June 21st, 2013

1. SendData数据都是对的但是laser并没有反应，而且显示的是已经successfully send data.
2. SendPriorityData always have something wrong, error code = 119, not available.
3. 在一次执行完send(sDataSocket)之后，DataSocket变成了一个很大的数，但是当前的这次数据传送的是成功的。？？

下一次send之前socket又变成了正常的而且这次传送是成功地，但是laser并没有显示出来。

在之后的某一次他就有laser显示了。

1. 在进入WriteData的时候dataSocket就变成了一个很大的数，之后紧接着就出错了。
2. 有时候在senddata的时候返回值的话可能就出错了，就是invalid reading memory.
3. Assert:

Void assert (int expression);

If the argument “expression” of this macro with functional form compares equal to zero, a message is written to the standard error device and abort is called, terminating the program execution.

The specifics of the message shown depend on the particular library implementation, but it shall at least include: the expression whose assertion failed, the name of the source file, and the line number where it happened. A usual expression format is:

Assertion failed*: expression*, file *filename*, line line *number*.

This macro is disabled if, at the moment of including <assert.h>, a macro with the name NDEBUG has already been defined. This allows for a coder to include as many assert calls as needed in a source code while debugging the program and then disable all of them for the production version by simply including a line like:

#define NDEBUG

at the beginning of its code, before the inclusion of <assert.h>.

Therefore, this macro is designed to capture programming errors, not user or run-time errors, since it is generally disabled after a program exits its debugging phase.

Asserts are a way of explicitly checking the assumptions that your code makes, which helps you track down lots of bugs by narrowing down what the possible problems could be. They are typically only evaluated in a special “debug” build of your application, so they won’t slow down the final release version.

Let’s say you wrote a function that took a pointer as an argument. There’s a good chance that your code will assume that the pointer is non-NULL, so why not explicitly check that with an assertion?

#include <assert.h>

void function(int\* pointer\_arg)

{

assert(pointer\_arg != NULL);

...

}

An important thing to note is that the expressions you assert must never have side effects, since they won’t be present in the release build. So never do something like this:

assert(a++ == 5);

Some people also like to add little messages into their assertions to help give them meaning. Since a string always evaluates to true, you could write this:

assert((a == 5) && "a has the wrong value!!");

To avoid the side effect problem in MFC, you can use the VERIFY macro instead of ASSERT. VERIFY evaluates the expression in all versions but does not check the result in the Release version.