

Forum:[Hack Forums](#)

Thread: [My code smells and i'm unsure how to fix it.](#)

aeiro - December 20, 2019, 6:10 am

So here I am looking at my project lialis and i hate it. Its messy! its buggy! it just feels like an abomination to me. I've tried to modulate things a bit more but in the end i feel like it just needs an overhaul... a complete overhaul. like rewriting it from the bottom up once again like i did with the original version of this project. Unfortunately i'm not sure where i could actually begin to clean this project up. I could jump right into it and just re-imagine the code flow like i previously did but doing so most likely will put me in the same boat I'm in now.

To sum this up i'd like to ask those of you who have looked into lialis(or the theme engine) for advice on how i could write cleaner and more efficient code.

quick example of something I've done to clean up my code. i think i'm in the right direction but i feel like there's a lot i'm overlooking.

akali - December 20, 2019, 7:03 am

Whenever I've been annoyed with the design of the past code. I don't bother to fix it, I do a complete rewrite. Of both the design and the code. But one way I can see an improvement to your code is to use smart pointers. Also in C++17 you can use `wstring::data` to get the `wchar_t*` of the buffer instead of doing a manual conversion.

aeiro - December 20, 2019, 7:12 am

(Dec 20, 2019 08:03 AM)akali Wrote: [\[CITE\]](#) Whenever I've been annoyed with the design of the past code. I don't bother to fix it, I do a complete rewrite. Of both the design and the code. But one way I can see an improvement to your code is to use smart pointers. Also in C++17 you can use `wstring::data` to get the `wchar_t*` of the buffer instead of doing a manual conversion.

I was trying not to rewrite it but im pretty sure i've made the decision to rewrite it and start with the winsock portion first and create some kind of handing class for all of my socket needs.

Ill saw a video on smart pointers earlier was thinking about implementing them.

And holy shit you're right i didn't even think about using the data pointer in `wstring` to skip the entire loop and save time! Thank you for that.

Emp3hack - December 20, 2019, 7:27 am

Apart from following the updated guidelines and the new features of the language itself, what can be useful to consider is (language independent) :

- Modularity (There is a God class? Should you create a new class for X?)
- Cleanliness (Are there too many params? Dead code? Should I split this (big) function?)
- Redundancy (Is there code duplication? And feature duplication (different code but the same role)?)

- Readability

[OT]

- Performance

- Testing

[/OT]

Some general rules:

A good name for a class should be a noun (i.e. User)

A good name for function is a verb or a sentence with verb in it (i.e. login, searchPost)

SRP (Single response principle)

Avoid useless comment (I don't think it's your case by looking at your sample)

Try to avoid generic term like handler (i.e. ConnectionHandler -> Connection)

As a suggested reading, if you are interested, you could read Clean Code by Robert C. Martin.

The general rules list you read above is composed by some of the suggestion offered in this book.

aeiro - December 20, 2019, 7:49 am

(Dec 20, 2019 08:27 AM)Emp3hack Wrote: [\[CITE\]](#) Apart from following the updated guidelines and the new features of the language itself, what can be useful to consider is (language independent) :

- Modularity (There is a God class? Should you create a new class for X?)

- Cleanliness (Are there too many params? Dead code? Should I split this (big) function?)

- Redundancy (Is there code duplication? And feature duplication (different code but the same role)?)

- Readability

[OT]

- Performance

- Testing

[/OT]

Some general rules:

A good name for a class should be a noun (i.e. User)

A good name for function is a verb or a sentence with verb in it (i.e. login, searchPost)

SRP (Single response principle)

Avoid useless comment (I don't think it's your case by looking at your sample)

Try to avoid generic term like handler (i.e. ConnectionHandler -> Connection)

As a suggested reading, if you are interested, you could read Clean Code by Robert C. Martin.

The general rules list you read above is composed by some of the suggestion offered in this book.

I watched a seminar on the new modular feature. it was an interesting addition and i may be able to take advantage of it in a rewrite.

there is a bit of code duplication but there are slight changes to each dupe, if you know what i mean? i could definitely clean this up. possibly with modularity across the client/server portion.

As with The handler portion i started out doing that in my earlier projects and have for the most part gotten rid of it. heh i guess that one stuck.

Speaking of Robert C. Martin a seminar i watched earlier about a concept called SOLID mentioned him as "Uncle Bob". Hearing that name again tells me i should add this book to my library.

still understanding modules but I was thinking something like a modular winsock wrapper.

ex..

module sockets.connection

module sockets.remote

if i do this i'm going to have to separate non socket related code though.

should I implement those into their own separate modules also?

this will force me to rethink my entire server client code.

which also got me thinking.

Since i've got multiple sockets open for each socket, one for remote one to keep open for commands ect (not even sure if this is a proper way its just what i came up with), should i open all of the sockets that could potentially be opened at first connection or should i open and close them

as needed?

akali - December 20, 2019, 11:24 am

(Dec 20, 2019 08:49 AM)aeiro Wrote: [\[CITE\]](#) this will force me to rethink my entire server client code. which also got me thinking.

Since i've got multiple sockets open for each socket, one for remote one to keep open for commands ect (not even sure if this is a proper way its just what i came up with), should i open all of the sockets that could potentially be opened at first connection or should i open and close them as needed?

You don't need multiple sockets per connection. One socket per connection is fine. Since you can just parse packets in your read method. You can just use predefined structures and packets ids to indicate what your incoming /outgoing data could look like.

```
Code enum PacketIdentifiers{
    kPacketRequestFileFromDir = 0xff,
    kPacketDownloadFile
```

```
};
struct RequestFilePacket{
    char pathtofile[MAX_PATH];
};
```

```
struct DownloadFilePacket{
    unsigned short port;
    bool execute;
    char remotehost[MAX_PATH];
    char remotefilepath[MAX_PATH];
    char savetofilepath[MAX_PATH];
};
```

Then before I sent any data to the client, I make sure to always send a packet id first so it knows the incoming data/sizes.

```
Code PacketIdentifiers id = kPacketDownloadFile;
send(s,(char*)&id,sizeof(id),0);
DownloadFilePacket packet = {0};
packet.port = 80;
packet.execute = true;
strcpy(packet.remotehost,"someurl.com");
strcpy(packet.remotefilepath,"/remotefile.exe");
strcpy(packet.savetofilepath,"C://Users/someusers//desktop//remotefile.exe");
send(s,(char*)&packet,sizeof(packet),0);
```

Then on your recieving end, you would just parse each packet in their own method/whatever you want to use.

```
Code enum PacketIdentifiers{
    kPacketRequestFileFromDir = 0xff,
    kPacketDownloadFile
};
```

```
struct DownloadFilePacket{
    unsigned short port;
    bool execute;
    char remotehost[MAX_PATH];
    char remotefilepath[MAX_PATH];
    char savetofilepath[MAX_PATH];
};
```

```
PacketIdentifiers id;
while(1){
    recv(s,(char*)&id,sizeof(id),0);
    switch(id){
```

```

case kPacketDownloadFile:{
    DownloadFilePacket packet. = {0};
    recv(s,(char*)&packet,sizeof(packet),0);
    //Do whatever needs to be done for that packet
}
}

```

}This is all just example and your code shouldn't look exactly like that but that's how you should send packets/commands.

aeiro - December 20, 2019, 1:10 pm

(Dec 20, 2019 12:24 PM)akali Wrote: [\[CITE\]](#) You don't need multiple sockets per connection. One socket per connection is fine. Since you can just parse packets in your read method. You can just use predefined structures and packets ids to indicate what your incoming /outgoing data could look like.

yes but shouldn't i keep a dedicated socket open for strictly sending each frame captured to not "clog" up other things and to have true async?

<https://github.com/HFaeiro/Lialis/blob/fbec6d2db9a63ecb1f4753e821a4610aabf097fd/Lialis/LialisClient/RemoteView.cpp#L212>

Yes i admit i shouldn't have this dedicated remoterecv socket open strictly to receive remote commands. that can easily be done from the original socket.

Emp3hack - December 21, 2019, 6:18 am

(Dec 20, 2019 02:10 PM)aeiro Wrote: [\[CITE\]](#) yes but shouldn't i keep a dedicated socket open for strictly sending each frame captured to not "clog" up other things and to have true async?

<https://github.com/HFaeiro/Lialis/blob/fbec6d2db9a63ecb1f4753e821a4610aabf097fd/Lialis/LialisClient/RemoteView.cpp#L212>

Yes i admit i shouldn't have this dedicated remoterecv socket open strictly to receive remote commands. that can easily be done from the original socket. Here there are my 2 cents:

We all agree that given N client, for sure you should use at least (N + 1) socket on the server side. So we can say that we have 1 ListenNewClient socket and N ServeClient sockets that could be used to send/receive simple commands .

Now, answering your question, you can have async both with 1 socket or 20 sockets for client.

In fact the real bottleneck should be:

- Bandwidth: Given you have a 10kbit/s minimum speed, you can have 1 socket sending at 10 kbit/s or you can have 2 socket sending at 1kbit/s and 9kbit/s and it will be the same for the network layer. It's not relevant (and it's also unknown) for the network layer how many socket and which protocol are being sent, they are just bit.

- CPU: Parallelism is capped, we can't run an infinite number of thread and pretend to be executed in an infinite pipeline.

After receiving the data we should think about processing it ASAP. What I mean is that, assuming we got this 10kbit/s of data on the receiver buffer (again, please note that there is no difference between 1 socket and 2 socket), we should be quickly at its elaboration to free the buffer and go on.

So what about the differences?

In real you are introducing some overhead when adding an other thread and socket for the same client.

If you have many thread per client you are losing some CPU cycles for the context switch of threads and the time needed to establish the new TCP connection.

As a personal opinion I think that you could spawn a new thread and open a new socket on the server for some "intensive" communication feature (like a video streaming could be). But just because it could be useful to isolate (logically) different communication channels.

It's useless to say that I would use without any doubt the ServeClient socket for commands similar to getComputerInfo or getCurrentUser.

As an addition I looked at just some piece of your code, and maybe there are some potential issue with mutex.

Since you are not using lock_guard (which usage is highly suggested when possible), you should be careful on releasing the locks.

For example #178, #183 show some of this cases.

<https://github.com/HFaeiro/Lialis/blob/fbec6d2db9a63ecb1f4753e821a4610aabf097fd/Lialis/Lialis/Remote.cpp#L178>

And if I got it right (I don't have so much time to do a deep analysis of the code, the lines #208,209 shouldn't be something like:

```
if(try_lock)
unlock()
```

to avoid undefined behaviour caused by unlocking an already unlocked mutex?

The last one thing I noticed were some nested if that could be avoided by specifying a multiple condition in the head of the if statement or by "anticipating" the failure condition. This is an example applied to your code: <https://pastebin.com/GHQi2RGr>

PS: I'm happy that you know uncle Bob too ?

aeiro - December 21, 2019, 12:32 pm

(Dec 21, 2019 07:18 AM)Emp3hack Wrote: [\[CITE\]](#) Here there are my 2 cents:

there are a lot of good talking points you've mentioned.

yes i agree too many sockets and too many threads can and will slow down the cpu processing time and will start to lag.

currently this system created a thread to listen for commands. then if the client gets the remote command it will reconnect to the server by creating a new socket setting up this socket called the remote receive socket. this socket will then setup a new thread just to listen to remote commands. when this remote receive socket gets created the server tells the client to once more create a new socket called a remote send socket to dedicate sending the frame data to the server. so at the moment there are 2 threads running on the client process. one for listening to server commands and one for listening to remote commands. when the client gets a remote command to send a frame it will then create a new thread to take the screen and then send the data over then the thread exits to wait for the next command.

this is currently being rewritten to only create one remotesocket for the frame data. there is no need to create a new socket just to wait for separate commands when these can be processed inside the first client socket.

now on your point about the mutexing. i threw that in there because i was having problems with the istream data getting overwritten by a new frame before i could even draw it to the screen. so i was learning ways to lock the object in a multithreaded environment. still don't think im doing it right.

but for your statement that i using "if(try_lock)

unlock()" i could unlock an already unlocked mutex is incorrect if i understand what its doing correctly. i added this because its basically a check that says if mutex is unlocked lock it and then i

proceed to unlock it because i need to use this data. it seems there is no chance that it could unlock an already unlocked mutex object.

Emp3hack - December 21, 2019, 4:47 pm

(Dec 21, 2019 01:32 PM)aeiro Wrote: [\[CITE\]](#) now on your point about the mutexing. i threw that in there because i was having problems with the istream data getting overwritten by a new frame before i could even draw it to the screen. so i was learning to ways to lock the object in a multithreaded environment. still don't think im doing it right. but for you statement that i using "if(try_lock)

unlock()" i could unlock an already unlocked mutex is incorrect if i understand what its doing correctly. i added this because its basically a check that says if mutex is unlocked lock it and then i proceed to unlock it because i need to use this data. it seems there is no chance that it could unlock an already unlocked mutex object. Talking about the try_lock() I think you should be careful, this damn things are really infamous!

In fact you should not assume that the 2 lines of your code are run in an atomic way.

You can think of it in this way, let's imagine a scenario, for the sake simplicity, with just 2 thread that runs the following code

ThreadA piece of code:

```
..  
lock()  
// do something  
unlock()  
...
```

ThreadB pseudocode:

```
..  
try_lock()  
// do something  
unlock()  
..
```

What when something like this happen?

[ThreadA is running]

lock() is executed

[Context switch: ThreadB is running]

ThreadB suspended (ready status in the CPU scheduler) right after try_lock() call so it fails (return false)

[ThreadA is running]

unlock() is executed

[ThreadB is running]

unlock() is executed ???

As regards the first paragraph I'll read it tomorrow, it's late on my zone lel

BTW this is the reason why the RAII syntax was added, almost all people hates the cares needed for multithreading synchronisation problem with the standard mutex/semaphore, so much margin for mistakes.