RAS TO AMCS

July 21st, 2005

1. Details on the current version:

Here is a wrap-up of what is currently working on the current and first version of the code, and how it is done.

The program has currently 5 classes that are being used:

- TStatusControl_AMCS
- TConnection_AMCS
- TClientAMCS
- TVSP_AMCS
- TMessages_AMCS

There is also the main function located in TRasToAMCS.cpp.

For now, here the sequence of what the program is doing.

A) The program is currently using predefined VSPs, since the VSP file structure and ReadVSP is not determined at the moment. The constructor of the VSP objects is now setting defaults attributes, which are 4 IPs, a port, a system ident, site ident, and timeouts values.
B) The setup function is ready, except for the bulletin and event queue. It takes the VSP attributes, and fills the enabledIPs table.
C) It is able to ping, connect to AMCS, generate greetings, timeout, receive back those messages from AMCS. It is able to reconnect if a physical or software communication was lost.
D) An default event array was created, with default strings in it, simulating the return of the isEventAMCS() method. This array includes the fields for the changed status points. The generateStatusMsg will take this table and generate a status. This works really well, and is really fast.
E) 5 defaults tests points were created on the AMCS demo PC, which are RAS state, RAS Mode, RAS freespace, radar source 1 and radar source 2. They all receive simulated status.
F) The program checks the connection and socket on a regular basis and is able to detect :

- A physical connection lost event within a range of :

ping time< detection time < ping time + timeoutCheckEvents attribute

-A software communication lost:

50 seconds < detection time < 50 seconds + timeoutCheckEvents

2. How the Ping works

The ping command is done using a system command. Using : "system(command)", where command is: ping –c 1 enabledIPs[index]. This command returns 0 or 256 if ok or not. Only one packet is sent. If not successful, a counter is increased. Then another packet is sent. If it is ok, the function will return true. If not , the counter is one again increased and the function will return false. This is very efficient and fast from what I saw during the tests.

3. How the check socket state works

Since everything is on Linux, no properties or method of winsock.h can be used. That means it was more complicated to check the socket state programmatically. I choose a fast way that works really well and can be as fast as Linux to change the state of the socket. It is once again a system command.
I do a netstat only for TCP, and grep for the active port on which the connection was established. I output the result of that command to a temp file. Then I search for the word ESTABLISHED in that file.

Linux will change the ESTABLISHED socket state to CLOSE_WAIT after a connection loss, physical or not, in about 50 seconds to one minute. I've seen cases where it took only 30 seconds, but the usual time is 50 seconds to one minute. It then stays in the CLOSE_WAIT or FIN_WAIT for a really long time, unless the socket is closed.

4. How the generateStatusMsg works

The generateStatusMsg will use the event_array[] (array of char*) to generate the message.. That means that each index is a string that can be easily added at the end of each other using the sprintf and binary operators.

One index of that table looks like
"|dataID|status|color|text|label|English text|French text|*|"

That is similar to what was established to be included in the bulletin structure. Using this table, in a while loop, I add those strings one after another, as many times as there are events. The rest of the message is similar to greetings and timeout messages, so it is generated the same way.

5. Why there is no timer anymore?

Because there is no need for it. Everything can be done using the sleep() call. Timer would not have been interrupted anyway. There were supposed to be 3 timers. Here's how they can be replaced by sleep() calls.

Message timer: The message timer was supposed to be used to wait for the socket to make sure that the AMCS had time to send its message back before I read it. We absolutely don't need a start and stop timer and its and handler for this. This timer is really short anyway.

TCP Connection Timer: The connection timer was supposed to be used in case that the connect function hangs. But this function is just like read or receive, it can be set to a non-block. It will return successful or not, when you try it. This timer won't even be replaced by a sleep(). It won't be used at all, except during the reconnection process. To avoid hammering the servers, I do a sleep call between each try.

Check status Timer: The check status timer is supposed to let the program know that it's time to check the bulletin for new events. The thing is, the program do absolutely nothing during this time. It doesn't need to be interrupted by a timer handler function, because it just waits until a new check. So again a sleep call will replace the timer.