES

The common layer server routines are invoked via a lnj from a layer server. Except for the terminate task routine all the routines will return to the calling routine.

2.4.1 TERMINATE TASK ROUTINE

The terminate task routine is called by a layer server when it wishes to terminate it's task level, the routine resets the start address to the iorb preprocessing routine, note: this routine is not used by the megabus interface services when it is operating at interrupt level. The routine requires the following input parameters:

none

The routine supplies the following output parameters:

none

The routine performs the following functions:

1. Retrieve the null address from .zhcomm and place it into $b4.

2. Call the terminate task routine, this routine will terminate the level, and the start address will be set to the next instruction. Therefore, the next instruction is a branch to the iorb preprocessor routine.

2.4.2 BUILD LCB ROUTINE

The build lcb routine will fill in certain common fields in the lcb from the rct and iorb. The routine requires the following input parameters:

$b1 = a(lcb)
$b4 = a(iorb)
$b2 = a(rct)
$b3 = a(tt)
$b5 = return address

The routine supplies the following output:

$b1 = a(lcb)
$b4 = a(iorb)
$b2 = a(rct)
$b3 = a(tt)

The routine performs the following function:

1. The routine fills in the following fields of the lcb:

   cb_pri - clears the field.

   cb_ncb - set the field to null.

cb_rct - sets the pointer to the rct into the field, the pointer to the rct is retrieved from the input parameteri.

cb_lit - sets the pointer to the lit into the field, the pointer to the lit is retrieved from the tt.

cb_icw - sets the field from bits in the lit ll id2 word. word

cb_cts - clears the field.

cb_fss - clears the field.

cb_cbs - clears the left byte, sets the number of buffers into the right byte, number of buffers is known from the iorb.

cb_lrs - copies the rb_dad field from the iorb.

cb_lls - copies the lst_ls field in the local sap table, pointer to the local sap table is retrieved from the rct.

cb_tng - copies the rb_rng field from the iorb.

cb_adr - set the field from the iorb or buffer discriptor after calling the exec absoultizing routine.

cb_rng - set the field from the iorb or buffer discriptor.

3.2.4.3 VALIDATE IORB ROUTINE

The validate iorb routine will validate certain fields of the iorb. The routine requires the following input parameters:

$b4 = a(iorb)
$b2 = a(rct)
$b5 = return address

The routine supplies the following output parameters:

$b4 = a(iorb)
$b2 = a(rct)
$r1 = status
0 - iorb is validate
4 - iorb contains invalid parameters

The routine performs the following function:

The fields this routine validates will be determined when the layer servers are speced.

3.2.5 POWER FAIL RESTART ROUTINES

3.2.5.1 FIRST POWER FAIL ROUTINE

The 1th pf routine will be invoked by the executive pf restart routine. The exec pf restart routine will retrieve the s_inpf field in the scb and then inj to the address specified. The idms ist code will set up the pointer in the s_inpf field to be the 1th pf routine entry point. The routine also requires clm to place the tcb pointer of the lowest lan interrupt level into the cd_tcb field of the cd. The 1th pf routine requires the following input parameters:

$b5 = return address
no stack

The routine supplies the following output parameters:

none

The 16 experiences a power failure, the executive pf restart routine is invoked at system level 2. The executive performs a inj to the address specified in the s_infp field in the scb. The 1th power fail routine performs the following:

1. Retrieve the pointer to the cd from the scb.
2. Retrieve the tcb pointer for the lowest lan interrupt level from the cd.
3. Places the address of the 2nd lan pfrs routine into the p-counter in the tcb of the lowest lan interrupt level.
4. Performs a lev (to emulate an interrupt) to invoke the lowest lan interrupt level.
5. Returns to the executive pfrs routine.

3.2.5.2 SECOND POWER FAIL ROUTINE

The 2nd pfrs routine is invokes as a result of the lev performed by the 1th pfrs routine. The routine requires the following input parameters:

none

The 2nd pfrs routine supplies the following output parameters:

none

The 2nd pfrs routine performs the following function:

1. Retrieves the pointer to the cd from the scb.

2. Retrieves the pointer to the ud from the cd.

3. For each entry, the rct associated with it is retrieved, and all requests active on that rct (irb queue) are posted back to the user with a power fail error status, unless the icb's completion bit is set, then the request is posted back successfully.

4. All structures are cleaned up, they are placed back into the state they were at before any local sap was activated.

5. All lan tcb are checked, and if they are active, (i.e. the level had been interrupted) their p-counters are changed to the exit level routine.

Notes:

1. The users must reactivate, at this time the controllers will be loaded (in the exact manner as done in any activate).

2. Any subsequent requests issued to the lan subsystem (other than activate local sap requests) will be posted back to the user with an software not, loaded error.

4 PDL

TBD

5 ISSUES

1. Quility of service
2. States
3. Flow control
4. Events in a co enviroment
5. Lost icbs, icb nak'd when controller is down
6. Mutiple cpus
7. Group tsaps
8. Errors need to be defined
9. System preprocessor (exec's won't be ready until 4.1)

10. Different icb, iorb for different functions

11. Define the function specific word, 4 bits iorb function, 12 fs

12. Ldis and idms need to be seperate bound units because of the ist code the idms has to run.

13. Define the iass and iassi ec's.

14. Define events.

QUESTIONS

1. Does zqeasd handle a buffer list?

2. Does exec need return status in the iorb or does he place it in himself from $r1?