4

N 55 8 1C

C++ Exercises
Set 7

Author(s): Olivier Gelling, Leon Lan, Mohammad Habibi
Previously rated by Frank
14:52

October 27, 2025

55

Exercise 55:

Write the Lock class and a program that locks a file and appends a line of text to it.

We do exactly this, choosing not to refactor subroutines like copying the string into a char *, as the exercise specifies that no more members are needed than the ones mentioned. We had some trouble juggling the conversions from char *s to strings and back and forth, and as such hope this works. There might be room for improvements and optimisation by either leaning more on char * and less on strings, but that is a matter for another time.

We note that checking for memory leaks with valgrind makes the program fail to open the lock file! This might be due to the way valgrind runs the input commands? Maybe it runs from a different directory, as we specifically see our validate function output the debug message that there was an issue with the lock file itself. We do not see any memory leaks luckily!

? It shouldn't doesn't do that for me.

We have chosen to remove the debug messages, but the "lock successful" and "lock failed" will remain as part of the program's feedback.

We have left the option to add a second commandline argument to select the directory in which the .lck file will be stored out. Adding it in would require some extra shenanigans with memory allocation that seems irrelevant to this exercise's scope, but the code works and can be un-commented for testing!

Can't see why it does that for you.

As for writing a library:

We make Lock's library by first compiling all the source files into object files, we can do this with:
g++ -std=c++26 -Wall -Werror -c *.cc // what happens to main.cc?

Or by using make or icmbuild library. We can then make a library from this by running:
ar rcs liblock.a */*/*.o ? confusing. Why not use a build utility?

(since our object files were placed in tmp/o/). We can now either link directly to this library, or place it somewhere and link to it from there.

Our class is made within the lock/ directory, separating it from the main program. We could install the library in /usr/lib, and the header in /usr/include, but since it will be growing over time that is likely a bit premature. It would greatly simplify including and linking it though. Instead, for now, we will place it somewhere else high up the file structure. My projects are usually in a directory like so:
~/Projects/Olivier/FBC++/set7/55

We will make a new directory:
~/orgutility

which has subdirectories lib/ and include/. We will store our headers and libraries inside here for now, and move them over to /usr/ when they are ready!

```
orgutility$ tree
.
        include
                lock.h
        lib
            liborgutils.a
```

We include the header in angled brackets inside main.ih, and then call the compiler to link the project like so:

```
g++ -I$HOME/orgutility/include main.cc -L$HOME/orgutility/lib -lorgutility -o
lock.out
```

This compiles on our end and gives us the program exactly as such. We will turn the angled brackets back into double quotes to satisfy the mailhandler though, as it likely won't pick up on our lock.h otherwise.

We will henceforth expand on this library.

Listing 1: lock/lock.h

```
#ifndef INCLUDED_LOCK_
#define INCLUDED_LOCK_

#include <string>

class Lock
{
    int d_filedesc = -1;
    public:
        Lock(std::string const &path);
        Lock(std::string const &path, std::string lockDir);
        ~Lock();

        int open(std::string pathStr);
        bool valid();

    private:
        static std::string stringName(std::string const &path1,
                                      char *(*name)(char* pathPH));
        std::string lockPath(std::string const &path,
                             std::string const &lockDir);
};

#endif
```

Listing 2: lock/lock.ih

```
#include "lock.h"
#include <filesystem>
#include <libgen.h>          // for basename/dirname
#include <fstream>
#include <fcntl.h>           // for open()
#include <sys/file.h>        // for flock()
#include <unistd.h>          // for close()
#include <iostream>
#include <cstring>           // for strcpy

using namespace std;
```

Not needed BH.

Listing 3: lock/lock1.cc

```
#include "lock.ih"
```

```
    // by

Lock::Lock(string const &path)
:
    Lock(path, stringName(path, dirname))    // get string filename
{}                                           // and delegate
```

Listing 4: lock/lock2.cc

```
#include "lock.ih"

    // by

Lock::Lock(string const &path, string lockDir)
{
    string fullName = lockPath(path, lockDir);

    d_filedesc = open(fullName);

    cerr << (valid() ? "lock successful\n" : "lock failed\n");
}
```

Listing 5: lock/lock3.cc

```
#include "lock.ih"

    // by

Lock::~Lock()
{
    if (valid())                         // IF locked file, close/unlock
        ::close(d_filedesc);
}                                        // Else nothing needed to be done
```

Listing 6: lock/open.cc

```
#include "lock.ih"

    // by lock2.cc

int Lock::open(string pathStr)
{
    filesystem::path pathObj(pathStr);       // for exists()
    char const *pathC = pathStr.c_str();     // for open()
    int filedesc;                            // temp storage of d_filedesc

    filedesc = (filesystem::exists(pathObj) ? ::open(pathC, O_RDWR)
                        : ::open(pathC, O_CREAT | O_TRUNC | O_RDWR, 0600));

    return filedesc;
}
```

Listing 7: lock/valid.cc

```
#include "lock.ih"

    // by

bool Lock::valid()
{
    if (d_filedesc == -1 || flock(d_filedesc, LOCK_EX | LOCK_NB) == -1)
        return false;                    // d_fd == -1 means no file
                                         // else check locking success
    return true;
}                       // Using a ternary here would probably be a bit much
```

Listing 8: lock/stringname.cc

4C

```
#include "lock.ih"

    // by lock1.cc lockpath.cc

string Lock::stringName(string const &path1, char *(*name)(char *pathPH))
{
    char fileName[path1.length() + 1];

    strcpy(fileName, path1.c_str());

    return string(name(fileName));
}
```

strcpy is C: use string facilities

Listing 9: lock/lockpath.cc

```
#include "lock.ih"

    // by

string Lock::lockPath(string const &path, string const &lockDir)
{
    string baseName = stringName(path, basename);

    return lockDir + "/" + baseName + ".lck";
}
```

Listing 10: main.ih

```
#include "lock/lock.h"
//#include <lock.h>      // when compiling with a static library and header
                         // placed in our orgutility directory
#include <fstream>
#include <iostream>

using namespace std;
```

Listing 11: main.cc

```
#include "main.ih"

int main(int argc, char **argv)
{
    if (argc == 1)
        return 1;

    // For testing manually handing the .lck file's directory:
    // string directory = argv[2];
    // Lock fileLock(argv[1], directory);

    Lock fileLock(argv[1]);
    if (!fileLock.valid())
    {
        cerr << "could not open file\n";
        return 1;
    }

    fstream writeFile(argv[1], ios::app);

    string appendText;
    cerr << "? ";

    getline(cin, appendText);
    writeFile << appendText << '\n';
}
```

prefer using named operators