REMOTE OBSERVING PROJECT
Status Report No. 2

A. Balestra, P. Marcucci, M. Pucillo,
P. Santin, G. Sedmak, C. Vuerli

November 21st, 1990

Astronomical Observatory of Trieste, 1335

Introduction

The work presented in this report was carried on by staff of the Astronomical Observatory of Trieste (OAT), as a part of a collaboration agreement with the European Southern Observatory (ESO). The aim of this collaboration is to perform test remote observations from Trieste, Italy, with the ESO telescope NTT, located in Chile, in the course of 1991, using leased links going via ESO Garching, Germany.

This study is based on the NTT control and acquisition system [1], designed and implemented at ESO and presently in operation. In particular the code of *Pool Management Software* [2] is taken from there and is now being ported, in agreement with staff of the Electronic Group of ESO, for the implementation on UNIX machines using the C language.

Notes on the conversion of the Pool software

According to the agreement between ESO and OAT mentioned above, the software for the management of the Pool has been completely rewritten, starting from the original one, written in Fortran on a HP 900/A computer system, to an almost standard, portable code based on C (Kernigham & Ritchie) language for Unix based computer systems. Even if the main goal was the implementation of a portable code, it is assumed that, for any discrepancy among the possible different flavours of Unix, the SUN workstation is the target computer system.

The following comments apply to the conversion :

The data structure has been completely redesigned, maintaining the same functionality.

To avoid ambiguities, all integer variables are defined as

- `int` or
- `int[]`.

Character strings are defined as `char[]` and are null terminated.

Logical variables are defined as `int` with the convention

- `0 FALSE`
- `!0 TRUE`.

The equivalence between the C structures and the old Fortran variables is listed below.

Definitions of memory data structures for the shared memory of Pool programs used by all pool related programs to share memory segments

C	FORTRAN
struct file_desc	
{	
char plfname [PL_FNAMELEN];	ibuf(1)...ibuf(23)
char frecname [PL_FRNAMELEN];	ibuf(25)...ibuf(29)
char frectype [PL_FINAMELEN];	ibuf(30)...ibuf(34)
char fildef[PL_FDNAMELEN];	ibuf(47)...ibuf(51)
char passwd [PL_PWLEN];	ibuf(35)...ibuf(37)
int node;	ibuf(24)
int fhandle;	????
int fixflg;	ibuf(38)
int globflg;	ibuf(39)
int statflg;	ibuf(40)
int diskflg;	ibuf(41)
int maxrec;	ibuf(42)
int userec;	ibuf(43)
int nbytes;	ibuf(44)
int nitems;	ibuf(45)
int rwrec;	ibuf(46)
int recnam0;	ibuf(52)
int itmnam0;	ibuf(53)
int plsav;	ibuf(54)
int rootpos;	ibuf(59)
int nlock;	ibuf(60)
int locker;	ibuf(61)
int delflg;	ibuf(62)
int rec0;	ibuf(63)
int item0;	ibuf(64)
char fill [166];	####
};	

```
typedef struct file_desc FDESC;
```

C	FORTRAN
struct rec_type	
{	
char itmname[PL_ITMNAMELEN];	ibuf(1)...ibuf(3)
char itmtyp;	ibuf(4)
int itmsiz;	ibuf(5)
int itmlen;	ibuf(6)
int rwflg;	ibuf(7)
int firstw;	ibuf(8)
int firstb;	ibuf(9)
int itmnum;	ibuf(15)
int itmpos;	ibuf(16)

```
   char    fill [28];                  | ####
   };                                  |
-------------------------------------------------------------------------

typedef struct rec_type IDESC;

-------------------------------------------------------------------------
              C                        |              FORTRAN
---------------------------------------|---------------------------------
                                       |
struct rec_name                        |
   {                                   |

   char    recname[PL_RECNAMELEN];     | ibuf(1)...ibuf(3)
   int     recnum;                     | ibuf(6)
   int     recpos;                     | ibuf(7)
   int     locker;                     | ibuf(8)
   char    fill[12];                   | ####
   };                                  |
-------------------------------------------------------------------------

typedef struct rec_name RDESC;

-------------------------------------------------------------------------
              C                        |              FORTRAN
---------------------------------------|---------------------------------
                                       |
struct scratch                         | integer*2  scratch(28)
   {                                   |
   int    nfiles;                     | scratch(1)
   int    nlock;                      | scratch(3)
   int    node;                       | scratch(4)
   int    updtim;                     | scratch(5)...scratch(6)
   int    plstat;                     | scratch(7)
   };                                  |
-------------------------------------------------------------------------

typedef struct scratch  MISC;

-------------------------------------------------------------------------
              C                        |              FORTRAN
---------------------------------------|---------------------------------
                                       |
struct id_tab                          | integer*2  idtab(64,4)
   {                                   | idtab(k,1) : id     k = 1..64
   int    nentries;                   | idtab(k,2)          k = 1..64
   int    subid;                      | idtab(k,3)          k = 1..64
   int    futuse[1];                  | idtab(k,4)          k = 1..64
   };                                  |
-------------------------------------------------------------------------

typedef struct id_tab  IDTAB;


struct pool_comm
  {
  int      routines[SUBR_LEN];
  MISC     misc;
  IDTAB    idtab [IDTAB_LEN];
```

```
    FDESC     fdesc [FDESC_LEN];
    IDESC     idesc [IDESC_LEN];
    RDESC     rdesc [RDESC_LEN];
    };

  typedef struct pool_comm PLCOMM;

  /*****************************************************************/

  PLCOMM   *plcomm;                          /* general Pool Common
Structure */

  /*****************************************************************/
```

For the disk files (both hard disk and Ram disk) the stream concept is adopted. As a consequence, the record definition is a logical one, and a flexible read/write mechanism is used (e.g. for items, arrays). As an example the format of the Root file has been changed, since we do not need any more a record to write the first -nfiles- variable; the new format is :

nfiles(rec1)(rec2)...(recn).

The Unix unbuffered functions for I/O are used. They are less efficient with respect the correspondent buffered C I/O function, but the sharing mechanism of the Pool makes it mandatory.

The old protocol has been maintained for the inter-process communication with the Pool Manager.

The references in the following are to the list of open problems/questions about the code conversion reported in the *Status and open questions* section.

No Fortran application has been foreseen on top of the pool library, so no Fortran to C library is provided.

A separate document, reported in the Appendix A, presents the standards we follow in software preparation at the Trieste Observatory.

A table of the current status of the conversion is given below:

Old FORTRAN names	Author	New C names
	PS	pool.h
	PS	pldef.h
	PS	pldat.h
	PS	pllocdat.h
	PS	plsmatt.c
	PS	plsmdet.c
	PS	mvbits.c
addentry.ftn	PS	addentry.c
askitem.ftn	CV	plaskitem.c
*-- bckpp.ftn		
changerec.ftn		
*-- ckrec.ftn		
*-- cmsn1.ftn		
compare.ftn	PS	compare.c
*-- compo.ftn	MP	plcomp.c
dcod.ftn	PS	--------
*-- ec.ftn		
*-- ec1.ftn		
enterpool.ftn	PS	plsbin.c
exitpool.ftn	PS	plsbout.c
fastread.ftn	PS	--------
fastwrite.ftn	PS	--------
fromfile.ftn		
getaddress.ftn	PS	pladget.c
getdcb.ftn	PS	--------
getfileinfo.ftn		
getinfo.ftn		
getnames.ftn		
*-- inifi.ftn	CV	plflinit.c
io.ftn		
itnamtonum.ftn		
itnumtonam.ftn		
libcomp.ftn		
*-- lipoo.ftn		
*-- lipu.ftn		
lock.ftn	PS	lock.c
*-- modpool.ftn		
*-- modpool_old.ftn		
mvbuftocar.ftn		
*-- pomgr.ftn	PS	plmgr.c
*-- poolclient.ftn		
pooldelfile.ftn	PS	plfldel.c
poolinit.ftn	PS	plinit.c
poollockfile.ftn	PS	plfllock.c
poollockrecord.ftn	PS	plrclock.c
poolreadarray.ftn	PS	plarrd.c
poolreaditem.ftn	PS	plitrd.c
poolreadrecord.ftn	PS	plrcrd.c
poolreinit.ftn		
poolreporterror.ftn	CV	plerr.c

```
*-- poolstat.ftn                AB    plstat.c
    poolunlockfile.ftn          PS    plflunlk.c
    poolunlockrecord.ftn        PS    plrcunlk.c
    poolwritearray.ftn          PS    plarwr.c
    poolwriteitem.ftn           PS    plitwr.c
    poolwriterecord.ftn         PS    plrcwr.c
*-- pret.ftn                 ------------------
    printitem.ftn
    printrecord.ftn
*-- puflp.ftn
    qreaditem.ftn           PS    plitqrd.c
    qreadrecord.ftn         PS    plrcqrd.c
    qwriteitem.ftn              PS    plitqwr.c
    qwriterecord.ftn        PS    plrcqwr.c
*-- razbu.ftn
*-- razpool.ftn
    recnamtonum.ftn
    recnumtonam.ftn
*-- remotepoolserver.ftn
*-- restorfile.ftn
    rrec.ftn                PS    --------
*-- savefile.ftn
*-- setmarlas.ftn
*-- sos.ftn
*-- unlok.ftn
*-- wait.ftn
    where.ftn               PS    where.c
    where_hash.ftn
    wrec.ftn                PS    --------
    writeitem.ftn               PS    itemwrite.c
*-- wtfp.ftn
```

Status and open questions

POMGR --- plmgr

PS - All data structures redefined.

PS - IdAdd of various programs substituted with a constant value, like the keys of message queues.

PS - All variables transformed to int

PS - Changed the root file format : nfiles,rec1,...recn,..

PS - The node name in the file pathnames must be defined

PS - I/O : UNIX functions used, less efficient, but consistent with file sharing

PS - idtab is present in a loop on150

PS - -7654 magig number ???

PS - see point # 13 on original list : incoherent logic

PS - flag delflgset to1 (0 in the original file)

PS - classrep has been substituted by answflg in the communication protocol, used as an answer wait flag.

PS - idtab structure has been modified : it is an array where each program has a private slot (=key). No more dynamic entry search

PS - following idtab structure modification actions 7 and 8 have been modified, too. 7 is activated by plinit, behaves as a communication check and is used to zero the entry. The same for action 8.

WRITEITEM --- itemwrite

PS - The control if it is a RAM disk file is based on dcb ?? Why not with the flag ?

PS - progid has been canceled from the call

GETDCB

PS - withdrawn, substituted with the introduction of the fhandle parameter, data file handle, in fdesc.

WHERE ---- where

PS - call parameters changed, according with ADDENTRY

COMPARE

PS - withdrawn

ADDENTRY --- addentry

PS - in the original file n2 is modified in output, nutil is not upgraded

FASTREAD

PS - withdrawn, substituted with RAM disk functions

FASTWRITE

PS - withdrawn, substituted with RAM disk functions

RREC

PS - withdrawn, substituted with UNIX functions

WREC

PS - withdrawn, substituted with UNIX functions

POOLINIT --- plinit

Following the modification of idtab (see plmgr), plinit no more asks the pool manager for an entry, but is used to initialize communications and to zero idtab

EXITPOOL --- plsubout

PS -

GETADDRESS --- pladget

PS - Arguments have been modified, separating names and numbers.

PS - if enterpool fails it exits and calls exitpool !! in this way it decrements once too many !!

PS - ??? clarify the signed '+1'

PS - ???? if rec and item are defined by names, the exit value is defined in posr and posi. Where are they defined and passed back ????

LOCK --- lock

PS - call sequence has been changed, according to getaddress/pladget

PS - password logic must be revised

PS - error management in IPC must be revised (EVERYWHERE!!)

POOLLOCKFILE --- plfllock

PS - call sequence has been changed according to lock/getaddress/pladget

POOLLOCKRECORD --- plrclock

PS - call sequence has been changed according to lock/getaddress/pladget

POOLUNLOCKFILE --- plflunlk

PS - call sequence has been changed according to lock/getaddress/pladget

POOLUNLOCKREC --- plrcunlk

PS - call sequence has been changed according to lock/getaddress/pladget

POOLDELFILE --- plfldel

PS - call sequence has been changed according to lock/getaddress/pladget

POOLREADRECORD --- plrcrd

PS - call sequence has been changed

PS - fnum, rnum should be saved???

POOLWRITERECORD --- plrcwr

PS - call sequence has been changed

POOLREADITEM --- plitrd

PS - call sequence has been changed

PS - fnum, rnum, inum should be saved ???

POOLWRITEITEM --- plitwr

PS - call sequence has been changed

QREADRECORD --- plrcqrd

PS - call sequence has been changed

QWRITERECORD --- plrcqwr

PS - call sequence has been changed

PS - password control logic : initi* is magic ?

PS - lock control logic : if it is locked by another program it asks plmgr anyway, just to get an error : useless !

PS - control on RAM and on type 2 ???

QREADITEM --- plitqrd

PS - call sequence has been changed

PS - if item type = X(string) what is the length in bytes and what the size ???

QWRITEITEM --- plitqwr

PS - call sequence has been changed

PS - lock control logic : if it is locked by another program it asks plmgr anyway, just to get an error : useless !

PS - control on RAM and on type 2 ???

QREADARRAY --- plarqrd

PS - the control on pool revision is missing ?? (like oter qrd)

QWRITEARRAY --- plarqwr

PS - the control on pool revision is missing ?? (like oter qrd)

POOLSTAT --- plstat

AB - fourth option (show pool access) has no path to it

AB - how about substituting dispatchlock and goprv ?

Auxilary software tools

Interfaces with UNIX utilities - SYSV flavour

Two interfaces have been written in order to easily use the sysV facilities reagarding the shared memory, the semaphores and the messages. These modules contain a certain number of functions that allow normal operations, such as open, close, get informations, and the operations related to the particular module (e.g. read and write for shared memory or set and clear for semaphores). The functions supply a high degree of flexibility, allowing the user to manage successfully many tasks; only for esotheric programming the use of the original functions may be useful.

An important remark regards key and identifiers: the user is free to use any integer, positive and different from zero, as a key; the identifiers returned by the open calls are unique for every key.

SHMLIB module

This module is an interface between the shared memory system calls of UNIX sysV (shmget, shmop, shmctl) and a generic user.

The actions allowed by the library are:

- a. open a shared memory segment.
- b. attach and detach a shared memory segment to and from the user data area.
- c. write and read on and from a shared memory segment.
- d. get informations about a shared memory segment.
- e. close a shared memory segment.

a. open

A shared memory segment can be opened in two ways: the first requires that the segment has been already opened by another (or the same) user, the second creates the segment if it does not exist or simply opens it, if already created. Actually it is not possible only to create a segment returning error if already existent.

The user is free to use any integer 0 as a key; the function returns a segment identifier that is the same for all the processes (if the key is the same). The total number of segments and their size are system limited.

b. attach and detach

Once the segment has been created, it is necessary to have its address in the user data area. This is accomplished by the shmadd call; with the returned address, all the normal operations with pointers (e.g. memcpy, strcpy, ...) are possible. If the segment is no longer necessary to the user it is possible, but not necessary, to detach it from the user data area. Also the number of attached segment to a process is system limited.

c. write and read

The library includes two functions in order to easily accomplishing the read and write operations, but the user is free to use other system calls, as memcpy or bcopy, to do it. The library functions allows the use of an offset useful to correctly positioning data in the segment; the writing routine also checks if there is enough space in the segment to perform the requested write operation; an error is returned if not enough space is available.

d. get informations

It is possible to obtain two types of informations about a previously created segment: its size and the number of currently attached processes. The former information is useful to avoid that the write operation fail or is not completely performed; the latter is useful to avoid failure of a close operation.

e. close

After a close function is performed, the segment is no longer accessible by anyone. The close operation is successful only if there are no more processes attached to the segment and who is performing the operation has the rights to do it. Otherwhise, the segment is not closed and an error is returned. The super-user can close the segment anyway.

SEMLIB module

The interface towards sysV semaphores allows simple use of this utility; it should be noticed that semaphores are created as sets, i.e. more semaphores can have the same key and an operation can be atomically performed on a whole set. Another feature to be pointed out is that the main utility of semaphores is strictly connected with shared memory.

Five tasks are provided:

- a. open a semaphores set.

- b. free a semaphores set.
- c. set and clear a semaphores set.
- d. get informations about a semaphores set.
- e. close a semaphores set.

a. open

Only a limited number of semaphores set can be created. Opening is performed in two ways: open only if set is already existent, open if existent and create if not. In the future it will be possible to only create a set, returning error if not already existent.

b. free

This function allows the user to put all the semaphores in a set to the same value. The operation is atomically performed.

c. set and clear

The library implements all the possible operations on semaphores.

These are of three types:

- add a value (always possible)
- subtract a value (possible if the semaphore value is greater than it)
- zero operation (go through only if semaphore value is zero)

All the operations can be atomically performed on the whole set of semaphores or only on a part of them. Blocking and no-blocking mode are both possible: in blocking mode the function will not return until the operation is possible; otherwhise the function returns with a warning.

d. get informations

The only information tha can be recovered is the value of a semaphore. It is not possible to get informations about a whole set.

e. close

The operation is possible only if the user has the rights to do it; the super-user always can.

IPMLIB module

Easy interprocess communication is performed by mean of this library. The user is copletely free to use his own queues structure. Pay attention to system defined parameters as the maximum size of messages.

Four kinds of operation are available:

- a. open a message queue.
- b. get informations about a message queue.
- c. send and receive messages.
- d. close a message queue.

a. open

Open operation is provided in two ways: the first opens the queue only if it is already existent, the second creates the queue if nobody did it, just opening it if this is not true. The possibility of only create without opening if already existent will be implemented in the near future.

b. get informations

Three informations about a queue can be obtained. The first is the total size in bytes of an already opened queue; the second is the number of messages pending on that queue; the third is the size, in bytes, of the remaining space on it.

c. send and receive

c.1 send

Sending a message is actually the operation of writing on a certain queue. This is accomplished specifying in the call who is the sender, the identifier of the queue to write on, how many bytes to send and the text of the message. A check is performed if there is enough space in the queue for the message.

c.2 receive

As send is a write operation, receive is a read operation. The user has to supply the identificator of the queue on which the operation must be performed, the maximum number of bytes to read and the timeout. On return, the sender type, the number of bytes actually read and the message will be provided. Depending on the timeout value, the operation will wait until a message is available (timeout = -1), wait n seconds (timeout = n 0) returning a warning if no message is available, wait 0 seconds returning immediately either a message is present or not.

d. close

When a queue is no longer useful, it can be closed: only who has the proper rights can do it. The super-user, as always, can do it anyway.

Interfaces with UNIX utilities - BSD flavour

IPTLIB module

Easy exploitation of BSD streams sockets is accomplished by mean of this library. Inter- node communications are possible and easy to implement. The only requirement is the definition of a service and his own port in the file /etc/services. The use of streams sockets with TCP protocol guarantees sequenced, reliable, two ways connection based byte streams.

An important remark regards the socket descriptors: they behave the same way as normal file descriptors; all the operations possible on the file descriptors are also allowed on socket. This last statement could not be true in some cases: this is due to different implementation of the socket library. The best way to manage this is reading carefully related system manuals.

Four tasks are provided:

- a. open a socket.
- b. accept a connection.
- c. attempt to connect.
- d. close a connection.

The send and receive tasks are not provided because the usage of system calls like read, write, send, recv is straightforward; an implementation of "ad hoc" routines would make heavier the module employ.

a. open

The open function accomplishes three operations: the creation of a socket, its binding to a name and the declaration of the readiness of the socket to accept connection. The latter action also defines a queue where a limited number of requests are stored.

Among the requested parameters are the remote host name and the service name; these names must be defined in the two files /etc/hosts and /etc/services.

The user is also requested to specify if the calling program is a server or a client: in the first case the local address of the socket is set to "wildcard address" and the listen operation is performed; otherwise the local address is set to the host name

and no listen is done.

b. accept

The accept operation is performed by the server program. It creates a new socket through which successive communications will take place. The default behaviour is blocking until a request is arrived; non blocking mode will be implemented in the near future. On return the function supplies the internet address of the client in . notation (i.e. xxx.yyy.www.zzz).

c. connect

An attempt to connect is the complement to the accept operation. The client must perform it in order to establish the connection with the server. The client will communicate on the same socket created by the open operation.

d. close

Once a connection is no longer of interest, it can be easily closed. A close operation will result in a reading of 0 bytes for the other process; this condition can be safely used as a signal of connection reset.

After a close is called, any unsent data are sent before the socket is actually closed. Any unreceived data are lost.

RAMDISK LIBRARY

PURPOSE

The ramdisk library can be used in the interprocess communication among processes running on the same machine. Two or more processes on the same machine may exchange data and, more widely, messages using common locations of the RAM, in which all communicating processes may write new informations or read the contained information. The ramdisk library is based on these concepts.

IMPLEMENTATION

The UNIX Sys V Operating System allows the processes to share portions of the RAM (Shared Memory) through a set of calls to the kernel of the system. The portions in which the Shared Memory is divided are named segments and each segment has associated a segment identifier. The System calls allow the processes to get a segment identifier, to link a segment to the local data area of the process, to unlink a segment and to perform some operations on previously linked segments.

Using the system calls, a library named SHMLIB (Shared Memory Library) was implemented; through the SHMLIB library all possible operations on the segments of the shared memory can be easily implemented.

The RAMDISK library then was written using the SHMLIB library together with another library named SEMLIB (Semaphores library) used in order to resolve concurrent accesses to the same segment by more processes at the same time.

More precisely, more processes can read from the same segment at the same time (more readers) but only one process at a time can write (one writer only); furthermore a writer locks all readers.

Three semaphores for each shared memory segment were used realizing the scheme above. The three semaphores are:

- A : writers controller (binary)
- B : number of actually operating writers (binary)
- C : number of actually operating readers (not binary)

The scheme is implemented in the following way.

READERS:

- Pass if : A whatever value ; B = 0 ; C whatever value
- Entry operations : C = C + 1
- Exit operations : C = C - 1

WRITERS:

- Pass if : A = 1 ; B whatever value ; C = 0
- Entry operations : A = A - 1 ; B = B + 1
- Exit operations : A = A + 1 ; B = B - 1

STRUCTURE

The RAMDISK library is constituted by eight functions:

```
static long RDMINI ()
```

Prepares the ramdisk environment for a process

```
static long RDMINF (n_el,filename,size,flags,p_el)
```

Attempts to create a new shared memory segment

- n_el - the key for the new segment
- filename - the name of file that has to be loaded into the ramdisk
- size - additional space for the new segment

- flags - in which way the segment must be opened (see below)
- p_el - pointer to an item of the segment 1

```
long RDMOPN (filename,flags,p_size,p_key)
```

Attempts to open a segment for the specified file

- filename - the name of file to be opened
- flags - in which way the segment must be opened (see below)
- p_size - the total size of the segment
- p_key - the key of the opened segment

```
long RDMREA (key,ptr,nbytes)
```

Reads a specified number of bytes from a shared memory segment

- key - the key of the segment
- ptr - where the read bytes will be stored
- nbytes - the number of bytes to be read

```
long RDMWRI (key,ptr,nbytes)
```

Writes a specified number of bytes into a shared memory segment

- key - the key of the segment
- ptr - where the bytes to be written will be read
- nbytes - the number of bytes to be written

```
long RDMSEK (key,dist,mode)
```

Sets the offset (pointer) inside a segment

- key - the key of the segment
- dist - set the value of the offset (inside the segment)
- mode - in which way the value of 'dist' must be interpreted
  - 0: new offset = dist
  - 1: new offset = old offset + dist
  - 2: new offset = end of file + dist

```
long RDMFLU (key,filename)
```

Saves the content of the specified segment

- key - the segment key

- filename - the name of saving file

```
long RDMCLO (key,filename,mode)
```

Attempts to close a segment

- key - the segment key
- filename - the name of saving file
- mode - in which way the segment must be closed

The last six functions constitute the RAMDISK user interface because only these functions are visible to the user programs. The first two static functions (RDMINI and RDMINF) are internal functions (called by other functions of the library).

REQUIREMENTS

Using the RAMDISK library, the RAMDISK.H header file must be included. The ramdisk.h file defines some useful constants to be employed by user programs.

We can divide the ramdisk constants in three groups:

Opening mode constants:

These constants can be or'ed forming a bit-mask that will be passed to RDMOPN function in order to establish in which way a segment must be opened. The constants are:

RDM_READONLY : The process can only read the just opened segment

RDM_RDWR : Both read and write operations allowed

RDM_APPEND : The process write at the end of the file

RDM_CREATE : Create the segment if it does not exist

RDM_SCRATCH : Segment used writing/reading temporary informations

Closing mode constants:

These constants can be or'ed forming a bit-mask that will be passed to RDMCLO function in order to establish in which way a segment must be closed. The constants are:

RDM_DETACH : Detach (unlink) the segment

RDM_SAVE : Save the segment in a disk file before detach it

RDM_DELETE : Attempts to delete a segment (success if no further processes attached)

Error codes:

The error codes can be used by the user program in order to test the execution status of the ramdisk functions. The error codes are long values, so all ramdisk functions are declared as long functions. As well the declaration of the ramdisk functions is contained in the ramdisk.h header file.

USING RAMDISK LIBRARY

As explained above, a program using the ramdisk library must include the ramdisk.h header file. When a file must be loaded on the shared memory, the rdmopn function has to be called returning the segment key in which the specified file resides, and the total segment size.

When the program calls a ramdisk function, we suggest to test alwais the execution status in order to know if the function worked well. If the called function fails, the error returned allow you to understand the reason that caused the failure.

After the file was opened, the other functions can be used on the segment. We recommend to close the segment after the completion of the operations on the segment; dead processes still attached prevent the segment deletion.

Hardware Procurement and Tests

The status of the communication hardware procurement is as follows :

- 2 Retix 4820 remote bridge — installed and tested
- 2 Streamline 7600/4 multiplexors — installed and tested
- 2 voice compressors/codecs — installed and tested
- Video system — expected installation : end 1990

Following ESO decision on SAT-PRISME system we have issued an order to IBC to procure a PRISME-consultation system consisting of a Display Control and a Decoder Board. This hardware will be procured on the funds (60 MLit) given by Italian C.R.A. to Astronomical Observatory of Trieste for 1990 programme.

Moreover another order has been issued to IBC to procure the complete set of hardware needed to the acquisition and transmission of images, so that the complete system will be tested on site before the tests with ESO. This hardware will be procured using funds available to the Astrophysical Technologies Research Group of the Astronomical Observatory of Trieste.

References

[1] Raffi G., Biereichel P. Gilli B., Gustafsson B., Roche J., Wirenstrand K.
NTT control/acquisition system as a prototype for the VLT
SPIE Conference, Tucson, 1990

[2] NTT documentation - AsterX Software Manual
ESO publication , 1990