CTOS COURSE

NAGW-1333

by

D. R. Lefebvre, L. Page and J. R. Noseworthy

Rensselaer Polytechnic Institute
Electrical, Computer, and Systems Engineering Department
Troy, New York 12180-3590

November 1992

CIRSSE REPORT #128

CTOS Course – 20 Nov'92

Instructors: Don Lefebvre
Lance Page
Russ Noseworthy

CIRSSE

CTOS Course Objectives

After course should be able to:

- design distributed applications
- code event handler functions
- build CTOS configuration files
- startup/shutdown CTOS applications
- use CVCS software management tools
- know where to seek additional information

Course does not cover:

- details of CTOS implementation
- using MCS, VSS or PN Controllers

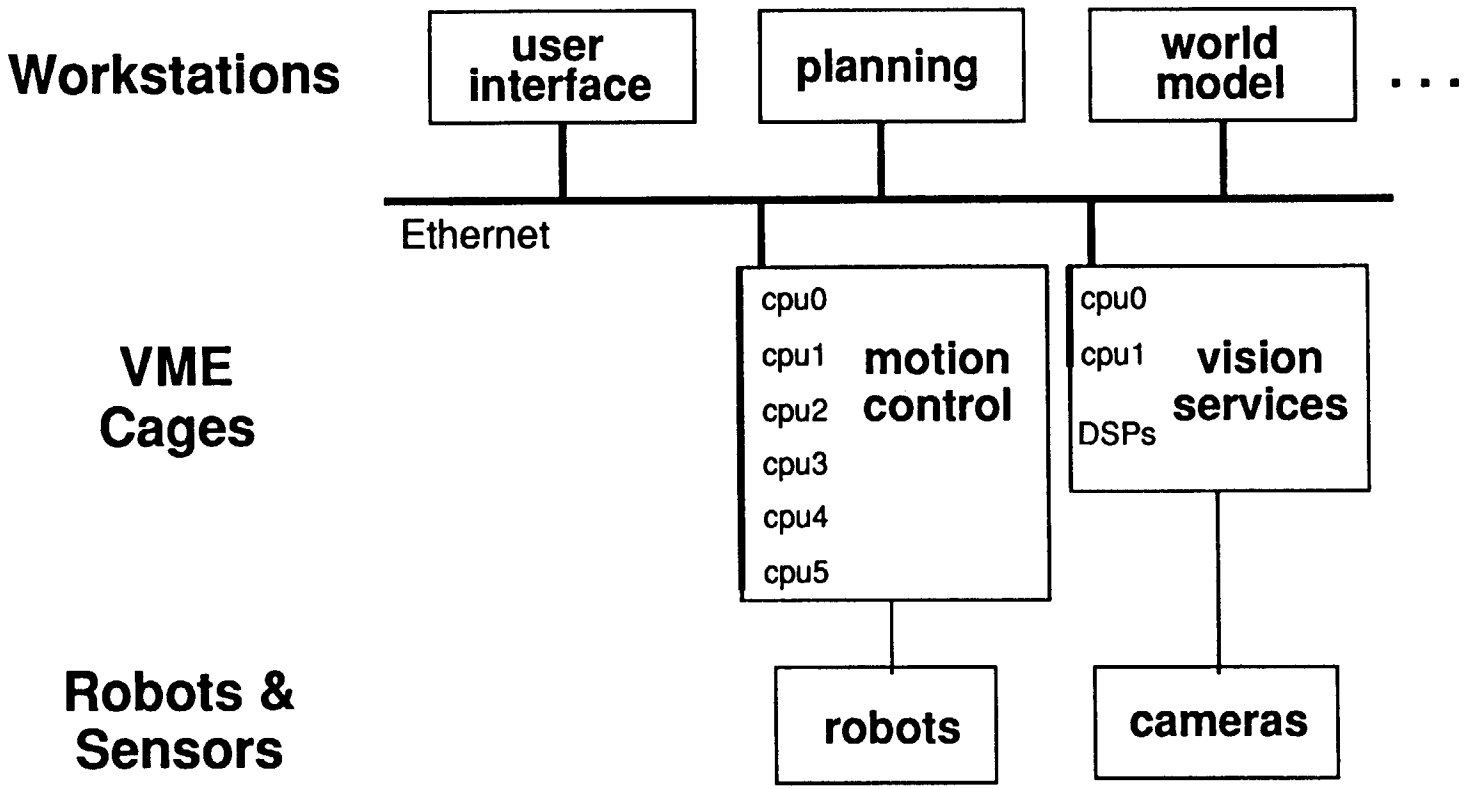
CTOS Course Schedule

9:00	CTOS Overview	30 min.
9:30	CTOS Tasks & Applications	30
10:00	Building & Sending CTOS Messages	40
10:40	BREAK	20
11:00	Message Data Management	20
11:20	X-window I/O	20
11:40	CTOS Configuration	30
12:10	LUNCH	50
1:00	CTOS Startup	20
1:20	Advanced Tools & Services	40
2:00	MCS, VSS, PN Controllers	20
2:20	Software Management Tools	30
2:50	BREAK (move to lab)	10
3:00	Exercises	2 hrs
5:00	DONE	

CIRSSE Testbed Operating System (CTOS) Overview

CIRSSE

Goal: to Build Distributed Applications



Common Characteristics of Complex Robotic Systems

- **Distributed** — multiple loosely-coupled processors, possible widely separated spatially
- **Heterogeneous** — various processor architectures, operating systems, communication protocols
- **Wide range of time scales** — 5 ms control loops, to aperiodic runtime planning, to human time-scale interactions

Reasons For Distributed Systems

- Computing power by dividing workload
- Reliability/fault tolerance via redundancy
- Physical system is spatially distributed
- Access to specialized hardware

Distributed Real-Time Issues

- Coordinating independent tasks on multiple computers
- Synchronizing startup, execution, and shutdown
- Communicating between diverse operating systems & architectures
- Sharing data between tasks
- Supporting real-time execution of control tasks
- Managing multiple versions of software
- Acquiring high level of developer expertise

Principal CTOS Services

- Data-driven distribution of tasks
- Synchronized startup of the application
- High-freq. periodic execution of real-time tasks
- Message-based communication between tasks
- Requester-server data communication protocol
- Centralized X window-based user interface

Definitions

Chassis: UNIX workstation or VME cage, can contain multiple CPUs

CPU: an individual processor

Process: general term for independent thread of execution

Task: process with CTOS resources (message queue, etc.)

Task Function: user-written part of CTOS task that processes messages (a.k.a. event handler function)

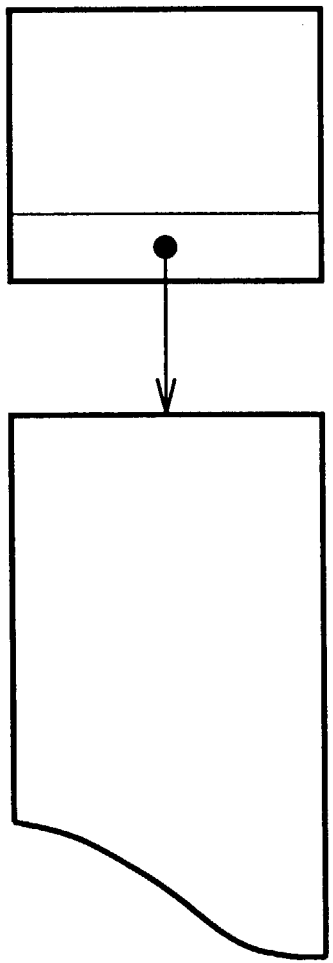
Message: small, fixed-size data structure sent between tasks

Message Data: variable-size block of data accompanying message

Application: collection of communicating CTOS tasks on one or more computers

CTOS Messages

Message

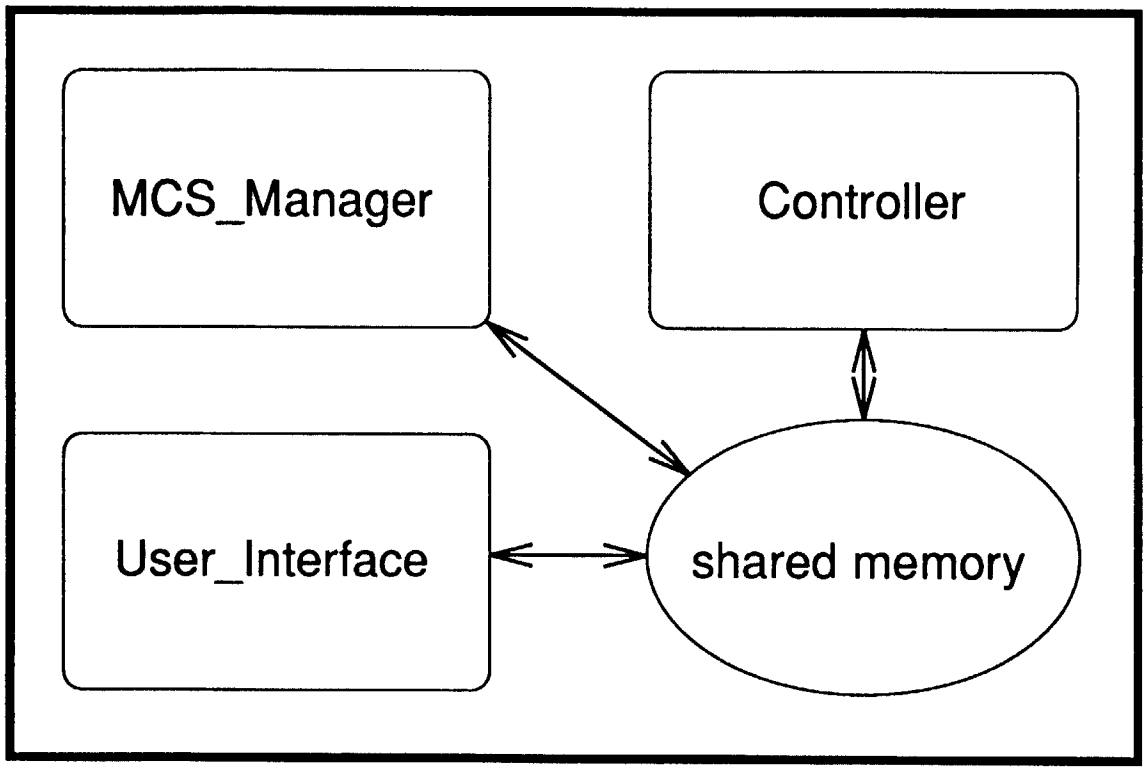


fixed-size structure containing
routing information plus data pointer

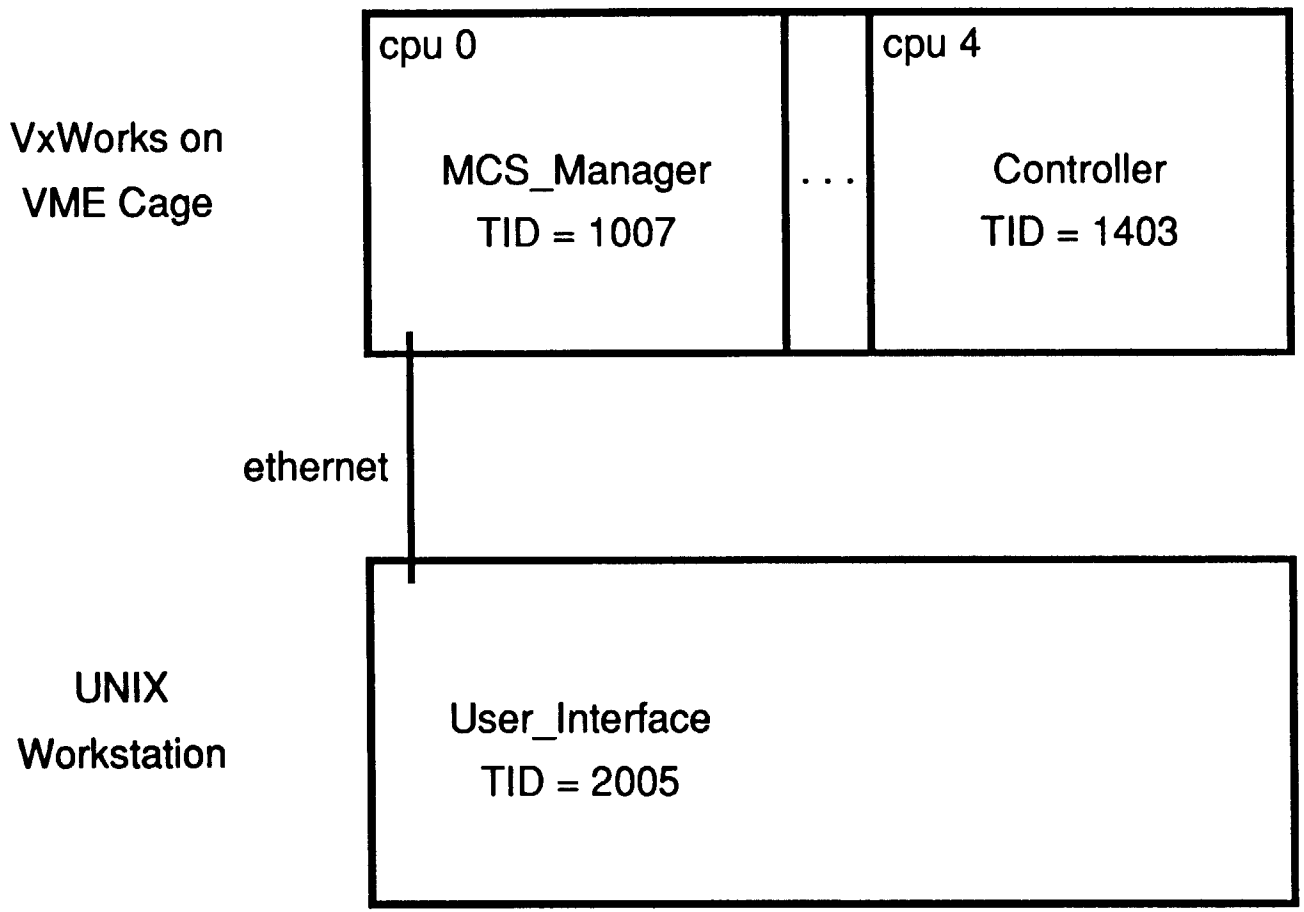
Message Data

application-specific data block
of any size

Simplified Teleoperation Application



Tasks are Connected by Symbolic Names



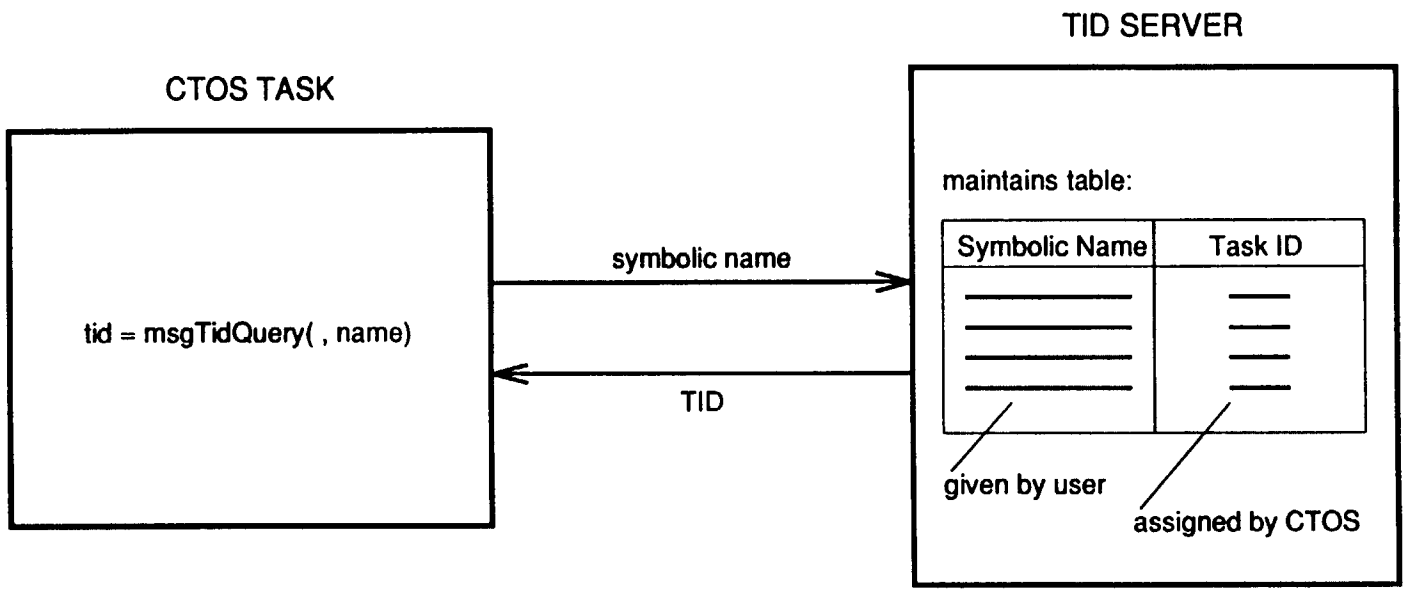
CTOS Application Configuration File

```
%  Config file for Robot Teach Pendant

%  identify which computers are used
CHASSIS   vx0   1
CHASSIS   earth
SEQUENCER earth

%  start task on workstation
PREFIX  earth  0
TASK  User_Interface  /home/username/mydir/progif

%  start controller tasks
PREFIX  vx0
CHDIR  /home/username/mydir/
0  LOAD  mcsSupportLib.o
0  TASK  MCS_Manager  mcsMsgHandler  100
4  LOAD  mcsControlLib.o
4  TASK  Controller   mcsPIDcntrl    100
```

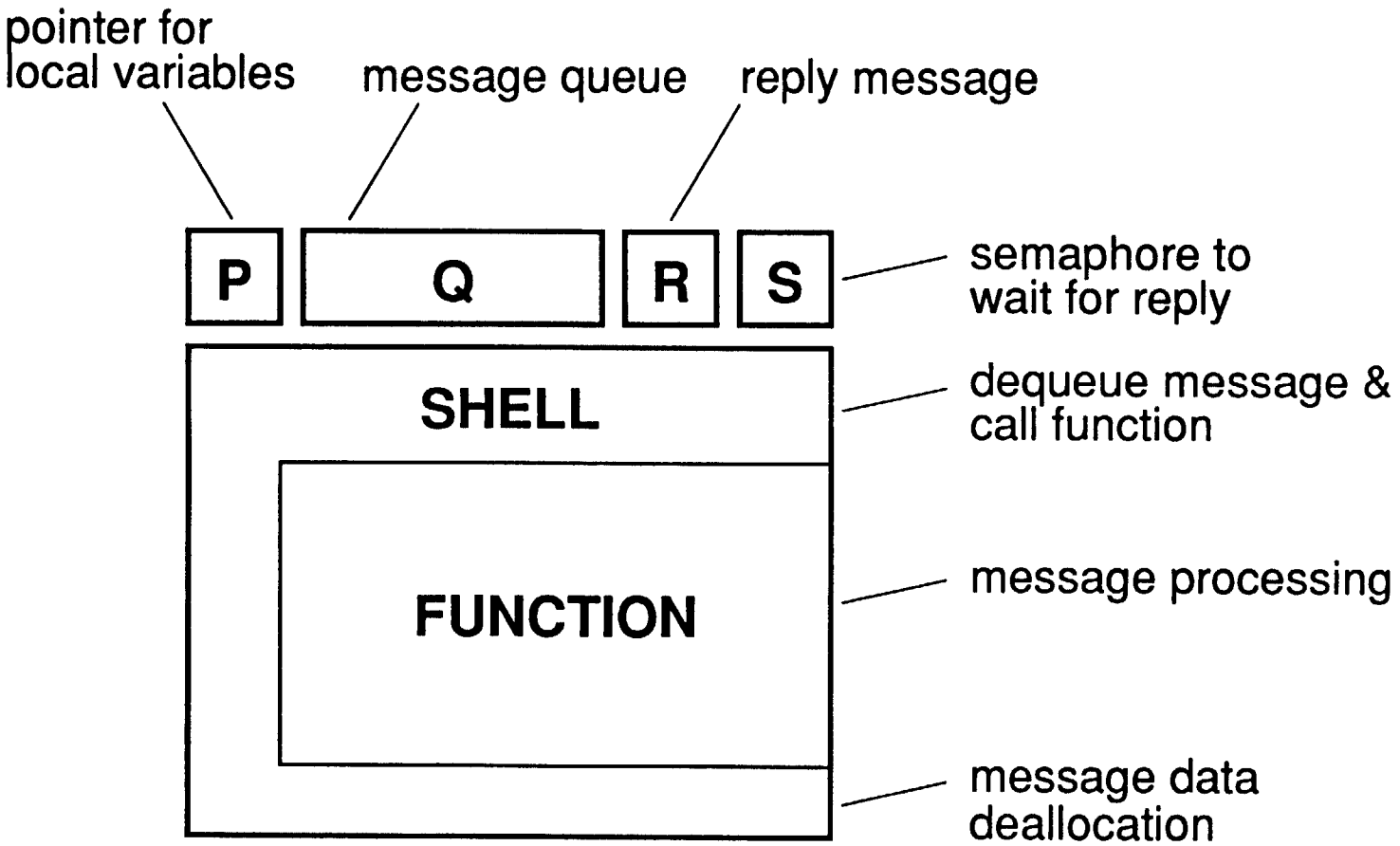
CTOS Tasks

References: Tech Memo 5:1
Man pages: msgLib

Task Identification



Components of a CTOS Task



Format of CTOS Task Function

```
#include <ctos.h>                              /* include CTOS services          */

CtosTask(myFunctionName)                       /* required for UNIX CTOS shell   */

int  myFunctionName (TID_TYPE myTid, MSG_TYPE *msg)    /* standard prototype */
{
    switch (msg->command)                      /* determine requested processing */
    {
    case MSG_AINIT:                            /* do application initialization  */
        break;

    case MSG_AEXEC:                            /* begin application execution    */
        break;

    case MSG_MYMSG:                            /* process appl-specific message  */
        break;
    }
    return (msgDefaultProc(myTid, msg));       /* default message processing     */
}
```

Default Message Processing

- `msgDefaultProc()` provides default message processing
  - acknowledges standard CTOS messages
  - generates default reply if message sender is still waiting
- Typically end CTOS task function with:
  `return(msgDefaultProc(myTid, msg));`
- Ending `case` statement with `break` passes current message to default processing
- Ending `case` statement with `return(0)` bypasses default message processing

It is recommended that all CTOS task functions send all messages to default processing.

Example CTOS Task Function

```
 1  #include <ctos.h>
 2  #include <recLib.h>
 3  #ifdef OS_UNIX
 4  #  include <stdio.h>
 5  #  include <string.h>
 6  #endif
 7  #ifdef OS_VXWORKS
 8  #  include <stdioLib.h>
 9  #  include <strLib.h>
10  #endif
11
12  CtosTask(myFunctionName)
13
14  #define  MSG_HELLO          (MSG_USER+100)
15  #define  MSG_WHOAREYOU      (MSG_USER+101)
16
17  int  myFunctionName (TID_TYPE myTid, MSG_TYPE *msg)
18  {
19      static TID_TYPE  partnerTid;
20      MSG_TYPE         myMsg;
21      char            *greeting, phrase[]="Hello, partner!";
22      char            *answer;
23
```

```
24      switch (msg->command)
25      {
26      /*  During initialization, find TID of task named     */
27      /*  'CommPartner'                                     */
28      /*                                                    */
29      case MSG_AINIT:
30          partnerTid = msgTidQuery(myTid, "CommPartner");
31          break;
32
33      /*  When application is ready to execute, send a      */
34      /*  message to 'CommPartner', and display its reply.  */
35      /*                                                    */
36      case MSG_AEXEC:
37          /*  build the message  */
38          greeting = malloc(strlen(phrase)+1);
39          strcpy(greeting, phrase);
40          msgBuild(&myMsg,                /* message storage  */
41                   partnerTid,            /* destination task */
42                   myTid,                 /* source task      */
43                   MSG_HELLO,             /* message command  */
44                   (void *)greeting,      /* message data     */
45                   strlen(greeting)+1,    /* size of data     */
46                   MF_REPLYWAIT);         /* request a reply  */
47
48          /*  send the message and then display reply  */
49          answer = (char *)msgSend(&myMsg);
50          recInfo(myTid, "%s\n", answer);
51          break;
```

```
52
53      /*  Reply to user-defined message with my TID.          */
54      /*                                                      */
55      case MSG_WHOAREYOU:
56          recInfo(myTid, "Received '%s' from task %x\n",
57                  (char *)msg->data, msg->source);
58          msgReply(msg, (void *)myTid, MS_NONE, MF_STANDARD);
59          return(0);
60      }
61
62      /*  send all other messages to default processing       */
63      /*                                                      */
64      return (msgDefaultProc(myTid, msg));
65  }
```

Types of CTOS Tasks

- Server Task
  - waits for request message, then performs service
  - results returned in reply message
  - most common type, e.g. TID Server
- Monitor Task
  - self-triggered periodic processing & service messages
  - service msg / periodic processing / sleep / send MSG_MONITOR to self
  - alternatively use `msgQueueCount()` to check for CTOS msgs
- Hybrid Event-Handler
  - monitor CTOS msg queue plus other event queues
  - can respond to X-events or TelRIP objects
  - CTOS task spawns child process to handle events

Types of CTOS Tasks, Con't

- Interface Task
  - custom interface to non-CTOS task
  - typically use shared memory or pipes
  - commonly used as managers for "synchronous tasks" on VME chassis
- Application Executive
  - generates sequence of requests to servers
  - may be driven by user input
  - usually application-specific

CTOS Applications

References: Tech Memo 5:2
Man pages: ctos_boot_phases

Application Synchronization

- CTOS applications are composed of many independent system-level and user-developed tasks that may be distributed across several UNIX workstations and VME chassis
- To coordinate startup, operation, and shutdown CTOS enforces several *phases of execution*
  - all tasks transition between phases in unison
  - guaranteed conditions exist at start of each phase
- Start of execution phase signaled by CTOS broadcast message
  - execution phase in effect until all tasks have acknowledged completion
  - **all CTOS tasks must acknowledge all application phase messages, except** `MSG_AEXEC`
  - correct processing is obtained if all messages are passed to `msgDefaultProc()`

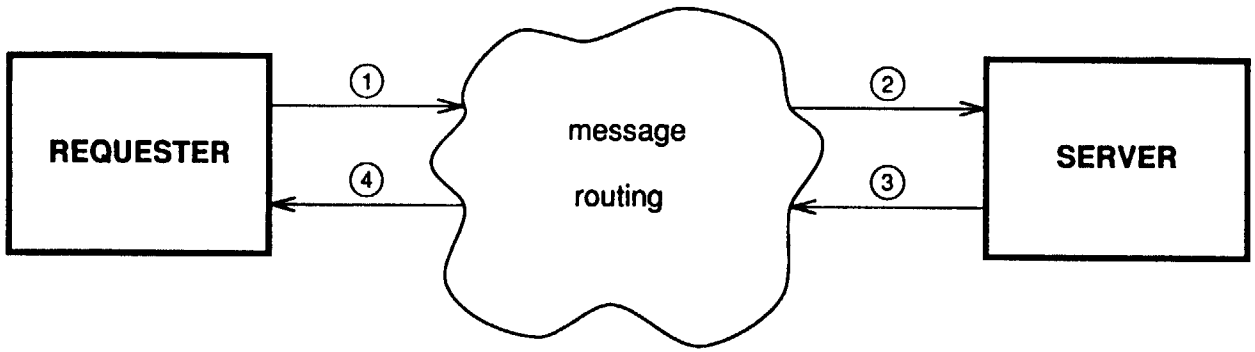
Purpose & Conditions of Application Phases

- Process Initialization (PINIT)
  - all CTOS services guaranteed to exist, but states of other tasks are unknown
  - used to initialize individual tasks, e.g. read data files or set up data structures
- Application Initialization (AINIT)
  - all tasks guaranteed to have completed process initialization
  - used to perform inter-task initializations, e.g. register with servers
- Application Execution (AEXEC)
  - all tasks have completed initialization and are ready for normal operation
  - most tasks are servers and so will not respond to AEXEC
  - Application Executive task takes control at start of AEXEC

Application Phases, Con't

- Application Termination (ATERM)
  - inverse of AINIT phase
  - used to release shared inter-task resources
- Process Termination (PTERM)
  - inverse of PINIT phase
  - used to free remaining resources owned by the task
  - VME cage reboots when `MSG_PTERM` message is broadcast
- CTOS system uses two additional phases
  - CINIT (prior to PINIT) used to initialize CTOS servers
  - CTERM (following PTERM) used to reset CTOS system
- User may implement additional phases via REPLY-WAIT broadcast (see tech memo #5)

Requester-Server Communication



1. Requester sends message and then blocks
2. Message is routed to Server's message queue
3. Server dequeues msg, performs service, and sends reply
4. Receipt of reply unblocks Requester, reply data delivered as return value of `msgSend()`

Requester-Server Architecture

- Most common flow control for CTOS applications
- Most tasks are servers that wait for request messages
- Request messages may be generated by several sources:
  - user interaction
  - hardware event
  - periodic processes
  - system management
  - application control program (application executive)
- Application can be designed in coarse-grain object-based manner
  - task encapsulates all of the object's data and methods
  - protocol for communicating with Server can be encapsulated in RPC (Remote Procedure Call)

Development Steps

- Identify major operations and data flows
  - use standard software engineering techniques
  - consider an object-based approach
  - identify standard servers that may be reused
- Group operations into tasks
  - define initialization requirements and the phases where they should occur
  - logically group family of related operations into one task, e.g. all methods operating on an object go into same task
  - concurrent operations should be in separate tasks
  - consider single manager task for operations that must be serialized, e.g. to control access to a resource
  - consider reentrancy issues if multiple tasks are to share the same event handler function
  - assign a unique symbolic name to each task

Development Steps, Con't

- Describe inter-task communications
  - define messages and data being passed
  - roughly, each message corresponds to a different operation or result
  - identify when replies are required and which task owns message data memory
  - identify communication partners (who sends message and who receives it)
  - draw diagram showing tasks and message exchanges, and check sequences for deadlocks
  - keep high volume communications on same CPU if possible, or at least on same chassis

Development Steps, Con't

- Write event handler functions
  - write interface functions to encapsulate building request messages
  - avoid hardcoding configuration information, such as directory names, by passing data from configuration files via `args` command.
  - build event handlers to interface to synchronous processes
  - design application executive to perform main execution sequence

- Build configuration files
  - assign tasks to chassis and CPUs
  - for dynamically linked VxWorks, dependencies of function calls determine order to load object modules
  - consider prelinking VxWorks modules for each CPU to save loading time

CTOS Message Structure

References: Tech Memo 6:2
Man pages: msgLib, message_commands, message_flags, msgFlag_macros

CIRSSE

Message Structure

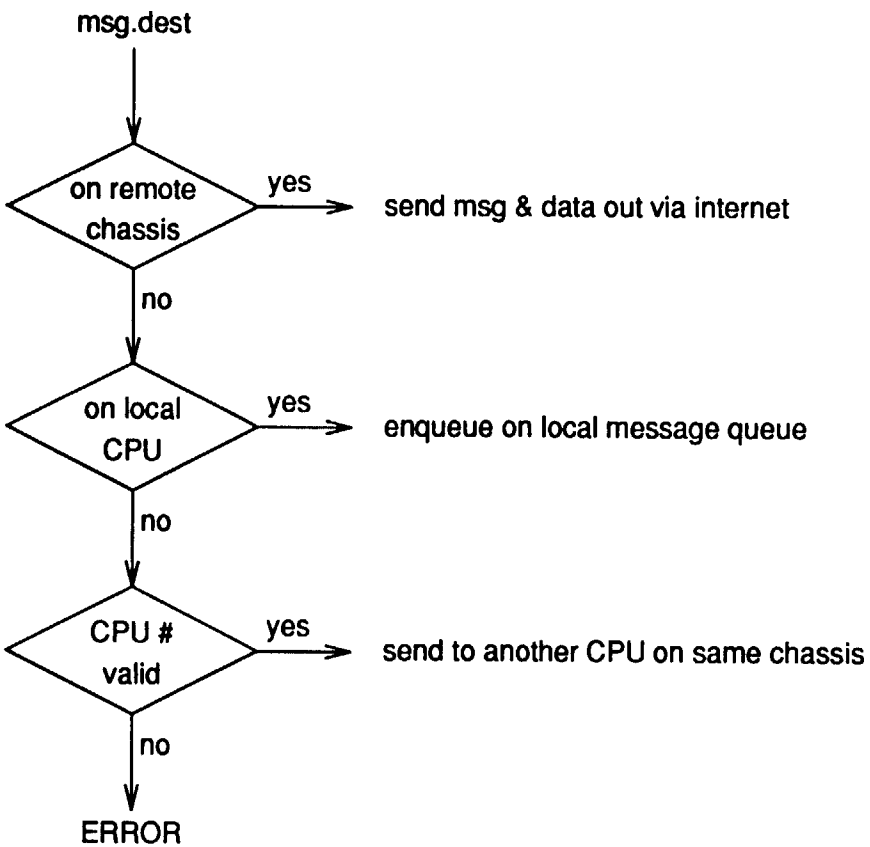
```
typedef struct
{
   TID_TYPE        dest;
   TID_TYPE        source;
   CMD_TYPE        command;
   void           *data;
   int             datasize;
   FLAG_TYPE       flags;
}
MSG_TYPE;
```

dest	TID of destination (receiving) task
source	TID of source (sending) task
command	indicates function of message
data	points to additional message data
datasize	byte length of additional data
flags	specifies message handling options

Destination TID used for Message Routing

TID = Chassis # + CPU # + Local Task #

up to 16 chassis	up to 16 CPUs	up to 256 local tasks



Message Commands

- msg.command used to indicate function of message
  - `CMD_TYPE` is 2-byte unsigned int → over 65,000 cmds
  - usually equate to a predefined constant
- Message command conventions
  - upper case names begining with `MSG_`
  - subapplications added prefix, e.g. `MSG_MCS_`
  - assigned as offset in block of commands
- Standard messages
  - execution phases: `MSG_PINIT`, `MSG_AINIT`, `MSG_AEXEC`, etc.
  - CTOS services: `MSG_QUERY_TID`, `MSG_REC_INFO`, etc.
- User-defined messages
  - assigned as offset to `MSG_USER`, e.g.

```
#define  MSG_MY_BLOCK      MSG_USER
#define  MSG_MY_MESSAGE    (MSG_MY_BLOCK+1)
#define  MSG_ANOTHER_MSG   (MSG_MY_BLOCK+2)
```

Message Data

- msg.data is pointer to a contiguous block of memory of length msg.datasize:

```
MSG_TYPE        myMsg;
static double   value;

myMsg.data      = (void *)&value;
myMsg.datasize = sizeof(double);
```

- Alternatively, 4 bytes of data may be transmitted in msg.data:

```
static int  number;

myMsg.data      = (void *)number;
myMsg.datasize = MS_NONE;
```

- Data may be retrieved from received messages by dereferencing the data pointer:

```
value  = *(double *)msg->data;
number = (int)msg->data;
```

Message Data, Con't

- Format of message data is not identified in the message
  - msg.data is void pointer
  - responsibility of message sender and receiver to define content of message data
- Data structures and arrays may be transmitted if contiguous
  - generally, precludes the inclusion of pointers
  - must package linked list into contiguous block and rebuild list at receiving task
- CTOS provides a mechanism for automatically deallocating message data after use
  - controlled by `MEMOWNER` field of message flag

Message Flags

- TYPE
  - selects message passing mechanism used to deliver message
  - *users should only use MF_TYPE_NORMAL*
- REPLY_WAIT
  - indicates whether sending task wishs to wait for reply
  - effects return value of `msgSend()`
- SEND_WAIT *is used by system only*
- MEMOWNER controls message data deallocation
- PRIORITY
  - effects ordering of messages
  - MF_PRI_NORMAL is sufficient for most messages

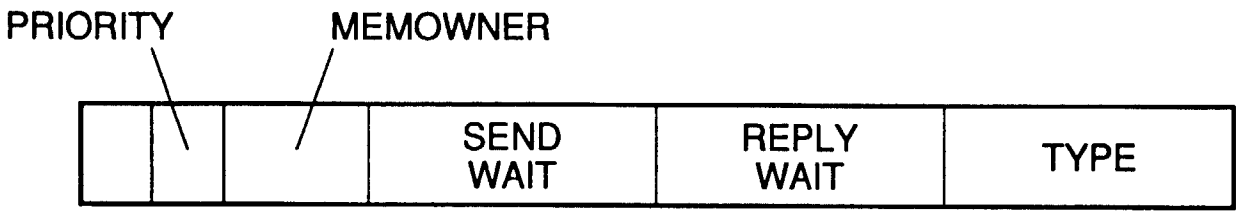
Programmers are encouraged to use predefined message flags rather that defining their own.

Message Flag Values

FIELD	VALUE	ACTION
TYPE	MF_TYPE_NORMAL MF_TYPE_REPLY † MF_TYPE_BC_NORMAL † MF_TYPE_BC_REPLY †	Normal message Reply message Broadcast message Reply to broadcast
REPLY_WAIT	MF_REPLY_WAIT_NO MF_REPLY_WAIT_FOR	No reply expected Block task and wait for a reply
SEND_WAIT †	MF_SEND_WAIT_NO MF_SEND_WAIT_FOR	Fail if cannot send immediately Wait until sent
MEMOWNER	MF_MEMOWNER_RECEIVER MF_MEMOWNER_SENDER MF_MEMOWNER_COPY	Receiving task owns memory Sending task owns memory Receiver owns copy of memory
PRIORITY	MF_PRI_NORMAL MF_PRI_URGENT	Add to back of message queue Add to front of message queue

(† For system use only)

Message Flag Bit Fields, Masks & Macros



FIELD	BITS	MASK	MACRO
TYPE	0-3	MFM_TYPE	msgTypeFlagSet()
REPLY_WAIT	4-7	MFM_REPLY	msgReplyFlagSet()
SEND_WAIT	8-11	MFM_SEND_WAIT	msgSendFlagSet()
MEMOWNER	12-13	MFM_MEMOWNER	msgMemownerFlagSet()
PRIORITY	14	MFM_PRIORITY	msgPriorityFlagSet()
(unused)	15		

Predefined Message Flags

FLAG NAME	ALT. NAME †	REPLY_WAIT	MEMOWNER	PRIORITY
MF_STD_RCVR	MF_STANDARD	NO	RECEIVER	NORMAL
MF_STD_SNDR	MF_MEMKEEP	NO	SENDER	NORMAL
MF_STD_COPY		NO	COPY	NORMAL
MF_WAIT_RCVR	MF_REPLYWAIT	YES	RECEIVER	NORMAL
MF_WAIT_SNDR	MF_KEEPWAIT	YES	SENDER	NORMAL
MF_WAIT_COPY		YES	COPY	NORMAL
MF_SYS_RCVR	MF_SYSTEM	NO	RECEIVER	URGENT
MF_SYS_SNDR	MF_SYSTEMKEEP	NO	SENDER	URGENT
MF_SYS_COPY		NO	COPY	URGENT

(† old names that are being phased out)

Building Messages

Can assign values to members of message structure directly:

```
MSG_TYPE  myMsg;

myMsg.dest     = rcvrTid;
myMsg.source   = myTid;
myMsg.command  = MSG_COMMAND;
myMsg.data     = (void *)&value;
myMsg.datasize = sizeof(value);
myMsg.flags    = MF_WAIT_RCVR;
```

Or, can use a CTOS function:

```
msgBuild(&myMsg,                /* message storage  */
         rcvrTid,               /* destination task */
         myTid,                 /* source task      */
         MSG_COMMAND,           /* message command  */
         (void *)&value,        /* message data     */
         sizeof(value),         /* size of data     */
         MF_WAIT_RCVR);         /* request a reply  */
```

CTOS Message Passing

References: Tech Memo 6:3–:6,:8–:9
Man pages: msgLib, msgSend

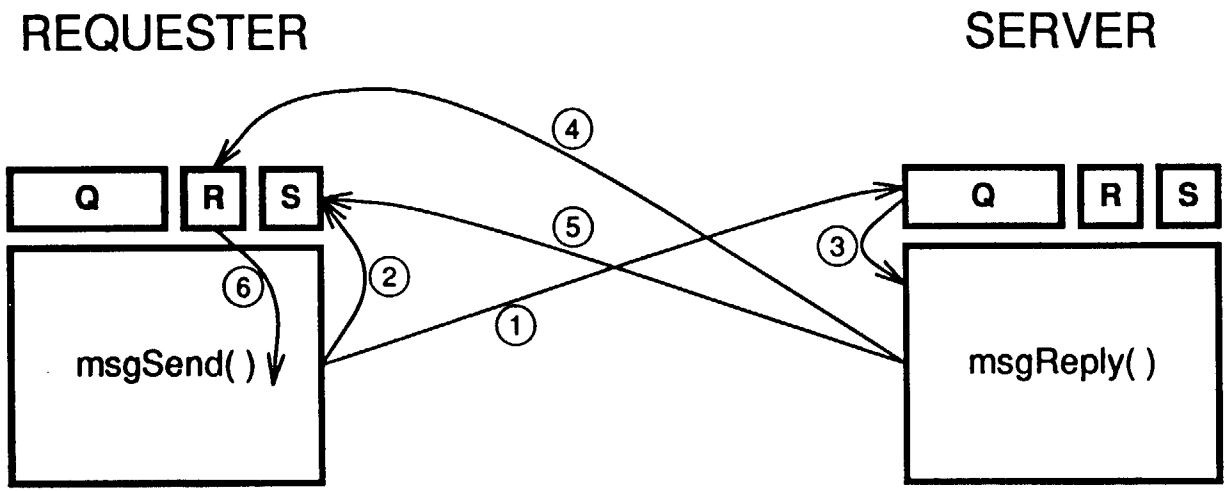
The msgSend() Function

- Normal messages sent via call to the msgSend() function, e.g. `result = msgSend(&myMsg);`
  - pointer to MSG_TYPE message structure is only argument
  - handling options specified in msg.flags
- **msgSend()** performs three functions:
  - message routing
  - blocking for replies
  - message data management
- MF_REPLY_WAIT_NO flag specifies "one-way" message
  - **msgSend()** returns immediately
  - return value of **OK** or **ERROR** is status of sending msg out
- MF_REPLY_WAIT_FOR flag specifies task is to wait for reply
  - task calling **msgSend()** blocks after sending out message
  - arrival of reply message unblocks sending task
  - return value is pointer to reply message data

The msgReply() Function

- msgReply() is used to reply to a previously received message, e.g. `status = msgReply(msg, reply_data, datasize, memowner);`
  - input arguments specify message originator and reply data
  - return value of `OK` or `ERROR` is status of sending reply out
- Reply message data follows same rules as normal message data
  - data must be contiguous
  - when only acknowledgement is needed specify `NULL` data and `MS_NONE` datasize
- Reply data received as return value of original `msgSend()` call
  - dereference pointer to access reply data
  - use `msgReplyGet()` to access reply MSG_TYPE structure
- CTOS protects against incorrect replies
  - `msgReply()` returns `ERROR` if no reply was expected.
  - `msgDefaultProc()` will generate "missing" reply

Reply-Wait Mechanism



① Send message

② Block sending task

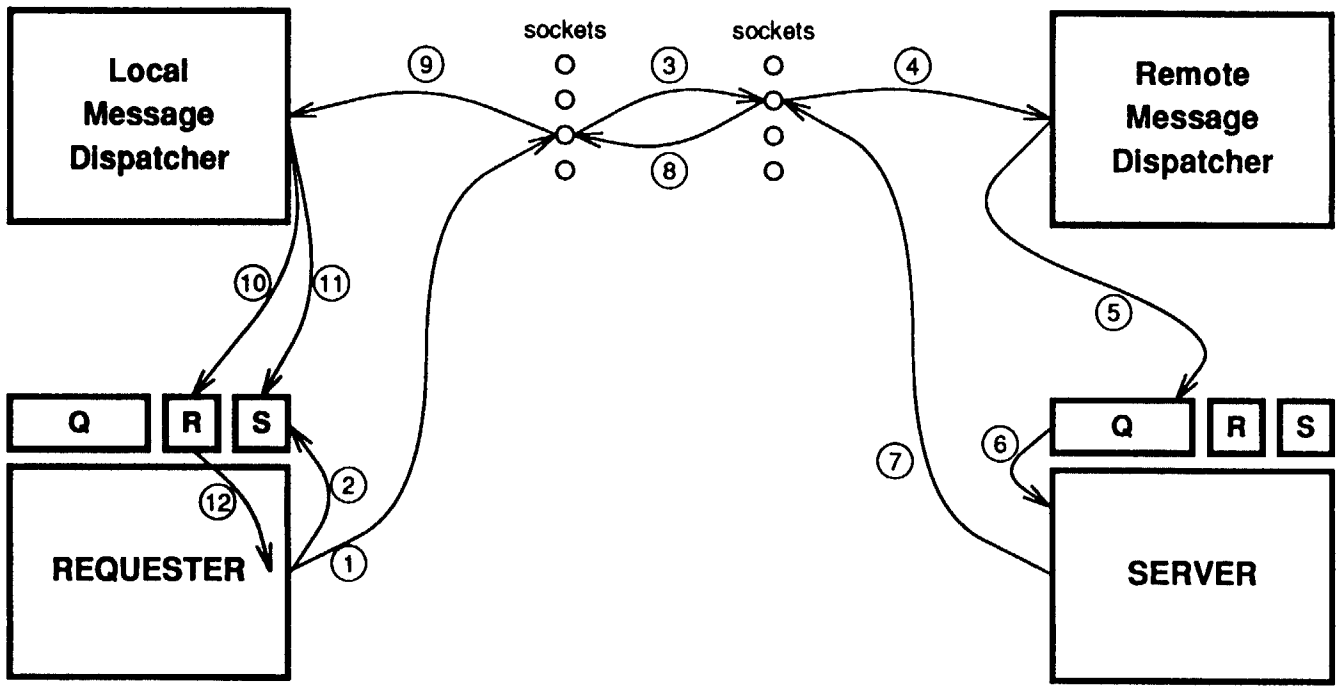
③ Receive message

④ Send reply

⑤ Unblock sending task

⑥ Return reply data

Reply-Wait Routing for Remote Tasks



Uses of Reply-Wait Mechanism

- Task Synchronization
  - can use reply-wait messages when sequence of task executions must be controlled
  - overcomes limitations of semaphores for distributed applications
  - application-wide synchronization can be accomplished via broadcasting

- Encapsulating requester-server exchanges
  - can write function that handles all communications with Server
  - only need to input request data
  - function returns results from Server
  - reduces errors in message construction
  - allows change to requester-server protocol without change to function argument list
  - similar to Remote Procedure Call (RPC)

Example RPC Function

```
int  myRPC (TID_TYPE myTid, double some_data)
{
    static TID_TYPE  svrTid = 0;

    /*  server TID initialization  */
    if ((svrTid == 0) &&
        ((svrTid = msgTidQuery(myTid,"Server_name")) == 0))
        return (ERROR);

    /*  send request message & return result */
    return (msgBuildSend(svrTid, myTid, MSG_SERVICE_THIS,
                         (void *)&some_data, sizeof(double),
                         MF_WAIT_SNDR));
}
```

The msgBroadcast() Function

- `msgBroadcast()` sends the same message to all tasks within the scope of the broadcast, e.g.
  `nzAcks = msgBroadcast(&myMsg, scope);`
  - first argument points to message to be broadcast
  - scope can be `MB_LOCAL`, `MB_CHASSIS`, or `MB_APPLICATION`
  - return value depends on MF_REPLY_WAIT flag

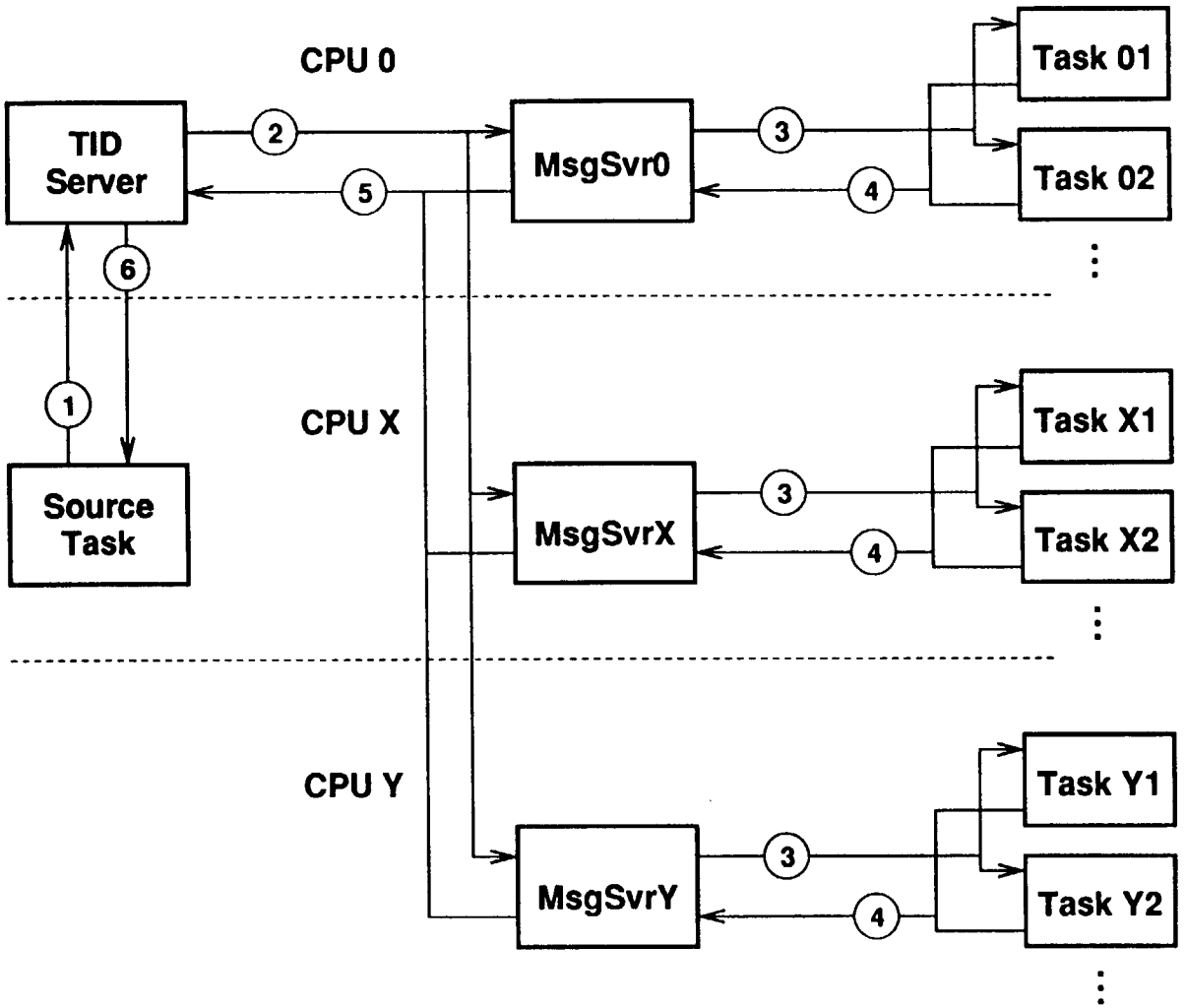
- MF_REPLY_WAIT_NO flag specifies "one-way" message
  - `msgBroadcast()` returns immediately
  - return value of `OK` or `ERROR` is status of sending out request to broadcast

- MF_REPLY_WAIT_FOR flag specifies task is to wait for reply from all tasks in scope of broadcast
  - task calling `msgBroadcast()` blocks after sending out request
  - reply messages are counted by Message Servers
  - return value is number of non-zero acknowledgements

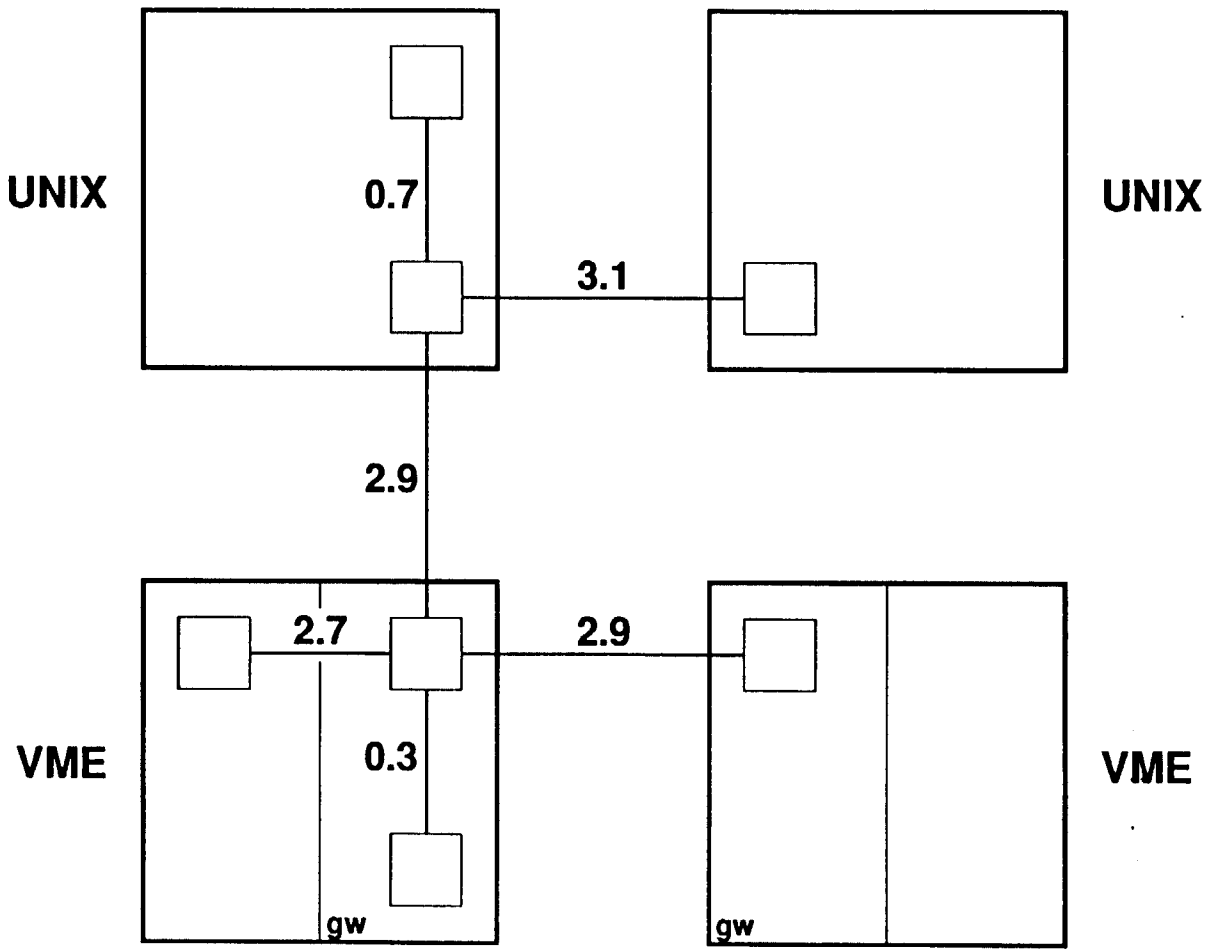
The msgBroadcast() Function, Con't

- Broadcast originator does not receive the message
  - if needed, **msgSend()** to self before **msgBroadcast()**
  - never send message with MF_REPLY_WAIT_FOR to self since will deadlock!
- Sending task must own data accompanying broadcast message
  - receiving tasks share copy of broadcast data
  - MF_MEMOWNER_RECEIVER is coerced to _SENDER
- All tasks must respond to broadcast-with-reply-wait
  - **msgAcknowledge(msg)** is simple way to zero-ack
  - pass this message to **msgDefaultProc()** for zero-ack
  - must use **msgReply()** for non-zero acknowledgement
- Can not reply with data to a broadcast message
  - send a normal message to the broadcast originator
  - nzack returned by **msgBroadcast()** tells how many to expect

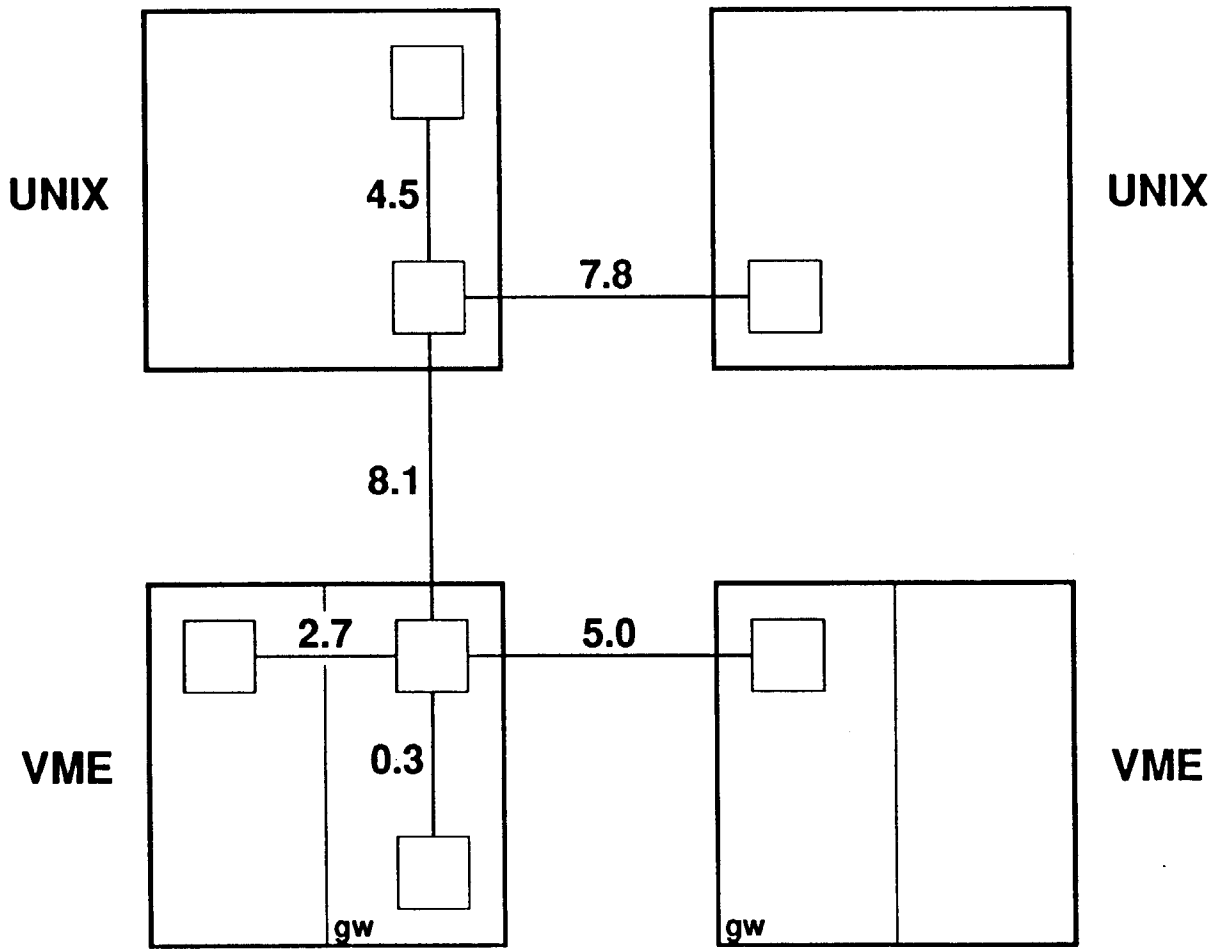
Routing of Broadcast Message
(VME Chassis)



Latency of Normal Messages With No Data (ms)



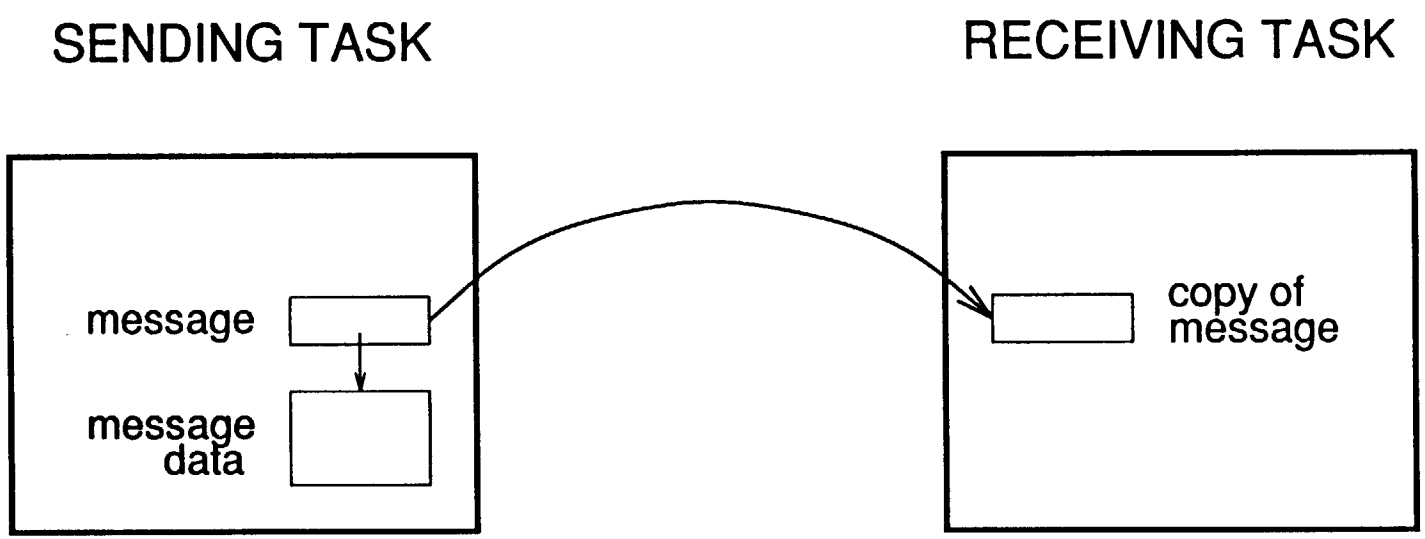
Latency of Normal Messages With 1000 Bytes of Data (ms)



Message Data Management

References: Tech Memo 6:7
Man pages: msgLib

When Can Message Data Be Deallocated?



When has receiver finished with data ?

Was data dynamically deallocated ?

How to free data not in local address space ?

CTOS Message Data Management

- CTOS provides automatic deallocation as programming convenience so user can avoid explicit management of message data
  - handling of message data is functionally the same across CPUs and chassis
- Message sender specifies "owner of data" via message flags
  - MEMOWNER_SENDER: sending task manages data
  - MEMOWNER_RECEIVER: receiving task manages data
  - MEMOWNER_COPY: data copied, receiver manages
- Must consider *persistence* of data
  - must dynamically allocate data for MEMOWNER_RECEIVER
  - statically declared variable OK for MEMOWNER_SENDER, but usually don't want to use local (stack) variables
  - local variables are OK if both MEMOWNER_SENDER and REPLY_WAIT_FOR are set
  - any variable storage type OK for MEMOWNER_COPY

Task's Perspective of Message Data

MEMOWNER	SENDING TASK	RECEIVING TASK
SENDER	No deallocation by CTOS	Data is READ_ONLY
RECEIVER	Unavailable after msgSend()	Freed when exit task function
COPY	No deallocation by CTOS	Freed when exit task function

Message Data Management Functions

- msgDataKeep( )
  - prevents deallocation of message data
  - returns new pointer to message data
  - receiving task responsible for later deallocation
- msgDataCopy( )
  - makes local copy of message data
  - returns pointer to newly allocated storage
  - does not copy the message itself
- msgCopy( )
  - makes local copy of message
  - returns pointer to newly allocated storage
  - does not copy message data, so msg->data pointer wrong
  - to copy message and data:

```
static MSG_TYPE  *mymsg;
mymsg        = msgCopy(msg);
mymsg->data = msgDataCopy(msg);
```

Reply Message Data

- Reply messages are handled differently than normal messages
  - can store only one reply message at a time (no queue)
  - only one normal message per call of task function
  - may receive multiple replies in each task function call
- Reply message data is deallocated when:
  - call to msgSend( ) will generate a reply (REPLY_WAIT set)
  - and, previous reply message had MEMOWNER_RECEIVER or _COPY
- Therefore, it is recommended that return values from reply messages be saved immediately upon receipt
- RPC functions can cause reply data to be lost
  - RPCs send messages to servers and receive data in replies
  - as a result, previous replies may be deallocated
  - use msgReplyPush( ) and msgReplyPop( ) in your RPC functions to prevent this problem

Message Data Deallocation on VME Chassis

- VME chassis message data deallocation is functionally same as on UNIX chassis
- Message data deallocation may be turned on/off on VME chassis
  - presently, deallocation is off at startup
  - to turn on, add command to application configuration file:

```
chas_args   vx0   MSG_DATA_DEALLOCATION = on
```

CTOS X-Window I/O

Man pages: recLib, recSvr

recLib — 'Recording library'

Major components:

- recWindow: A text I/O X-Window for the whole application.
- recSvr: The CTOS task which controls the recWindow.
- recLib: The programmer's interface to the recSvr.

Text output:

```
STATUS recInfo    (TID_TYPE tid, const char fmt[], ...);
STATUS recWarning (TID_TYPE tid, const char fmt[], ...);
STATUS recError   (TID_TYPE tid, const char fmt[], ...);
```

Identical actions, but different semantics in the source code. These prepend the specified string, with the calling task's tid and name.

recLib— keyboard-input functions

Blocking input:

```
char *recPrompt (TID_TYPE tid, char *inbuf, const char fmt[], ..
```

Blocks until the user enters a string, and fills `inbuf` with the string entered by the user.

`fmt, ...` describe the prompt-line in printf() style.

Should not ignore return value– NULL indicates error.

Non-blocking input request:

```
char *recPromptPost (TID_TYPE tid, REC_POST_FLAGS flags,
                     CMD_TYPE cmdReply, const char fmt[], ...);
```

Requests the recSvr to send this task a message, when user enters a string. Each prompt-line has a unique (`tid`, `cmdReply`) pair.

Flag values: `REC_POST_UNBLOCKED` (basic), `REC_POST_CONTINUOUS`, `REC_POST_CANCEL`, or `REC_POST_ECHO`.

Configuring the recWindow

Selecting the display:

- Default (using app_win) is the console from which app_win was run.
- Specify display by sending a task-arg of "`DISPLAY = ...`" to the recSvr.

Select other options using task-arg of "`REC_OPTIONS = ...`".

Example:

```
prefix SequHost 0
args recSvr DISPLAY = mars:0.0
args recSvr REC_OPTIONS = -name farmAnimals -geom 600x500
args recSvr REC_OPTIONS += -rv -iconic -fn 10x20
```

CTOS Configuration

References: Tech Memo 16:2&:6
Man pages: ctos_config

CIRSSE

Types of config-files

Application config-file:

One per application— At minimum, specifies which chassis are in the application and the names of the chassis config-files.

Chassis config-files:

Optionally, the CPU-specific commands for a chassis may be placed in a file separate from the application config-file.

System config-files:

Chassis-level config-files which are *not* application-dependent. Established at installation time.

Application-wide Config-file commands

`chassis <chassis> [chassis-num] [chassis-cfg-file-name]`

`chassis-num` only needed for vxWorks chassis.

`sequencer <chassis>`

One per application; must be UNIX for multi-chassis.

`chas_args <chassis> <string...>`

Special bootstrap options. Rarely used.

`prefix [chassis [CPU]]`

Affects the CPU-specific commands which follow it.

CPU-specific Config-file commands

Format for all CPU-specific commands:

```
<chassis> <cpu-num> <command-name> [args...]
```

Key words for `<chassis>`:

`all` — All chassis.

`SequHost` — The sequencer chassis.

Special CPU value:

`-1` — All CPUs.

The 'task' command

UNIX tasks:

`<chassis> <CPU> task <symbolic-name> <program-name> [bts-mode]`

`<chassis> <CPU> systask <symbolic-name> <program-name> [bts-mode]`

Ignores `PTERM` phase.

VxWorks tasks:

`<chassis> <CPU> load <obj-file>`

Dynamically links the object-code.

`<chassis> <CPU> task <symbolic-name> <function-name> <priority>`

Uses a previously `loaded` event-handler function.

More CPU-specific Config-file commands

`<chassis> <CPU> chdir <new-directory>`

`<chassis> <CPU> include <cfg-file-name>`

`<chassis> <CPU> args <symbolic-name> <string...>`

Send a `MSG_TASK_ARGS` message to the task, before `PINIT`.

`<vxworks-chassis> <CPU> echo <string...>`

`<vxworks-chassis> <CPU> share <obj-file> <hex-address>`

Shared memory— all CPUs load the `obj-file` in the same physical memory. Recommend CPU = -1, and `hex-address` = 0x0.

`<vxworks-chassis> <CPU> logo <logo-file-name>`

`<vxworks-chassis> <CPU> connect <hostname> <CPU_num>`

For the system config-file only– connects the CPUs together.

Hints for Distributing Tasks

1. Heavy communication should be *local*.

2. Heavy CPU-loads should be *apart*.

3. Polling and continuous tasks are anti-social.

4. CTOS makes it easy to move tasks around.

METHOD 1: Monolithic configuration file.

```
% File: clifDemo.cfg
% ------------------
chassis vx0 1
chassis earth
sequencer earth

PREFIX SequHost 0
args  recSvr   REC_OPTIONS    = -rv -geometry 500x800+0 -name Clif_Demo
args  recSvr   XCTOS_DISPLAY = earth:0.0

PREFIX earth 0
task  clifClientHandler /usr2/testbed/exp/unix/demos/clif/clifClientUnix
task  gsmServer         /usr2/testbed/exp/unix/bin/sun4/gsmServer
systask viewer REVROOTDIR/unix/bin/sun4/xctosParent
args    viewer XCTOS_PROG = REVROOTDIR/unix/bin/sun4/fxctosviewer
args    viewer XCTOS_DISPLAY = DISPLAY

PREFIX vx0
-1 share /usr2/testbed/exp/vxworks/share/680x0/mcsShare.o
-1 load  /usr2/testbed/exp/vxworks/lib/680x0/mcsLib.o

0 load /usr2/testbed/exp/vxworks/lib/680x0/clifManagerLib.o
0 task clifManager clifMsgHandler 90

1 load /usr2/testbed/exp/vxworks/lib/680x0/gripLib.o
1 task gripL gripEventHandler 150
1 task gripR gripEventHandler 150

1 load /usr2/testbed/exp/vxworks/lib/680x0/tgen.o
1 task tgen tgenMsgHandler 100

2 load /usr2/testbed/exp/vxworks/lib/680x0/pumaLib.o
2 load /usr2/testbed/exp/vxworks/lib/680x0/chanPuma.o
2 task chanLPmaDrv chanLPumaMsgHandler 100

3 load /usr2/testbed/exp/vxworks/lib/680x0/pumaLib.o
3 load /usr2/testbed/exp/vxworks/lib/680x0/chanPuma.o
3 task chanRPmaDrv chanRPumaMsgHandler 100
```

METHOD 2: Subordinate chassis-config files.

```
% File: clifDemo.cfg
% ------------------
chassis vx0 1 /usr2/testbed/exp/vxworks/demos/clif/clif.mcs
chassis earth
sequencer earth

PREFIX SequHost 0
args  recSvr       REC_OPTIONS   = -rv -geometry 500x800+0 -name Clif_Demo
args  recSvr       XCTOS_DISPLAY = earth:0.0

PREFIX earth 0
task clifClientHandler /usr2/testbed/exp/unix/demos/clif/clifClientUnix

task  gsmServer /usr2/testbed/exp/unix/bin/sun4/gsmServer
```

```
% File: clif.mcs
% --------------
PREFIX vx0
-1 share /usr2/testbed/exp/vxworks/share/680x0/mcsShare.o
-1 load  /usr2/testbed/exp/vxworks/lib/680x0/mcsLib.o

0 load /usr2/testbed/exp/vxworks/lib/680x0/clifManagerLib.o
0 task clifManager clifMsgHandler 90

PREFIX vx0 1
load /usr2/testbed/exp/vxworks/lib/680x0/gripLib.o
task gripL gripEventHandler 150
task gripR gripEventHandler 150

PREFIX vx0 1
load /usr2/testbed/exp/vxworks/lib/680x0/tgen.o
task tgen tgenMsgHandler 100

PREFIX vx0 2
load /usr2/testbed/exp/vxworks/lib/680x0/pumaLib.o
load /usr2/testbed/exp/vxworks/lib/680x0/chanPuma.o
task chanLPmaDrv chanLPumaMsgHandler 100

PREFIX vx0 3
load /usr2/testbed/exp/vxworks/lib/680x0/pumaLib.o
load /usr2/testbed/exp/vxworks/lib/680x0/chanPuma.o
task chanRPmaDrv chanRPumaMsgHandler 100
```

METHOD 3: Subordinate config files, using include

```
% File: clifDemo.cfg
% ------------------
chassis vx0 1
chassis earth

sequencer earth

PREFIX SequHost 0
args  recSvr       REC_OPTIONS    = -rv -geometry 500x800+0 -name Clif_Demo
args  recSvr       XCTOS_DISPLAY = earth:0.0

PREFIX earth 0
task clifClientHandler /usr2/testbed/exp/unix/demos/clif/clifClientUnix
task gsmServer         /usr2/testbed/exp/unix/bin/sun4/gsmServer

PREFIX vx0
-1 include /usr2/testbed/exp/vxworks/demos/clif/clif.mcs
```

METHOD 4: Subordinate chassis-config files and CVCS.

```
% File: Imakefile on unix side.
% ----------------------------
LDLIBS += -lrec -lmsg -lbts -lctos -lmcs -lclifClient
LDLIBS += -lkntpt -lconfig -ltranParams -lkin -ltrans

AllTarget(clifClientUnix clifDemo.cfg)

UNIXBinTarget(clifClientUnix,)
ConfigFileTarget(clifDemo.cfg, clifDemo.cfg_src)

% File: clifDemo.cfg_src
% ----------------------
#define DISPLAY earth:0.0
#define HOST    earth
/* Note: REVROOTDIR and TARGETDIR are provided by CVCS. */

chassis vx0 1 REVROOTDIR/vxworks/demos/clif/clif.mcs_fast
chassis HOST
sequencer HOST

PREFIX SequHost 0
args  recSvr       REC_OPTIONS    = -rv -geometry 500x800+0 -name Clif_Demo
args  recSvr       XCTOS_DISPLAY = DISPLAY

PREFIX HOST 0
task    clifClientHandler TARGETDIR/clifClientUnix
task    gsmServer         REVROOTDIR/unix/bin/sun4/gsmServer
```

METHOD 4, continued

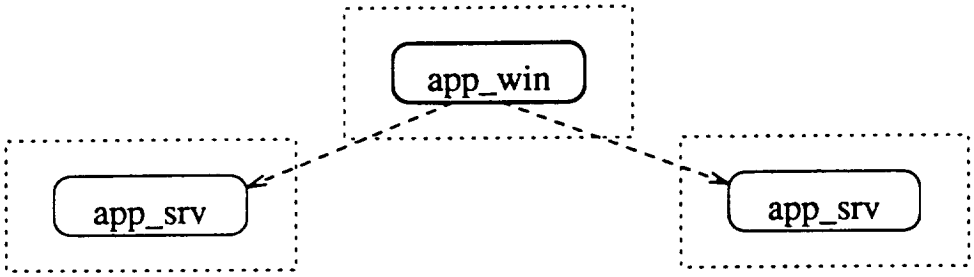
```
% File: Imakefile on vxworks side.
% --------------------------------
AllTarget(clif.mcs_fast)
ConfigFileTarget(clif.cfg,          clif.cfg_src)
FastLoadTarget(clif.mcs_fast,clif.cfg,clifServer,clif.cfg)
...

% File: clif.mcs_src
% ------------------
PREFIX vx0
-1 share REVROOTDIR/vxworks/share/680x0/mcsShare.o
-1 load  REVROOTDIR/vxworks/lib/680x0/mcsLib.o

0 load REVROOTDIR/vxworks/lib/680x0/clifManagerLib.o
0 task clifManager clifMsgHandler 90

PREFIX vx0 1
load REVROOTDIR/vxworks/lib/680x0/gripLib.o
task gripL gripEventHandler 150
task gripR gripEventHandler 150

PREFIX vx0 1
load REVROOTDIR/vxworks/lib/680x0/tgen.o
task tgen tgenMsgHandler 100

PREFIX vx0 2
load REVROOTDIR/vxworks/lib/680x0/pumaLib.o
load REVROOTDIR/vxworks/lib/680x0/chanPuma.o
task chanLPmaDrv chanLPumaMsgHandler 100

PREFIX vx0 3
load REVROOTDIR/vxworks/lib/680x0/pumaLib.o
load REVROOTDIR/vxworks/lib/680x0/chanPuma.o
task chanRPmaDrv chanRPumaMsgHandler 100
```

CTOS Startup

References: Tech Memo 16:3
Man pages: app_srv, app_win, app_bts, app_cleanup

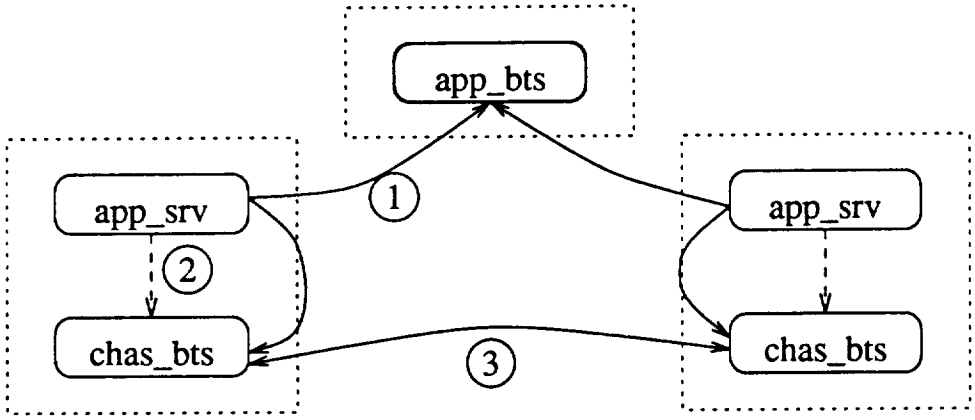
CIRSSE

CTOS BOOTSTRAP



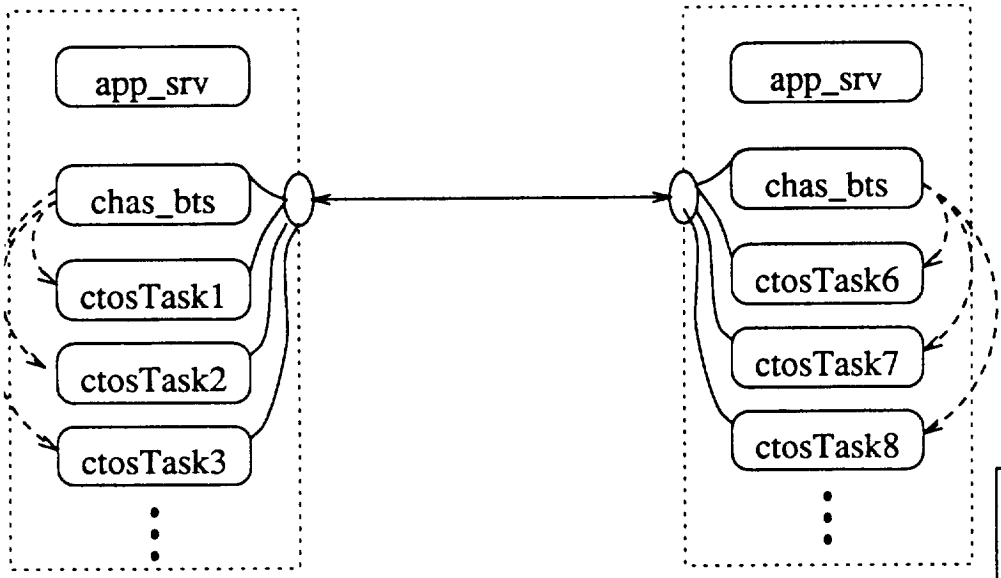
Startup app_srv.

app_srv ready.



Connecting

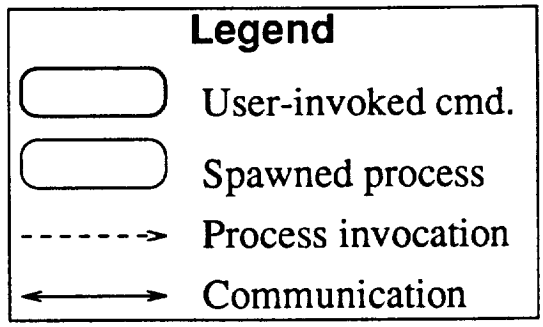
1. app_bts connects to app_srv
2. app_srv spawns chas_bts.
3. chas_bts's are completely connected.



Running

Each task has a msg. queue

Tasks inherit interchassis sockets from chas_bts



UNIX Commands

`app_srv [-a auth] [-p perm] [-u user]`

Server, to spawn CTOS processes on behalf of applications.

`app_win [xterm-options...] <chassis> [app_srv-options...]`

Preferred way to invoke `app_srv`.

`app_bts <cfg-file>`

Bootstraps an application from any UNIX machine.

`app_cleanup [cfg-file] [-i | -t | -a | -h]`

Cleans up System V IPC resources, after an application crashes.

`ipcs`

System V command, for examining IPC status.

Bootstrapping Notes

1. Only one `app_srv` may run on a chassis at a time.
2. Multiple applications may run concurrently on a Unix chassis (not on a VxWorks chassis), from the same `app_srv`.
3. Running `app_cleanup` for a chassis on which any application is running will crash that application.
4. `app_win` should be run from a "local" shell. (It uses the DISPLAY environment variable.)

Startup on VxWorks

Reset sequence on a vxWorks chassis:

- Load special vxWorks/ CTOS kernel
- Spawn CTOS boot processes
- Execute CTOS system config-file (incl. inter-processor connections)
- Print CTOS logo
- CPU-0 waits for connection requests (and emulates an `app_srv`)

Reset conditions:

- Hardware reset (The "red button" on CPU 0.)
- Controlled application shutdown*
- `app_cleanup`*

* — Software-resets may be disabled.

ctoskernel Command

```
ctoskernel - set CTOS kernel called by VxWorks

usage: ctoskernel  chassisname  -option|-f CTOS kernel file

Valid chassis names are:
cat             --   CAT Control System
mcs             --   Motion Control System Cage
mrs             --   Mobile Robot System
vss             --   Vision Services System

Available options are:
-d              --   default (stable) CTOS kernel
-e              --   experimental CTOS kernel
-f              --   user-specified CTOS kernel
-h              --   help
-o              --   old multi-chassis CTOS kernel
-r              --   released CTOS kernel
-s              --   stable CTOS kernel
-t              --   test CTOS kernel (incl. msg data deallocation)
-v              --   old single-chassis CTOS kernel
-x              --   experimental CTOS kernel


> ctoskernel mcs -x
ctoskernel done, mcs chassis will use following CTOS kernel on next reboot:
        /usr2/testbed/exp/vxworks/src/ctos/kernels/ctosKernel.o.
```

Advanced Tools & Services

- Task Configuration with TASK_ARGS

  References: Tech Memo 16:6
  Man pages: msgStringAssign

- Hybrid Tasks: XCTOS & CTRIP

  Files: ctoshLib.h, xctosParent.c

- Real-Time Services on VME Chassis

  References: Tech Memo 4
  Man pages: ipbLib, syncLib

Using Task-args

Use task-args to specify flags and/or string values.

Robotic application example:
Select robot speed with SPEED, between 0 and 1.
Select robot driver with ROBOT $\in$ {CLIF, PENDANT, GSM}.
Enable watchdog, flagged by WATCHDOG.

CTOS config-file:

```
...
PREFIX earth 0
task robotBrain roboBrain
args robotBrain SPEED = 1.
args robotBrain ROBOT = CLI     % This is silly,
args robotBrain ROBOT +=F       %   but okay.
args robotBrain WATCHDOG
```

Using Task-args, cont.

CTOS-task source code for the same example:

```
...
CtosTask(robThoughtHandler)
int robThoughtHandler (TID_TYPE tid, MSG_TYPE *msg)
{
  static char    *strSPEED=NULL, *strROBOT=NULL;
  static BOOL     flagWATCHDOG = FALSE;
  static double   speed        = .25;
  static enum {CLIF, PENDANT, GSM} robotDevice  = CLIF;

  switch(msg->command)
    { case MSG_TASK_ARGS:
        if      (msgStringAssign ((char*)msg->data,"SPEED", &strSPEED));
        else if (msgStringAssign ((char*)msg->data,"ROBOT", &strROBOT));
        else if (!strcmp((char*)msg->data,"WATCHDOG")) flagWATCHDOG = TRUE;
        break;

      case MSG_PINIT:
        if (strSPEED != NULL) sscanf(strSPEED, "%lf", &speed);
        if (strROBOT != NULL)
          { if      (!strcmp(strROBOT, "CLIF"))    robotDevice = CLIF;
            else if (!strcmp(strROBOT, "PENDANT")) robotDevice = PENDANT;
            else if (!strcmp(strROBOT, "GSM"))     robotDevice = GSM;
          }
...
```

Hybrid tasks

Some kinds of events:

- CTOS msgs.
- X-events.
- TelRIP events.
- IO events.
- Timer events.

Techniques for handling multiple kinds of events:

1. Polling.
   - Busy wait.
   - Poll by time (e.g. 'xtripDisplay').
   - Poll type A when type B arrives (suitable when communication is primarily in one direction).
2. Hybrid event handler— Convert one type of event into another.

Hybrid tasks— `ctoshLib`

CTOS-hybrid (ctosh) task consists of:

- Parent process— For each incoming message:
  1. Put message into special queue.
  2. Trigger an event that the child can detect.
- Child process— an event-handler primarily for some other kind of event, but which also has access to CTOS msgs. and `msgLib`.

Parent must be customized for the type of hybrid.

Examples:

1. 'xctos' — e.g. recSvr, GSM.
2. 'ctrip' — CTOS–TelRIP bridge.

CTOS Real-Time Services on VME Chassis

- Synchronous Services
  - supports periodic, synchronized execution of VxWorks tasks
  - synchronous task is spawned by "interface-type" CTOS task
  - provision for "overrun task" to handle exception condition
  - chassis clock & individual task clock rates set by user
  - see Tech Memo #4 or syncLib manual pages

- Inter-Processor Blocks
  - VxWorks semaphores don't work between CPUs
  - use IPBs to "data-synchronize" tasks between CPUs
  - all tasks waiting on IPB are unblocked simultaneously
  - see manual pages for ipbCreate, ipbTake & ipbUnblock

- Shared Memory
  - on CPU 0 or dedicated board
  - VME memory is dual-addressed
  - shared variables must be predefined (not dynamically alloc)
  - shared variable referenced in code like ordinary **`extern`** variable

Synchronous Services Example

```
/*************************************************************
  userfcn - interface CTOS task function
*************************************************************/
extern TID_TYPE  parent;            /* TID of parent    */
extern SEM_ID    semSync;           /* for sync process */
extern BOOL      runSync;
void  syncTask ();

int userfcn (TID_TYPE myTid, MSG_TYPE *msg)
{
    static SYNC_HANDLE  hSync;

    switch (msg->command)
    {
    case MSG_PINIT:          /* set up synchronous task */
        parent = myTid;
        hSync = syncProcSpawn(&semSync, syncTask,"Sync_Task",
                        0, NULL, NULL, "", SYNC_OVR_MILD,
                        &runSync, 1, 2000);
        if (hSync == ERROR)
            printf ("ERROR: Could not create Sync Task\n");
        break;
    }
    return (msgDefaultProc(myTid, msg));
}
```

Synchronous Services Example

```
/*************************************************************
  syncTask
*************************************************************/
TID_TYPE  parent;                    /* TID of parent    */
SEM_ID    semSync;                   /* for sync process */
BOOL      runSync = FALSE;

void  syncTask ()
{
    MSG_TYPE  msg;
    char      s[80];

    while (TRUE)
    {
        /*  block on semaphore  */
        runSync = FALSE;
        if (semTake (semSync, WAIT_FOREVER) == ERROR)
            logMsg("***  ERROR: Invalid semaphore  ***\n");

        /*  send message  */
        sprintf (s, "Hi daddy");
        msgBuild (&msg, parent, parent, MSG_STRING,
                  s, strlen(s), MF_STANDARD);
        msgPost (&msg);
    }
}
```

Use of Shared Memory

- Write & compile module containing shared variable definitions
  - variables should be initialized else compiler removes
  - use VxWorksShareTarget command in Imakefile

```
/*  File:  shareMem.c  */
char    shareString[] = "This is predefined";
double  shareVector[9] = {0.0};
```

- Reference shared variables as `extern` variables

```
/*  File:  useMem.c  */
extern char    shareString[];
extern double  shareVector[];
recInfo(t,"I got this from shared memory: %s", shareString);
for (i=0; i<9; i++)
    shareVector[i] = initVec();
```

- Load shared memory module onto all CPUs

```
%  VME chassis config file
prefix  vme_xyz
-1  share  shareMem.o
```

Standard CTOS Applications

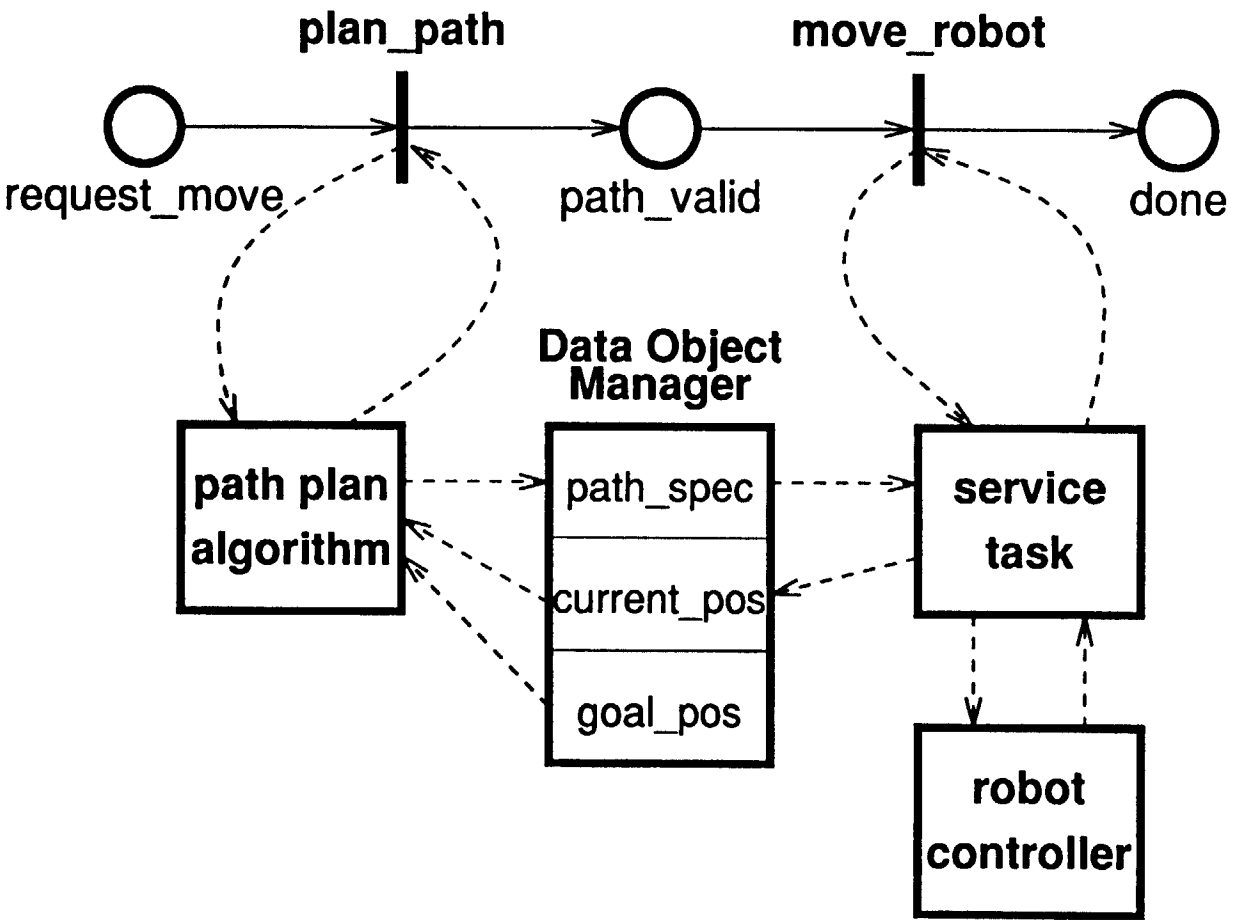
- Petri Net Controllers
- Motion Control System (MCS)
- Vision Services System (VSS)

References: CIRSSE Report 108

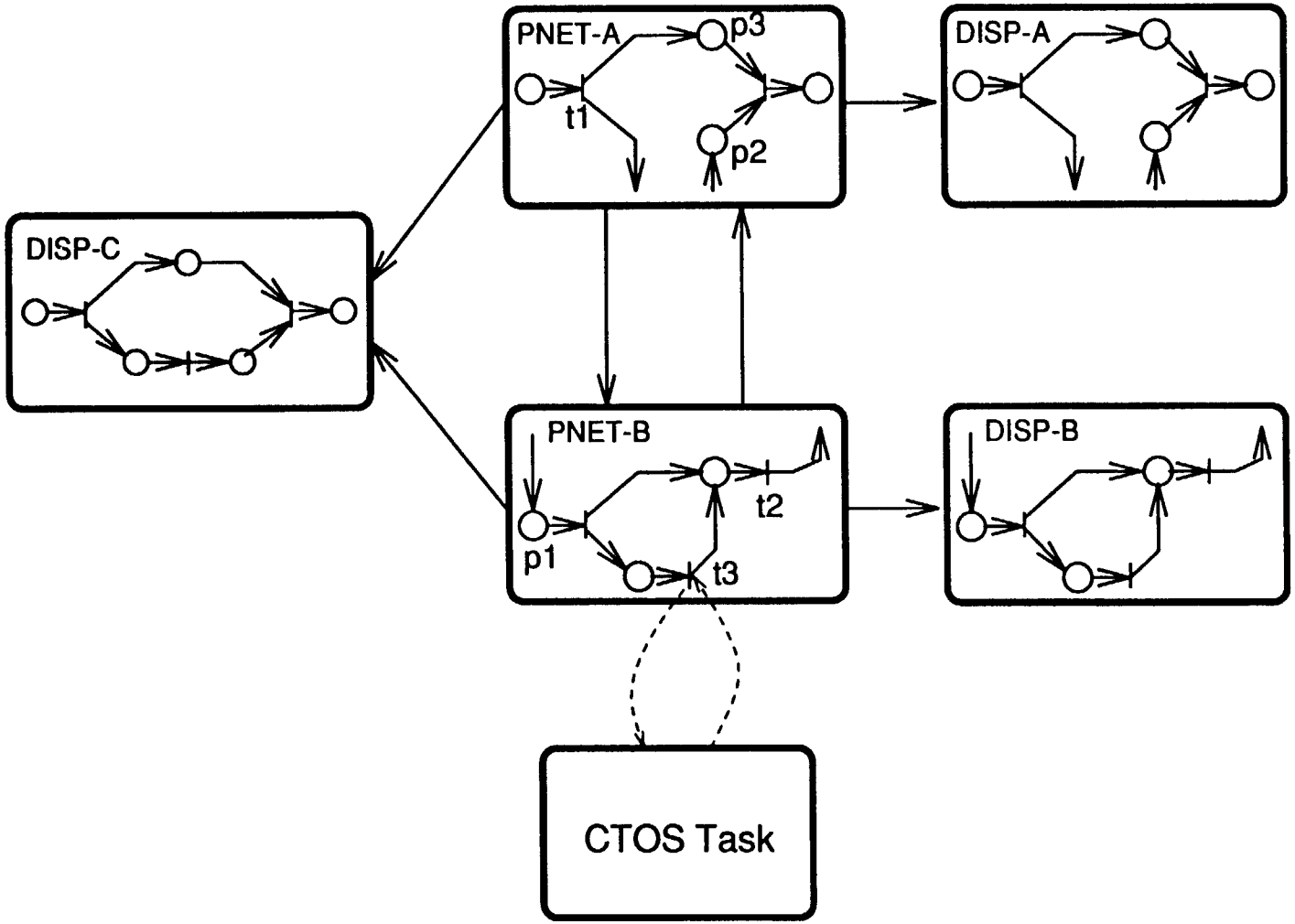
Petri Net Controllers

- Distributed application coordinated by Petri net Controller
  - PN represent parallelism of distributed robotic applications
  - PNC can be built hierarchically from simplier subsystems
  - can serve as intuitive operator interface
  - state of system and precedence of operations read from PN graph
- PN Controller is "application executive" task
  - generates messages to servers to sequence operations
  - generates messages based on operator input
- CTOS facilitates integration
  - PN player & displays can be distributed
  - Data Object Manager uses CTOS to implement distributed shared memory
  - PN servers communicate with Motion Control System & Vision Services

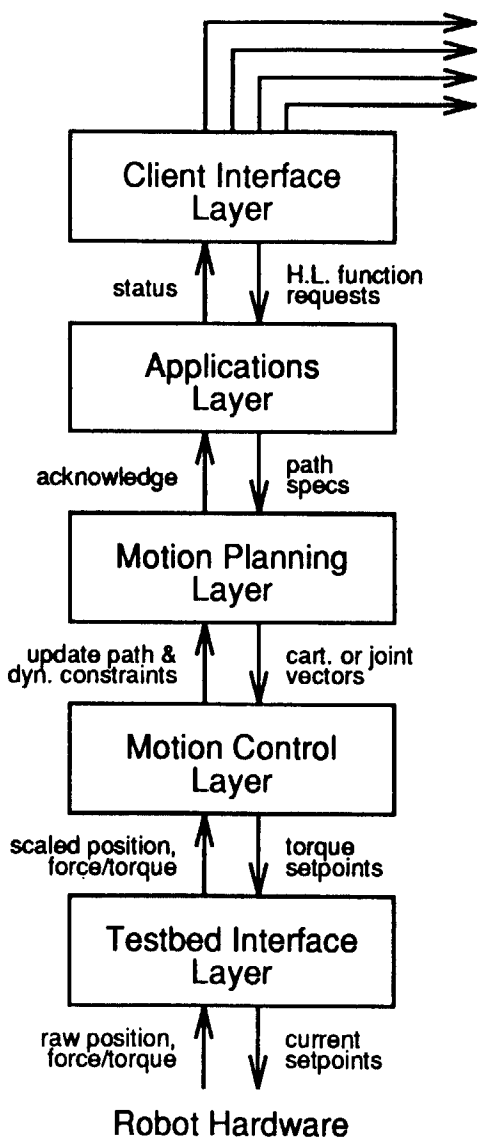
PN Interface to CTOS Tasks



Distributed PN Players & Displays



Motion Control System Overview



- Interactive operation over internet
- Emulation of VAL, etc.
- Direct programming via Petri nets
- Communication with Coordination Level

- Object-oriented multiprocessor architecture

- High-level functions (calibration, safety, logging, etc.)
- User-developed functions

- Cartesian/joint-interpolated trajectories
- Smoothing of segment transitions
- Multiple-arm cooperation

- Control algorithms providing gross & fine path trajectory tracking

- Synchronized reads/writes to HW
- Hide hardware dependencies

Motion Control System

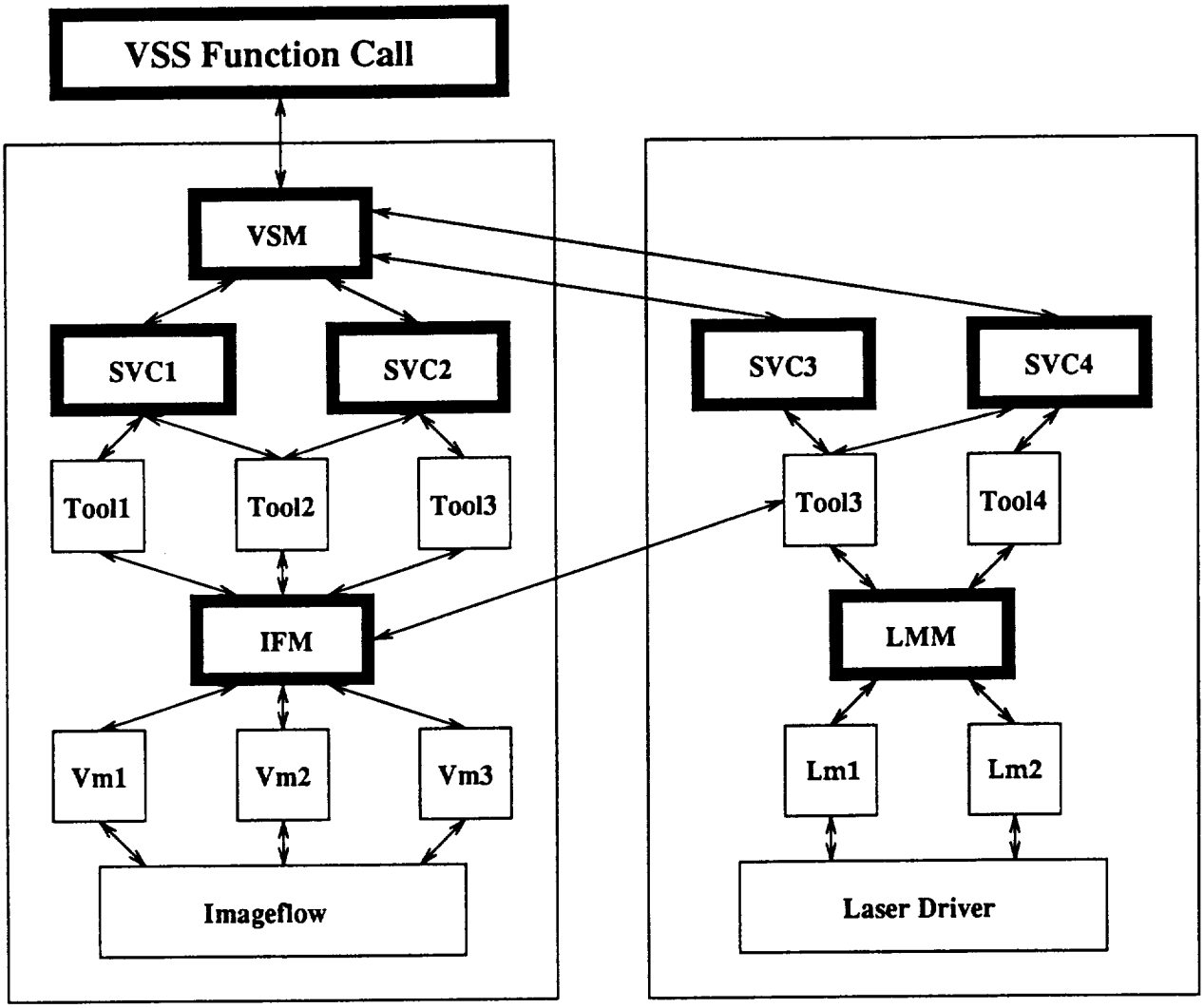
- Interface to manipulators of CIRSSE testbed
  - controls all 18 DOF
  - supports all aspects of motion control research
  - provides high-level interface for Coordination-Level research
- MCS is distributed application on multiprocessor VME chassis
  - access controller hardware on same chassis
  - divide workload over multiple processors
- CTOS facilitates integration
  - MCS uses real-time support & multiprocessor communication
  - MCS communicates with PN Controllers & Vision Services in multi-computer applications
- MCS uses CTOS real-time support extensively
  - lower 3 layers composed of synchronous tasks
  - channel drivers (interface layer) run at 5.4 ms
  - control loops data-synchronized with channel drivers by IPBs

Vision Services System

VSS is the software framework for accessing the Datacube and Laser Scanner hardware.

Features:

- Supports low and medium-low level vision operations, from frame-grabbing & display control, to finding struts or fiducial mark patterns.
- vss-level functions are implemented as RPCs; encapsulates the whole vision system in a server model.
- Multilayer implementation: services, tools, and vision (or laser) modules.
- Dynamic linking scheme only loads the necessary pieces of VSS for an application. (Dependencies are handled automatically.)
- Connected both internally and externally by CTOS.



VSM	=	Vision Services Manger
SVC	=	Service
IFM	=	Imageflow Manager
LMM	=	Laser Module Manager
Vm	=	Vision Module (has an Init Func and an Exec Func)
Lm	=	Laser Module

Vision Services System, cont

A sample request at each layer of VSS:

```
STATUS vssNodePattFind (TID_TYPE           mytid,
                        const VSS_SOURCE *source,
                        SVCNPF_PATT_ENUM  pattern,
                        BLOB            **pattListPtr,
                        BOOL             *flagsOriented);

STATUS toolStripe      (TID_TYPE           svcTid,
                        unsigned           num_pts,
                        const LASER_DATA *point);
typedef struct
 { MODULE_ID      vmId;
   IMAGE_ID       ovly;
   GFX_CLEAR_TYPE clearType;
   GFX_LINE      *lineList;
   int            numLines;
   GFX_MARKER    *markerList;
   int            numMarkers;
} VM_GFX_DRAW_ARG;
```

Software Engineering Tools

References: Tech Memo 16:5
Man pages: cmkmf, cvcs

Why Does Software Need Managing?

- "Hello World"-type software projects need no real management
  - Single programmer
  - User(s) need only the executable
  - Small number of source files/directories
  - Infrequent changes
- Any large dynamic multi-programmer multi-user software project needs software management to maintain sanity
- CIRSSE's working software tree currently has:
  - 187 directories
  - containing 1,083 *.[ch] files
  - totaling 229,339 lines of (bug free) code
  - written by $\approx$ 17 people
  - used by $>$ 17 people

The Joys of Make

- *make* - maintain, update, and regenerate related programs and files

- The user specifies targets to build and on what the targets depend as well as the rule(s) used to build targets

```
all:    prog1 prog2 course.ps

prog1:  prog1.o prog1.c prog1.h
        gcc -o prog1 prog1.c

prog2:  prog2.o prog2.h extraFile.o extraFile.h

course.dvi:     course.tex swmanag.tex
        slitex course.tex

swmanag.tex:

%.ps:   %.dvi
        dvips -o $@ $<

clean:
     rm -f *.o prog1 prog2 *.dvi *.ps *.aux *.log
```

The Joys of Make-ing it with Sun Microsystems

- Specifying every header file on which a given source file depends by hand is a nightmare
  - File1 includes File2 which includes FileA and FileB which include ...
- If the rule for building a target changes, it is often desirable for that target to rebuild
- Enter the nice people at Sun with their special .KEEP_STATE target
- .KEEP_STATE does the right thing :)
- *make* is not standard :(

How can *Imake* make *make* more manageable

- Your strange and mysterious makefiles frighten and confuse me
- 90% of *make* commands are repeated over and over
- "Let's use the C preprocessor to hide *make* details with macros and call it **imake**‡"†
- Someone creates *imake* template files "once and for all"
- It is customary to have yet another program invoke *imake* and tell it from where to get its template files
- Here that program is called: **cmkmf** for CIRSSE Make Makefile
- Note that any legal *make* command may appear in an Imakefile
- Only one Imakefile per directory

†Somebody from MIT

‡for no apparent reason

Common *cmkmf* macros

AllTarget(targets) macro which forces *make* to build **targets** when *make* or *make all* is invoked

UNIXBinTarget(target,objlist) macro to link group of UNIX object files **(objlist)** into 1 executable. Executable file is specified as **target**. **objlist** may be omitted if **target** built solely from **target.c**

UNIXLibTarget(target,objlist) macro to combine **objlist** into library archive named **target**, which is assumed to be prefixed with lib and suffixed with .a. **objlist** may be omitted if **target** built solely from **target.c**

DefineString(macro,string) used to pass defined constant strings to cpp. Useful when a source file requires that a directory be compiled into it and that directory is subject to frequent change: DefineString(TARGETDIR, "$(PWD)"). Note that string should be surrounded by double quotes. *cmkmf* automatically defines REVROOTDIR, REVSYMNAME, and TARGETDIR, so any .c or .h file may reference them as if they had been #defined.

Common *cmkmf* macros (continued)

VxWorksBinTarget(target,objlist) macro to link a group of VxWorks object files **(objlist)** into one executable named **target**. if | **objlist** | > 1 then **target** cannot be a file in **objlist**. **objlist** may be omitted if **target** built solely from **target.c**

VxWorksLibTarget(target,objlist) macro identical to **VxWorksBinTarget**. Logically separates VxWorks object files used as libs from those object files which are VxWorks executables

VxWorksShareTarget(target,objlist) macro to create a VxWorks shared object module **(target)** from **objlist**. **target** must never be a file in **objlist**. **objlist** may be omitted if **target** built solely from **target.c**

ConfigFileTarget(cfgTarget,cfgSource) macro creates a CTOS config file **(cfgTarget)** from a source file **(cfgSource)** by passing the **cfgSource** through cpp

MakeSubdirs(subdirs) macro which invokes *cmkmf* in each **subdir** in the order of appearance. Only 1 per Imakefile.

Pre-Defined *make* macros

CPPFLAGS allows user to add directories in which to search for include files by assigning or appending **-I/pathname** to the macro

LDFLAGS allows user to add directories in which to search for libraries with which to link *before* the default directories. This is done by assigning or appending **-L/pathname** to the macro.

LDLIBS allows user to add directories in which to search for libraries with which to link *after* the default directories. Libraries with which to link may specified with this macro, **but NOT with LDFLAGS**. Additional directories are added by **-L/pathname** to the macro. Libraries are added by assigning or appending **-lname** to the macro.

CFLAGS allows the user to specify compiler options. It's **-O2** by default. May wish to assign **-g** to this macro for debugging

CFGPPFLAGS allows the user to specify additional directives to the C pre-processor used by the **ConfigFileTarget** macro described above.

REVROOTDIR holds the current revision root directory (e.g. /usr2/testbed/exp)

Example UNIX Imakefile

```
/* C-style comments are not passed on to the Makefile */

/**/# Here is how to pass a comment to the Makefile

CPPFLAGS += -I/any/path/include -I/another/path/include
LDFLAGS  += -L/any/path/before/lib
LDLIBS   += -lany -L/another/path/after/lib -lanother
LDLIBS   += -lm

AllTarget(libsampleS.a libsampleM.a sampleS sampleM)

UNIXLibTarget(libsampleS.a, )

UNIXBinTarget(sampleS, )

UNIXLibTarget(libsampleM.a, functions.o subs.o)

UNIXBinTarget(sampleM, sampleM.o io.o)
```

Example VxWorks Imakefile

```
/**/# Comments prefixed by C comment delimiters like
/**/# this line are copied into the Makefile generated
/**/# by imake.

/* Comments like this don't show up in the Makefile */

/* Note that any valid make commands are perfectly  */
/* acceptable.  For example, these are make macros  */
OBJS      = sampleSLib.o sampleMLib.o sampleSBin.o \
            sampleMBin.o

CPPFLAGS += -I/any/path/include -I/another/path/include

AllTarget($(OBJS))

VxWorksLibTarget(sampleSLib.o, )
VxWorksBinTarget(sampleSBin.o, )
VxWorksLibTarget(sampleMLib.o, functions.o subs.o)
VxWorksBinTarget(sampleMBin.o, main.o io.o)
```

VxWorks and UNIX macros may NOT BE MIXED

What's still missing?

- What do we do when we want to modify a source file but we don't want to risk screwing up what we've already done?
- What do we do when we have multiple programmers trying to work on the same set of source files?
- What do we do when we have users who can't constantly keep up with the latest changes?
- What do we do when we need to do all these things on a large scale?

◇ RCS or SCCS handle the first two points well for a single directory.

◇ CVCS builds upon RCS to address all 4 points.

What does CVCS do?

⇒ Provides revision control for an entire directory structure (and the files within)

  * ⋆ Many different revisions of the directory structure are allowed to exist simultaneously (e.g. stable and experimental)
  * ⋆ The files **and** the directories themselves may differ between revisions

⇒ Allows multiple users "idiot-proof" ability to edit and build in only one *Revision Tree*.

  * ⋆ Only one user may edit a single source file at any one time

⇒ Provides a central location for all the software (and documentation) developed at CIRSSE

⇒ Allows users to recompile the entire "Experimental" tree (or any of it's subtree) as their development demands while leaving other "Stable" tree(s) completely unchanged.

⇒ Allows controlled promotion of files, subtrees, or the entire tree from "Experimental" to "Stable"

Some example CVCS commands

newfile - Put a new file under CVCS revision control

checkedit - Get a writable copy of a CVCS revision controlled file

checkin - Update a CVCS revision controlled file to save changes

checkout - Get a read-only copy of a CVCS revision controlled file

createdir - Create a new directory in an Experimental CVCS revision tree

movedir - Move (or rename) a directory within an Experimental CVCS revision tree

scan - Display diagnostic information of files in a CVCS revision tree

sweep - Remove files which can be recovered from revision control in a CVCS revision tree

Good Software Development Practices

- Header files should be self-sufficient – but no more
  - #include everything in the header which is necessary to use the functions or structures defined therein – but no more
  - Do not put functions or structures which the user does not need in a "public" header file.
- Always protect header files from multiple inclusions
- Use libraries whenever possible to logically group related functions
- Agree on a consistent naming convention and use it
- Use meaningful comments, variable names, and function names
- Several small files are almost always better than one large file
- Don't ignore compiler warnings unless you're very sure you know what you're doing. Even then it is better to eliminate the warning than to ignore it.

List of CTOS Functions

List of CTOS Functions

NAME	PURPOSE
SENDING MESSAGES	
msgSend	send message to event handler task
msgPost	post message (returns immediately)
msgBroadcast	broadcast message to all tasks
msgReply	reply to received message
msgAcknowledge	acknowledge a received message
msgBuildSend	build then send a message
BUILDING MESSAGES	
msgBuild	set all members of message structure
msgMemownerFlagSet	set MEMOWNER field of msg.flag
msgPriorityFlagSet	set PRIORITY field of msg.flag
msgReplyFlagSet	set REPLY_WAIT field of msg.flag
msgSendFlagSet	set SEND_WAIT field of msg.flag
msgTypeFlagSet	set TYPE field of msg.flag

List of CTOS Functions, Con't

WORKING WITH TIDS	
msgTidQuery	find task id from symbolic task name
msgTidQueryLocal	msgTidQuery restricted to local chassis
msgTidGetChassis	get CHASSIS field of task id
msgTidGetCpu	get CPU field of task id
msgTidGetLocal	get LOCAL field of task id
msgTidSetChassis	set CHASSIS field of task id
msgTidSetCpu	set CPU field of task id
msgTidSetLocal	set LOCAL field of task id
MEMORY MANAGEMENT	
msgCopy	make local copy of message
msgDataCopy	make local copy of message data
msgDataKeep	keep message data (prevents dealloc)
msgDataFree	free message data
msgReplyGet	get pointer to reply message
msgReplyPop	pop reply from stack
msgReplyPush	push reply onto stack
msgVarPtrSet	set pointer to saved variables
msgVarPtrGet	get pointer to saved variables

List of CTOS Functions, Con't

I/O & STRING OPERATIONS	
recInfo	print message to record window
recWarning	print warning message to record window
recError	print error message to record window
recPrompt	get input from record window
recPromptPost	get input from rec window (nonblocking)
msgErrorLog	send a string to local Error Server
msgErrorExit	send error message then exit application
msgMyTaskName	get a task's symbolic name
msgStringAssign	parse string for keyword
msgString	format string containing msg components
QUEUE OPERATIONS	
msgDequeue	read message directly from local queue
msgQueueCount	count messages in local queue
msgRequeue	put message directly into local queue
MISC PROCESSING	
msgAckAINIT	explicit acknowledgement of AINIT message
msgApplicationExit	request shutdown of the application
msgDefaultProc	default processing for system messages

Exercises

1. Single task reporting CTOS phases
2. Multichassis VxWorks + UNIX appl.
3. Sending messages to other tasks
4. Developing a server task
5. Writing a RPC function

CTOS Course Exercises

1. Simple UNIX Event Handler Task

   - Description: Single-task application which reports CTOS phases.
   - Lessons:
     - format of event handler function
     - output to X-window
     - CTOS execution phases
     - using Imakefile & cmkmf to compile/link
     - building configuration files
     - starting CTOS application
   - Initial Setup:
     (a) Change to the directory created for your team. For example, Team 1 uses `/usr2/testbed/course/team1/`. All code written for the exercises should be put in this directory.
     (b) Use `epath` command to set up path to 'experimental' directory tree.
   - Procedure:
     (a) Write a standard-format CTOS task function to print a string when it receives MSG_PINIT, MSG_AINIT, MSG_AEXEC, MSG_ATERM, and MSG_PTERM messages.
     (b) Print your message string using `printf`, and the same string using `recInfo`. This will demonstrate the different output locations of these two functions.
     (c) Create an Imakefile and use `cmkmf` command to compile and link your task function.
     (d) Write a CTOS application configuration file. Include the following line to start an Application Executive task as part of your application:

     ```
     task  AppExec  /usr2/testbed/course/servers/appexec
     ```

     (e) Start an app_server with Permissions='Anyone' as shown below - replacing `workstation_name` with the one assigned by the instructors.

     ```
     app_win  workstation_name  -p 0
     ```

     (f) Run your application via `app_bts` command, and observe the printed output (content and location).
   - Hints/Suggestions:
     - If you want to keep code from exercises separate you can create directories below your 'team' directory, e.g. ex1/, ex2/, etc.; but this is not necessary.
     - Experiment with formatting the string, e.g. could use the `msgMyTaskName` function to identify your task by name in the message string.
     - Imakefile needs only 3 statements: `LDLIBS`, `AllTarget` and `UNIXBinTarget`.

- If you share an app_server you must configure the recServer to direct output to your terminal, e.g.:

  ```
  prefix  SequHost 0
  args  recSvr  XCTOS_DISPLAY = xterm8:0.0
  ```

- When sharing a workstation with another team only the first call to app_win will succeed, but both teams will be able to use it. If you are unable to share an app_server it may not have been started with the `-p 0` option.
- If your application ends abnormally it may be necessary to use **app_cleanup** to release IPC resources used by CTOS.

2. Multi-chassis VxWorks+UNIX Application

   - Description: Port UNIX task to VxWorks, and combine all teams' tasks into one application.
   - Lessons:
     - task args for configuring a task
     - compatibility between UNIX and VxWorks
     - Imakefiles for both UNIX and VxWorks
     - multi-chassis application config files
   - Procedure:
     (a) Modify your CTOS task function from Exercise 1 to save a string sent to it via an **args** command in the chassis config file, and to display the string during PINIT phase.
     (b) Add the **args** command to your config file and test it.
     (c) Create a subdirectory for a VxWorks version of your task function (this is necessary because the UNIX and Vxworks Imakefiles are different).
     (d) Port your CTOS task function to run under VxWorks on the VME cage by isolating UNIX- or VxWorks-specific code within conditional compilation directives, e.g.:

     ```
     #ifdef OS_UNIX
     #include <unix_stuff.h>
     #endif
     ```

     (e) Update Imakefiles and compile/link.
     (f) As a group, build an application configuration file to start UNIX and VxWorks tasks for all teams in the same CTOS application. A VME CPU will be assigned to each team by the instructor.
     (g) Before starting the application, rlog into your VME CPU, e.g.:

     ```
     rlogin  vx2
     ```

     (h) Run the application and observe the printed output on app_servers, VME consoles (via rlogin), and recWindow.

- Hints/Suggestions:
  - Using the **msgStringAssign** function is the best way to parse the args string.
  - The available compiler constants are OS_UNIX and OS_VXWORKS.
  - Rather than copying code into the vxworks subdirectory consider creating a symbolic link to the code in the 'team' subdirectory so that code changes are immediately reflected in both locations. The symbolic link can be created with the following command:

    ```
    ln -s ../myfcn.c .
    ```

  - **cmkmf** will make both your team and vxworks subdirectories if you add a `MakeSubdirs()` statement to the Imakefile in the team subdirectory.
  - Designate one team to assemble input from each team to build the group's application configuration file. Put this file in directory `/usr2/testbed/course/`.
  - Remember to include the AppExec task and to configure the RecWindow to display on the workstation from which the application will be run.
  - Hung applications are reset by `app_cleanup`. Remember to wait for the VME cage to reboot.
  - You will need to rlog into the VME CPU after every reboot.

3. Send Messages to Other Tasks

- Description: Send & receive messages among tasks of multi-chassis application.
- Lessons:
  - finding TIDs of other tasks
  - saving data between calls to EH function
  - building and sending messages
  - accessing data contained in messages
- Procedure:

  (a) Teams are assigned symbolic names for their tasks based on their team number; for example, team 1's task is named `Team1`.

  (b) Modify your CTOS task function from Exercise 2 to query for the TID of other teams' tasks during AINIT phase. Find the TIDs of the two teams numerically higher and lower that yours (i.e. team 2 finds TIDs of team 1 and team 3). Your task should report the result to the recWindow, and save the TIDs for later use.

  (c) During AEXEC phase, build and send messages to the tasks queried earlier. In the message send the string received via the config file **args** command. Use the predefined message command of `MSG_STRING`.

  (d) Add processing for receipt of a `MSG_STRING` **message**. Your task should display the received string and the TID of the source task.

(e) As in the previous exercise, build an application configuration file that will start an application containing a task from each team and the Application Executive. Remember to include an **args** command for each team's task.

(f) Run it.

- Hints/Suggestions:
  - We will not use the VME cage for the remainder of the exercises so you may remove references to VxWorks tasks from the Imakefile and config files.
  - Keep the messages identifying execution phases if you like (may be good for debugging), but delete printf's and just use recInfo's.
  - You can test your task's message passing by having it send a message to itself, and running it as a two-task application (your task plus the usual AppExec).

4. Develop a Server Task

- Description: Write a server and a simple task to test it.
- Lessons:
  - requester-server architecture
  - server message protocol
- Procedure:

(a) Design a server to perform a simple operation. For example, your server might receive two floating point numbers, multiply them together, and return the result.

(b) Define a message command for the request message to the server. Offset the message command value from MSG_USER+100 by your team number (i.e. team 5 should use MSG_USER+100+5).

(c) Define a data structure to contain the data transmitted in the request message to the server. A data structure for return data may also be needed unless the server returns a simple data type such as a single floating point value.

(d) Develop a server task function which receives your request message containing input data, performs its operation, and returns the result in a reply message.

(e) Write a task function that builds and sends a request message to your server and displays the returned results.

(f) Compile, link and run your server and test task to ensure correct operation.

- Hints/Suggestions:
  - Each team works alone for this exercise, but in the next exercise we will combine results. Keep in mind that other teams will later use your server.
  - Use of a header file is *highly* recommended for this exercise, and will be required for the next one.
  - Next exercise we will assign symbolic names to servers based on team number (e.g. **Team3_Svr**) - you might as well start using this name now.

- Can avoid the need of a data structure for server reply data if you return just a simple data type. However, you will need to specify a way to identify error conditions; for example by defining a certain return value to be an error indication.
- Hint: you should use the reply-wait mechanism to block the test task until the server replies, so you will use a MF_WAIT_SENDER, _RECEIVER, or _COPY message flag for the request message.
- A good server should check all inputs to ensure that they are valid. For instance, what happens if the received request message has a NULL msg.data pointer?
- Try several requests to the server, perhaps including invalid inputs or inputs that result in an illegal server operation.

5. Write a RPC Function

- Description: Encapsulate access to your server as a remote procedure call so others can use it.
- Lessons:
  - RPC functions
  - header files and libraries
- Procedure:
  (a) Name your server from Exercise 4 based on your team number, i.e. the server written by team 3 will get symbolic name `Team3_Svr`.
  (b) Encapsulate the building of the server request message and all communication with the server into a RPC function, and put this function in a separate file. Compile it as a library (see Imakefile `UNIXLibTarget()` command).
  (c) Create a header file which contains only the definitions needed to access your RPC function (potentially as little as the function prototype). Put enough documentation in the header file so that users from the other teams will be able to understand how to use your RPC.
  (d) Build symbolic links in directory `/usr2/testbed/course/servers/` to your RPC header and library, and to the executable code for your server.
  (e) Modify your task function from Exercise 4 to call your RPC function, and test it.
  (f) Experiment with calling the RPC functions developed by the other teams. To do so you will need to include the header file in your test task that calls the RPC, and to start their server task when you run the application.
- Hints/Suggestions:
  - The server developed in Exercise 4 can probably be used without modification for this exercise.

- Pick a name for your RPC function and library that will avoid conflicting with other teams' names (we don't want ten versions of myRPC). Including your team number in the RPC name is a good way to accomplish this (myRPC3 is OK, although boring).
- When using other team's RPCs you will need to include their header files and link with their libraries. Add the following lines to your Imakefile to enable cmkmf to find the RPCs:

```
CPPFLAGS += -I/usr2/testbed/course/servers/
LDLIBS   += -L/usr2/testbed/course/servers/
```

- If the linker cannot resolve a reference to a RPC function it may be because the library was not named in the Imakefile LDLIBS statement.
- In a week or so the directories used during the exercises will be deleted. Copy any files you'd like to keep into your own account soon.

Exercise Solutions

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                        SOLUTION TO EXERCISE 1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

/*
** File:        ex1.c
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     Solution to Exercise 1
*/

#include <stdio.h>
#include <ctos.h>
#include <recLib.h>

CtosTask(ex1)


/*****************************************************************************
  ex1 - CTOS task function
*****************************************************************************/
int ex1 (TID_TYPE myTid, MSG_TYPE *msg)
{
    switch (msg->command)
    {
    case MSG_PINIT:
        printf("%s received PINIT\n", msgMyTaskName(myTid));
        recInfo(myTid, "%s received PINIT\n", msgMyTaskName(myTid));
        break ;

    case MSG_AINIT:
        printf("%s received AINIT\n", msgMyTaskName(myTid));
        recInfo(myTid, "%s received AINIT\n", msgMyTaskName(myTid));
        break ;

    case MSG_AEXEC:
        printf("%s received AEXEC\n", msgMyTaskName(myTid));
        recInfo(myTid, "%s received AEXEC\n", msgMyTaskName(myTid));
        break ;

    case MSG_ATERM:
        printf("%s received ATERM\n", msgMyTaskName(myTid));
        recInfo(myTid, "%s received ATERM\n", msgMyTaskName(myTid));
        break ;

    case MSG_PTERM:
        printf("%s received PTERM\n", msgMyTaskName(myTid));
        recInfo(myTid, "%s received PTERM\n", msgMyTaskName(myTid));
        break ;
    }
    return (msgDefaultProc(myTid, msg)) ;
}

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

/*
** File:        Imakefile
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     Imakefile for Exercise 1
*/

LDLIBS += -lrec -lmsg -lbts -lctos

AllTarget(ex1)

UNIXBinTarget(ex1, ex1.o)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

%% File:        app.cfg
%% Written By:  Don Lefebvre
%% Date:        Nov 18, 1992
%% Purpose:     Application configuration file for Exercise 1

chassis   pluto
sequencer pluto

prefix SequHost 0
args   recSvr  XCTOS_DISPLAY = xterm8:0.0

prefix  pluto 0
task  AppExec  /usr2/testbed/course/servers/appexec
task  MyTask1  /usr2/testbed/course/answers/ex1/ex1
task  MyTask2  /usr2/testbed/course/answers/ex1/ex1
task  MyTask3  /usr2/testbed/course/answers/ex1/ex1
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                          SOLUTION TO EXERCISE 2
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

/*
** File:        ex2.c
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     Solution to Exercise 2
*/

#ifdef OS_UNIX
#  include <stdio.h>
#endif OS_UNIX

#ifdef OS_VXWORKS
#  include <stdioLib.h>
#endif OS_VXWORKS

#include <ctos.h>
#include <recLib.h>

CtosTask(ex2)


/*****************************************************************************
  ex2 - CTOS task function
*****************************************************************************/
int ex2 (TID_TYPE myTid, MSG_TYPE *msg)
{
    static char  *text = NULL;

    switch (msg->command)
    {
    case MSG_TASK_ARGS:
        msgStringAssign(msg->data, "TEXT", &text);
        break;

    case MSG_PINIT:
        printf("%s received PINIT\n", msgMyTaskName(myTid));
        recInfo(myTid, "%s received PINIT\n", msgMyTaskName(myTid));

        if (text != NULL)
            recInfo(myTid, "Received via TASK_ARGS: '%s'\n", text);
        break ;

    case MSG_AINIT:
        printf("%s received AINIT\n", msgMyTaskName(myTid));
        recInfo(myTid, "%s received AINIT\n", msgMyTaskName(myTid));
        break ;

    case MSG_AEXEC:
        printf("%s received AEXEC\n", msgMyTaskName(myTid));
        recInfo(myTid, "%s received AEXEC\n", msgMyTaskName(myTid));
        break ;

    case MSG_ATERM:
        printf("%s received ATERM\n", msgMyTaskName(myTid));
        recInfo(myTid, "%s received ATERM\n", msgMyTaskName(myTid));
        break ;

    case MSG_PTERM:
        printf("%s received PTERM\n", msgMyTaskName(myTid));
        recInfo(myTid, "%s received PTERM\n", msgMyTaskName(myTid));
        break ;
    }
    return (msgDefaultProc(myTid, msg)) ;
}

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

/*
** File:        Imakefile
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     UNIX Imakefile for Exercise 2
*/

LDLIBS += -lrec -lmsg -lbts -lctos

AllTarget(ex2)

UNIXBinTarget(ex2, ex2.o)

MakeSubdirs(vxworks)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

/*
** File:        vxworks/Imakefile
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     VxWorks Imakefile for Exercise 2
*/

AllTarget(ex2.o)

VxWorksBinTarget(ex2.o, )

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

%% File:        app.cfg
%% Written By:  Don Lefebvre
%% Date:        Nov 18, 1992
%% Purpose:     Application configuration file for Exercise 2

chassis   vx0   1
chassis   pluto
sequencer pluto

prefix  SequHost 0
args    recSvr  XCTOS_DISPLAY = pluto:0.0

%%  start Application Executive
%%
prefix  SequHost 0
task    AppExec  /usr2/testbed/course/servers/appexec


%%  Team 1 task config  ------------------------------------------
%%
%%  UNIX chassis configuration
%%
prefix  pluto 0
chdir   /usr2/testbed/course/answers/ex2/
task    MyUnixTask  ex2
args    MyUnixTask  TEXT = test of task args to UNIX
%%
```

```
%%  VxWorks chassis configuration
%%
prefix  vx0
1  chdir /usr2/testbed/course/answers/ex2/vxworks/
1  load  ex2.o
1  task  MyVxWorksTask  ex2  100
1  args  MyVxWorksTask  TEXT = test of task args to VxWorks


%%  Team 2 task config  ------------------------------------------------
%%
%%  UNIX chassis configuration
%%
%%  VxWorks chassis configuration


%%  Team 3 task config  ------------------------------------------------
%%
%%  UNIX chassis configuration
%%
%%  VxWorks chassis configuration
```

```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
                        SOLUTION TO EXERCISE 3
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

/*
** File:        ex3.c
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     Solution to Exercise 3
*/

#ifdef OS_UNIX
#  include <stdio.h>
#  include <string.h>
#endif OS_UNIX

#ifdef OS_VXWORKS
#  include <stdioLib.h>
#  include <strLib.h>
#endif OS_VXWORKS

#include <ctos.h>
#include <recLib.h>

CtosTask(ex3)


/*****************************************************************************
  ex3 - CTOS task function
*****************************************************************************/
int ex3 (TID_TYPE myTid, MSG_TYPE *msg)
{
    static TID_TYPE  tid1, tid2, tid3;
    static char     *text = NULL;
           char     *textcopy;
           MSG_TYPE  mymsg;
           int       len;

    switch (msg->command)
    {
    case MSG_TASK_ARGS:
        msgStringAssign(msg->data, "TEXT", &text);
        break;

    case MSG_PINIT:
        recInfo(myTid, "%s received PINIT\n", msgMyTaskName(myTid));

        if (text != NULL)
            recInfo(myTid, "Received via TASK_ARGS: '%s'\n", text);
        break ;

    case MSG_AINIT:
        recInfo(myTid, "%s received AINIT\n", msgMyTaskName(myTid));

        if ((tid1 = msgTidQuery(myTid, "Team1")) == 0)
            recInfo(myTid, "Could not find tid of Team 1\n");
        else
            recInfo(myTid, "Team 1 TID= %x\n", tid1);

        if ((tid2 = msgTidQuery(myTid, "Team2")) == 0)
            recInfo(myTid, "Could not find tid of Team 2\n");
        else
            recInfo(myTid, "Team 2 TID= %x\n", tid2);

        if ((tid3 = msgTidQuery(myTid, "Team3")) == 0)
            recInfo(myTid, "Could not find tid of Team 3\n");
        else
            recInfo(myTid, "Team 3 TID= %x\n", tid3);
        break ;

    case MSG_AEXEC:
        recInfo(myTid, "%s received AEXEC\n", msgMyTaskName(myTid));

        if (text != NULL)
        {
            len = strlen(text) + 1;
            msgBuild(&mymsg, tid1, myTid, MSG_STRING, text, len, MF_STD_SNDR);
            msgSend(&mymsg);

            mymsg.dest  = tid2;
            mymsg.flags = MF_STD_COPY;
            msgSend(&mymsg);

            textcopy = strdup(text);
            msgBuildSend(tid3, myTid, MSG_STRING, textcopy, len, MF_STD_RCVR);
        }
        else
            recInfo(myTid, "%s never got TASK_ARGS text\n",
                    msgMyTaskName(myTid));
        break ;

    case MSG_STRING:
        recInfo(myTid, "'%s' received from task %x\n",
                (char *)msg->data, msg->source);
        break;

    case MSG_ATERM:
        recInfo(myTid, "%s received ATERM\n", msgMyTaskName(myTid));
        break ;

    case MSG_PTERM:
        recInfo(myTid, "%s received PTERM\n", msgMyTaskName(myTid));
        break ;
    }
    return (msgDefaultProc(myTid, msg)) ;
}

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

/*
** File:        Imakefile
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     UNIX Imakefile for Exercise 3
*/

LDLIBS += -lrec -lmsg -lbts -lctos

AllTarget(ex3)

UNIXBinTarget(ex3, ex3.o)

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

%% File:        app.cfg
%% Written By:  Don Lefebvre
%% Date:        Nov 18, 1992
%% Purpose:     Application configuration file for Exercise 3
```

```
chassis   pluto
sequencer pluto

prefix  SequHost 0
args    recSvr  XCTOS_DISPLAY = xterm8:0.0

%%  start Application Executive
%%
prefix  SequHost 0
task    AppExec  /usr2/testbed/course/servers/appexec


%%  Team 1 task config  ------------------------------------------------
%%
prefix  pluto 0
chdir   /usr2/testbed/course/answers/ex3/
task    Team1  ex3
args    Team1  TEXT = Aloha from Team 1


%%  Team 2 task config  ------------------------------------------------
%%
prefix  pluto 0
chdir   /usr2/testbed/course/answers/ex3/
task    Team2  ex3
args    Team2  TEXT = Howdy partner from Team 2


%%  Team 3 task config  ------------------------------------------------
%%
prefix  pluto 0
chdir   /usr2/testbed/course/answers/ex3/
task    Team3  ex3
args    Team3  TEXT = G'day from Team 3
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                          SOLUTION TO EXERCISE 4
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

/*
** File:        ex4.c
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     Solution to Exercise 4
*/

#ifdef OS_UNIX
#  include <stdio.h>
#  include <string.h>
#endif OS_UNIX

#ifdef OS_VXWORKS
#  include <stdioLib.h>
#  include <strLib.h>
#endif OS_VXWORKS

#include <ctos.h>
#include <recLib.h>

#include "team0svr.h"

CtosTask(ex4)


/*****************************************************************************
  ex4 - CTOS task function
*****************************************************************************/
int ex4 (TID_TYPE myTid, MSG_TYPE *msg)
{
    static TID_TYPE  svrTid;
           OPERAND   opr;
           double    result;

    switch (msg->command)
    {
    case MSG_AINIT:
        if ((svrTid = msgTidQuery(myTid, SVR_NAME)) == 0)
            recInfo(myTid, "Could not find %s tid\n", SVR_NAME);
        break ;

    case MSG_AEXEC:
        recInfo(myTid, "%s received AEXEC\n", msgMyTaskName(myTid));

        opr.operand1 =  3.0;
        opr.operand2 =  5.0;
        if (svrTid != 0)
        {
            result = *(double *)msgBuildSend(svrTid, myTid, MSG_REQUEST,
                                             (void *)&opr, sizeof(OPERAND),
                                             MF_WAIT_SNDR);
            if (result == SVR_ERROR)
                recInfo(myTid, "ERROR: could not get result from server\n");
            else
                recInfo(myTid, "result: %f * %f = %f\n",
                        opr.operand1, opr.operand2, result);
        }

        opr.operand1 =   9.27;
        opr.operand2 = -64.88;
        if (svrTid != 0)
        {
            result = *(double *)msgBuildSend(svrTid, myTid, MSG_REQUEST,
                                             (void *)&opr, sizeof(OPERAND),
                                             MF_WAIT_SNDR);
            if (result == SVR_ERROR)
                recInfo(myTid, "ERROR: could not get result from server\n");
            else
                recInfo(myTid, "result: %f * %f = %f\n",
                        opr.operand1, opr.operand2, result);
        }
        break ;
    }
    return (msgDefaultProc(myTid, msg)) ;
}

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

/*
** File:        team0svr.h
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     Header file for TEAM 0 SERVER for Exercise 4
*/

#define  MSG_REQUEST   (MSG_USER + 99)

#define  SVR_NAME      "Team0_Svr"
#define  SVR_ERROR     (-0.123456789)

typedef struct
{
    double  operand1;
    double  operand2;
}
OPERAND;

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

/*
** File:        team0svr.c
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     TEAM 0 SERVER for Exercise 4
*/

#ifdef OS_UNIX
#  include <stdio.h>
#  include <string.h>
#endif OS_UNIX

#ifdef OS_VXWORKS
#  include <stdioLib.h>
#  include <strLib.h>
#endif OS_VXWORKS

#include <ctos.h>
#include <recLib.h>

#include "team0svr.h"

CtosTask(team0svr)
```

```
/*****************************************************************************
  team0svr - CTOS server task function:  multiplies two double FP numbers
*****************************************************************************/
int team0svr (TID_TYPE myTid, MSG_TYPE *msg)
{
          OPERAND  *opr;
          double    result = SVR_ERROR;

    switch (msg->command)
    {
    case MSG_AEXEC:
        recInfo(myTid, "Team 0 Server is available\n");
        break ;

    case MSG_REQUEST:
        recInfo(myTid, "Team 0 Server received request from Task %x\n",
                msg->source);

        opr = (OPERAND *)msg->data;
        if (opr != NULL)
        {
            result = opr->operand1 * opr->operand2;
            recInfo(myTid, "Team 0 Server result: %f * %f = %f\n",
                    opr->operand1, opr->operand2, result);
        }
        else
            recInfo(myTid, "ERROR at Team 0 Server: no operands received\n");

        msgReply(msg, (void *)&result, sizeof(result), MF_STD_COPY);
        break;

    }
    return (msgDefaultProc(myTid, msg)) ;
}
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
/*
** File:        Imakefile
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     UNIX Imakefile for Exercise 4
*/

LDLIBS += -lrec -lmsg -lbts -lctos

AllTarget(ex4 team0svr)

UNIXBinTarget(ex4     , ex4.o     )
UNIXBinTarget(team0svr, team0svr.o)
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
%% File:        app.cfg
%% Written By:  Don Lefebvre
%% Date:        Nov 18, 1992
%% Purpose:     Application configuration file for Exercise 4

chassis   pluto
sequencer pluto

prefix  SequHost 0
args    recSvr  XCTOS_DISPLAY = xterm8:0.0

%%  start Application Executive
%%
prefix  SequHost 0
task    AppExec  /usr2/testbed/course/servers/appexec


%%  Team 0 task config  ------------------------------------------------
%%
prefix  pluto 0
chdir   /usr2/testbed/course/answers/ex4/
task    Team0      ex4
task    Team0_Svr  team0svr
```

```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
                            SOLUTION TO EXERCISE 5
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

/*
** File:        ex5.c
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     Solution to Exercise 5
*/

#ifdef OS_UNIX
#  include <stdio.h>
#  include <string.h>
#endif OS_UNIX

#ifdef OS_VXWORKS
#  include <stdioLib.h>
#  include <strLib.h>
#endif OS_VXWORKS

#include <ctos.h>
#include <recLib.h>

#include "mult0rpc.h"

CtosTask(ex5)


/*****************************************************************************
  ex5 - CTOS task function
*****************************************************************************/
int ex5 (TID_TYPE myTid, MSG_TYPE *msg)
{
    double    result;

    switch (msg->command)
    {
    case MSG_AEXEC:
        result = multiplyRPC(myTid, 3.0, 5.0);

        if (result == SVR_ERROR)
            recInfo(myTid, "ERROR: could not get result from server\n");
        else
            recInfo(myTid, "result: %f * %f = %f\n", 3.0, 5.0, result);

        result = multiplyRPC(myTid, 9.27, -64.88);

        if (result == SVR_ERROR)
            recInfo(myTid, "ERROR: could not get result from server\n");
        else
            recInfo(myTid, "result: %f * %f = %f\n", 9.27, -64.88, result);
        break ;
    }
    return (msgDefaultProc(myTid, msg)) ;
}

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

/*
** File:        mult0rpc.h
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     Header file for TEAM 0 RPC for Exercise 5
*/

/*****************************************************************************
**    multiplyRPC() multiplies two double-precision floating point numbers
**
**    Inputs:   tid  - TID of calling task
**              opr1 - first  operand for multiply
**              opr2 - second operand for multiply
**
**    Returns:  result of multiplying opl * opr2,
**              or SVR_ERROR if an error occurs
**
*****************************************************************************/

#define  SVR_ERROR    (-0.123456789)

double  multiplyRPC (TID_TYPE tid, double opr1, double opr2);

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

/*
** File:        mult0rpc.c
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     TEAM 0 RPC library for Exercise 5
*/

#include <ctos.h>
#include <recLib.h>

#include "mult0rpc.h"
#include "team0svr.h"


double  multiplyRPC (TID_TYPE tid, double opr1, double opr2)
{
    static TID_TYPE  svrTid;
           OPERAND   opr;
           double    result;

    /*  server TID initialization  */
    if ((svrTid == 0) &&
        ((svrTid = msgTidQuery(tid, SVR_NAME)) == 0))
        return(SVR_ERROR);

    /*  send request message & return result  */
    opr.operand1 = opr1;
    opr.operand2 = opr2;
    result = *(double *)msgBuildSend(svrTid, tid, MSG_REQUEST,
                                     (void *)&opr, sizeof(OPERAND),
                                     MF_WAIT_SNDR);
    return(result);
}

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

/*
** File:        team0svr.h
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     Header file for TEAM 0 SERVER for Exercise 5
*/

#define  MSG_REQUEST  (MSG_USER + 99)
```

```
#define  SVR_NAME     "Team0_Svr"

typedef struct
{
    double  operand1;
    double  operand2;
}
OPERAND;

/*
** File:        team0svr.c
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     TEAM 0 SERVER for Exercise 5
*/

#ifdef OS_UNIX
#  include <stdio.h>
#  include <string.h>
#endif OS_UNIX

#ifdef OS_VXWORKS
#  include <stdioLib.h>
#  include <strLib.h>
#endif OS_VXWORKS

#include <ctos.h>
#include <recLib.h>

#include "mult0rpc.h"
#include "team0svr.h"

CtosTask(team0svr)


/*****************************************************************************
  team0svr - CTOS server task function:  multiplies two double FP numbers
*****************************************************************************/
int team0svr (TID_TYPE myTid, MSG_TYPE *msg)
{
          OPERAND  *opr;
          double    result = SVR_ERROR;

    switch (msg->command)
    {
    case MSG_AEXEC:
        recInfo(myTid, "Team 0 Server is available\n");
        break ;

    case MSG_REQUEST:
        recInfo(myTid, "Team 0 Server received request from Task %x\n",
                msg->source);

        opr = (OPERAND *)msg->data;
        if (opr != NULL)
        {
            result = opr->operand1 * opr->operand2;
            recInfo(myTid, "Team 0 Server result: %f * %f = %f\n",
                    opr->operand1, opr->operand2, result);
        }
        else
            recInfo(myTid, "ERROR at Team 0 Server: no operands received\n");

        msgReply(msg, (void *)&result, sizeof(result), MF_STD_COPY);
        break;

    }
    return (msgDefaultProc(myTid, msg)) ;
}

/*
** File:        Imakefile
** Written By:  Don Lefebvre
** Date:        Nov 18, 1992
** Purpose:     UNIX Imakefile for Exercise 5
*/

CPPFLAGS += -I/usr2/testbed/course/servers/
LDLIBS   += -L/usr2/testbed/course/servers/

LDLIBS   += -lrec -lmsg -lbts -lctos -lmult0

AllTarget(libmult0.a  ex5  team0svr)

UNIXBinTarget(ex5       , ex5.o      )
UNIXBinTarget(team0svr , team0svr.o)
UNIXLibTarget(libmult0.a, mult0rpc.o)

%% File:        app.cfg
%% Written By:  Don Lefebvre
%% Date:        Nov 18, 1992
%% Purpose:     Application configuration file for Exercise 5

chassis   pluto
sequencer pluto

prefix  SequHost 0
args    recSvr  XCTOS_DISPLAY = xterm8:0.0

%%  start Application Executive
%%
prefix  SequHost 0
task    AppExec  /usr2/testbed/course/servers/appexec


%%  start team 0 test task
%%
prefix  pluto 0
chdir  /usr2/testbed/course/answers/ex5/
task   Team0      ex5


%%  start servers used by test task
%%
prefix  pluto 0
chdir  /usr2/testbed/course/servers/
task   Team0_Svr  team0svr
%task   Team1_Svr  team1svr
%task   Team2_Svr  team2svr
```