Scramnet and AO datataking

phil perillat

I. Overview.

A. What is scramnet?

Scramnet is a reflective memory system consisting of nodes (sbus/vmebus cards) connected in a ring configuration via fiber optics. Each node contains memory (128k bytes in our case). When any node writes into its memory, the information is propagated around the ring so that it appears in the same memory location in all the other nodes on the ring. Ring transfer rates are 5 to 12 Mb/sec (depending on the protocol used).

B. How is it used at the observatory?

The data acquisition system at the observatory consists of 3 VME crates and 2 sun workstations. Scramnet is used to pass device configuration and status from the cpus that control the devices to the cpus that need this information for generating headers, displaying device state, and for logging.

II. Scramnet memory layout.

A. Scramnet memory blocks.

Scramnet memory is divided into blocks. Each block contains information relating to a particular datataking function (eg. turret, tiedown, az/gr/ch drives..etc). Blocks have the following properities:

1. Block properties:
   - Blocks start on 16 bytes boundaries for alignment purposes.
   - There is a 16 byte header at the start of each block.
   - The data follows the header and is double buffered (this cuts down on collsions between writers and readers. See the protocol below).
   - The protocol used to access scramnet allows only a single program to write to a given block but permits multiple readers (see protocol).

B. Locating memory blocks within scramnet memory.

The scramnet memory layout is defined in the file: *~phil/vw/devs/scrm/scrmLoad.h.* It defines an array of structures. Each structure holds the start position, block length, and single buffer length of a block.

```
typedef struct {
    int     offsetBytes;   /* from start of scrmnet.computed at boot up*/
    int     lenBlk;        /* len blk. including blk header*/
    int     lenBuf;        /* 1 buffer in bytes*/
} SCRM_ADR_INFO;
```

```
static SCRM_ADR_INFO scrmAdrAr[]= {
        {    0,   2048, sizeof(SCRM_B_PNT) } , /* SCRM_PNT_IND 0 */
        {    0,   2048, sizeof(SCRM_B_AGC } , /* SCRM_AGC_IND 1 */
        {    0,   3072, sizeof(SCRM_B_TT)   } , /* SCRM_TT_IND    2 */
        {    0,   2048, sizeof(SCRM_B_PNT) } , /* SCRM_TIE_IND  3 */
            ...
             ....
          }
```

scrmAdrAr[i].offsetBytes is the starting position relative to the start of scramnet memory for block *i*. These values are filled in at boot time on vxWorks and at program startup on the Sun.

scrmAdrA[i].lenBlk is the entire block length including the header and two data buffers. It should be rounded up to be a multiple of 16 bytes. Block i+1 will be located lenBlk bytes from the start of block i. lenBlk is normally made larger than hdrlen + 2*lenBuf. This allows you to change lenBuf without modifying the locations of any blocks that follow this one.

scrmAdrA[i].lenBuf is the length of a single copy of the data buffer. the length is defined via a structure SCRM_B_XXX. The structures are in *~phil/vw/h/scrm-BlkXXX.h* where *XXX* is the abbreviation for the block in question.

The index for scrmAdrA determines which block to access. These indices are defined in *~phil/vw/h/scrm.h*.

C. Locating the data buffer within the memory block.

Each block contains a header followed by a header. The block header is:

```
typedef struct {
    int    rdInd;      /* 0 or 1*/
    int    filler;
    int    modCnt[2];  /* 1 for each buffer*/
} SCRM_BLK_HDR;
```

rdInd tells readers which of the two buffers to read from. modCnt gets updated every time the writer modifies the block.

III. Synchronizing access to scramnet.

Scramnet memory is accessed by different computers. Because of this, local semaphores can't be used to synchronize the accesses. The properties of the protocol are:

1. Writers never block when trying to write into scramnet memory.
2. Readers access memory without blocking and they can tell whether the block was being modified while they read it (so they can try again).

There are two data buffers in the block. The block header has a read index pointing to the current read buffer and a modCnt modification counter that gets updated after every write to the buffer. The access algorithm is:

w1. writer computes write index to be (rdIndex + 1) %2.
w2. writer sets the modCnt for buffer its is modifying to be negative.
w3. writer modifies buffer.
w4. writer makes modCnt positive then increments it.
w5. writer sets rdIndex to index of buffer it just udpated.

r1. reader grabs rdIndex and stores current modCnt for this buffer.
r2. reader inputs the data.
r3. reader checks that modCnt for the buffer in use has not changed.

Since there are two buffers, an overlap only occurs if the reader held the buffer for longer than two writes. Since many of the blocks are updated once a second , the chances of getting bad data are reduced.

IV. Code to access scramnet.

The code to access scramnet is stored in *~phil/vw/devs/scrm/scrmLoad.c.* It is run on the sun and on vxWorks. It uses the structure defined in *~phil/vw/h/scrm.h:*

```
typedef struct {
  short       initialized; /* 12345*/
  short       blkInd; /* which scrmblock 0..15 user requested*/
  int         blkOff; /* offset scrmBloclk from start of scrm memory*/
  volatile SCRM_BLK_HDR *pblkHdr;/* ptr to start of header in scrmBlk*/
  int         bufInd; /* which buffer in block we are using*/
  int         modCnt; /* for buffer we pnt to */
  char        *pbuf;   /* ptr to start of data*/
} SCRM_BLK_INFO;
```

A. code for a writer:

The example below modifies the AGC block of

```
#include <scrmLib.h>
#include <scrmBlkAgc.h>
SCRM_BLK_INFO blkI;

If (scrmWrGBlkI(SCRM_AGC_IND,sizeof(SCRM_B_AGC),&blkI) == ERROR)
{ ... error processing }
```

This will initialize the blk_info structure. The index of the block to use is in ~phil/vw/h/scrm.h. The block structure is define in scrmBlkAgc.h. Errors will occur if a bad index is passed in, or the buffer length has changed since the program was last compiled. This call needs to be done once before a writer program can access scramnet. To modify a block in scramnet use:

```
if (scrmWBeg(&blkI)==ERROR)return(ERROR);
bcopy((char*)&newData,blkI.pbuf,sizeof(SCRM_B_AGC));/*load newdat */
scrmWrEnd(&blkI);   /* updates rdInd, modCnt */
```

B. code for a reader.

The example below reads from the AGC block

```
#include <scrmLib.h>
#include <scrmBlkAgc.h>
SCRM_BLK_INFO blkI;
int modCnt;
```

```
If (scrmRdGBlkI(SCRM_AGC_IND,&blkI) == ERROR) { ... error processing }
```

This will initialize the blk_info structure. Errors will occur if a bad index is passed in.. This call needs to be done once before a writer program can access scramnet. To read a block in scramnet use:

```
if (scrmRBeg(&blkI)==ERROR)return(ERROR);/* probably being written*/
bcopy(blkI.pbuf,(char*)&newData,sizeof(SCRM_B_AGC));/*load newdat */
if (scrmREnd(&blkI,&modCnt) == ERROR ) { ../* block was being modified*/
```

When reading an error can occur at Rbeg or Rend. If an error occurs at Rbeg then you can not use the blkI.pbuf to do the copy (since it wasn’t initialized).

V. Differences between vxWorks and Solaris.

A. vxWorks bootUp.

vxWorks has a device driver that runs on then first cpu of the crate that has the VME scramnet board. The driver is *~phil/vw/devs/scrm/scrmDrv.c.* On bootup it makes sure that the board is installed and then initializes the scramnet board.

The driver cpu and other cpus in the crate must then call aoScrmInit to compute the scrmAdrAr values. All of this is done at boot time in the configuration file for the cpu. The defines in the cpu startup file that control scramnet are:

#define INCLUDE_SCRAMNET ... if the board will access scramnet

#define INCLUDE_SCRAMNET_DRIVER ... if this cpu runs the driver.

B. vxWorks compiling.

When compiling vxWorks routines, <scrmLib.h> and <scrmBlkXXX.h> need to be included (in ~phil/h). The systran Scramnet routines are only needed by the driver.

C. Solaris.

On the solaris cpus scramnet memory is moved to local shared memory once a second by the mshmd daemon (~phil/vw/datatk/shm/mshmD.c). This daemon is linked with the systran scramnet code and scrmLoad.c. Other programs then use the local shared memory access routines to get at this information.

VI. Files and program locations.

~phil/vw/devs/scrm - directory holding ao scramnet code

- aoScrmStat.c - vxW/Sun return the scramnet status register in ascii as the return value. on vxW only use on driver cpu.
- scrmBlkLen.c - vxW- prints out current buflen, blockLens.
- scrmDbg.c - vxW/Sun - dumps scramnet board status, registers. On vxW is should only be run on the driver cpu.
- scrmDev.c - vxW device driver.
- scrmIo.c - vxW diagnostic to time io transfer rates. Runs independant of driver. Should not be run while network in use.
- scrmLnk.c - vxW/Sun diagnostic to insert or remove scramnet board from the ring. Don't use while network in use.
- scrmLoad.c -vxW/Sun contains the ao routines to access scramnet. aoScrmInit,scrmWBeg,scrmRBeg, etc...
- scrmLoad.h -vxW/Sun defines scramnet memory layout via scrmAdrAr[].
- scrmstat.sc -Sun. a script that runs on the sun and logs the scramnet board status register for every scramnet board on the network once a minute.
- scrmVwAgc.c -vxW. Dumps the agc scramnet block to stdout.
- scrmVwTst1.c -vxW. Diagnostic to see if a reader ever hangs up. Loops doing scrmRBeg, delay, scrmREnd checking for errors.
- scrmVwTst2.c -vxW. Diagnostic to see if a reader ever hangs up. Loops doing scrmRBeg, delay, scrmREnd checking for errors.

~phil/vw/h - vxW/Sun ao include files.

- scrm.h - defines scm block indices.SCRM_BLK_HDR, SCRM_BLK_INFO.
- scrmLib.h - prototypes for scrmLoad.c
- scrmBlkAgc.h - az/greg/ch scrm block structure.
- scrmBlkPnt.h - pointing program scrm block structure.
- scrmBlkTie.h - tiedown program scrm block structure.
- scrmBlkTt.h - turret scrm block structure.

~phil/vw/load - vxWorks load modules.

- scrmLoad.o -vxW. load module for scrmLoad.c

~phil/vw/shm/ -Sun. shared memory routines. mshmD shared memory daemon.

/usr/local/sys/scramnet/vxPhil - vxW. systran vxWorks code version 1
/usr/local/sys/scramnet/vxV1.03Phil - vxW. systran vxWorks code version 1.03