IntraCluster Communications
Module 7

Introduction

Students will learn ICC concepts of *associations* and *connections*, how they work and how to use them for communications within a cluster. Students will review a couple of ICC applications and be able to analyze the design considerations associated with ICC programming.

Objectives

Students should be able to:

- Describe *associations* and *connections* with respect to ICC.
- Set up simple ICC communications within program control.

Topics

- Intra-Cluster Communications (ICC)
- ICC Concepts
- ICC Programming Considerations

Intra-Cluster Communications (ICC)

Intra-cluster communications allows applications to communicate between nodes within a OpenVMScluster, without the use of any layered network software or special licenses.

ICC Features

- "Easy-to-use" system service interface for inter-node (within a cluster), inter-process communication.
- Interface is callable from all access modes.
- Independent of any networking products.
- Works within a single, non-clustered node.
- Supports clients and servers.
- Minimal privileges are required:
    - Client need NETMBX
    - Server needs NETMBX and SYSNAM
- Registry for servers and services, supporting security interfaces.

ICC Concepts

ASSOCIATION

- Named link between application and ICC.
- Name is associated with a node and must be unique within a node.
- Provides an identification for a server to remote nodes.
- Created by calling **`SYS$ICC_OPEN_ASSOC`** or **`SYS$ICC_CONNECT`**.
- Client may use default association.

NODE

- Stand-alone system or cluster member.
- Identified by SCSNODE name.

CONNECTION

- Link between two associations.
- Established by `SYS$ICC_CONNECT` (client) and `SYS$ICC_ACCEPT` (server).
- Communications within a connection may be asynchronous or synchronous.
- Multiple connections may be active simultaneously on the same association.
- A non-default association must have a connection AST routine associated with it to allow for accepting/rejecting incoming connections.

SERVER

- Defines the original association.
- Accepts/rejects connections.
- Must supply a connect AST routine for association.
- May operate totally asynchronously.
- Must be able to handle **multiple** incoming connection requests.

CLIENT

- Connects to an existing association.
- May define its own association (act as server as well as client) or use default association.

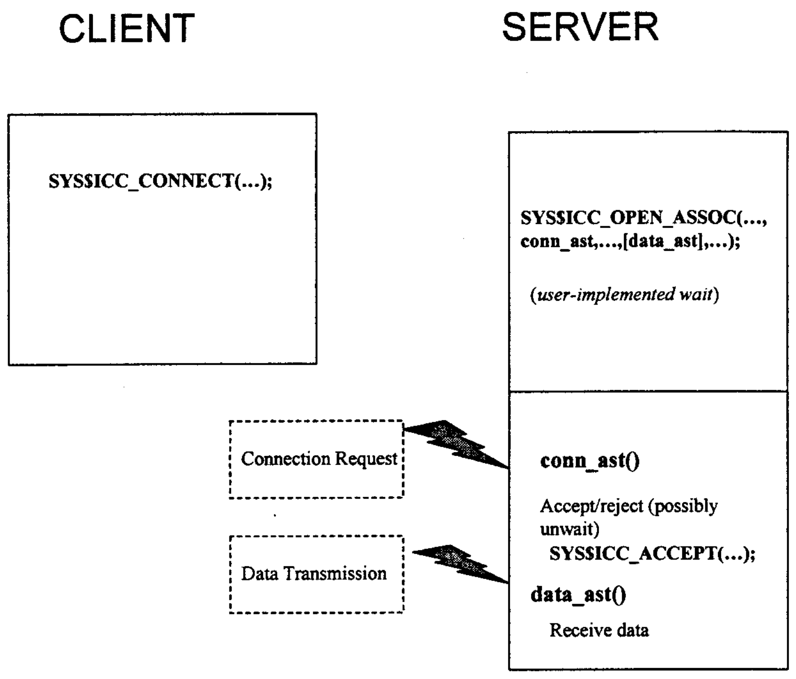
ICC SIMPLE CLUSTERWIDE REGISTRY

- Provides method to register association names under a single logical name, allowing multiple servers.

HANDLES

- Identify the association or connection.
- Used with accepts/rejects (associations).
- Used with receives, transmits, transceives, and replies (connections).

ICC Model


CLIENT
SERVER
SYS$ICC_CONNECT(...);
SYS$ICC_OPEN_ASSOC(..., conn_ast,...,[data_ast],...);
(user-implemented wait)
Connection Request
Data Transmission
conn_ast()
Accept/reject (possibly unwait)
SYS$ICC_ACCEPT(...);
data_ast()
Receive data

ICC Routines

Association

```
int SYS$ICC_OPEN_ASSOC(unsigned int *assoc_handle, void
*assoc_name, void *logical_name,void
(*conn_event_rtn)(__unknown_params), void (*disc_event_rtn)
(__unknown_params), void (*recv_rtn)(__unknown_params),
unsigned int maxflowbuf_cnt, unsigned int prot);
```

```
int SYS$ICC_CLOSE_ASSOC(unsigned int assoc_handle);
```

Connection

```
int SYS$ICC_CONNECT[W](struct _ios_icc *ios_icc, void
(*astadr)(__unknown_params), __int64 astprm, unsigned int
assoc_handle, unsigned int *conn_handle, void *remote_assoc,
void *remote_node, unsigned int user_context, char *
conn_buf, unsigned int return_buf_len, unsigned int
*retlen_addr, unsigned int flags);
```

```
int SYS$ICC_ACCEPT(unsigned int conn_handle, char
*accept_buf, unsigned int accept_len, unsigned int
user_context, unsigned int flags);
```

```
int SYS$ICC_REJECT(unsigned int conn_handle, char
*reject_buf, unsigned int reject_buf_len, unsigned int
reason);
```

```
int SYS$ICC_DISCONNECT[W](unsigned int conn_handle, struct
iosb iosb, void (*astadr)(_unknown_params),__int64 astprm,
char *disc_buf, unsigned int disc_buf_len);
```

ICC Routines (continued)

Communication

```
int SYS$ICC_TRANSMIT[W](unsigned int conn_handle, struct
ios_icc *ios_icc, void(*astadr)(__unknown_params),__int64
astprm, char *send_buf, unsigned int send_len);
```

```
int SYS$ICC_RECEIVE[W](unsigned int conn_handle, struct
ios_icc *ios_icc, void(*astadr)(__unknown_params),__int64
astprm, char *recv_buf, unsigned int recv_buf_len);
```

```
int SYS$ICC_TRANSCEIVE[W](unsigned int conn_handle, struct
ios_icc *ios_icc, void(*astadr)(__unknown_params),__int64
astprm, char *send_buf, unsigned int send_len);
```

```
int SYS$ICC_REPLY[W](unsigned int conn_handle, struct
ios_icc *ios_icc, void(*astadr)(__unknown_params),__int64
astprm, char *reply_buf, unsigned int reply_buf_len);
```

User-supplied AST routines

```
void conn_disc_evt(unsigned int event_type, unsigned int
conn_handle, unsigned int data_len, char *data_bfr, unsigned
int P5, unsigned int P6, char *P7);
```

```
void data_evt(unsigned int message_size, unsigned int
conn_handle, unsigned int user_context);
```

ICC Programming Considerations

Associations

- Server must:
  - Declare and register an association name or open an association to a pre-registered namespace.
  - Be prepared to handle multiple incoming connection requests.
  - Declare a connection AST routine.
- Server may:
  - Declare a disconnection AST routine (may be the same routine as connection, distinguished by event_type of **`ICC$C_EV_CONNECT/ICC$C_EV_DISCONNECT`**).
  - Declare a receive data AST routine. This routine must be capable of handling data for multiple connections. The **`user_context`** data can be used to provide unique information for each specific connection.
- Client may:
  - use the default association, where a name will be generated of the form **`ICC$PID_`***nnnnnnnn*.
- Declaring and registering an association name requires SYSNAM privilege (can be done through **`ICC$STARTUP.COM`**).
- Opening an association requires NETMBX privilege.
- When an association is disconnected, all participants must disconnect, either implicitly through image rundown or explicitly.

ICC Programming Considerations (continued)

Connections

- Client opens connections through **`SYS$ICC_CONNECT[W]`**.
- Server opens connections through **`SYS$ICC_ACCEPT`**.
- Server must be able to handle multiple incoming connections.
- Up to 262,144 connections per system are allowed, subject to BYTLM quota.
- Unique connection handle is passed to connection AST routine on each unique connection request.
- All user mode connections are disconnected at image rundown.
- Inner mode disconnections are the responsibility of the inner mode code, but will be disconnect on process termination.

Data Transmission

- Client will normally issue a transmit/transceive to initiate communications followed by a receive.
- Server will normally issue a receive to accept data and respond to client transmission.
- Receiving of data may, and most likely will, be done in user specified AST routine.
- Transceive is basically transmit/receive combination.
- Reply is basically a transmit in response to a Request Handle passed through the `ios_icc` structure to a receive (written by Transceive).

ICC Programming Considerations (continued)

General

- Completion status, as well as subarguments, is passed through the **`ios_icc`** structure (more sophisticated **`iosb`**).
- Completion status should always be checked and handled in asynchronous requests.
- Data transmission is asynchronous and a wait (W) form of the service or completion ASTs should be used to guarantee that the data has arrived.
- The **`SS$_SYNCH`** status, similar to the **`$ENQ`** system service, can be used to eliminate the need for AST delivery when Transmit, Receive, and Reply services are used (**`ICC$M_SYNCH_MODE`** flag must be set on Connect or Accept to put this into effect).
- The following argument to **`SYS$ICC_OPEN_ASSOC`** maintains communication bounds:

  `maxflowbuffcnt`tradeoff memory/quotas vs. potential stalls of all connections.
- The size of transfers is limited to 1MB, dependent on size of receive buffer.

Very Simple ICC Example (continued)

```
int main(void)
{
    int      status;
    int      i;
    int      a_handle;
    int      c_handle;
    $DESCRIPTOR(ra_name,"VERY_SIMPLE_SERVER");
    $DESCRIPTOR(node_pmt,"Server node: ");
    static   char   node[MAX_NODE_NAME];
    struct   dsc$descriptor_s r_node = {sizeof(node),
                    DSC$K_DTYPE_T,DSC$K_CLASS_S,node};
    int      acc_len;
    struct _ios_icc        icc_iosb;
    char     info[MAX_BUFF];

/* Get remote node name to connect with. */
      status = lib$get_input(&r_node,&node_pmt,&r_node.dsc$w_length);
      check(status);
```

- Requesting a connection.

```
/* Connect. */
      printf("connect request.\n");
      status = sys$icc_connectw(&icc_iosb,0,0,
                        ICC$C_DFLT_ASSOC_HANDLE, &c_handle,
                        &ra_name,&r_node,0,0,0,info,sizeof(info),0,0);
      check(status);
      check(icc_iosb.ios_icc$w_status);
/* Display the value received in the return buffer. */
      printf("Connect buffer data: \"%s\"\n",info);
```

Very Simple ICC Example (continued)

- Receiving 5 messages from the server.

```
/* Accept and display the first 5 messages sent to us. */
      for(i=0;i<NUM_MSGS;i++)
      {
            status = sys$icc_receivew(c_handle,
                        &icc_iosb,0,0,info,sizeof(info));
            check(status);
            check(icc_iosb.ios_icc$w_status);
            printf("Read %d bytes\n",icc_iosb.ios_icc$l_rcv_len);
            printf("Read \"%s\"\n",info);
      }
```

- Disconnecting.

```
/* Disconnect. */
      status = sys$icc_disconnect(c_handle,&icc_iosb,0,0,0,0);
      if(status != SS$_DISCONNECT)
      {
            check(status);
            check(icc_iosb.ios_icc$w_status);
      }
      return(EXIT_SUCCESS);
}
```

Very Simple ICC Example (continued)

```
$ cc very_simple_client
$ link very_simple_client
$ type very_simple_server.c
/*
    Very simple illustration of setting up an association and
    accepting a connect request.
    This is the server used with the "very_simple_client" program.
    Note, server must be run before the client.
    SYSNAM privilege is required to set up an association.
*/
#include <syidef.h>
#include <stdlib.h>
#include <starlet.h>
#include <iccdef.h>
#include <iledef.h>
#include <stdio.h>
#include <descrip.h>
#include <lib$routines.h>
#include <iosbdef.h>
#include <string.h>
#include <ssdef.h>
#define check(STS) if(!((STS)&1)) sys$exit(STS)
#define MAX_BUFF 1000
/* Prototype for the REQUIRED connection ast routine. */
void conn_ast(unsigned int event_type,unsigned int c_handle,
             unsigned int data_len,char *data_buff,
             unsigned int p5, unsigned int p6, char *p7);
```

Very Simple ICC Example (continued)

```
/* Static storage is used to pass connection handle back to main.
   This is a little unstructured, but there is little alternative in this
   simple design.*/
static int  glob_c_handle;
int main(void)
{
    int     status;
    int     a_handle;
    $DESCRIPTOR(a_name,"VERY_SIMPLE_SERVER");
    int     acc_len;
    char    u_data[] = "Fan mail from a flounder.";
    struct  _ios_icc     icc_iosb;
    char    buffer[BUFSIZ];
    int     i;
```

Very Simple ICC Example (continued)

- Server establishing an association.

```
/* Establish an association for VERY_SIMPLE_SERVER.
   NOTE: the connection AST is not optional for a server.
*/
     printf("Setting up association\n");
     status = sys$icc_open_assoc(&a_handle,&a_name,0,0,conn_ast,
              0,0,0,0);
     check(status);
```

- User defined wait.

```
/* Go to sleep until somebodey wants to "hear" what we have to say. */
     printf("Waiting to accept connect request.\n");
     sys$hiber();
```

- Accepting the connection request

```
/* Accept the client's connection request. Send along a little
  message that she/he can read after completing the connect request.
   Handle passed to us through the connection AST routine.
*/
      status = sys$icc_accept(glob_c_handle,u_data,
                   sizeof(u_data),0,0);
      check(status);
```

- Transmitting until disconnect received.

```
/* Read from the keyboard and send along until we are disconnected
   by the client.
   NOTE: due to the very simple design, we will read in one more
   user message than the client will accept!
*/
```

Very Simple ICC Example (continued)

```
    do
    {
        printf("Enter string: ");
        gets(buffer);
        status = sys$icc_transmitw(glob_c_handle,&icc_iosb,
              0,0,buffer,strlen(buffer)+1);
        if(status != SS$_WRONGSTATE)
        {
              check(status);
              check(icc_iosb.ios_icc$w_status);
        }
        else
        {
              printf("Last item (%s) was not transmitted.\n",buffer);
        }
    }
    while(status != SS$_WRONGSTATE);
    return(EXIT_SUCCESS);
}
```

Very Simple ICC Example (continued)

- AST routine to handle connection request.

```
/* Connection AST routine. */

#define MAX_UNAME 12

void conn_ast(unsigned int event_type, unsigned int c_handle, unsigned int
    data_len, char *data_buff, unsigned int p5, unsigned int p6, char *p7)

{

    char      user[MAX_UNAME+sizeof('\0')];

    static  int    conn_count = 0;

    int       status;

    char      rej_buf[] = "I'm too busy too deal with you now!";

/* Get and print the connecting user name.  Note: you must copy this info

   prior to calling any other ICC system services. */

    strncpy(user,p7,MAX_UNAME);

    user[MAX_UNAME] = '\0';

/* Guarantee only one connection is accepted by this process. */

    if(conn_count == 0)

    {

      printf("Accepting connection for user: %s\n",user);

    /* Display connecting process' pid. */

      printf("With pid: %8x\n",p6);

    }

    else

    {

    /* Ignore any after first connection. */

      printf("Rejecting connection for user: %s\n",user);

    /* Display connecting process' pid. */

      printf("With pid: %8x\n",p6);

      status = sys$icc_reject(c_handle,rej_buf,sizeof(rej_buf),0);

      check(status);

      return;

    }
```

Very Simple ICC Example (continued)

```
    conn_count++;
    printf("Accepting connection for user: %s\n",user);
/* Display connecting process' pid. */
    printf("With pid: %8x\n",p6);
/* Pass the connection handle back to main. */
    glob_c_handle = c_handle;
/* wake up and get to work. */
    sys$wake(0,0);
}
```

Very Simple ICC Example (continued)

```
$ cc very_simple_server
$  link very_simple_server
$
```

- Example of node ALLEN as server, node VEDDER as client.

Node ALLEN session

```
$ write sys$output f$getsyi("SCSNODE")
ALLEN
$ run very_simple_server
Setting up association
Waiting to accept connect request.
Accepting connection for user: BAE
With pid: 2040011f
Enter string: Now is the
Enter string: time for all
Enter string: good people to come to
Enter string: the aid of their
Enter string: OpenVMScluster!!!
Enter string: This will be ignored
Last item (This will be ignored) was not transmitted.
$
```

Node VEDDER session

```
$ sh proc
10-AUG-1998 04:15:32.08   User: BAE          Process ID:   2040011F
                          Node: VEDDER       Process name: "BAE"
```

Very Simple ICC Example (continued)

```
Terminal:              RTA1:  (1054::SYSTEM)
User Identifier:       [BUNDER,BAE]
Base priority:         4
Default file spec:     ALLEN$DKA200:[BAE]
Number of Kthreads: 1

Devices allocated:  VEDDER$RTA1:
$ write sys$output f$getsyi("SCSNODE")
VEDDER
$ r very_simple_client
Server node: ALLEN
connect request.
Connect buffer data: "Fan mail from a flounder."
Read 11 bytes
Read "Now is the"
Read 13 bytes
Read "time for all"
Read 23 bytes
Read "good people to come to"
Read 18 bytes
Read "the aid of their "
Read 14 bytes
Read "OpenVMScluster!!!"
$
```

More Complex ICC Example

The following example is written more in the spirit of a true client/server application.

In the example:

- The server:
  - Runs as a detached process on all nodes within the cluster.
  - Operates asynchronously.
  - Accepts multiple connection requests.
  - Responds to all data requests by all clients, in AST routine, by sending the time in all access modes to any client requesting the information.
- The client:
  - Activated via a foreign command accepting a sampling interval and number of iterations in which to sample data.
  - Determines which nodes are in the cluster.
  - Connects to the server on each node (including self).
  - Sends request for CPU time statistics to all nodes and reads data.
  - Displays the data and goes to sleep before repeating last step for user-specified number of iterations.

Sample client interface

```
$ cpu_stat 5 3
Connecting with ALLEN
Connecting with VEDDER
    Node: Kern Exec  Sup User   IS MPSY Idle Up time (sec)
ALLEN   :   4%   1%   0%   6%   3%   0%  84%       14185
VEDDER  :   0%   0%   0%   0%   0%   0%  96%       13583
ALLEN   :   0%   0%   0%   1%  10%   0%  86%       14190
VEDDER  :  32%  13%   0%  30%   5%   0%  18%       13588
ALLEN   :   0%   0%   0%   1%  13%   0%  84%       14195
VEDDER  :  20%   4%   0%  33%   5%   0%  35%       13593
$
```

More Complex ICC Example (continued)

- Client software

```
$ type clu_cpu.h
typedef struct cpu_stat
{
    __int64 kern;
    __int64 exec;
    __int64 super;
    __int64 user;
    __int64 i_state;
    __int64 true_is;
    __int64 mp_sync;
    __int64 idle;
    __int64 old_tot;
    int up_time;
} CPU_STAT;
#define CPUSRV_C_CPU_STAT 100

$ type cpu_client.c
/*
    This is a little more elaborate illustration of a client
    that needs to connect to multiple servers.
    The program connects to the association CPU_SERVER on all
    nodes, including self and gets CPU time statistics and displays
    over user specified interval for user specified iterations.
    Runs as a foreign command, requiring the following symbol:
    $ CPU_STAT=="$device:[dir]CPU_CLIENT"
    Command format:
    $ CPU_STAT interval iterations
*/
```

More Complex ICC Example (continued)

```
#include <syidef.h>
#include <stdlib.h>
#include <starlet.h>
#include <iledef.h>
#include <stdio.h>
#include <unistd.h>
#include <descrip.h>
#include <lib$routines.h>
#include <iosbdef.h>
#include <iccdef.h>
#include "clu_cpu.h"
#include <ssdef.h>
#define MAX_NODES 255
#define LINES_PER_REFRESH 24
#define REQ_ARGS 3
#define INTV_ARG 1
#define NI_ARG 2
#define check(STS) if(!((STS)&1)) sys$exit(STS)
#define MAX_NODE_NAME 15
#define MAX_BUFF 1000
void get_node_list(struct dsc$descriptor_s **nodes,int *nc);
const __int64 PCT_FACT = 100;
```

More Complex ICC Example (continued)

```
int main(int argc,char **args)
{
    int      status;
    int      i,j;
    int      a_handle;
    $DESCRIPTOR(ra_name,"CPU_SERVER");
    static   char   node[MAX_NODE_NAME];
    struct   dsc$descriptor_s r_node = {sizeof(node),
       DSC$K_DTYPE_T,DSC$K_CLASS_S,node};
    int      acc_len;
    struct _ios_icc      icc_iosb;
    int      info = CPUSRV_C_CPU_STAT;
    struct dsc$descriptor_s     *node_list[MAX_NODES];
    int      c_handles[MAX_NODES];
    int      node_count;
    CPU_STAT *new_cpu_stats;
    CPU_STAT *old_cpu_stats;
    CPU_STAT calc;
    int      interval;
    int      num_iterations;
    __int64  tot_cpu;
    __int64  tmp_tot;
    int      lines;
/* Validate that we received both user specified arguments. */
    if(argc != REQ_ARGS)
    {
      fprintf(stderr,"Bad argument count, "
             "Format: cpu_stat interval iterations\n");
      return(EXIT_FAILURE);
    }
```

More Complex ICC Example (continued)

```
/* Convert the arguments to binary. */
     interval = atoi(args[INTV_ARG]);
     num_iterations = atoi(args[NI_ARG]);
/* Determine all nodes in cluster, including self. */
     get_node_list(node_list,&node_count);
/* Allocate arrays of new and old CPU_STAT structures for each node. */
     new_cpu_stats = (CPU_STAT *)malloc(node_count*sizeof(*new_cpu_stats));
     if(!new_cpu_stats)
     {
       perror("Cannot allocate new_stats buffer");
       return(EXIT_FAILURE);
     }
     old_cpu_stats = (CPU_STAT *)malloc(node_count*sizeof(*old_cpu_stats));
     if(!old_cpu_stats)
     {
       perror("Cannot allocate old_stats buffer");
       return(EXIT_FAILURE);
     }
/* Connect to all nodes, including ourself. */
     for(i=0;i<node_count;i++)
     {
       printf("Connecting with %.*s\n",node_list[i]->dsc$w_length,
                            node_list[i]->dsc$a_pointer);
       status = sys$icc_connectw(&icc_iosb,0,0,ICC$C_DFLT_ASSOC_HANDLE,
                      c_handles+i,&ra_name,node_list[i],0,0,0,0,0,0,0);
       check(status);
       check(icc_iosb.ios_icc$w_status);
     }
```

More Complex ICC Example (continued)

```
/* Get and display stats for each user specified iteration. */
    for(i=0,lines=0;i<num_iterations;i++)
    {
    /* Get stats for each node. */
      for(j=0;j<node_count;j++)
      {
      /* Notify the server that we want CPU stats. */
             status = sys$icc_transmitw(c_handles[j],&icc_iosb,0,0,
                    (char *)&info,
                    sizeof(int));
             check(status);
             check(icc_iosb.ios_icc$w_status);
      /* Note: no handle specified. */
             icc_iosb.ios_icc$l_req_handle = 0;

      /* Get the stats. */
             status = sys$icc_receivew(c_handles[j],&icc_iosb,0,0,
                    &new_cpu_stats[j],sizeof(new_cpu_stats[j]));
             check(status);
             check(icc_iosb.ios_icc$w_status);

      /* Put up a header every 24 lines. */
             if((lines % LINES_PER_REFRESH ) == 0 && !j)
               printf("%8s: %4s %4s %4s %4s %4s %4s %4s %10s\n",
                 "Node","Kern","Exec","Sup","User","IS","MPSY","Idle",
                 "Up time (sec)");
      /* Calculate total CPU time.  Note: MP_SYNCH does not count
         against total, since it is charged against Kernel mode time.
      */
```

More Complex ICC Example (continued)

```
        tot_cpu =
                new_cpu_stats[j].kern +
                new_cpu_stats[j].exec +
                new_cpu_stats[j].super +
                new_cpu_stats[j].user +
                new_cpu_stats[j].i_state;
/* Factor the idle time out of the IS time. */
        new_cpu_stats[j].true_is = new_cpu_stats[j].i_state -
                new_cpu_stats[j].idle;
/* Bump line counter. */
        lines++;
/* If first display show stats since last boot in typical
   UNIX style.
*/
        if(i==0)
        {
          printf("%8.*s: %3d%% %3d%% %3d%% %3d%% %3d%% %3d%%"
                " %3d%% %10d\n",
                 node_list[j]->dsc$w_length,
                 node_list[j]->dsc$a_pointer,
                 (int)(new_cpu_stats[j].kern*PCT_FACT/tot_cpu),
                 (int)(new_cpu_stats[j].exec*PCT_FACT/tot_cpu),
                 (int)(new_cpu_stats[j].super*PCT_FACT/tot_cpu),
                 (int)(new_cpu_stats[j].user*PCT_FACT/tot_cpu),
                 (int)(new_cpu_stats[j].true_is*PCT_FACT
                              /tot_cpu),
                 (int)(new_cpu_stats[j].mp_sync*PCT_FACT
                              /tot_cpu),
                 (int)(new_cpu_stats[j].idle*PCT_FACT/tot_cpu),
                 new_cpu_stats[j].up_time);
```

More Complex ICC Example (continued)

```
        old_cpu_stats[j].old_tot = tot_cpu;
      }
/* If not first display, show change since last iteration.*/
      else
      {
        calc.kern = new_cpu_stats[j].kern -
                   old_cpu_stats[j].kern;
        calc.exec = new_cpu_stats[j].exec -
                   old_cpu_stats[j].exec;
        calc.super = new_cpu_stats[j].super -
                   old_cpu_stats[j].super;
        calc.user = new_cpu_stats[j].user -
                   old_cpu_stats[j].user;
        calc.idle = new_cpu_stats[j].idle -
                   old_cpu_stats[j].idle;
        calc.true_is = new_cpu_stats[j].true_is -
                   old_cpu_stats[j].true_is;
        calc.mp_sync = new_cpu_stats[j].mp_sync -
                   old_cpu_stats[j].mp_sync;
/* Make sure that we save the old total for next iteration. */
        tmp_tot = tot_cpu;
        tot_cpu -= old_cpu_stats[j].old_tot;
        old_cpu_stats[j].old_tot = tmp_tot;

        printf("%8.*s: %3d%% %3d%% %3d%% %3d%% %3d%% %3d%%"
            " %3d%% %10d\n",
             node_list[j]->dsc$w_length,
             node_list[j]->dsc$a_pointer,
```

More Complex ICC Example (continued)

```
                    (int)(calc.kern*PCT_FACT/tot_cpu),
                    (int)(calc.exec*PCT_FACT/tot_cpu),
                    (int)(calc.super*PCT_FACT/tot_cpu),
                    (int)(calc.user*PCT_FACT/tot_cpu),
                    (int)(calc.true_is*PCT_FACT/tot_cpu),
                    (int)(calc.mp_sync*PCT_FACT/tot_cpu),
                    (int)(calc.idle*PCT_FACT/tot_cpu),
                   new_cpu_stats[j].up_time);
            }
/* Save stats for next iteration. */
            old_cpu_stats[j].kern = new_cpu_stats[j].kern;
            old_cpu_stats[j].exec = new_cpu_stats[j].exec;
            old_cpu_stats[j].super = new_cpu_stats[j].super;
            old_cpu_stats[j].user = new_cpu_stats[j].user;
            old_cpu_stats[j].true_is = new_cpu_stats[j].true_is;
            old_cpu_stats[j].mp_sync = new_cpu_stats[j].mp_sync;
            old_cpu_stats[j].idle = new_cpu_stats[j].idle;
              }
   /* Go to sleep for user specified interval. */
    sleep(interval);
   }
/* Disconnect. */
   for(j=0;j<node_count;j++)
   {
    status = sys$icc_disconnect(c_handles[j],&icc_iosb,0,0,0,0);
    if(status != SS$_DISCONNECT)
    {
            check(status);
            check(icc_iosb.ios_icc$w_status);
            }
   }
   return(EXIT_SUCCESS);
}
```

More Complex ICC Example (continued)

```
/* Routine to get a list of node names. Fills in an array of pointers
   to descriptors, which are allocated off the heap.  Caller is
        responsible for deallocating heap space if necessary.  Also returns
        count of total nodes.
*/
#define MAX_NODE_NAME 15
#define WILD_SYI -1
void  get_node_list(struct dsc$descriptor_s **nodes,int  *nc)
{
     int      status;
     int      next_node = 0;
     int      node_count;
     int      max_node;
     iosb     syi_iosb;
     int      csid = WILD_SYI;
     ile3     items[] = {{MAX_NODE_NAME,SYI$_SCSNODE},{0,0}};
     do
     {
/* Allocate a descriptor and fill it in.  Also initialize item list
   for GETSYI call.
*/
       nodes[next_node] = malloc(sizeof(struct dsc$descriptor_s));
       if(!nodes[next_node]) sys$exit(SS$_INSFMEM);
       items[0].ile3$ps_bufaddr =
               nodes[next_node]->dsc$a_pointer =
                     malloc(MAX_NODE_NAME);
       if(!nodes[next_node]->dsc$a_pointer) sys$exit(SS$_INSFMEM);
       items[0].ile3$ps_retlen_addr =
                     &nodes[next_node]->dsc$w_length;
```

More Complex ICC Example (continued)

```
       nodes[next_node]->dsc$b_dtype = DSC$K_DTYPE_T;
       nodes[next_node]->dsc$b_class = DSC$K_CLASS_S;
/* Wild card and get name of all nodes. */
       status = sys$getsyiw(0,&csid,0,items,&syi_iosb,
                        0,0);
       if(status !=SS$_NOMORENODE) check(status);
       next_node++;
     }
     while(status !=SS$_NOMORENODE);
/* Free up last unused descriptor. */
     free(nodes[--next_node]->dsc$a_pointer);
     free(nodes[next_node]);

/* Return updated count. */
     *nc = next_node;
}
$
```

More Complex ICC Example (continued)

- Server software

```
$ type cpu_server.c
/*
        This is an event driven server example that grabs CPU time
        stats and returns them to clients.
        It can be run as a detached process.
        DISCLAIMER:
        It contains a kernel mode routine which is NOT supported nor
        recommended!  If you change any of it there is a GOOD CHANCE
        YOU WILL CRASH THE SYSTEM.
*/
#include <syidef.h>
#include <stdlib.h>
#include <starlet.h>
#include <iccdef.h>
#include <iledef.h>
#include <stdio.h>
#include <descrip.h>
#include <lib$routines.h>
#include <iosbdef.h>
#include <string.h>
#include "clu_cpu.h"
#include <cpudef.h>
#include <mutexdef.h>
#include <pcbdef.h>
#include <ssdef.h>
#include <sch_routines.h>
#define check(STS) if(!((STS)&1)) sys$exit(STS)
#define MAX_BUFF 1000
/* Prototype for the REQUIRED connection ast routine. */
```

More Complex ICC Example (continued)

```
/* Establish an association for CPU_SERVER.
   NOTE: the connection AST is not optional for a server.
*/
        status = sys$icc_open_assoc(&a_handle,&a_name,0,0,conn_ast,
                        disc_ast,get_data_ast,0,0);
        check(status);
/* Go to sleep until somebodey wants to "hear" what we have to say. */
        sys$hiber();
/* We run until stopped. */
        return(SS$_NORMAL);
}
/* Connection AST routine. */
#define MAX_UNAME 12
void    conn_ast(unsigned int event_type,unsigned int c_handle,
                        unsigned int data_len,char *data_buff,
                        unsigned int p5, unsigned int p6, char *p7)
{
        int     status;
/* Accept the client's connection request.
*/
        status = sys$icc_accept(c_handle,0,0,0,0);
        check(status);
}
```

More Complex ICC Example (continued)

```
void    disc_ast(unsigned int event_type,unsigned int c_handle,
                    unsigned int data_len,char *data_buff,
                    unsigned int p5,
                    unsigned int p6,
                    char *p7)

{
        int     status;
        struct  _ios_icc        iosb;
/* Accept the client's connection request.*/
        status = sys$icc_disconnectw(c_handle,&iosb,0,0,0,0);
        if(status != SS$_DISCONNECT && iosb.ios_icc$w_status!= SS$_DISCONNECT)
        {
                check(status);
                check(iosb.ios_icc$w_status);
        }
}
int     get_cpu_stats(CPU_STAT *);
void    get_data_ast(unsigned msg_size,unsigned int c_handle,
                unsigned int user_ctx)
{
        int     buffer;
        struct  _ios_icc        icc_iosb;
        static CPU_STAT send_cpu_stat;
        int     status;
        struct
        {
                int     argc;
                CPU_STAT        *stat;
        }  k_args = {1,&send_cpu_stat};
```

More Complex ICC Example (continued)

```
/* Determine the data requested by client.  Right now we support
   only CPU stats.
*/
           status = sys$icc_receivew(c_handle,&icc_iosb,
                                0,0,(char *)&buffer,sizeof(buffer));
           check(status);
           check(icc_iosb.ios_icc$w_status);
           switch(buffer)
           {

               case CPUSRV_C_CPU_STAT:
/* Get aggreate CPU stats for all CPUs on this node. */
                      status = sys$cmkrnl(get_cpu_stats,&k_args);
                      check(status);
/* Sending them back to whoever requested them. */
                      status = sys$icc_transmitw(c_handle,&icc_iosb,
                                0,0,&send_cpu_stat,sizeof(send_cpu_stat));
                      check(status);
                      check(icc_iosb.ios_icc$w_status);
                      break;
/* Right now a bad data type "crashes" the server. */
               default:
                      sys$exit(SS$_BADPARAM);
           }
}
```

More Complex ICC Example (continued)

```
/* Kernel mode routine to collect and return the CPU stats. */
extern CPU      *smp$gl_cpu_data[];
extern int      exe$gl_abstim;
#define MAX_CPUS 32
#define KERN 0
#define EXEC 1
#define SUPER 2
#define USER 3
#define IS 4
#define MPSY 5
extern unsigned int smp$gl_active_cpus;
extern MUTEX SMP$GQ_CPU_MUTEX;
extern PCB      *CTL$GL_PCB;
/* User supplied "CALL"able mutex unlocker, since none are currently
   available in OpenVMS.
*/
void    sch_bae$unlock_quad(MUTEX *,PCB *);
int     get_cpu_stats(CPU_STAT *pcs)
{
        CPU     **cpu_ptr = (CPU **)&smp$gl_cpu_data;
        unsigned int active_cpus = smp$gl_active_cpus;
        int     i;
/* Clear mode accumulators. */
        pcs->kern = pcs->exec = pcs->super = pcs->user = pcs->i_state =
        pcs->mp_sync = pcs->idle = 0;
/* Make sure that CPUs do not go away while we are looking at them. */
        sch_std$lockr_quad(&SMP$GQ_CPU_MUTEX,CTL$GL_PCB);
```

More Complex ICC Example (continued)

```
/* Look at all CPUs.  The active_cpus variable will terminate
   the loop, after we have processed all active CPUs.
*/
        for(i=0;active_cpus && (i<MAX_CPUS);i++,active_cpus>>=1)
        {
                if(active_cpus & 1)
                {
                        pcs->kern += cpu_ptr[i]->cpu$q_kernel[KERN];
                        pcs->exec += cpu_ptr[i]->cpu$q_kernel[EXEC];
                        pcs->super += cpu_ptr[i]->cpu$q_kernel[SUPER];
                        pcs->user += cpu_ptr[i]->cpu$q_kernel[USER];
                        pcs->i_state += cpu_ptr[i]->cpu$q_kernel[IS];
                        pcs->mp_sync += cpu_ptr[i]->cpu$q_kernel[MPSY];
                        pcs->idle += cpu_ptr[i]->cpu$q_nullcpu;
                }
        }
/* Make sure that we give up the mutex. */
        sch_bae$unlock_quad(&SMP$GQ_CPU_MUTEX,CTL$GL_PCB);
/* Return up time in seconds. */
        pcs->up_time = exe$gl_abstim;
        return(SS$_NORMAL);
}
```

More Complex ICC Example (continued)

```
$ @cpu_build
Building client.
Building server.
$
```

- Server creation on node ALLEN

```
$ run/uic=bae/priv=(TMPMBX,SYSNAM,CMKRNL,WORLD,SYSPRV)/proc=CPU_SERVER -
$_ sys$sysdevice:[bae]cpu_server
%RUN-S-PROC_ID, identification of created process is 20400126
$ sh sys
OpenVMS I7.2  on node VEDDER  11-AUG-1998 01:43:50.31  Uptime  0 03:34:01
  Pid    Process Name    State  Pri      I/O       CPU        Page flts  Pages
20400101 SWAPPER         HIB     16        0   0 00:00:00.17          0      0
20400106 CLUSTER_SERVER  HIB     15       13   0 00:00:00.15         64     74
...
20400121 _RTA2:          CUR  0  10    13086   0 00:01:48.55      12412    125
20400126 CPU_SERVER      HIB      8        4   0 00:00:00.12         73     83
$
```

- Server creation on node VEDDER

```
$ run/uic=bae/priv=(TMPMBX,SYSNAM,CMKRNL,WORLD,SYSPRV)/proc=CPU_SERVER -
_$ sys$sysdevice:[bae]cpu_server
%RUN-S-PROC_ID, identification of created process is 2020013B
$ sh sys
OpenVMS I7.2  on node ALLEN  11-AUG-1998 01:50:26.77  Uptime  0 03:50:40
  Pid    Process Name    State  Pri      I/O       CPU        Page flts  Pages
20200101 SWAPPER         HIB     16        0   0 00:00:00.31          0      0
20200106 CLUSTER_SERVER  HIB     15       13   0 00:00:00.25         61     76
...
2020012D _FTA4:          HIB     10    14368   0 00:00:09.46        335     96
20200132 _FTA5:          LEF      8     1601   0 00:00:06.88       1618     70
2020013B CPU_SERVER      HIB      9        4   0 00:00:00.15         71     83
$
```

ICC Security Considerations

By default all users may connect to an association.

- The **`prot`** argument to **`SYS$ICC_OPEN_ASSOC`** can be used to limit access to the server. Values:
  - `0` all users may connect
  - `1` requires that user is in the system, owner, or group category to connect
  - `2` requires that user is in system or owner category to connect
- **`SET SECURITY`** may override these values while the association is open.
- Can create permanent security objects for associations using:

```
@SYS$MANAGER:ICC$CREATE_SECURITY_OBJECT
```

- This command procedure may be used to:
  - Restrict client access to a permanent security object (in this case *node::association*).
  - Open up server access to a permanent security object for a user.
  - The OPEN security attribute allows the creation of an association.
  - The ACCESS security attribute allows connection to an association.
  - Permanent security objects go away when the system reboots so you can use **`ICC$SYSTARTUP.COM`**, called by **`ICC$STARTUP.COM`** to recreate the objects at boot time. **`ICC$SYSTARTUP.COM`** contains additional examples of setting up security controls.

ICC Security Example

- User BAE does not have SYSNAM privilege and cannot run the VERY_SIMPLE_SERVER program that is attempting to create the VERY_SIMPLE_SERVER association.

```
$ set host 0
 Welcome to OpenVMS (TM) Alpha Operating System, Version I7.2
Username: bae
Password:
   Welcome to OpenVMS (TM) Alpha Operating System, Version I7.2 on node ALLEN
    Last interactive login on Tuesday, 11-AUG-1998 11:30:04.49
    Last non-interactive login on Tuesday, 11-AUG-1998 11:49:44.92
$ run very_simple_server
Setting up association
%SYSTEM-F-NOPRIV, insufficient privilege or object protection violation
```

- User BAE however does have SYSPRV privilege.

```
$ sh proc/priv

 12-AUG-1998 15:44:07.16   User: BAE            Process ID:   20200129
                           Node: ALLEN          Process name: "BAE"


 Authorized privileges:
  NETMBX       OPER        SETPRV      SYSPRV      TMPMBX


 Process privileges:
  NETMBX                 may create network device
  OPER                   may perform operator functions
  SETPRV                 may set any privilege bit
  SYSPRV                 may access objects via system protection
  TMPMBX                 may create temporary mailbox
```

ICC Security Example (continued)

```
Process rights:
 INTERACTIVE
 REMOTE
 VMS$MEM_RESIDENT_USER


System rights:
 SYS$NODE_ALLEN
$
```

- A permanent security object is set up for BAE to have open/access rights on the association VERY_SIMPLE_SERVER RUNNING on node ALLEN.

```
$ @sys$manager:icc$create_security_object allen::very_simple_server -
_$ /acl=(id=bae,access=open+access)
```

- Now BAE can create the association only from node ALLEN.

```
$ r very_simple_server
Setting up association
Waiting to accept connect request.
 Interrupt


$
```

- Viewing the security object for ALLEN::VERY_SIMPLE_SERVER.

```
$ sh secu/object=icc allen::very_simple_server

ALLEN::VERY_SIMPLE_SERVER object of class ICC_ASSOCIATION
     Owner: [BUNDER,BAE]
     Protection: (System: AO, Owner: AO, Group: A, World: A)
     Access Control List:
           (IDENTIFIER=[BUNDER,BAE],ACCESS=ACCESS+OPEN)
$
```

ICC Security Example (continued)

- Access is still denied for user BAE on node VEDDER. Another ACL could be set up using `ICC$CREATE_SECURITY_OBJECT.COM`. The node designation `ICC$::` could be used to grant cluster-wide access.

```
$ set host vedder


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


Username: bae
Password:
   Welcome to OpenVMS (TM) Alpha Operating System, Version I7.2 on node VEDDER
    Last interactive login on Wednesday, 12-AUG-1998 15:41:55.94
    Last non-interactive login on Tuesday, 11-AUG-1998 11:49:44.92
$
$ r very_simple_server
Setting up association
%SYSTEM-F-NOPRIV, insufficient privilege or obje           n violation
$
```