Object-Oriented Design & Programming Coursework: MailPunk

Holger Pirk

November 26, 2018

1 Introduction

In this coursework, we model a realistic software development project: the high-level objective of the project is to develop an email client. To keep things simple, the client shall have only limited functionality: logging in to a mail server (using the IMAP protocol), viewing emails and allowing their deletion.

Your responsibility is restricted to the middle part of the software stack: maintaining the (temporary) data of the application while it is running (i.e., everything that is in main memory). While this limits your effort, it forces you to adhere to a strictly defined programming interface (the set of classes and their functions) both in terms of the functionality you have to provide as well as the library you can build upon.

Specifically, the user interface is implemented as a terminal/console based application using the `FinalCut` library. The connection to the IMAP server is implemented using the `libetpan` library. Both have their quirks: you don't need to worry about `FinalCut` since this is taken care of for you. `Libetpan`, however is a `C` library. As such, it has no support for classes, automatic memory management, templates and the like. You task is, therefore, to wrap the existing `Libetpan` interface in some nice object-oriented abstractions.

2 Getting started

To get started log in to a lab machine and run the following sequence of commands:

```
git clone git@gitlab.doc.ic.ac.uk:OODP2018/MailPunk.git
cd MailPunk/
mkdir build
cd build/
cmake ..
cmake --build .
## At this point, it will run a while (it compiles the dependencies)
```

This build will fail with the following message:

```
[ 57%] Built target buildetpan
[ 60%] Building CXX object CMakeFiles/MailPunk.dir/MailPunk.cpp.o
[ 64%] Building CXX object CMakeFiles/MailPunk.dir/imap.cpp.o
[ 67%] Building CXX object CMakeFiles/MailPunk.dir/UI.cpp.o
[ 71%] Linking CXX executable MailPunk
CMakeFiles/MailPunk.dir/UI.cpp.o: In function `UI::refreshMailList()::{lambda(finalcut::FWidget*,
 ↪  void*)#1}::operator()(finalcut::FWidget*, void*) const':
UI.cpp:(.text+0xd23): undefined reference to `IMAP::Message::getBody[abi:cxx11]()'
```

```
CMakeFiles/MailPunk.dir/UI.cpp.o: In function `UI::refreshMailList()':
UI.cpp:(.text+0x1105): undefined reference to `IMAP::Session::getMessages()'
UI.cpp:(.text+0x124c): undefined reference to `IMAP::Message::getField(std::__cxx11::basic_string<char,
 ↪  std::char_traits<char>, std::allocator<char> >)'
UI.cpp:(.text+0x12d1): undefined reference to `IMAP::Message::getField(std::__cxx11::basic_string<char,
 ↪  std::char_traits<char>, std::allocator<char> >)'
CMakeFiles/MailPunk.dir/UI.cpp.o: In function `auto UI::loginClicked(finalcut::FWidget*)::{lambda(auto:1*,
 ↪  auto:2*)#2}::operator()<finalcut::FWidget, void>(finalcut::FWidget*, void*) const':
UI.cpp:(.text+0x176f): undefined reference to `IMAP::Message::deleteFromMailbox()'
CMakeFiles/MailPunk.dir/UI.cpp.o: In function `UI::loginClicked(finalcut::FWidget*)':
UI.cpp:(.text+0x1803): undefined reference to `IMAP::Session::Session(std::function<void ()>)'
UI.cpp:(.text+0x18b5): undefined reference to `IMAP::Session::connect(std::__cxx11::basic_string<char,
 ↪  std::char_traits<char>, std::allocator<char> > const&, unsigned long)'
UI.cpp:(.text+0x19e7): undefined reference to `IMAP::Session::login(std::__cxx11::basic_string<char,
 ↪  std::char_traits<char>, std::allocator<char> > const&, std::__cxx11::basic_string<char,
 ↪  std::char_traits<char>, std::allocator<char> > const&)'
UI.cpp:(.text+0x1a95): undefined reference to `IMAP::Session::selectMailbox(std::__cxx11::basic_string<char,
 ↪  std::char_traits<char>, std::allocator<char> > const&)'
CMakeFiles/MailPunk.dir/UI.cpp.o: In function `UI::~UI()':
UI.cpp:(.text+0x2a4e): undefined reference to `IMAP::Session::~Session()'
collect2: error: ld returned 1 exit status
CMakeFiles/MailPunk.dir/build.make:148: recipe for target 'MailPunk' failed
make[2]: *** [MailPunk] Error 1
CMakeFiles/Makefile2:141: recipe for target 'CMakeFiles/MailPunk.dir/all' failed
make[1]: *** [CMakeFiles/MailPunk.dir/all] Error 2
Makefile:83: recipe for target 'all' failed
make: *** [all] Error 2
```

This means you need to define these functions (either inline of in the cpp file).

3 Your task

Your task is to implement the imap classes `Session` and `Message`. You are free to change anything in the `imap.hpp` and `imap.cpp` file.

3.1 Testing your solution

Once you (think you) have a running solution, you can test it against a mailserver I have created for you. The server has the IP `146.169.46.139` and it holds an account for each student. The username is created from your regular login name by appending the word `mail`. For example, my login is `hlgr` making `hlgrmail` my mail login. I will send you your password in a separate mail. Assuming you received `olNiWYR3J4iJI` as your password, you can run the MailPunk client from the `build` directory. You can pass in the user, server and password like this:

3.1.1 Starting the client

```
USER=hlgrmail SERVER=146.169.46.139 PASSWORD=olNiWYR3J4iJI ./MailPunk
```

You can find a video of a working solution at [`https://youtu.be/L_RcILrNB7E`].

3.1.2 Sending test emails

Your email client should now only allow you to view emails but also delete them. It is, therefore, handy to be able to send new mails to your inbox. You can do that using the following script (feel free to modify it):

```
(echo From: $USER@`hostname`; echo "Subject: A test email"; echo; echo "Here is the first line of the body";
 ↪  echo "and the second") | curl smtp://146.169.46.139 --mail-rcpt hlgrmail@localhost -T -
```

4 Submitting your solution

Submit `imap.hpp` and `imap.cpp` to Cate. You are free to make any changes you like to those files but they **must** compile on the lab machines using the procedure outlined above. Modify any of the other files at your own peril.

Take particular care to make sure there are no memory leaks.

5 Libetpan cheat sheet

`Libetpan` is very complex and somewhat poorly documented. I, therefore, did my best to extract only the relevant parts of the documentation (covering the part of the API I used in my solution). This way, you don't have to dig through the source code too much (though there will be some of that). You can also have a look at the official example: [`https://github.com/dinhviethoa/libetpan/blob/master/tests/imap-sample.c`].

5.1 CLists

CLists are an attempt to achieve the functionality of C++ lists in C. They are a bit cumbersome but the interface should be familiar.

```
/**
 * A function (macro) that get's the content of an iterator over a clist
 */
clist_content(clistiter)


clist_begin(result->st_info_list)
clist_count(fetch_result)
clist_next(cur)
clist_append
```

5.2 Functions

These are the functions that are useful to implement the solution. Look up the exact interfaces in the file `mailimap_types_helper.h`, `mailimap_types.h`, `mailimap.h`, `mailimap_helper.h` and `mailimap_socket.h`. Feel free to ask questions if you feel that the documentation is insufficient.

You will find that most of the functions that deal with data structures follow a pattern: their prefix is usually close to what we would use as a class name (something like `mailimap_status_att_list_`). The remainder describes what the function does like `_new` to create `_free` to free or `_add` to add to a list. Beware that `_new` functions allocate and can leak memory.

5.2.1 Utilities

```
/**
 * A utility function I defined to turn an error code (int r) into a thrown exception
 */
check_error(r, "could not connect");
```

5.2.2 Flagging/Marking Messages (e.g., to delete them)

```
/*
  this function creates an empty list of flags
*/
mailimap_flag_list_new_empty();

/*
  this function adds a flag to the list of flags

  @return MAILIMAP_NO_ERROR will be returned on success,
  other code will be returned otherwise
*/
mailimap_flag_list_add(flag_list, d);

/*
  this function creates a \Deleted flag
*/
mailimap_flag_new_deleted();

/*
  mailimap_store_att_flags is the description of the STORE operation
  (change flags of a message)

  this function creates a store attribute to set the given flags
  - flag_list is the list of flags to change
*/
mailimap_store_att_flags_new_set_flags(flag_list);
mailimap_store_att_flags_free(store);

/*
   mailimap_uid_store()

   This function will alter the data associated with some messages
   (flags of the messages).

   @param session          IMAP session
   @param set              This is a list of message unique identifiers.
   @param store_att_flags  This is the data to associate with the
     given messages

   @return the return code is one of MAILIMAP_ERROR_XXX or
     MAILIMAP_NO_ERROR codes
*/
mailimap_uid_store(session->imap, set, store);

/*
   This function will permanently remove from the selected mailbox
   message that have the \Deleted flag set.

   @param session IMAP session

   @return the return code is one of MAILIMAP_ERROR_XXX or
     MAILIMAP_NO_ERROR codes
*/
mailimap_expunge(session->imap);
```

5.2.3 Session Management

```
/*
   mailimap_new()

   This function returns a new IMAP session.
   @param progr_rate  When downloading messages, a function will be called
     each time the amount of bytes downloaded reaches a multiple of this
     value, this can be 0.
   @param progr_fun   This is the function to call to notify the progress,
     this can be NULL.

   @return an IMAP session is returned.
 */
mailimap_new(0, NULL)

/**
 * connect to the sever
 */
mailimap_socket_connect(imapSession, server.c_str(), port)

/*
   mailimap_free()

   This function will free the data structures associated with
   the IMAP session.

   @param session   IMAP session
 */
mailimap_free(imap);

/*
   mailimap_login()

   This function will authenticate the client.

   @param session   IMAP session
   @param userid    login of the user
   @param password  password of the user

   @return the return code is one of MAILIMAP_ERROR_XXX or
    MAILIMAP_NO_ERROR codes
*/
mailimap_login(imap, userid.c_str(), password.c_str())
/*
   mailimap_logout()

   This function will logout from an IMAP server by sending
   a LOGOUT command.

   @param session IMAP session

   @return the return code is one of MAILIMAP_ERROR_XXX or
     MAILIMAP_NO_ERROR codes
*/
mailimap_logout(imap);
```

5.2.4 Working with mailboxes

```
/*
   mailimap_select()

   This function will select a given mailbox so that messages in the
   mailbox can be accessed.

   @param session          IMAP session
   @param mb  This is the name of the mailbox to select.

   @return the return code is one of MAILIMAP_ERROR_XXX or
     MAILIMAP_NO_ERROR codes
```

```
*/
mailimap_select(imap, mailbox.c_str())


/*
  mailimap_mailbox_data_status is the list of information returned
  when a STATUS of a mailbox is requested

  - mailbox is the name of the mailbox, should be allocated with malloc()

  - status_info_list is the list of information returned
*/
mailimap_mailbox_data_status_free(result);
```

5.2.5 Getting status information (like the number of messages) from a mailbox

```
/*
  mailimap_status_att_list is a list of mailbox STATUS request type

  - list is a list of mailbox STATUS request type
    (value of elements in the list can be MAILIMAP_STATUS_ATT_MESSAGES,
    MAILIMAP_STATUS_ATT_RECENT, MAILIMAP_STATUS_ATT_UIDNEXT,
    MAILIMAP_STATUS_ATT_UIDVALIDITY or MAILIMAP_STATUS_ATT_UNSEEN),
    each element should be allocated with malloc()
*/
mailimap_status_att_list_new_empty();
mailimap_status_att_list_free(status_att_list);


/*
  this function adds status attributes to the list

  @return MAILIMAP_NO_ERROR will be returned on success,
  other code will be returned otherwise
*/
mailimap_status_att_list_add(status_att_list, MAILIMAP_STATUS_ATT_MESSAGES);

/*
   mailimap_status()

   This function will return informations about a given mailbox.

   @param session          IMAP session
   @param mb               This is the name of the mailbox
   @param status_att_list  This is the list of mailbox information to return
   @param result           Pointer to list to be filled with returned values

   @return the return code is one of MAILIMAP_ERROR_XXX or
     MAILIMAP_NO_ERROR codes
*/

mailimap_status(mailimap * session, const char * mb,
    struct mailimap_status_att_list * status_att_list,
    struct mailimap_mailbox_data_status ** result);
```

5.2.6 Others

```
/*
  set is a list of message sets

  - list is a list of message sets
*/
mailimap_set_free(set);
```

```
mailimap_set_new_interval(1, 0);
mailimap_set_new_single(uid);

/*
  this function creates a mailimap_fetch_att structure to request
  the unique identifier of a message
*/
mailimap_fetch_att_new_uid()

/*
  mailimap_fetch()

  This function will retrieve data associated with the given message
  numbers.

  @param session    IMAP session
  @param set        set of message numbers
  @param fetch_type type of information to be retrieved
  @param result     The result of this command is a clist
    and it is stored into (* result). Each element of the clist is a
    (struct mailimap_msg_att *).

   @return the return code is one of MAILIMAP_ERROR_XXX or
     MAILIMAP_NO_ERROR codes
*/
mailimap_fetch(imap, set, fetch_type, &fetch_result);

/*
  this function creates a mailimap_fetch_att structure to request
  unique identifier of a message
*/
mailimap_fetch_att_new_uid()

/*
   mailimap_fetch_list_free()

   This function will free the result of a fetch command.

   @param fetch_list  This is the clist containing
     (struct mailimap_msg_att *) elements to free.
*/
mailimap_fetch_list_free(fetch_result);
/*
  this function adds a given fetch attribute to the mailimap_fetch
  structure

  @return MAILIMAP_NO_ERROR will be returned on success,
  other code will be returned otherwise
*/
mailimap_fetch_type_new_fetch_att_list_add(fetch_type, mailimap_fetch_att*);
/*
  this function creates a mailimap_fetch_type structure
*/
mailimap_fetch_type_new_fetch_att_list_empty();

/*
  this function creates a mailimap_fetch_att structure to request
  the body of a message
*/
mailimap_fetch_att_new_body()

/*
  mailimap_header_list is a list of headers that can be specified when
  we want to fetch fields
*/
```

```
mailimap_header_list_new(headerList);

/*
  this functions creates a mailimap_section structure to describe
  a list of headers to be fetched
*/
mailimap_section_new_header_fields(headers)


/*
  mailimap_fetch()

  This function will retrieve data associated with the given message
  numbers.

  @param session    IMAP session
  @param set        set of message unique identifiers
  @param fetch_type type of information to be retrieved
  @param result     The result of this command is a clist
    and it is stored into (* result). Each element of the clist is a
    (struct mailimap_msg_att *).

   @return the return code is one of MAILIMAP_ERROR_XXX or
     MAILIMAP_NO_ERROR codes
*/
mailimap_uid_fetch(session->imap, set, fetch_type, &fetch_result);
```

5.3 useful snippets

```
/**
 * get a mailimap_status_info value from the first clist (clist_begin gets the first element)
 */
 auto value = ((struct mailimap_status_info*)clist_content(clist_begin(thelist)))->st_value;

/**
 * get the list of attributes from a clist element
 */
 auto msg_att = (struct mailimap_msg_att*)clist_content(element);

/**
 * cast the content of a clist element to a message attribute item
 */
 auto item = (struct mailimap_msg_att_item*)clist_content(cur);
```