Introduction

Synchronization provides a method for guaranteeing consistent data and coordination of threads of code. We will concentrate on process and thread oriented synchronization considerations. Students will also learn basic methods for inter-process communication.

Objectives

Students should be able to:

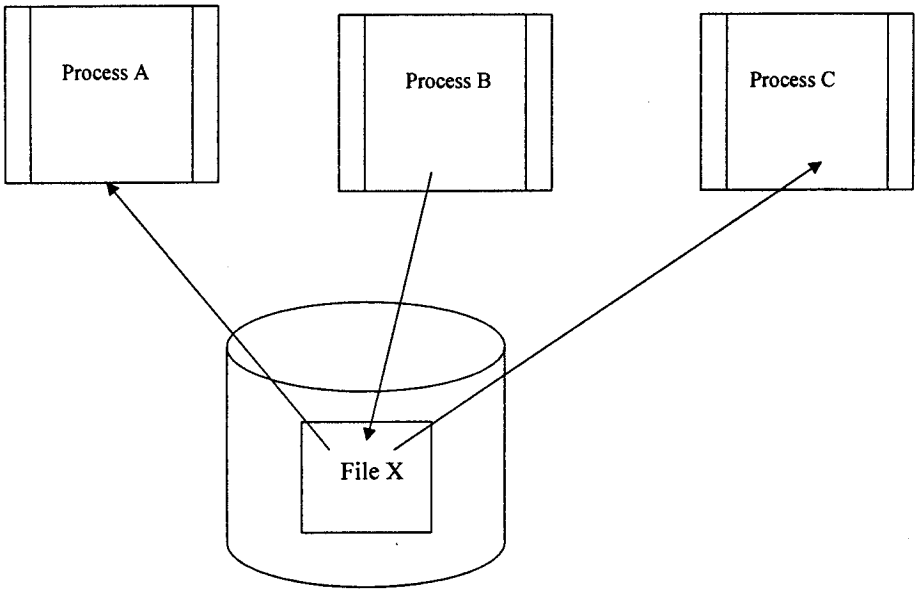
- Describe how synchronization is implemented using locks.
- Translate logical names from a program.
- Describe communication methods using mailboxes.
- Perform basic mailbox communication.

Topics

- Lock Management
- Common Event Flags
- Logical Names
- Mailboxes

Locking Concepts

The need for locking



Support for locking

- Locks are requested on named resources.
- **SYS$ENQ[W]** system service is used to request locks.
- **SYS$DEQ** system service is used to release locks.
- Locking services allow for multiple readers and exclusive writers, with several request modes.
- Locks may be converted to preserve value block information.
- Locks have built-in deadlock detection.
- Locks support a sub-lock hierarchy.
- Locks function cluster-wide.

File-related locking

- **RMS** and the file system (**XQP**) lock implicitly.
- **RMS** supports waiting for locked resource to become available by the bit **RAB$V_WAT** in **RAB$L_ROP** field of Record Access Block (**RAB**). Default is not to wait, but to return error status.

The OpenVMS Lock Manager

The OpenVMS Lock Manager can be used by processes.

- System-wide availability.
- Regulates access to a named object.
- Requires implementation agreement.

Lock Management System Services

- $ENQ[W]
  - Grants access to lock if mode is allowed.
  - Queues incompatible lock requests.
  - Lock mode defines owner access.
- $DEQ
  - Dequeues granted locks.
  - Cancels non-granted requests.
- $GETLKI
  - Returns information about lock database.

Enqueueing and Dequeueing Locks

Lock Modes

Mode of Currently Granted Lock						
Mode of Currently Granted Lock	**NL**	**CR**	**CW**	**PR**	**PW**	**EX**
NL	Yes	Yes	Yes	Yes	Yes	Yes
CR	Yes	Yes	Yes	Yes	Yes	No
CW	Yes	Yes	Yes	No	No	No
PR	Yes	Yes	No	Yes	No	No
PW	Yes	Yes	No	No	No	No
EX	Yes	No	No	No	No	No

SYS$ENQ Call Format

```
SYS$ENQ [efn] ,lkmode ,lksb ,[flags] ,[resnam] ,[parid]
,[astadr] ,[astprm] ,[blkast] ,[acmode] ,[rsdm_id]
,[nullarg]
```

Argument	Meaning
efn	Event Flag to set on completion
lkmode	lock mode LCKK_NLMODE, LCKK_CRMODE, LCKK_CWMODE, LCKK_PRMODE, LCKK_PWMODE, LCKK_EXMODE
lksb	Lock Status Block
flags	LCK$M_NOQUEUE, LCK$M_SYNCSTS, LCKM_SYSTEM, LCKM_VALBLK, LCK$M_CONVERT, LCK$M_NODLCKWT, LCK$M_NODLCKBLK, LCK$M_EXPEDITE, LCKM_QUECVT, LCKM_NOQUOTA
resnam	Resource name
parid	Parent Lock ID
astadr	Address of AST routine
astprm	Parameter passed to AST routine
blkast	Address of blocking AST routine
acmode	Access mode
rsdm_id	Lock resources names are implicitly accessible by LCK$M_SYSLCK. Resource domains allow a finer granularity for visibility of resource names. See: sys$set_resource_domain

Lock Features

The Lock Value Block

- The lock value block is used to identify changes to a resource.
- It is written by first holding a protected write or exclusive lock on a resource. Then dequeueing the lock and passing the value block argument or converting the lock and passing the value block from the lock status block (lksb).
- To obtain the value block, request a lock and use the flag **LCK$M_VALBLK**. The value block will be returned as a part of the lock status block.

Lock Conversions

- Lock conversions preserve the integrity of a resource and its value block.
- NULL locks give the effect of no resource ownership, but support keeping the resource information around.

Deadlock detection

- Deadlock conditions are automatically detected, after **DEADLOCK_WAIT** (**SYSGEN** parameter) seconds. A victim is chosen and is responsible for dequeueing all locks.

Lock Example (continued)

```
int main(void)
{
    Record  rec;
    lksb    lock_stat;
    $DESCRIPTOR(lock2,"BAE$REC2");
    int     i;
    FILE    *fp;
    int     status;
    int     num_obj;
    float   stall = 20.0;
    int     wait_ast(void);
/* Open the file for shared access, with no locking. Assumes file exists*/
    fp = fopen("grunge.dat","r+b","acc",uopen,&status);
    if(!fp)
    {
      perror("Cannot open grunge.dat\n");
      sys$exit(SS$_ABORT);
    }
/* Set a timer. */
    status = sys$setimr(0,five_secs,wait_ast,0,0);
    sys_stat(status);
/*  Request a lock on the record. */
    status = sys$enqw(0,LCK$K_PRMODE,&lock_stat, 0,&lock2,0,0,0,0,0,0,0);
    sys_stat(status);
    sys_stat(lock_stat.lksb$w_status);
/* Cancel the timer. */
    sys$cantim(0,0);
```

Lock Example (continued)

```
/* Go to target record. */
    status = fseek(fp,sizeof(rec)*REC_TO_READ,SEEK_SET);
    if(status == -1)
    {
      perror("Bad seek");
      sys$exit(SS$_ABORT);
    }
/* Read the record. */
    num_obj = fread(&rec,sizeof(Record),NUM_RECS,fp);
    if(num_obj == NUM_RECS)
    {
      printf("Record read\n");
      printf("Name: %s\nAge %d\n",rec.name,rec.age);
    }
    else
    {
      printf("Error reading records\n");
    }
    lib$wait(&stall);

    return(SS$_NORMAL);
}
```

Lock Example (continued)

```
/* Timer ast routine. */
int  wait_ast(void)
{
     int      status;
/* Note our impatience. */
     puts("Waiting...");
/* Reset the timer. */
     status = sys$setimr(0,five_secs,wait_ast,0,0);
     sys_stat(status);
     return(SS$_NORMAL);
}

/* User open to allow shared read/write from C. */
int  uopen(int *garb,struct FAB *fab,struct RAB *rab)
{
     int      status;
     fab->fab$b_shr = FAB$M_UPI|FAB$M_SHRPUT;
     return(SS$_NORMAL);
}

$
```

Lock Example (continued)

```
$ type file_writer.c
/* Sample locking program.  Takes out protected write locks. */
#include <stdio.h>
#include <ssdef.h>
#include <starlet.h>
#include <descrip.h>
#include <lckdef.h>
#include <lksbdef.h>
#include <lib$routines.h>
#define MAX_NAME 80
#include <rms.h>
static struct RAB *rabptr;
typedef struct      _record
{
    char     name[MAX_NAME];
    int      age;
}Record;
#define sys_stat(STS) if(!((STS)&1)) sys$exit(STS)
int     uopen(int *garb,struct FAB *fab,struct RAB *rab);
#define NUM_RECS  sizeof(recs)/sizeof(*recs)

int main(void)
{
    Record  recs[] = {{"Dylan Allen",29},
                      {"Eddie Vedder", 32},
                      {"Neil Young",53}};
    lksb    lock_stat[NUM_RECS];
```

Lock Example (continued)

```
    $DESCRIPTOR(lock1,"BAE$REC1");
    $DESCRIPTOR(lock2,"BAE$REC2");
    $DESCRIPTOR(lock3,"BAE$REC3");
    struct dsc$descriptor_s *lock_ptrs[NUM_RECS];
    int     i;
    FILE    *fp;
    int     status;
    float   stall = 30.0;
    int     num_obj;
/* Open file.  Note: assumes file has been initialized.  */
    fp = fopen("grunge.dat","w+b","acc",uopen,&status);
    if(!fp)
    {
      perror("Cannot open grunge.dat\n");
      sys$exit(SS$_ABORT);
    }
    lock_ptrs[0] = &lock1;
    lock_ptrs[1] = &lock2;
    lock_ptrs[2] = &lock3;

/* Lock three "records" */
    for(i=0;i<NUM_RECS;i++)
    {
      status = sys$enqw(0,LCK$K_PWMODE,lock_stat+i,
                  0,lock_ptrs[i],0,0,0,0,0,0,0);
      sys_stat(status);
      sys_stat(lock_stat[i].lksb$w_status);
    }
```

Lock Example (continued)

```
/* Hold the locks. */
    lib$wait(&stall);
/* Write the three records. */
    num_obj = fwrite(recs,sizeof(Record),NUM_RECS,fp);
    if(num_obj == NUM_RECS)
    {
      printf("Records written\n");
    }
    else
    {
      printf("Error writing records\n");
    }

/* Give up locks. */
    for(i=0;i<NUM_RECS;i++)
    {
      status = sys$deq(lock_stat[i].lksb$l_lkid,0,0,0);
      sys_stat(status);
    }
    return(SS$_NORMAL);
}
```

Lock Example (continued)

```
/* useropen  AST routine, tallow file sharing. */
int     uopen(int *garb,struct FAB *fab,struct RAB *rab)
{
     int     status;
     fab->fab$b_shr = FAB$M_UPI|FAB$M_SHRPUT;
     rabptr = rab;
     return(SS$_NORMAL);
}
$
$
$ cc file_reader
$ link file_reader
$
$ cc file_writer
$ link file_writer
$
```

Lock Example (continued)

```
/* Hold the locks. */
    lib$wait(&stall);
/* Write the three records. */
    num_obj = fwrite(recs,sizeof(Record),NUM_RECS,fp);
    if(num_obj == NUM_RECS)
    {
      printf("Records written\n");
    }
    else
    {
      printf("Error writing records\n");
    }

/* Give up locks. */
    for(i=0;i<NUM_RECS;i++)
    {
      status = sys$deq(lock_stat[i].lksb$l_lkid,0,0,0);
      sys_stat(status);
    }
    return(SS$_NORMAL);
}
```

Lock Example (continued)

```
/* useropen  AST routine, tallow file sharing. */
int     uopen(int *garb,struct FAB *fab,struct RAB *rab)
{
    int     status;
    fab->fab$b_shr = FAB$M_UPI|FAB$M_SHRPUT;
    rabptr = rab;
    return(SS$_NORMAL);
}
$
$
$ cc file_reader
$ link file_reader
$
$ cc file_writer
$ link file_writer
$
```

Lock Example (continued)

```
$ spawn/nowait r file_writer
%DCL-S-SPAWNED, process ALLEN_1 spawned
$ spawn/nowait r file_reader
%DCL-S-SPAWNED, process ALLEN_2 spawned
$
Waiting...
$ show sys/sub
OpenVMS V7.0  on node ALLEN  23-JUL-1996 08:18:57.03  Uptime  0 12:08:57
  Pid     Process Name    State  Pri      I/O       CPU        Page flts  Pages
000000DE ALLEN_1         HIB      4       16    0 00:00:00.14       114    107  S
000000DF ALLEN_2         LEF      5       15    0 00:00:00.16       126    108  S
Waiting...
Waiting...
Records written
Record read
Name: Eddie Vedder
Age 32
$
$ spawn/nowait r file_reader
%DCL-S-SPAWNED, process ALLEN_1 spawned
Record read
Name: Eddie Vedder
Age 32
$
$ spawn/nowait r file_reader
%DCL-S-SPAWNED, process ALLEN_3 spawned
Record read
Name: Eddie Vedder
Age 32
$
```

Common Event Flags

Common event flags provide a cheap and simple form of inter-process synchronization.

- Common event flags have the following restrictions:
  - Not available for cluster-wide synchronization
  - Limited *Namespace*
  - No built-in deadlock detection
  - Limited to UIC group
- You must associate with common event flags, prior to use using:

```
sys$ascefc(efn, name,prot,perm)
```

- The **efn** argument to **sys$ascefc** uses a flag within a cluster to identify which cluster to use, i.e. **efn**s 64, 65, ..., 95 identify cluster 2, while efns 96, 97, ... 127 identify cluster 3.
- Common event clusters may be created permanently, if you have PRMCEB privilege. To delete a common event cluster use: **sys$delefc**.
- Temporary common event flag clusters are charged against your TQELM quota.
- To reuse a common event flag cluster you can disassociate by using **sys$dacefc**.

Common Event Flag Example

```
$
$ type synch_w_child.c
/* Example of creating synchronizing processes, using common event flags.
*/
#include <jpidef.h>
#include <iledef.h>
#include <descrip.h>
#include <ssdef.h>
#include <starlet.h>
#include <stdio.h>
#include <pqldef.h>
#define LIST_END 0
#define TERM_ARG 0
#define PID_ARG 1
#define TERM_MAX 15
#define BASE_PRI 4
#define sys_stat(STS) if(!((STS)&1)) sys$exit(STS)
#define PARENT_FLAG 64
#define CHILD_FLAG 65
#define CLUSTER_2_EF PARENT_FLAG
#define NUM_MSGS 5
typedef struct pqls
{
    char    item;
    int     value;
} pqls;
```

Common Event Flag Example (continued)

```
int main(void)
{
    $DESCRIPTOR(rel,"Relationship");
    $DESCRIPTOR(image,"synch_w_dad");
    $DESCRIPTOR(prcnam,"max");
    $DESCRIPTOR(my_prcnam,"dad");
    long    pid;
    long    my_pid;
    int     i;
    char    terminal[TERM_MAX+1];
    struct dsc$descriptor_s term_d =  {TERM_MAX,DSC$K_DTYPE_T,DSC$K_CLASS_S};
    ile3 jpi_list[] =
            {{sizeof(terminal),JPI$_TERMINAL,0,0},
            {sizeof(my_pid),JPI$_PID},
             {0,LIST_END}};
    int     status;
    pqls    pql_list[] = {{PQL$_WSDEFAULT,1024},
                  {PQL$_WSQUOTA,1024},
                  {PQL$_WSEXTENT,2048},
                  {PQL$_LISTEND}};

/* Reset process name. */
    status = sys$setprn(&my_prcnam);
    sys_stat(status);
/* Fill in the JPI list to get terminal name */
    jpi_list[TERM_ARG].ile3$ps_bufaddr = terminal;
    jpi_list[TERM_ARG].ile3$ps_retlen_addr = &term_d.dsc$w_length;
    jpi_list[PID_ARG].ile3$ps_bufaddr = &my_pid;
/* Init the terminal name descriptor. */
    term_d.dsc$a_pointer = terminal;
```

Common Event Flag Example (continued)

```
/* Get terminal name. */
    status = sys$getjpiw(0,0,0,jpi_list,0,0,0);
    sys_stat(status);
/* Create child. */
    status = sys$creprc(&pid,&image,0,&term_d,
            &term_d,0,pql_list,&prcnam,BASE_PRI,0,0,0);
    sys_stat(status);
    status = sys$ascefc(CLUSTER_2_EF,&rel,0,0);
    sys_stat(status);
    sys$clref(PARENT_FLAG);
/* Wait for child. */
    status = sys$waitfr(PARENT_FLAG);
    sys_stat(status);
    sys$clref(PARENT_FLAG);
/* Print 5 coordinated messages. */
    for(i=1;i<=NUM_MSGS;i++)
    {
      printf("\nParent message %d\n",i);
      sys$clref(PARENT_FLAG);
      sys$setef(CHILD_FLAG);
      sys$waitfr(PARENT_FLAG);
    }
    return(SS$_NORMAL);
}
$
```

Common Event Flag Example (continued)

```
$ type synch_w_dad.c
#include <stdio.h>
#include <starlet.h>
#include <ssdef.h>
#include <lib$routines.h>
#include <descrip.h>
#define PARENT_FLAG 64
#define CHILD_FLAG 65
#define CLUSTER_2_EF CHILD_FLAG
#define sys_stat(STS) if (!((STS)&1)) sys$exit(STS);
#define NUM_MSGS (sizeof(msgs)/sizeof(*msgs))
int main(void)
{
     float    stall = 10.0;
     $DESCRIPTOR(rel,"Relationship");
     int      status;
     int      i;
     char     *msgs[] = {"Dear Dad,","School is great!",
                    "Love you and Mom!",
                    "Please send MONEY!!!!",
                    "\tLove, Max"};

     status = sys$ascefc(CLUSTER_2_EF,&rel,0,0);
     sys_stat(status);
  /* Keep dad waiting. */
     lib$wait(&stall);
     sys$clref(CHILD_FLAG);
     sys$setef(PARENT_FLAG);
```

Common Event Flag Example (continued)

```
    for(i = 0; i<NUM_MSGS;i++)
    {
      sys$waitfr(CHILD_FLAG);
      sys$clref(CHILD_FLAG);
      puts(msgs[i]);
      sys$setef(PARENT_FLAG);
    }
    sys$clref(CHILD_FLAG);
    return(SS$_NORMAL);
}
$
$ run synch_w_child

Parent message 1
Dear Dad,

Parent message 2
School is great!

Parent message 3
Love you and Mom!

Parent message 4
Please send MONEY!!!!

Parent message 5
    Love, Max
$
```

Logical Names

Logical names can be used to pass information to processes and programs, as well as for device and file name independence.

- Logical names are stored in tables. The logical name **LNM$FILE_DEV** in the table **LNM$SYSTEM_DIRECTORY** identifies the order in which the logical name tables will be searched. The default order is to search:

1. LNM$PROCESS
2. LNM$JOB
3. LNM$GROUP
4. LNM$SYSTEM (search list for LNM$SYSTEM then LNM$SYSCLUSTER)

- Logical names may be created through the **sys$crelnm** system service.
- Logical name tables may be created through the **sys$crelnt** system service.
- Logical names may be translated, using the `sys$trnlnm` system service. Format of `sys$trnlnm`:

```
sys$trnlnm(attr,tabnam,lognam,acmode,itmlst)
```

SYS$TRNLNM Call Arguments

Argument	Meaning
attr	LNM$M_CASE_BLIND
tabnam	Table name
lognam	Logical name
itmlst	item list 3 format Basic item codes: LNM$_ATTRIBUTES (LNM$M_CONCEALED, LNM$M_CONFINE, LNM$M_CRELOG, LNMM_EXISTS, LNMM_NOALIAS, LNMM_TABLE, LNMM_TERMINAL) LNM$_CHAIN Process next adjacent item list LNM$_INDEX Current index LNM$_LENGTH Return length LNM$_STRING Equivalence string

Logical Name Translation Example

$ **type translate.c**

```
/* Sample logical name translation. */
#include <ssdef.h>
#include <iledef.h>
#include <lnmdef.h>
#include <starlet.h>
#include <lib$routines.h>
#include <descrip.h>
#include <string.h>
#define STRING 0
#define MAX_EQUIV 255
#define MAX_TRANS 10
#define sys_stat(STS) if(!((STS)&1)) sys$exit(STS)
int main(void)
{
    struct dsc$descriptor_s prog_logical=
      {sizeof("BAE$DEBUG")-1};
    $DESCRIPTOR(table,"LNM$FILE_DEV");
    char    equiv_s[MAX_EQUIV+1];
    char    new_log[MAX_EQUIV+1] = "BAE$DEBUG";
    int     status;
    struct  dsc$descriptor_s equiv = {sizeof(equiv_s)};
    int     index_level = 0;
    ile3    item_list[] = {{sizeof(equiv_s),LNM$_STRING},{0}};

/* Set up item list to translate BAE$DEBUG. */
    item_list[STRING].ile3$ps_bufaddr = &equiv_s;
    item_list[STRING].ile3$ps_retlen_addr = &equiv.dsc$w_length;
/* Set up descriptors for logical and equivalence names. */
    equiv.dsc$a_pointer = equiv_s;
    prog_logical.dsc$a_pointer = new_log;
```

Logical Name Translation Example (continued)

```
/* Translate until there are no more levels. */
    do
    {
    /* Translate using LNM$FILE_DEV table. */
      status = sys$trnlnm(0,&table,&prog_logical,0,item_list);
    /* if no more translations do not display. */
      if(status != SS$_NOLOGNAM)
      {
            sys_stat(status);
      /* Display the logical. */
            lib$put_output(&equiv);
      /* Copy equivalence name to logical for next
                  translation */
            prog_logical.dsc$w_length = equiv.dsc$w_length;
            equiv.dsc$w_length = sizeof(equiv_s);
            strncpy(new_log,equiv_s,MAX_EQUIV);
       /* Determine the current level of translation. */
            index_level++;
            if(index_level > MAX_TRANS)
                  sys$exit(SS$_TOOMANYLNAM);
       }
     }
     while(status !=SS$_NOLOGNAM);
/* If no translations generate an error message. */
     if(index_level)
     {
       return(SS$_NORMAL);
     }
```

Logical Name Translation Example (continued)

```
    else
    {
      return(SS$_NOLOGNAM);
    }
}
$
$
$ sh log bae$debug
%SHOW-S-NOTRAN, no translation for logical name BAE$DEBUG
$
$ r translate
%SYSTEM-F-NOLOGNAM, no logical name match
$
```

Logical Name Translation Example (continued)

```
$ def bae$debug allen
$ r translate
ALLEN
$
$ def allen albert
$ def albert einstein
$ r translate
ALLEN
ALBERT
EINSTEIN
$ define einstein allen
$ r translate
ALLEN
ALBERT
EINSTEIN
ALLEN
ALBERT
EINSTEIN
ALLEN
ALBERT
EINSTEIN
ALLEN
ALBERT
%SYSTEM-F-TOOMANYLNAM, logical name translation exceeded allowed depth
$
```

Clusterwide Logicals

Version 7.2 introduces clusterwide logical names.

- A logical name created in a cluster-wide logical name table is made available throughout all nodes in a VMScluster.
- Cluster-wide logical names/name tables are made available to all nodes in a VMS cluster, even nodes booting after the logical names have been created.
- They can be created on VAX or Alpha nodes.
- Works in a non-cluster environment. (i.e. The clusterwide table names exist and logical names may be placed in them, but they are not shared.)
- Requires V7.2 or greater for nodes to participate.
- Name tables are shared.
- The database is replicated on every node in a VMScluster (automatically).

Cluster-Wide Logicals Implementation

- New Logical name tables and logical names:

Name	Function
LNM$CLUSTER_TABLE	Parent name table for all cluster-wide name tables
LNM$CLUSTER	Logical name for LNM$CLUSTER_TABLE
LNM$SYSCLUSTER_TABLE	Default cluster-wide name table containing system logical names.
LNM$SYSCLUSTER	Logical name for LNM$SYSCLUSTER_TABLE

- The logical name **LNM$SYSTEM** is defined as a search list for **LNM$SYSTEM** and **LNM$SYSCLUSTER**
- Order of search using **lnm$file_dev:**

```
$ sh log/table=lnm$system_directory lnm$file_dev
   "LNM$FILE_DEV" = "LNM$PROCESS" (LNM$SYSTEM_DIRECTORY)
     = "LNM$JOB"
     = "LNM$GROUP"
     = "LNM$SYSTEM"
     = "DECW$LOGICAL_NAMES"
1  "LNM$SYSTEM" = "LNM$SYSTEM_TABLE" (LNM$SYSTEM_DIRECTORY)
     = "LNM$SYSCLUSTER"
2  "LNM$SYSCLUSTER" = "LNM$SYSCLUSTER_TABLE" (LNM$SYSTEM_DIRECTORY)
1  "DECW$LOGICAL_NAMES" [table] = "" (LNM$SYSTEM_DIRECTORY)
$
```

Cluster-Wide Logical Name API

No major changes are required to use cluster-wide logical names within an application!

- To create a cluster-wide logical name table, simple use a parent table that is cluster-wide.
- Cluster-wide logical name tables can be created at the DCL level using:
- **$CREATE/NAME_TABLE/PARENT_TABLE=LNM$CLUSTER_TABLE *table***
- To share cluster-wide logical names, simply put them in a cluster-wide logical name table.
- You will get a new attribute bit (**LNM$M_CLUSTERWIDE**) if you ask for attributes. (Also available in **F$TRNLNM**)
- The **LNM$M_INTERLOCKED** bit can be used to synchronize completely with modifications made by other nodes. (Uses the lock **LNM$CWLOGICALS**)

Cluster-Wide Logical Name API Example

```
$ type create_cluster_log.c
/*   Sample program to create a user-mode cluster-wide logical name. */
#include <ssdef.h>
#include <stdio.h>
#include <iledef.h>
#include <syidef.h>
#include <lnmdef.h>
#include <starlet.h>
#include <lib$routines.h>
#include <descrip.h>
#include <string.h>
#include <unistd.h>
#define MAX_NODE 256
#define check(STS) if(!((STS)&1)) sys$exit(STS)

int     main(void)
{
        static  char    equiv[] = "Check the market";
        $DESCRIPTOR(prog_logical,"BAE_TOKEN");
        static  char    node_str[MAX_NODE];
        struct  dsc$descriptor_s node =
                {sizeof(node)-1,0,0,node_str};
        $DESCRIPTOR(partab,"LNM$CLUSTER");
        $DESCRIPTOR(my_cw_tab,"TOKEN_TAB");
        int     status;
        ile3    lnm_items[] = {{sizeof(equiv)-1,LNM$_STRING,equiv,0},{0,0}};
        const   int     node_item = SYI$_NODENAME;
```

Cluster-Wide Logical Name API Example (continued)

```
/* Identify the node that we are running on. */
        status = lib$getsyi(&node_item,0,&node,&node.dsc$w_length);
        check(status);
        node_str[node.dsc$w_length] = '\0';
        printf("Creating logical name and table on node: %s\n",
                node_str);
/* Create the cluster-wide logical name, using LNM$CLUSTER as the
   parent table.
*/
        status = sys$crelnt(0,0,0,0,0,&my_cw_tab,&partab,0);
        check(status);
/* Create the logical name in our newly created cluster-wide logical name
   table.
*/
        status = sys$crelnm(0,&my_cw_tab,&prog_logical,0,lnm_items);
        check(status);
/* Take a cat nap. */
        printf("Sleeping for 10 minutes.\n");
        sleep(600);
        return(SS$_NORMAL);
}

$
```

Cluster-Wide Logical Name API Example (continued)

```
$ type  trans_cluster_log.c
/*  Sample program to translate a cluster-wide logical name.  */
#include <ssdef.h>
#include <stdio.h>
#include <iledef.h>
#include <syidef.h>
#include <lnmdef.h>
#include <starlet.h>
#include <lib$routines.h>
#include <descrip.h>
#include <string.h>
#include <unistd.h>
#define MAX_NODE 256
#define check(STS) if(!((STS)&1)) sys$exit(STS)
#define MAX_EQUIV 255
int     main(void)
{
        static  char    equiv[MAX_EQUIV];
        $DESCRIPTOR(prog_logical,"BAE_TOKEN");
        static  char    node_str[MAX_NODE];
        struct  dsc$descriptor_s node =
                {sizeof(node)-1,0,0,node_str};
        $DESCRIPTOR(my_cw_tab,"TOKEN_TAB");
        static unsigned short   elen;
        int     status;
        ile3    lnm_items[] = {{sizeof(equiv)-1,LNM$_STRING,equiv,&elen},
                                        {0,0}};
        const   int     node_item = SYI$_NODENAME;
 /* Guarantee that no changes are in progress. */
        int     lnm_attr = LNM$M_INTERLOCKED;
```

Cluster-Wide Logical Name API Example (continued)

```
/* Get our node name and identify where we are running. */
        status = lib$getsyi(&node_item,0,&node,&node.dsc$w_length);
        check(status);
        node_str[node.dsc$w_length] = '\0';
        printf("Translating logical name on node: %s\n", node_str);
/* Translate the logical name using the cluster-wide name table
   TOKEN_TAB
*/
        status = sys$trnlnm(&lnm_attr,&my_cw_tab,&prog_logical,0,lnm_items);
        check(status);
        equiv[elen] = '\0';
        printf("BAE_TOKEN == %s\n",equiv);
        return(SS$_NORMAL);
}
$ cc create_cluster_log
```

Cluster-Wide Logical Name API Example (continued)

```
$ link create_cluster_log,sys$library:vaxcrtl/lib
$ cc trans_cluster_log
$ link trans_cluster_log
$
$ copy trans_cluster_log.exe sys$common:[sysmgr]
$
```

- Run on node ALLEN.

```
$
$ run create_cluster_log
Creating logical name and table on node: ALLEN
Sleeping for 10 minutes.
```

- Run on node VEDDER.

```
$ set host vedder

 Welcome to OpenVMS (TM) Alpha Operating System, Version I7.2

Username: system
Password:
   Welcome to OpenVMS (TM) Alpha Operating System, Version I7.2 on node VEDDER
    Last interactive login on Wednesday,  5-AUG-1998 03:22:24.23
    Last non-interactive login on Wednesday,  5-AUG-1998 03:33:05.49
$
```

Cluster-Wide Logical Name API Example (continued)

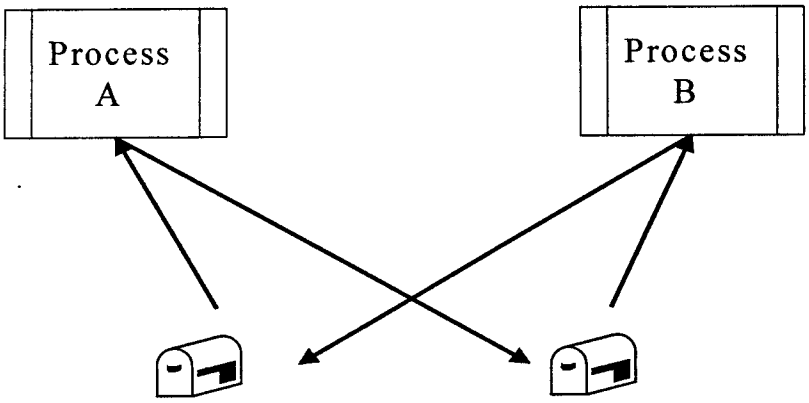
```
$ sh log/acc=user/table=token_tab bae_token
   "BAE_TOKEN" = "Check the market" (TOKEN_TAB)
$ r trans_cluster_log
Translating logical name on node: VEDDER
BAE_TOKEN == Check the market
$
$ write sys$output f$trnlnm("bae_token","token_tab",,"USER",,"CLUSTERWIDE")
TRUE
$
$ !After rundown of other program
$
$ r trans_cluster_log
Translating logical name on node: VEDDER
BAE_TOKEN == Check the market
$
$ sh log/acc=user/table=token_tab/full bae_token
   "BAE_TOKEN" [user] = "Check the market" (TOKEN_TAB)
$
```

Mailboxes

One form of interprocess communication is through mailboxes.

- A mailbox is a pseudo-device which can be read or written. Writes are queued to the mailbox until completed through reads.
- Mailboxes must be created by some process prior to use.
- Mailboxes are probably most easily accessed through the **sys$qio** system service.
- Mailbox access should be coordinated. Typically, only one process should read any given mailbox.

Typical Mailbox Design



Mailboxes (continued)

- The **sys$crembx** System Service format:

```
sys$crembx(prmflg,chan,maxmsg,bufquo,promsk,acmode,
lognam,flags,nullarg)
```

- Major arguments for **sys$crembx**:

Argument	Meaning
prmflg	if set mailbox is permanent requires PRMMBX privilege if temporary TMPMBX privilege is required
chan	channel identification for sys$qio
maxmsg	Maximum message size
bufquo	Mailbox size (default DEFMBXBUFQUO SYSGEN parameter)
lognam	Logical name

- LNM$TEMPORARY_MAILBOX defaults to LNM$JOB
  - Mailboxes may be deleted using **sys$delmbx**.

Mailbox Example

$ **type c_mbx.c**

```
/* Sample mailbox writer using C stdio.  NOTE: This is an illustation, not a
   recommended approach  */
#include <descrip.h>
#include <ssdef.h>
#include <starlet.h>
#include <stdio.h>
#include <string.h>
#define sys_stat(STS) if(!((STS)&1)) sys$exit(STS)
#define MBX_SIZE 60
int main(void)
{
    int     status;
    char    mbx_name[] = "DYLAN$MAIL";
    struct  dsc$descriptor_s mbx = {sizeof(mbx_name)-1};
    short   chan;
    FILE    *fp;
    char    mbx_data[MBX_SIZE+1] = "Fan mail from a flounder";
    int     i;

    mbx.dsc$a_pointer = mbx_name;
/* Create/open the mailbox. */
    status = sys$crembx(0,&chan,MBX_SIZE,0,0,0,&mbx,0,0);
    sys_stat(status);
/* Reopen for C stdio access. */
    fp = fopen(mbx_name,"r+b");
```

Mailbox Example (continued)

```
    if(!fp)
    {
      fprintf(stderr,"Cannot open mailbox\n");
      sys$exit(SS$_ABORT);
    }
/* Blast a message to the mailbox. */
    fwrite(mbx_data,1,sizeof(mbx_data),fp);
    return(SS$_NORMAL);
}
$
```

Mailbox Example (continued)

```
$ type c_mbx_read.c
/* Sample of using mailboxes from C
   NOTE: This is a kluge example for purely illustrative purposes, NOT a
     recommended method for using mailboxes.
*/
#include <descrip.h>
#include <ssdef.h>
#include <starlet.h>
#include <stdio.h>
#define sys_stat(STS) if(!((STS)&1)) sys$exit(STS)
#define MBX_SIZE 80
int main(void)
{
     int      status;
     char     mbx_name[] = "DYLAN$MAIL";
     struct   dsc$descriptor_s mbx = {sizeof(mbx_name)-1};
     short    chan;
     FILE     *fp;
     char     mbx_data[MBX_SIZE];

     mbx.dsc$a_pointer = mbx_name;
 /* Create the mailbox or assign a channel to it. */
     status = sys$crembx(0,&chan,MBX_SIZE,0,0,0,&mbx,0,0);
     sys_stat(status);
 /* Reopen the mailbox and access it through stdio */
     fp = fopen(mbx_name,"r+b");
     if(!fp)
     {
       fprintf(stderr,"Cannot open mailbox\n");
       sys$exit(SS$_ABORT);
     }
```

Mailbox Example (continued)

```
/* Read a fixed size block from the mailbox. */
    fread(mbx_data,1,MBX_SIZE,fp);
/* Print the info. */
    puts(mbx_data);
    return(SS$_NORMAL);
}
$
$ spawn/nowait r c_mbx_read
%DCL-S-SPAWNED, process ALLEN_1 spawned
$
$ show sys/sub
OpenVMS V7.0  on node ALLEN  23-JUL-1996 07:57:46.88  Uptime  0 11:47:47
  Pid    Process Name    State  Pri      I/O       CPU        Page flts  Pages
000000DC ALLEN_1         LEF      4       14    0 00:00:00.13      113    103  S
$ sh log dylan$mail

   "DYLAN$MAIL" = "MBA490:" (LNM$JOB_80D24A40)
$
$ run c_mbx
Fan mail from a flounder
$
$ show sys/sub
$
```

Termination Mailboxes

Termination mailboxes provide a mechanism of detecting the termination of a process, which you have created.

- When a process is deleted a termination accounting message is sent to the Job Controller process, if accounting has not been disabled for the process. Your process would receive the same message through the termination mailbox.
- The format of an accounting message is defined by **$ACCDEF/accdef.h**. The message includes the exit status of the last program run by the process.
- To detect termination of a created process:
  a. Create a Mailbox (**sys$crembx**)
  b. Determine the unit number of the mailbox (**sys$getdvi**)
  c. Pass the unit number of the mailbox to **sys$creprc**, in the mbxunt argument
  d. Post a read on the mailbox channel, using sys$qio.
- An example of termination mailboxes will be shown in the next course.