The e835 High Voltage Monitor

Robert McTaggart
Pennsylvania State University

February 12, 1996

Abstract

The e835 high voltage monitor is a program that runs concurrent with the data acquisition system and updates the run control concerning the state of the LeCroy 1440 High Voltage System. It is somewhat unique to most facets of the data acquistion system in that its actions do not depend on the state of the run control. For instance, one may stop taking data, but the monitor will keep checking the mainframes unless it is turned off manually by a button on the run control window.

We discuss the various error messages that the monitor will send to the run control and how the monitor was developed. The manner in which the monitor communicates these error messages will be followed by other e835 products in the data acquisition. We also demonstrate how this particular murmur client was constructed.

Contents

1 Development of the High Voltage Monitor

The fortran routines used by the monitor are based upon the original e835 High Voltage Program. The main program has been written in C, hv_monitor.c, and it calls three of the adjusted fortran subroutines. Actual terminal line communication with the LeCroy 1440 mainframes is performed by the same C program, io.c, that the regular high voltage program uses. The fortran routines, stored in checkhv.f, pass back to C the number of channels or mainframes with a problem, and the main program notifies the run control with the appropriate error message.

The communication of variables and strings between fortran and C is in general is compiler dependent, and several examples of such communication are found within the high voltage monitor. C-routines in io.c are called by the fortran routines in checkhv.f, and fortran routines in checkhv.f are called by the main C-routine in hv_monitor.c. Other C-routines in hv_monitor.c write specific error information into the error logfile.

2 Operation of the High Voltage Monitor

During initialization, the high voltage monitor will attempt to open the terminal line which connects fn835x with the daisy-chained LeCroy 1440 mainframes. If the monitor cannot open the terminal line, the monitor will turn itself off and must be restarted from the run control. In this case either the serial port has not been configured correctly, or there is something physically wrong with the serial port. In the future we hope to allow only one access at a time to the terminal line, so the open will fail if another such process is running. This will prevent someone from running the high voltage program to change voltages during a run, or running two monitor executables at the same time.

Also during this initialization, the monitor will read in a set of default voltages into an array and open a file in order to log any errors.

The monitor will then examine each of the 6 mainframes. First, the mainframe is addressed, and a 'status' command is sent, and the response is analyzed to determine subsequent action.

If the mainframe has been disabled or is currently in the 'off' state, then an error message describing the situation is logged to a file. Eventually, a message is sent to the run control saying how many mainframes have been disabled or are off. Then the next mainframe is addressed.

When a null response to the 'status' command is received, the buffer is read out several times(to empty it) and the mainframe is addressed again. If the string received after a 'mainframe' command is not correct, then run control will be notified how many mainframes do not respond. However, if this string is OK, then the 'status' command is tried once more. Failure after this second attempt will yield an error stating that the mainframe responded, but it did not acknowledge the 'status' command.

If the mainframe is found to be in the 'on' state, the monitor checks the actual voltages against a list of default voltages to ± 5 Volts, and against the demand voltages stored within each mainframe. If only the default voltage is different, then a message will be sent notifying run control how many channels have had their default voltages altered. If in addition the actual voltage does not match the demand value, then a message is generated stating how many channels have a hardware error. The mainframe in this case is either not measuring voltages correctly, or there may be a short circuit elsewhere.

Run control will only be notified how many channels have this kind of voltage error. More specific information, such as the channel number, the default, demand, and actual voltages will be written to a file instead of to the run control.

While writing and reading command strings to and from the terminal line, a write or read error may occur. This will not cause the program to shut down, but will generate a warning message. It may be the case that the terminal line is bad, it is connected to the wrong serial port(or misconnected), a channel in one of the mainframes is bad, or the controller to one of the mainframes is in need of repair.

After checking all of the mainframes with sundry fortran routines, a C routine processes the flags generated during this evaluation period, produces suitable error codes for the run control, and sends them to the murmur server.

Outside of a failure to open the terminal line, it is up to the run control to manually turn off the monitor and address the warning messages. In practice, the mainframes will have been turned on or off prior to the start of the data aquisition system via the e835 high voltage program along with any changes to the demand voltages. Some of these problems may be solved without having to stop the data acquisition system, and one may start taking data without running the high voltage monitor.

3 Construction of a Murmur Client

The main program has been written in C so that the monitor can communicate with the DART message reporter, murmur, used by all facets of the data acquisition system. Besides errors, general information from different programs may also be sent to the run control. We detail here how the communication with the high voltage monitor has been established with the run control in the form of a 'client'[1].

The first step in building a murmur client is to obtain a unique facility number. This number is not only unique to other programs used by e835, it is unique to all clients from any fixed-target experiment here at Fermilab.

Next, the client's home directory must be subdivided into bin, include, lib, and src subdirectories. The monitor's home directory is '/ dart/ daqman/ hv_monitor'. hv_monitor.c, checkhv.f, and io.c are placed into the src directory. Type "setenv HV_MONITOR_DIR '/ dart/ daqman/ hv_monitor' ". This will be used later.

In the include directory, a hv_monitor.msg file is created. This .msg file contains the error messages you want to send. One may also input different variables available only at run time into each message, which will be provided by hv_monitor.c.

To manufacture the murmur header files and databases for the main program, one executes the following command:

```
>$MURMUR_DIR/bin/murmessage hv_monitor.msg
$HV_MONITOR_DIR/lib
```

.

In the main program of the client, three header files must be included: mur_user.h, murmur_msg_c.h, and hv_monitor_msg_c.h. The high voltage monitor does not call any other programs that also use murmur message codes, but generally their header files are included as well.

The appendix demonstrates more completely how the error codes are sent from this client to the murmur server. For example, suppose one wishes to send a message that some mainframes are off. In the .msg file we have the string

```
OFF -sev W <(%d) mainframes are currently off.>
```

.

The header file hv_monitor_msg_c.h contains the hexadecimal codes to be used by the murmur server for each message. The following code need only be called once to setup the connection between the monitor and the murmur server:

```
mur_open();
mur_set_proc_name("HIGH VOLTAGE");
status=mur_init_global_stack(10);
```

.

Now one may send the message to the murmur server(err[2] is the number of mainframes that are off, and this number is placed where %d is located above):

mur_stack (MUR_STACK_GLOBAL, HV_OFF, 1, MUR_U_LONG, err[2]);
mur_report_stack (MUR_STACK_GLOBAL , MUR_STACK_FIFO, HV_OFF, NULL);

.

One may also add another string to the message not included in the .msg file by adding another argument to the mur_stack statement:

mur_stack (MUR_STACK_GLOBAL, HV_OFF, 2, MUR_U_LONG, err[2], MUR_APPEND, string);

The makefile in the src directory will place the executable for the monitor in the bin directory after linking io.c, checkhv.f, and hv_monitor.c with the required libraries and include files and compile these three programs.

The last thing to do is to concatenate all the individual product databases that are associated with your client into a murmur database. This is accomplished by the following set of commands typed in at your terminal:

```
>mkdir $HV_MONITOR_DIR/murdb
>cp $MURMUR_DIR/lib/*Db $HV_MONITOR_DIR/murdb
>rm $HV_MONITOR_DIR/murdb/murmur_ftokDb
>mkdir $HV_MONITOR_DIR/sysdb
>$MURMUR_DIR/bin/DbConcatenator -a $HV_MONITOR_DIR/lib
$HV_MONITOR_DIR/sysdb

>$MURMUR_DIR/bin/DbConcatenator -a $HV_MONITOR_DIR/murdb
$HV_MONITOR_DIR/sysdb
```

Other associated clients must have their lib directory concatenated with the main client's sysdb directory.

This will concatenate the client database with the murmur database that was just copied to $HV_MONITOR_DIR/murdb. However, to be included in the run control, the client database must be concatenated with the primary murmur databse, '/ dart/ daqman/ dart/ db/ murmur'.

4 Running the Murmur Client

Only one murmur server at a time can run on fn835x. Hence if a server is currently running and the monitor is started, any messages will be sent to that murmur server. However, if the command 'setenv MURMUR_SERVER_ADD fn835x' has not been given, uninterpreted messages(i.e. message codes that have not been converted into English) will be sent to stdout.

If the data acquisition system is not running, one may start it with the drt_fresh_start command. Here the configuration must be set together with the ip address of the terminal you want the windows to be displayed on.

If one does not wish to use the data acquisition system(and nobody else is using it), then one may use *start_mursrvr* and *stop_mursrvr* to start and stop the murmur server.

Furthermore, it is also possible to run the executable without starting the murmur server. But in this case the hexidecimal codes will be typed directly at your screen. Nevertheless this is an acceptable debugging option if others are running a murmur server.

Those on shift will be able to start and stop the monitor from a button-oriented menu. Thus one may restart the monitor if it crashes, or turn it off for high voltage maintenance(or to stop messages from being sent to murmur). This entails writing a separate .tk file that will be sourced by the main run control window.

References

[1] Laura Mengel et al. *Murmur Client User's Guide*. Fermilab Computing Division PN466. June30,1995.

A The High Voltage Monitor Program

```
#include <stddef.h>
#include <stdlib.h>
#include <fcntl.h>
#include ”mur_user.h”
#include ”mur_def.h”
#include ”murmur_msg_c.h”
#include ”hv_monitor_msg_c.h”

void main(int argc, char **argv)
{
MUR_MSG_STATUS status;      /* pointer for message stack*/
int i;                      /* counter */
int fdes;                   /*file descriptor for open*/
int x;
int openerr;                /*err opening terminal line*/
int readerr;                /*err reading terminal line*/
int writeerr;               /*err writing terminal line*/
int err[6];                 /*other mainframe errors*/
/*
err(1)->err[0]              default values changed
err(2)->err[1]              hardware error
err(3)->err[2]              # of mainframes off
err(4)->err[3]              # of mainframes disabled
err(5)->err[4]              # of mainframes not responding
err(6)->err[5]              # of mainframes responding without yielding a status
/
char string[132];
```

```
extern setuphv;
extern restore_def;
extern compare;

/*==============
INITIALIZATION
============== */
/*open connection to murmur server*/
mur_open();

/*set process name to associate with messages*/
mur_set_proc_name("HIGH VOLTAGE");

/*initiate space for the message stack*/
status=mur_init_global_stack(10);

/*Call the Fortran subroutine that opens the terminal line and
performs some initialization. */

openerr=0;
setuphv_( &openerr);
if(openerr != 0)
{
sprintf(string,"The high voltage monitor is now off.");
mur_stack (MUR_STACK_GLOBAL,HV_OPEN_FAIL,1,MUR_APPEND,string);
mur_report_stack (MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_OPEN_FAIL,NULL);
return;
}

/*Call the Fortran subroutine that reads in the default demand
voltages into an array for the comparison tests.*/

restore_def_( &readerr, &writeerr);

if(readerr != 0)
{
```

```
mur_report_stack (MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_READ_FAIL,NULL);
}

if(writeerr != 0)
{
mur_report_stack (MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_WRITE_FAIL,NULL);
}

x=1;

/*Continuously loop to check high voltages against their default values*/

while (x != 0)
{
compare_( &err, &readerr,&writeerr,&fdes);

if(readerr != 0)
{
mur_report_stack (MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_READ_FAIL,NULL);

}

if(writeerr != 0)
{
mur_report_stack (MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_WRITE_FAIL,NULL);

}


if(err[0]!= 0)
{
mur_stack (MUR_STACK_GLOBAL,HV_DEFAULT,1,MUR_U_LONG,err[1]);
mur_report_stack (MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_DEFAULT,NULL);
}

if(err[1]!= 0)
```

```
{
mur_stack (MUR_STACK_GLOBAL,HV_HARDWARE,1,MUR_U_LONG,err[1]);
mur_report_stack (MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_HARDWARE,NULL);

}

if(err[2] !=0)
{
mur_stack (MUR_STACK_GLOBAL,HV_OFF,1,MUR_U_LONG,err[2]);
mur_report_stack (MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_OFF,NULL);

}

if(err[3] !=0)
{
mur_stack (MUR_STACK_GLOBAL,HV_DISABLED,1,MUR_U_LONG,err[3]);
mur_report_stack (MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_DISABLED,NULL);
}

if(err[4] !=0)
{
mur_stack (MUR_STACK_GLOBAL,HV_NORESPONSE,1,MUR_U_LONG,err[4]);
mur_report_stack (MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_NORESPONSE,NULL);
}

if(err[5] !=0)
{
mur_stack (MUR_STACK_GLOBAL,HV_NOSTAT,1,MUR_U_LONG,err[5]);
mur_report_stack (MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_NOSTAT,NULL);
}
if(err[0]==0 && err[1]==0 && readerr==0 && writeerr==0 && err[4]==0 && err[5]==0)
{
mur_report_stack(MUR_STACK_GLOBAL,MUR_STACK_FIFO,HV_SUCCESS,NULL);
}

/* Perform the high voltage check once every 600 seconds*/
```

```
sleep(600);
}

return;
}
```

B The Monitor Makefile

```
INCLUDES= -I../include -I$(MURMUR_CLIENT_DIR)/include

../bin/hv_monitor: checkhv.o io.o hv_monitor.o
	f77 -g checkhv.o io.o hv_monitor.o\
	-o $@\
	-L$(MURMUR_CLIENT_DIR)/lib -lmuruser

hv_monitor.o: hv_monitor.c
	cc -c $(INCLUDES) -g -o $@ hv_monitor.c

checkhv.o: checkhv.f
	f77 -c $(INCLUDES) -g -o $@ checkhv.f

io.o: io.c
	cc -c $(INCLUDES) -g -o $@ io.c
```

C The Monitor Message File

```
-facility       hv_monitor    160
-prefix         HV

OPEN_FAIL       -sev E <The serial port could not be opened.>
READ_FAIL       -sev W <Error reading the 1440.>
WRITE_FAIL      -sev W <Error writing to the 1440.>
DEFAULT         -sev W <The default voltages of (%d) channels have been changed.>
HARDWARE        -sev E <A hardware error has occurred in (%d) channels.>
OFF             -sev W <(%d) mainframes are currently off.>
DISABLED        -sev W <(%d) mainframes have been disabled. >
NORESPONSE      -sev W <(%d) mainframes do not respond.>
NOSTAT          -sev W <(%d) mainframes respond, but don't return status.>
SUCCESS         -sev S <The high voltage is OK.>
-end
```