CS22510 - Assignment 1

Alexander Brown

March 24, 2011

1 File Layouts

1.1 logfile.log

The format for the logfile is:

`yyyy-mm-dd-hh-mm-ss_progr: uid action path/file`

1.2 .auth

The format of the .auth file is:

`user group password`

I also included a way of specifying comments (using the '`#`' charcter) to allow leeway.

2 Language Choices

I chose Java for the Staff Program as I believed it to be somewhat tougher with file handlling, but easier to work with Objects like Strings.

I then chose C++ for the somewhat more complex Administrator Program, as I knew I would be able to handle the complexity better due to being more practised with Object Orientated programming. This left the Student Program for C. This played out fairly well as it seemed to have the least string manipulation.

3 Program Description

All programs have the same initial behaviour; the config file is found and the base directory is read. Then the user is prompted for a user ID, followed by a password (which is masked by disabling the stdin echo in C and C++, and using a system password call in Java). This is then made into a string, with the name of the program to match how it would appear in the .auth file. The .auth file is then opened and read line by line until either the two strings match, in which case the user is logged in and passed to the menu, or the end of the file is reached, in which case an error message is printed and the program exits.

3.1 Adminstrator Program Description

The administrator has four primary functions; uploading an assignment, creating a staff user, creating a student user and reading the logfile. Uploading an assignment is a matter of specifying a filepath to the file to upload as an assignment. A File Handller static function is then called with the location to upload (repository/assignment/¡filename¿), which in turn calls the copy static function.

Creating a staff and student user is almost exactly the same and as such they both call a private function to create a general user, passing in the type of user they are as an enum. This then creates a new file handller object for the file, generates a random password and formats the information into a string. This string is then passed into the file handller's append function. This function creates a new lock for the file and appends the input string to the file. The file lock is automatically released due to the destructor. These functions then return the generated password, which is printed to the administrator. The create student function also creates a directory for the uid of the student in the students directory, followed by a results directory in this directory.

These two functions also append what they did to the log file in a similar way to how the .auth file is appended to.

Reading the log file involves reading each line of the logfile, spliting and formatting it, then printing it to the screen.

3.2 Student Program

The student program has three primary features; downloading the assignment, uploading a submission and viewing the marks for a submission.

Downloading the assignment is simply a matter of asking the user where it should be downloaded to, then copying from the assignment directory to that specified location.

Uploading a submission is also fairly simple, copying from the specified location of the submission to the student's directory.

Viewing the marks is also fairly simple, just searching the results directory of the student's directory for the file result.txt. If it's there, split and format it somewhat nicely, work out the grade achieved using if statements and printing.

Appending to the logfile is also fairly simple, first off the logfile is locked, then opened with the append mode on. Then writing the formatted string and closing and unlocking the file.

3.3 Staff Program

Much of the file handlling in the student program is modelled on how the admin program does the same. Refer to the above section for more information.

The Staff Program has three main features; listing all students that have made submissions, downloading the submission of a single student and giving marks for a specific student's submission.

Listing all students that have made submissions is a case of looping through the students directory, entrying each student's directory in turn and seeing if there's any files there other than the results directory. If there is it's printed.

Downloading is a case of locking the file, copying it to a specified location then unlocking it.

Giving marks is simply taking input from the keyboard and putting it into a text file in the right format.

4 Evaluation

At current the program is still fairly rough around the edges, it assumes all users are fairly competent and portable to systems other than *NIX, due to both the Makefile and the file locking. However it does meet the requirements fairly well and I have learned a fair amount about file handlling in all languages (unfortunately Java's new io libraries aren't yet fully available as they would have made some tasks much easier). I also took the time to learn a lot more about makefiles as it did make the project a lot easier to debug and run (and also being able to create the directory structure using BASH commands did make life a lot easier).

If I'd had more time I probably would have made changes to the file locking, creating a temporary ¡filename¿.lock file instead of relying on external, non-portable libraries to ensure locking was done correctly. Written inside these files would be the pid of the program locking them. Though this would probably produce a little more overhead, it would have made debugging easier.

I would also have changed to the way logging occured; possibly to the extent of using a threaded dqueue (new entries are added to the back of the dqueue, then all the while to top element is popped, attempted to be written to the logfile. If it fails it's put back onto the top of the queue, otherwise it's done with) to ensure loggin happens without affecting the user. Currently this probably wouldn't be a problem due to the small userbase, however it's likely the with increased traffic getting locked out of the logfile could be an issue and at current the program would either give up logging or wait until it can log, either way stopping the user from continuing.

Finally I would have made the admin program set up the directory stucture on the first run instead of the makefile to allow for increased portability.

5 Run Example

6 Example Usage

```
$ make

$ ./admin_program
Enter User ID: softly
softly's password:
Enter option code (h for help): h
        c [file]: Upload Assignment
        s [uid]: Create a new Staff User
        d [uid]: Create a new Student User
```

```
        l: Read the log file
Enter option code (h for help): c
Enter the file (including path) of the assignment: assignment.txt

$ ./admin_program
Enter User ID: softly
softly's password:
Enter option code (h for help): s
Enter the UID for the new staff user: fwl
Created Staff User: fwl
password: xf\xgDq\L

$ ./admin_program
Enter User ID: softly
softly's password:
Enter option code (h for help): d
Enter the UID for the new student user: adb9
Created Student User: adb9
password: pIgC2Aw7>]b

$ ./student_prog
Enter User ID: adb9
adb9's password:
Enter option code (h for help): h
        g - Get Assignment.
        s - Submit Assignment.
        m - View Marks.
        q - Quit.
Enter option code (h for help): g
Location to download the assignment to: download-assign.txt

$./student_prog
Enter User ID: adb9
adb9's password:
Enter option code (h for help): s
Location of file to submit: submission.txt

$ java -jar staff_prog.jar
Enter User ID: fwl
Password for fwl:
Enter option code (h for help): h
        l - List all students with submitted work.
        g - Download a specified student's work.
        m - Submit a mark for a specified student's work.
        q - Quit the program.
Enter option code (h for help): l
The following students have made submissions:
        adb9

$ ./admin_prog
Enter User ID: softly
softly's password:
Enter option code (h for help): d
Enter the UID for the new student user: crl9
Created Student User: crl9
password: qepA=\2236

$ ./student_prog
Enter User ID: crl9
crl9's password:
```

```
Enter option code (h for help): s
Location of file to submit: submission.txt

$ java -jar staff_prog.jar
Enter User ID: fwl
Password for fwl:
Enter option code (h for help): l
The following students have made submissions:
        crl9
        adb9

$ java -jar staff_prog.jar
Enter User ID: fwl
Password for fwl:
Enter option code (h for help): g
Enter the uid of the student to get the submission of: adb9
Enter the location you wish to download this submission to: adb9-to-mark.txt

$ java -jar staff_prog.jar
Enter User ID: fwl
Password for fwl:
Enter option code (h for help): m
Enter the uid of the student to submit a mark for: adb9
/home/softly/svn/assign_cs22510/trunk/repository/students/adb9/results/result.txt
Enter the mark (%): 75
Enter any comments:
A good piece of work, however some of the features I had asked for were not as good as expected. Also you n

$ ./student_prog
Enter User ID: adb9
adb9's password:
Enter option code (h for help): m
Results for user 'adb9':
        Mark: 75% (A-)
        Comments: A good piece of work, however some of the features I had asked for were not as good as ex

$ ./admin_prog
Enter User ID: softly
softly's password:
Enter option code (h for help): l
========================================
Date:           24/03/2011
Time:           16:01:13
User:           softly
Program:        init
Action:         Initilisation
File:           repository/
========================================
Date:           24/03/2011
Time:           16:01:13
User:           softly
Program:        init
Action:         Initilisation
File:           repository/logfile.log
========================================
Date:           24/03/2011
Time:           16:01:13
User:           softly
Program:        init
Action:         Initilisation
```

```
File:           repository/.auth
========================================
Date:           24/03/2011
Time:           16:01:13
User:           softly
Program:        init
Action:         Created user softly
File:           repository/.auth
========================================
Date:           24/03/2011
Time:           16:01:13
User:           softly
Program:        init
Action:         Initilisation
File:           repository/assignment/
========================================
Date:           24/03/2011
Time:           16:01:13
User:           softly
Program:        init
Action:         Initilisation
File:           repository/students/
========================================
Date:           24/03/2011
Time:           16:02:42
User:           softly
Program:        admin
Action:         Uploaded Assignment
File:           repository/assignment/assignment.txt
========================================
Date:           24/03/2011
Time:           16:03:55
User:           softly
Program:        admin
Action:         Created Staff User
File:           repository/.auth
========================================
Date:           24/03/2011
Time:           16:04:39
User:           softly
Program:        admin
Action:         Created Student User
File:           repository/.auth
========================================
Date:           24/03/2011
Time:           16:06:09
User:           adb9
Program:        stdnt
Action:         Downloaded assignment
File:           repository/assignment/
========================================
Date:           24/03/2011
Time:           16:22:04
User:           adb9
Program:        stdnt
Action:         Downloaded assignment
File:           repository/assignment/
========================================
Date:           24/03/2011
Time:           16:26:38
```

```
User:           adb9
Program:        stdnt
Action:         Uploaded assignment
File:           repository/students/adb9/
========================================
Date:           24/03/2011
Time:           16:27:27
User:           adb9
Program:        stdnt
Action:         Uploaded assignment
File:           repository/students/adb9/
========================================
Date:           24/03/2011
Time:           16:28:09
User:           adb9
Program:        stdnt
Action:         Uploaded assignment
File:           repository/students/adb9/
========================================
Date:           24/03/2011
Time:           16:28:41
User:           adb9
Program:        stdnt
Action:         Uploaded assignment
File:           repository/students/adb9/
========================================
Date:           24/03/2011
Time:           16:29:25
User:           fwl
Program:        staff
Action:         Listed Submissions
File:           repository/students/
========================================
Date:           24/03/2011
Time:           16:30:17
User:           softly
Program:        admin
Action:         Created Student User
File:           repository/.auth
========================================
Date:           24/03/2011
Time:           16:30:54
User:           crl9
Program:        stdnt
Action:         Uploaded assignment
File:           repository/students/crl9/
========================================
Date:           24/03/2011
Time:           16:32:58
User:           fwl
Program:        staff
Action:         Listed Submissions
File:           repository/students/
========================================
Date:           24/03/2011
Time:           16:33:48
User:           fwl
Program:        staff
Action:         Downloaded assignment
File:           repository/students/adb9/submission.txt
```

```
========================================
Date:           24/03/2011
Time:           16:35:44
User:           fwl
Program:        staff
Action:         Submitted mark for adb9
File:           repository/students/adb9/results/result.txt
========================================
```

7 Source code - Adminisrator Program

7.1 FileHandller.cpp

```
/* FileHandller.cpp Copyright (c) Alexander Brown, March 2011 */

#include <string>
#include <fstream>
#include <ios>
#include <iostream>

#include "FileHandller.h"
#include "FileLock.h"

using namespace std;

FileHandller::FileHandller(std::string file) : _filename(file) {
}

FileHandller::~FileHandller() {

}

bool FileHandller::append_file(std::string append_text) {
	try {
		FileLock lock(_filename);

		std::ofstream dest;
		dest.open(_filename.data(), ofstream::app);
		dest << append_text << endl;
		dest.close();
		// lock automatically freed and unlocked here.
		return true;
	} catch (std::string e) {
		//cout << "Append Failed: " << e << endl;
		return false;
	}
}

bool FileHandller::download_file(std::string dest_filename) throw (std::string) {
	try {
		FileLock lock(_filename);
		return FileHandller::copy(_filename, dest_filename);
		// lock automatically freed and unlocked here.
	} catch (std::string e) {
		cout << "Download Failed: " << e << endl;
	}
}

bool FileHandller::upload_file(std::string src_filename ,
		std::string dest_filename) throw (std::string){
	try {
		FileLock lock(dest_filename); //Lock the file if it exists.
	} catch (std::string e) {
		if(e == "File Descriptor Error") {
			// Do nothing - the file needs to be created instead, defying the point of locking it.
		} else {
			throw e; //Otherwise the file is locked.
		}
	}
	return copy(src_filename, dest_filename);
	// Lock automatically freed and unlocked here.
}

bool FileHandller::copy(std::string src_filename, std::string dest_filename) {
	std::ifstream src;
	std::ofstream dest;

	src.open(src_filename.data(), std::ios::binary);
	dest.open(dest_filename.data(), std::ios::binary);
```

```
        if (!src.is_open() || !dest.is_open()) {
            return false;
        }
        dest << src.rdbuf();

        dest.close();
        src.close();
        return true;
}
```

7.2 FileHandller.h

```
/* FileHandeller.h  Copyright (c) Alexander Brown, March 2011 */

#ifndef _FILEHANDLLER_H
#define	_FILEHANDLLER_H

/**
 * @brief A file handller for a single file.
 *
 * Due to the nature of the repository and the need for locking, it is best that
 * all file operations are handlled properly. This class enforces the locking
 * mechanism (calling the File Lock class) and performs basic operations with
 * files found in the repository.
 *
 * @author Alexander Brown
 * @version 1.0
 * @see FileLock
 */
class FileHandller {
public:
	/**
	 * Creates a file handller for a given file.
	 * @param filename	The file to work with.
	 */
	FileHandller(std::string filename);

	/**
	 * Destructor.
	 */
	virtual ~FileHandller();

	/**
	 * Uploads the local sorce file to the destination file in the repository.
	 * @param src_filename  The location of the local file to upload.
	 * @param dest_filename The destination of the source file.
	 * @see copy()
	 * @return		<code>true</code> - If the operation completed sucessfully
	 */
	static bool upload_file(std::string src_filename,
		std::string dest_filename) throw (std::string);

	/**
	 * Downloads the current file from the repository.
	 * @param dest_filename The location of the file to download to.
	 * @see copy()
	 * @return		<code>true</code> - If the operation completed sucessfully
	 */
	bool download_file(std::string dest_filename) throw (std::string);

	/**
	 * Appends a given string to the current file.
	 * @param append_text	The string to append.
	 * @return		<code>true</code> - If the operation completed sucessfully
	 */
	bool append_file(std::string append_text);
private:
	/** The current file in the repository. */
	std::string _filename;

	/**
	 * Copies the src file to the destination file.
	 * @param src_filename  The location of the file to copy.
	 * @param dest_filename The destination of the source file.
	 * @return		<code>true</code> - If the operation completed sucessfully
	 */
	static bool copy(std::string src_filename, std::string dest_filename);
};

#endif	/* _FILEHANDLLER_H */
```

7.3 FileLock.cpp

```
/* FileLock.cpp  Copyright (c) Alexander Brown, March 2011 */

#include <cstdlib>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <string>

#include "FileLock.h"

/**
 * Structure contaning details on the lock.
 * @author Neal Snooke
 */
struct flock* file_lock(short type, short whence) {
    static struct flock ret;
    ret.l_type = type;
    ret.l_start = 0;
    ret.l_whence = whence;
    ret.l_len = 0;
    ret.l_pid = getpid();
    return &ret;
}

FileLock::FileLock(std::string filename) throw (std::string) {
    _file_des = open(filename.data(), O_RDWR);

    if (_file_des == -1) {
        throw std::string("File Descriptor Error");
    }
    lock();
}

FileLock::~FileLock() {
    unlock();
    close(_file_des);
}

void FileLock::lock() throw (std::string) {
    _file_lock = file_lock(F_WRLCK, SEEK_SET);

    if (fcntl(_file_des, F_SETLK, _file_lock) == -1) {
        if (errno == EACCES || errno == EAGAIN) {
            throw std::string("Already locked by another process");
        } else {
            throw std::string("Unkown Error");
        }
    }
}

void FileLock::unlock() {
    fcntl(_file_des, F_SETLKW, file_lock(F_UNLCK, SEEK_SET));
}

int FileLock::get_locked_file() {
    return _file_des;
}
```

7.4 FileLock.h

```
/* FileLock.h  Copyright (c) Alexander Brown, March 2011 */

#ifndef _FILELOCK_H
#define	_FILELOCK_H

#include <string>

/**
 * @brief Locks a File so that only this process can use it.
 *
 * @author Alexander Brown
 * @author Neil Snooke (Locking Code).
 */
class FileLock {
public:
    /**
     * Creates a Lock on the given file.
     * @param filename  The file to lock.
     * @throws std::string If the lock fails.
     */
    FileLock(std::string filename) throw (std::string);

    /**
     * Destructs, freeing the lock in the process.
     */
    virtual ~FileLock();

    /**
     * Returns the file descriptor of the locked file.
     * @return The file descriptor of the locked file.
     * @deprecated Use of std::fstream makes this unecessary.
     */
    int get_locked_file();

private:
    /** A structure containing the details of the file lock. */
    struct flock* _file_lock;

    /** The file descriptor of the locked file. */
    int _file_des;

    /**
     * Performs the lock operation.
     * @throws std::string If the operation was not sucessful.
     */
    void lock() throw (std::string);

    /** Performs the unlock operation. */
    void unlock();
};

#endif	/* _FILELOCK_H */
```

7.5 Logger.cpp

```
/* Logger.cpp  Copyright (c) Alexander Brown, March 2011 */

#include <string>
#include <bits/basic_string.h>
#include <time.h>
#include <stdio.h>
#include <iostream>
#include <fstream>

#include "Logger.h"
#include "FileHandller.h"
#include "Setup.h"
#include "User.h"

using namespace std;

Logger::Logger() {
    _file_handller = new FileHandller(Setup::LOG_FILE);
}

Logger::~Logger() {
    delete _file_handller;
    _user = NULL;
}

void Logger::set_user(User* user) {
    _user = user;
}

void Logger::log(std::string activity, std::string filepath) {
        bool complete = _file_handller->append_file(generate_log_entry(activity, filepath));
        if(!complete) {
                log(activity, filepath);
        }
}

std::string Logger::generate_log_entry(std::string activity,
        std::string filepath) {
    std::string entry = encode_date_time();
    entry += " admin: ";
    entry += _user->get_uid() + "\t";
    entry += activity + "\t";
    int base_dir_pos = filepath.rfind(Setup::BASE_DIR);
    filepath = filepath.replace(base_dir_pos, base_dir_pos + Setup::BASE_DIR.length(), "");
    filepath = Setup::REPO_DIR + filepath;
    entry += filepath;


    return entry;
}

std::string Logger::encode_date_time() {
    time_t rawtime;
    struct tm * timeinfo;
    char buffer [20];

    time(&rawtime);
    timeinfo = localtime(&rawtime);
    strftime(buffer, 20, "%Y-%m-%d-%H-%M-%S", timeinfo);

    return std::string(buffer);
}

void Logger::read_log() {
    ifstream authstream(Setup::LOG_FILE.data());

    if (!authstream.is_open()) {
        throw string("Error opening '" + Setup::LOG_FILE + "'");
    }
    string line;
    getline(authstream, line);


    while (authstream.good()) {
        if (line.data()[0] != '#') {
            cout << "=======================================" << endl;
            std::string date_time = line.substr(0, 20);
            line = line.substr(20, line.length());
            date_time = decode_date_time(date_time);

            std::string user_type = line.substr(0,5);
            line = line.substr(7, line.length());

            std::string user = line.substr(0, line.find("\t"));
            line = line.substr(line.find("\t")+1, line.length());

            std::string action = line.substr(0, line.find("\t"));
```

```
            line = line.substr(line.find("\t")+1, line.length());

            cout << date_time;
            cout << "User:\t\t" << user << endl;
            cout << "Program:\t" << user_type << endl;
            cout << "Action:\t\t" << action << endl;
            cout << "File:\t\t" << line << endl;
        }
        getline(authstream, line);
    }
    cout << "=======================================" << endl;

    authstream.close();
}

std::string Logger::decode_date_time(std::string date_time) {
    struct tm tm;
    strptime(date_time.data(), "%Y-%m-%d-%H-%M-%S ", &tm);

    char buffer[80];
    strftime(buffer, 80, "Date:\t\t%d/%m/%Y\nTime:\t\t%H:%M:%S\n", &tm);

    return std::string(buffer);
}
```

7.6 Logger.h

```
/* Logger.h  Copyright (c) Alexander Brown, March 2011 */

#ifndef _LOGGER_H
#define _LOGGER_H

#include <string>

class User;
class FileHandller;

/**
 * @brief A class which handles all log file actions.
 *
 * This class deals with all operations that will be performed on
 * the logfile, ensuring the file is locked whilst these actions
 * are performed, etc.
 *
 * It also ensures that the same format is used, so that the
 * logfile can be read and formatted back.
 *
 * @author Alexander Brown
 * @version 1.0
 */
class Logger {
public:
    Logger();
    ~Logger();

    /**
     * Sets user to be the current User.
     *
     * @param user The current User.
     */
    void set_user(User *user);

    /**
     * Makes an entry to the logfile.
     *
     * @param activity  The activity that was performed.
     * @param filepath  The path to the file the activity was performed on.
     *
     * @see Logger::generateLogEntry(std::string, std::string)
     */
    void log(std::string activity, std::string filepath);

    /**
     * Prints the logfile in a human readable format.
     *
     * @see Logger::decodeDateTime(std::string)
     */
    void read_log(void);
private:
    /** Aggregation to the User */
    User * _user;

    FileHandller * _file_handller;

    /**
     * Generates the text for the log entry.
     *
     * @param activity  The activity that was performed.
     * @param filepath  The path to the file the activity was performed on.
     *
     * @return The textual form of a log entry.
     *
     * @see Logger::encodeDateTime()
     */
    std::string generate_log_entry(std::string activity, std::string filepath);

    /**
     * Encodes the current Date and Time into the string format used by the log
     * file.
     *
     * @return The current Date and Time into the string format used by the log
     * file.
     */
    std::string encode_date_time(void);

    /**
     * Decodes the Date and Time from a log entry into a human-readable format.
     *
     * @param date_time  The date and time to decode into a human readable
     * format
     *
     * @return The Date and Time from a log entry into a human-readable format.
     */
    std::string decode_date_time(std::string date_time);
```

```
};

#endif  /* _LOGGER_H */
```

7.7 main.cpp

```
/* main.cpp  Copyright (c) Alexander Brown, March 2011 */

#include <stdlib.h>
#include <stdio.h>
#include <iostream>

#include "Setup.h"
#include "User.h"
#include "Logger.h"

#include "UserCreator.h"
#include "UserType.h"

using namespace std;

#define FLAG_CREATE_ASSIGN  'c'
#define FLAG_CREATE_STAFF   's'
#define FLAG_CREATE_STDNT   'd'
#define FLAG_PRINT_LOG      'l'
#define FLAG_HELP           'h'
#define FLAG_NO_OPTION      'n'

void print_help() {
    cout << "\t" << FLAG_CREATE_ASSIGN << " [file]: Upload Assignment\n";
    cout << "\t" << FLAG_CREATE_STAFF << " [uid]: Create a new Staff User\n";
    cout << "\t" << FLAG_CREATE_STDNT << " [uid]: Create a new Student User\n";
    cout << "\t" << FLAG_PRINT_LOG << ": Read the log file\n";
}

int main(int argc, char** argv) {
    try {
        char flag = FLAG_NO_OPTION;

        Logger *logger = new Logger();
        User * user = User::login(logger);
        logger->set_user(user);
        Setup * setup = new Setup(logger);

        if (argc < 2) {
            //Allow the user to input their own choice rather than forcing them to use command line arguments.
            do {
                cout << "Enter option code (h for help): ";
                cin >> flag;
                switch (flag) {
                    case FLAG_CREATE_ASSIGN:
                        break;
                    case FLAG_CREATE_STAFF:
                        break;
                    case FLAG_CREATE_STDNT:
                        break;
                    case FLAG_PRINT_LOG:
                        break;
                    case FLAG_HELP:
                        print_help();
                        //Intended fallthrough so that the program doesn't continue and exit.
                    default:
                        flag = FLAG_NO_OPTION;
                }
            } while (flag == FLAG_NO_OPTION);
        } else {
            flag = argv[1][1];
        }

        std::string option;
        switch (flag) {
            case FLAG_CREATE_ASSIGN:
                if (argc < 3) {
                    cout << "Enter the file (including path) of the assignment: ";
                    cin >> option;
                } else {
                    option = argv[2];
                }
                setup->upload_assignment(option);
                break;
            case FLAG_CREATE_STAFF:
                if (argc < 3) {
                    cout << "Enter the UID for the new staff user: ";
                    cin >> option;
                } else {
                    option = argv[2];
                }
                user->add_staff(option);
                break;
            case FLAG_CREATE_STDNT:
                if (argc < 3) {
                    cout << "Enter the UID for the new student user: ";
                    cin >> option;
```

```
            } else {
                option = argv[2];
            }
            user->add_student(option);
            //TODO add something to create dir for the student.

            break;
        case FLAG_PRINT_LOG:
            logger->read_log();
            break;
        case FLAG_HELP:
            print_help();
            break;
        default:
            throw std::string("Unrecognised Option");
    }

    delete logger;
    delete user;
    delete setup;

    return (EXIT_SUCCESS);
  } catch (std::string e) {
    cout << argv[0] << ": " << e << endl;
    cout << "Useage: " << argv[0] << " [OPTION] [INPUT]" << endl;
    cout << "Try `" << argv[0] << " -h' for more information" << endl;
    return (EXIT_FAILURE);
  } catch (char const* c) {
    cout << c << endl;
  }
}
```

7.8 Setup.cpp

```
/* Setup.cpp  Copyright (c) Alexander Brown, March 2011 */

#include <string>
#include <iostream>
#include <fstream>

#include "Setup.h"
#include "FileHandller.h"
#include "Logger.h"

using namespace std;

std::string get_base_dir() {
    string line = "";
    ifstream config_file("config");
    if(config_file.is_open()) {
        getline(config_file, line);
    }
    line = line + "/";
    return line;
}

std::string get_repo_dir() {
    string line = "";
    ifstream config_file("config");
    if(config_file.is_open()) {
        getline(config_file, line);
        getline(config_file, line);
    }
    line = line + "/";
    return line;
}


// Change this to change the base directory.
const std::string Setup::BASE_DIR = get_base_dir();
const std::string Setup::REPO_DIR = get_repo_dir();

// Constant definitions.
const std::string Setup::AUTH_FILE = BASE_DIR+".auth";
const std::string Setup::LOG_FILE = BASE_DIR+"logfile.log";
const std::string Setup::ASSIGNMENT_DIR = BASE_DIR+"assignment/";
const std::string Setup::STUDENT_DIR = BASE_DIR+"students/";

Setup::Setup(Logger* logger): _logger(logger) {

}

Setup::~Setup() {
    _logger = NULL;
}

void Setup::upload_assignment(std::string src) throw (std::string) {
    int pos = src.find_last_of('/');
    std::string filename = src.substr(pos+1, src.length());

    FileHandller::upload_file(src, ASSIGNMENT_DIR+filename);

    _logger->log("Uploaded Assignment", ASSIGNMENT_DIR+filename);
}
```

7.9 Setup.h

```
/* Setup.h  Copyright (c) Alexander Brown, March 2011 */

#ifndef _SETUP_H
#define       _SETUP_H

#include <string>

class User;
class Logger;


/**
 * @brief A class to hadnle the setup of the repository.
 *
 * This class was orignally meant to handle the setup of the
 * retpository. However, due to a lack of time, I put the
 * majority of the code for this in the makefile as BASH
 * commands.
 *
 * Instead, this contains constants to locations in the
 * repository and allows for uploading of an assignment
 * file.
 *
 * @author Alexander Brown
 * @version 1.0
 */
class Setup {
public:
    /** The Base Directory of the repository. */
    static const std::string BASE_DIR;

    /** The Repository directory */
    static const std::string REPO_DIR;

    /** The location of the authentication file */
    static const std::string AUTH_FILE;

    /** The location of the log file */
    static const std::string LOG_FILE;

    /** The location of the assignment directory */
    static const std::string ASSIGNMENT_DIR;

    /** The location of the assignment directory */
    static const std::string STUDENT_DIR;

    /**
     * Aggregate the User and the Logger.
     * @param user      The User to aggregate to.
     * @param logger    The Logger to aggregate to.
     * @deprecated  Unneeded.
     */
    Setup(User *user, Logger *logger);

    /**
     * Aggregate the User and the Logger.
     * @param user      The User to aggregate to.
     * @param logger    The Logger to aggregate to.
     */
    Setup(Logger *logger);

    /** Removes the aggregation */
    virtual ~Setup(void);

    /** Sets up the entire repository. */
    void run_setup(void);

    /**
     * Allows an assignment to be uploaded.
     * @param src   The location of the local assignment file to upload.
     */
    void upload_assignment(std::string src) throw (std::string);

private:
    /**
     * Aggregation to the user
     * @deprecated  Unneeded.
     */
    User* _user;

    /** Aggregation to the logger */
    Logger* _logger;

    void create_dir_structure(void);
};

#endif       /* _SETUP_H */
```

7.10 User.cpp

```
/** User.cpp  Copyright (c) Alexander Brown, March 2011 */

#include <iostream>
#include <fstream>
#include <string>
#include <cstdio>

#include "Logger.h"

#include <termios.h>
#include <unistd.h>

#include "User.h"
#include "UserCreator.h"
#include "Setup.h"
#include "UserType.h"

using namespace std;

/**
 * Sets whether the stdin stream should echo to
 * the console or not.
 *
 * @param enable If stdin echo is enabled or not.
 *               defaults to true.
 * Note: Linux systems only.
 * @author Unknown (taken from Stack Overflow).
 */
void SetStdinEcho(bool enable = true) {
    struct termios tty;
    tcgetattr(STDIN_FILENO, &tty);
    if (!enable)
        tty.c_lflag &= ~ECHO;
    else
        tty.c_lflag |= ECHO;

    tcsetattr(STDIN_FILENO, TCSANOW, &tty);
}

User *User::_INSTANCE = NULL;

User::User(Logger *logger, std::string uid): _logger(logger), _uid(uid) {
}

User::~User() {
    _logger = NULL;
    _INSTANCE = NULL;
}

User* User::login(Logger *logger) throw (std::string) {
    // TODO decide if it would be best to break this function down further.
    if (_INSTANCE == NULL) {
        std::string user;
        std::string pass;

        cout << "Enter User ID: ";
        cin >> user;

        cout << "" << user << "'s password: ";

        SetStdinEcho(false);
        cin >> pass;
        SetStdinEcho(true);
        cout << endl;

        ifstream authstream(Setup::AUTH_FILE.data());

        if (!authstream.is_open()) {
            std::string t = "Error opening '" + Setup::AUTH_FILE + "'";
            throw t;
        }

        bool authenticated = false;
        string line;


        while (authstream.good()) {
            getline(authstream, line);
            if (line.data()[0] != '#') {
                if (line.find((user + " admin " + pass).data()) != string::npos) { // TODO See if this can be made more secure.
                    authenticated = true;
                }
            }
        }

        if (!authenticated) {
            std::string t = "Invalid Username/Password";
```

```
            throw t;
        }

        _INSTANCE = new User(logger, user);

        authstream.close();
    }
    return _INSTANCE;
}

std::string User::get_uid() {
    return _uid;
}

std::string User::add_user(std::string uid, UserType type) {
    UserCreator creator(uid, type);
    return creator.create_user();
}

std::string User::add_staff(std::string uid) {
    enum UserType type = STAFF;
    std::string pass = add_user(uid, type);
    _logger->log("Created Staff User", Setup::AUTH_FILE);
    cout << "Created Staff User: " << uid << endl << "password: " << pass << endl;
    return pass;
}

std::string User::add_student(std::string uid) {
    enum UserType type = STUDENT;
    std::string pass = add_user(uid, type);
    _logger->log("Created Student User", Setup::AUTH_FILE);
    cout << "Created Student User: " << uid << endl << "password: " << pass << endl;
    return pass;
}
```

7.11 UserCreator.cpp

```
/* UserCreator.cpp  Copyright (c) Alexander Brown, March 2011 */

#include <stdlib.h>
#include <time.h>
#include <sys/stat.h>
#include <sys/types.h>

#include "UserCreator.h"
#include "Setup.h"
#include "FileHandller.h"

UserCreator::UserCreator(std::string uid, enum UserType type) : _uid(uid), _type(type) {
    _file_handller = new FileHandller(Setup::AUTH_FILE);
}

UserCreator::~UserCreator() {
    delete _file_handller;
}

std::string UserCreator::create_user() {
    std::string passwd = generate_password();
    std::string entry = _uid + " ";
    switch (_type) {
        case STAFF:
            entry += "staff ";
            break;
        case STUDENT:
            entry += "stdnt ";
            mkdir((Setup::STUDENT_DIR+_uid+"/").data(), 0744);
            mkdir((Setup::STUDENT_DIR+_uid+"/results/").data(), 0744);
            break;
        default:
            entry = "#" + entry; //Something went wrong so comment it out.
    }
    entry += passwd;
    _file_handller->append_file(entry);
    return passwd;

}

std::string UserCreator::generate_password() {
    srand(time(NULL));
    int passwd_length = 8 + (rand() % 5);
    std::string passwd = "";

    for (int i = 0; i < passwd_length; i++) {
        /* Modulo of the last character of the characters we want (z, as
         * lowercase is after uppercase and numbers) minus the first character,
         * plus one to make it encompass them all. Plus the starting character).
         */
        char c = rand() % ('z' - '0' + 1) + '0';
        passwd += c;
    }

    return passwd;
}
```

7.12 UserCreator.h

```
/* UserCreator.h Copyright (c) Alexander Brown, March 2011 */

#ifndef _USERCREATOR_H
#define _USERCREATOR_H

#include <string>

#include "UserType.h"

class User;
class Logger;
class FileHandller;

/**
 * @brief Handles the creation of users.
 *
 * This classes defines the behaviour to use when creating
 * new users, ensuring this behaviour is standardised.
 *
 * @author Alexander Brown
 * @version 1.0
 */
class UserCreator {
public:
	/** Constructs a UserCreator with the given uid and type. */
	UserCreator(std::string uid, enum UserType type);

	/** Destructs the UserCreator. */
	~UserCreator();

	/** Writes the user to the .auth file */
	std::string create_user();
private:
	/** The file handller */
	FileHandller * _file_handller;

	/** The User ID of the user to create */
	std::string _uid;

	/** The generated password for the User */
	std::string _password;

	/** The type of the User */
	enum UserType _type;

	/** Generates a random password */
	std::string generate_password();
};

#endif  /* _USERCREATOR_H */
```

7.13 User.h

```
/* User.h  Copyright (c) Alexander Brown, March 2011 */

#ifndef _USER_H
#define _USER_H

#include <string>
#include "UserType.h"

// Define the Logger class to stop strange dependancy errors.
class Logger;

/**
 * @brief Defines the User for the Administator Programs and associated methods.
 *
 * This class is mostly used to control the logging in and out of the User, as
 * well as a few base actions (adding Users) so most items can be set up without
 * too much hastle.
 *
 * This class is a singleton, hence the private constructor, and can only be
 * instantiated through the login method.
 */
class User {
public:
    /**
     * Logs in a user.
     *
     * @param logger     The aggregation to the logger.
     *
     * @return The single instance of the User. Note that if the User is already
     * logged in then this will still return the instance, just without any
     * ability to re-login.
     */
    static User* login(Logger *logger) throw (std::string);

    /**
     * Destructor. As ISTANCE is the only pointer that needs to be freed in this
     * class and is the one this destructor will be called upon, this doesn't
     * acutally do much, other than setting a few nulls.
     */
    ~User(void);

    /**
     * Adds a member of staff to the program.
     *
     * @param uid    The User ID of the Staff to add.
     *
     * @param The auto-generated password.
     *
     * @see User::addUser(std::string, enum UserType)
     */
    std::string add_staff(std::string uid);


    /**
     * Adds a student to the program.
     *
     * @param uid    The User ID of the Student to add.
     *
     * @param The auto-generated password.
     *
     * @see User::addUser(std::string, enum UserType)
     */
    std::string add_student(std::string uid);

    /**
     * Returns the User ID of the current User.
     *
     * @return The User ID of the current User.
     */
    std::string get_uid(void);

private:
    /** The single instance of this class */
    static User * _INSTANCE;

    /** Aggregation to the logger. */
    Logger * _logger;

    /** The User ID of the User, used for logging purposes. */
    std::string _uid;

    /**
     * Private Constructor to ensure this remains a singleton.
     *
     * @param logger     The aggregation to the Logger.
     * @param uid        The User ID of this User.
     */
```

```
    User(Logger *logger, std::string uid);

    /**
     * Adds a user to the program, returning the auto-generated password.
     *
     * @param uid   The User ID of the user to create.
     * @param type  The type of the User.
     *
     * @param The auto-generated password.
     */
    std::string add_user(std::string uid, enum UserType type);
};

#endif  /* _USER_H */
```

7.14 UserType.h

```
/* UserType.h  Copyright (c) Alexander Brown, March 2011 */

#ifndef _USERTYPE_H
#define _USERTYPE_H

enum UserType {
    STAFF = 0,
    STUDENT = 1
};

#endif  /* _USERTYPE_H */
```

8 Source code - Staff Program

8.1 InvalidUserException.java

```
package uk.ac.aber.users.adb9.cs22510.staff_program;

/**
 * An Exception for throwing when a Username or Password is incorrect.
 * <p>
 * Thrown from {@link User#login(Logger)} so that the program doesn't have
 * to compare ugly things like {@code null}s to check if a User logged in.
 *
 * @author Alexander Brown
 * @version 1.0
 */
public class InvalidUserException extends Exception {
	/** The Serial Version UID for serialisation potential. */
	private static final long serialVersionUID = -239788711036612392L;

	/** The default error message. */
	private static final String DEFAULT_MESSAGE = "Invalid User/Password";

	/** Displays the default error message */
	public InvalidUserException() {
		super(DEFAULT_MESSAGE);
	}
}
```

8.2 Logger.java

```
package uk.ac.aber.users.adb9.cs22510.staff_program;

import java.text.SimpleDateFormat;
import java.util.Date;

import uk.ac.aber.users.adb9.cs22510.staff_program.io.FileHandller;

/**
 * Defines the logging class.
 * <p>
 * This class handles all entries that have to be made to the
 * {@linkplain StaffProgram#LOG_FILE logfile}, controlling how the entries
 * are made and the format in which they are made.
 *
 * @author		Alexander Brown
 * @version		1.0
 *
 * @see FileHandller
 */
public class Logger {
	/** Link back to the User */
	private User user;

	/**
	 * Sets the link to the User.
	 * @param user		The User to set the link back to.
	 */
	public void setUser(User user) {
		this.user = user;
	}

	/**
	 * Makes an entry to the logfile.
	 *
	 * @param activity		The activity to be logged.
	 * @param filepath		The path to the file that the activity was
	 * performed on.
	 *
	 * @see generateLogEntry()
	 */
	public void log(String activity, String filepath) {
		FileHandller handller = new FileHandller(StaffProgram.LOG_FILE);
		handller.appendFile(generateLogEntry(activity, filepath));
		handller.close();
	}

	/**
	 * Generates the text for a log entry
	 *
	 * @param activity		The activity to be logged.
	 * @param filepath		The path to the file that the activity was
	 * performed on.
	 *
	 * @see encodeDateTime()
	 */
	private String generateLogEntry(String activity, String filepath) {
		String file = filepath.replace(StaffProgram.BASE_DIR, "");

		String[] path = StaffProgram.BASE_DIR.split("/");
		String base = path[path.length-1];

		String entry = String.format("%s staff: %s\t%s\t%s/%s\n",
			encodeDateTime(),
			user.getUID(),
			activity,
			base,
			file);
		return entry;
	}

	/**
	 * Takes the date and time and encodes them into the string format
	 * used by the log.
	 *
	 * @return An encoded date time.
	 */
	private String encodeDateTime() {
		Date d = new Date();
		SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd-kk-mm-ss");
		return df.format(d);
	}
}
```

8.3 StaffProgram.java

```
package uk.ac.aber.users.adb9.cs22510.staff_program;

import java.util.Scanner;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class StaffProgram {
	private static final File CONFIG_FILE = new File("config");

	public static final String BASE_DIR = getBaseDir();
	public static final String LOG_FILE = BASE_DIR + "logfile.log";
	public static final String AUTH_FILE = BASE_DIR + ".auth";
	public static final String STUDENT_DIR = BASE_DIR + "students/";

	/**
	 * Defins the list option
	 * @see User#list()
	 */
	private static final char LIST = 'l';

	/**
	 * Defines the get option
	 * @see User#get(String uid, String destPath);
	 */
	private static final char GET  = 'g';

	/**
	 * Defines the mark option
	 * @see User#mark(String uid);
	 */
	private static final char MARK = 'm';

	/** Defines the quit option */
	private static final char QUIT = 'q';

	/**
	 * Defines the help option
	 * @see help()
	 */
	private static final char HELP = 'h';

	/** Defines the Unrecognised option, used to continue looping the menu. */
	private static final char NONE = ' ';

	public static void main(String args[]) {
		Logger logger = new Logger();
		User user = null;
		try {
			user = User.login(logger);
		} catch (InvalidUserException e) {
			System.err.println(e.getMessage());
			System.exit(1);
		}

		logger.setUser(user);

		char option = NONE;
		Scanner in = new Scanner(System.in);

		while (option == NONE) {
			System.out.print("Enter option code (h for help): ");
			option = in.next().charAt(0);
			String uid = "";
			switch (option) {
				case LIST:
					user.list();
					break;
				case  GET:
					System.out.print("Enter the uid of the student to get the submission of: ");
					uid = in.next();
					System.out.print("Enter the location you wish to download this submission to: ");
					String file = in.next();
					user.download(uid, file);
					break;
				case MARK:
					System.out.print("Enter the uid of the student to submit a mark for: ");
					uid = in.next();
					user.mark(uid);
					break;
				case QUIT:
					break;
				case HELP:
					StaffProgram.help();
					option = NONE;
```

```
                        break;
                    default:
                        System.out.println("Unrecognised option '"+option+"'");
                        option = NONE;
                }
            }
        }

        /** Prints out the help options */
        private static void help() {
            System.out.printf("\t%s - List all students with submitted work.\n"
            +"\t%s - Download a specified student's work.\n"
            +"\t%s - Submit a mark for a specified student's work.\n"
            +"\t%s - Quit the program.\n",
            LIST,
            GET,
            MARK,
            QUIT);
        }

        /**
         * Gets the Base Directory from the config file
         * @return The path of the Base Directory
         */
        private static String getBaseDir() {
            try {
                String line = "";
                if(CONFIG_FILE.exists()) {
                    Scanner s = new Scanner(new InputStreamReader(
                                new FileInputStream(CONFIG_FILE)));
                    line = s.nextLine();
                }

                return line+"/";
            }
            catch (IOException ioe) {
                ioe.printStackTrace();
                return "";
            }
        }
}
```

8.4 User.java

```
package uk.ac.aber.users.adb9.cs22510.staff_program;

import java.io.Console;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Scanner;

import uk.ac.aber.users.adb9.cs22510.staff_program.io.FileHandller;

public class User {
	/** The Singleton Instance. */
	static private User INSTANCE = null;

	/**
	 * Logs in a User.
	 * <p>
	 * This is the only way of creating a User, and ensures that the singleton
	 * design patter this class uses is kept strict.
	 * <p>
	 * As expected from a login method, the system asks for a User ID and password
	 * (the latter of which is concealed by the Console), then checks them against
	 * the {@linkplain StaffProgram#AUTH_FILE authentication file} and acts
	 * accordingly.
	 *
	 * @param logger	The link back to the Logger.
	 *
	 * @throws InvalidUserException	If the User ID or connected password is
	 * incorrect.
	 *
	 * @see Console
	 */
	public static User login(Logger logger) throws InvalidUserException {
		Console c = System.console();
		if(c == null) {
			System.err.println("No Console.");
			System.exit(2);
		}

		Scanner s = new Scanner(System.in);
		System.out.print("Enter User ID: ");
		String uid = s.next();

		System.out.printf("Password for %s: ", uid);
		char[] charPass = c.readPassword("");
		String passwd = String.valueOf(charPass);

		FileHandller f = new FileHandller(StaffProgram.AUTH_FILE);
		File auth = f.getFile();
		String line;
		Scanner reader = null;

		try {
			reader = new Scanner(new InputStreamReader(
					new FileInputStream(auth)));
			while (reader.hasNext()) {
				line = reader.nextLine();
				if (line.equals(String.format("%s staff %s", uid, passwd))) {
					INSTANCE = new User(uid, logger);
				}
			}
			reader.close();
		} catch (IOException eio) {

		}

		f.close();

		if (INSTANCE == null) {
			throw new InvalidUserException();
		}

		return INSTANCE;
	}

	/** The User ID of the current User */
	private String uid;

	/** The link to the Logger */
	private Logger logger;

	/** The File Handller this class makes use of */
	private FileHandller fileHandller;

	/**
```

```
 * Creates a new User, with the given User ID and a link to a Logger.
 * <p>
 * Note that this is private to facilitate the singleton design pattern this class uses.
 *
 * @param uid               The User ID of this User.
 * @param logger        The link to the Logger.
 */
private User(String uid, Logger logger) {
        this.uid = uid;
        this.logger = logger;
}

/**
 * Returns the User ID of the current User.
 * @return The User ID of the current User.
 */
public String getUID() {
        return uid;
}

/**
 * Lists all Students that have made submissions.
 * <p>
 * Works by looping through all directories in the {@linkplain StaffProgram#STUDENT_DIR student directory}
 * looking for those with more than 1 file contained within them (bearing in mind a directory for results
 * is created in the student's directory when it's created). And writing their username (which is handily
 * the name of the directory too) to a String.
 * <p>
 * Assuming any Students have made a submission, this String is then printed, otherwise a message to tell
 * the user than no Students have made submissions yet is displayed.
 *
 * @see FileHandller#getFile()
 * @see File#listFiles()
 */
public void list() {
        //fileHandller = new FileHandller(StaffProgram.STUDENT_DIR);
        //TODO - Cannot currently lock directories.
        File studentDir = new File(StaffProgram.STUDENT_DIR); //fileHandller.getFile();
        File[] students = studentDir.listFiles();
        boolean submission = false;
        String submissions = "The following students have made submissions:\n";
        for(File student:students) {
                if(student.listFiles().length > 1) {
                        submission = true;
                        submissions += "\t" + student.getName() + "\n";
                }
        }
        if(submission) {
                System.out.print(submissions);
        } else {
                System.out.println("No Students have made submissions");
        }

        logger.log("Listed Submissions",StaffProgram.STUDENT_DIR);

        //fileHandller.close();
}

public void download(String uid, String destination) {
        File studentDir = new File(StaffProgram.STUDENT_DIR);
        File[] studentDirs = studentDir.listFiles();

        for(File student:studentDirs) {
                if(student.getName().equals(uid)) {
                        File[] studentFiles = student.listFiles();
                        if(studentFiles.length > 1) {
                                for(File assignFile:studentFiles) {
                                        if(!(assignFile.getName().equals("results"))) {
                                                fileHandller = new FileHandller(assignFile.getAbsolutePath());
                                                fileHandller.downloadFile(destination);
                                                fileHandller.close();
                                                logger.log("Downloaded assignment", assignFile.getAbsolutePath());
                                                return;
                                        }
                                }
                        }
                }
        }
        System.out.println("Could not find assigment of student with uid: "+uid);
}

public void mark(String uid) {
        File studentDir = new File(StaffProgram.STUDENT_DIR);
        File[] studentDirs = studentDir.listFiles();
        for(File student:studentDirs) {
                if(student.getName().equals(uid)) {
                        File[] studentFiles = student.listFiles();
                        if(studentFiles.length > 1) {
                                File markFile = new File(student.getAbsolutePath()+"/results/result.txt");
                                if(!markFile.exists()) {
```

```
                                try {
                                        System.out.println(markFile.getAbsolutePath());
                                        markFile.createNewFile();
                                        fileHandller = new FileHandller(markFile.getAbsolutePath());
                                        Scanner in = new Scanner(System.in);
                                        System.out.print("Enter the mark (%): ");
                                        int mark = in.nextInt();
                                        in.nextLine();
                                        System.out.println("Enter any comments: ");
                                        String comment = in.nextLine();

                                        fileHandller.appendFile(String.format("%d%%\n%s\n", mark, comment));
                                        fileHandller.close();

                                        logger.log("Submitted mark for "+uid, markFile.getAbsolutePath());
                                        return;
                                } catch (IOException e) {
                                        System.err.println(e);
                                }
                        }
                }
        }

        System.err.println("Cannot submit mark: Student does not exist or has not yet submitted");
    }
}
```

9 Source code - Student Program

9.1 file_lock.c

```
/**
 * @author Alexander Brown
 */

#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>

#include "file_lock.h"

/**
 * @author Neal Snooke
 */
struct flock* file_lock(short type, short whence) {
    static struct flock ret;
    ret.l_type = type;
    ret.l_start = 0;
    ret.l_whence = whence;
    ret.l_len = 0;
    ret.l_pid = getpid();
    return &ret;
}

int lock(char * filepath) {
        printf("Locking file...\n");
        int file_descriptor = open(filepath, O_RDWR);

        if(file_descriptor == -1) {
                printf("Could not open %s: File Descriptor Error\n", filepath);
                return -1;
        }

        struct flock* fl = file_lock(F_WRLCK, SEEK_SET);

        if(fcntl(file_descriptor, F_SETLK, fl) == -1) {
                if(errno == EACCES || errno == EAGAIN) {
                        printf("Unable to lock %s: File is already locked by another process.\n", filepath);
                } else {
                        printf("Unable to lock %s: Unknown error.\n", filepath);
                }
                return -1;
        } else { // Lock has been obtained.
                return file_descriptor;
        }
}

int unlock(int file_descriptor) {
        if(file_descriptor == -1) {
                printf("Cannot unlock file: File Descriptor Error\n");
                return 0;
        } else {
                fcntl(file_descriptor, F_SETLKW, file_lock(F_UNLCK, SEEK_SET));
                close(file_descriptor);
                return 1;
        }
}
```

9.2 file_lock.h

```
/**
 * @brief Creates a lock on a file.
 *
 * @author Alexander Brown
 * @version 1.0
 */

/**
 * @brief Locks a file.
 *
 * @param filepath The path to the file.
 * @return The file descriptor of the file to lock, -1 if anything fails.
 */
int lock(char * filepath);

/**
 * @brief Unlocks a file.
 *
 * @param file_descriptor
 * @return
 *         1 - If the file unlocked sucessfully.
 *         0 - If not.
 */
int unlock(int file_descriptor);
```

9.3 logger.c

```
/*
 * logger.c copyright (c) Alexander Brown, March 2011
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

#include "utils.h"
#include "logger.h"
#include "file_lock.h"

#ifndef MAX_ENTRY_SIZE
#define MAX_ENTRY_SIZE 100
#endif

char * logfile;

void get_logfile(void) {
	if(logfile != NULL) {
		return;
	}
	logfile = malloc(115 * sizeof(char));
	char * base_dir = malloc(85 * sizeof(char));
	get_base_dir(base_dir);
	strcat(logfile, base_dir);
	strcat(logfile, "/logfile.log");
	free(base_dir);
}

void log(char * activity, char * uid, char * filepath) {
	get_logfile();
	int fd = lock(logfile);

	FILE * logfile = fdopen(fd, "a");
	printf("File open.\n");

	char * entry = malloc(MAX_ENTRY_SIZE * sizeof(char));
	generate_log_entry(entry, activity, uid, filepath);
	fprintf(logfile, "%s", entry);
	fclose(logfile);

	unlock(fd);
	free(entry);
}

void generate_log_entry(char * entry, char * activity, char * uid, char * filepath) {
	//yyyy-mm-dd-hh-mm-ss stdnt: %uid	%activity	%filepath
	encode_date_time(entry);
	strcat(entry, " stdnt: ");
	strcat(entry, uid);
	strcat(entry, "\t");
	strcat(entry, activity);
	strcat(entry, "\t");
	strcat(entry, filepath);
	strcat(entry, "\n");
}

void encode_date_time(char * entry) {
	time_t rawtime;
	struct tm * timeinfo;
	char time_str[21];

	time(&rawtime);
	timeinfo = localtime(&rawtime);
	strftime(time_str, 21, "%Y-%m-%d-%H-%M-%S", timeinfo);
	strcat(entry, time_str);
}

void free_logger() {
	free(logfile);
}
```

9.4 logger.h

```
#ifndef _LOGGER_H
#define _LOGGER_H

/**
 * Logs an activity.
 *
 * @param activity      The activity performed.
 * @param uid           The user ID of the user.
 * @param filepath      The path to the file that the activity was performed upon.
 */
void log(char * activity, char * uid, char * filepath);

void generate_log_entry(char * entry, char * activity, char * uid, char * filepath);

void encode_date_time(char * entry);

void free_logger(void);

#endif /* _LOGGER_H */
```

9.5 main.c

```
/**
 * @author Alexander Brown
 */

#ifndef LOGIN_FAILURE
#define LOGIN_FAILURE 12
#endif

#include <stdlib.h>
#include <stdio.h>
#include "user.h"
#include "logger.h"

#ifndef MENU_OPTIONS
#define MENU_OPTIONS
#define QUIT      'q'
#define HELP      'h'
#define GET       'g'
#define SUBMIT      's'
#define MARK      'm'
#define NONE      ' '
#endif

static char uid[7];

int main(int argc, char ** argv) {
	get_logfile();
	printf("Enter User ID: ");
	scanf("%6s", uid);

	if(!(login(uid) == 1)) {
		printf("Invalid Username/Password\n");
		return 12;
	}

	if(argc < 2) {
		char option = NONE;
		char path[100];
		while(option == NONE) {
			printf("Enter option code (h for help): ");
			fflush(stdin);
			scanf("%1s", &option);
			getchar(); // getchar() apparently stops this looping infinitely on my machine.
			switch(option) {
				case GET :
					printf("Location to download the assignment to: ");
					scanf("%98s", path);
					if(download_assignment(path) == 1) {
						log("Downloaded assignment", uid, "repository/assignment/");
					}
					break;
				case SUBMIT:
					printf("Location of file to submit: ");
					scanf("%98s", path);
					if(submit_assignment(path, uid) == 1) {
						strcpy(path, "repository/students/");
						strcat(path, uid);
						strcat(path, "/");
						log("Uploaded assignment", uid, path);
					}
					break;
				case MARK:
					view_marks(uid);
					break;
				case HELP:
					printf("\t%c - Get Assignment.\n\t%c - Submit Assignment.\n\t%c - View Marks.\n\t%c - Quit.\n",
						GET,
						SUBMIT,
						MARK,
						QUIT);
					option = NONE; // To allow looping
					break;
				case QUIT:
					break;
				default:
					printf("Unrecognised option '%s'\n", &option);
					option = NONE;
			}
		}
	}
	free_logger();
	return EXIT_SUCCESS;
}
```

9.6 user.c

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>
#include <sys/stat.h>
#include <string.h>

#include <termios.h>
#include <unistd.h>

#include "user.h"
#include "utils.h"
#include "file_lock.h"

void set_stdin_echo(int enable) {
    struct termios tty;
    tcgetattr(STDIN_FILENO, &tty);
    if (!enable)
        tty.c_lflag &= ~ECHO;
    else
        tty.c_lflag |= ECHO;

    (void) tcsetattr(STDIN_FILENO, TCSANOW, &tty);
}

int login(char * uid) {
        set_stdin_echo(0);
        char passwd[13];
        printf("%s's password: ", uid);
        scanf("%12s", passwd);
        set_stdin_echo(1);

        char * base_dir = malloc(115 * sizeof(char));
        get_base_dir(base_dir);
        strcat(base_dir, "/.auth");

        int auth_fd = lock(base_dir);

        if(auth_fd == -1) {
                printf("Error: .auth file not found");
                free(base_dir);
                return 0;
        }

        FILE * auth = fdopen(auth_fd, "r");
        char line[100];
        char user_line[100];
        int autherised = 0;

        strcpy(user_line, uid);
        strcat(user_line, " stdnt ");
        strcat(user_line, passwd);
        strcat(user_line, "\n");
        while(fgets(line, 100, auth) != NULL) {
                if(strcmp(user_line, line) == 0) {
                        autherised = 1;
                }
        }

        fclose(auth);
        unlock(auth_fd);
        free(base_dir);
        return autherised;
}

int download_assignment(char * dest_path) {
        char * src_path = malloc(115 * sizeof(char));
        DIR * assignment_dir;
        struct dirent * dir_entry;
        struct stat entry_stat;
        int has_file = 0;

        get_base_dir(src_path);
        strcat(src_path, "/assignment/");
        assignment_dir = opendir(src_path);

        while((dir_entry = readdir(assignment_dir)) != NULL) {
                printf("%s\n",dir_entry->d_name);
                if(stat(dir_entry->d_name, &entry_stat) == -1) {
                }

                if(S_ISREG(entry_stat.st_mode)) {
                        strcat(src_path, dir_entry->d_name);
                        has_file = 1;
                        break;
                }
        }
```

```
        closedir(assignment_dir);

        if(has_file == 0) {
                printf("No assignment file found\n");
                return 0;
        }

        int ret = copy(src_path, dest_path);
        free(src_path);
        return ret;
}

int submit_assignment(char * src_path, char * uid) {
        char * dest_path = malloc(115 * sizeof(char));
        get_base_dir(dest_path);
        strcat(dest_path, "/students/");
        strcat(dest_path, uid);
        strcat(dest_path, "/");

        char * tokens = strtok(src_path, "/");
        char * filename;

        while(tokens != NULL) {
                filename = tokens;
                tokens = strtok(NULL, "/");
        }
        strcat(dest_path, filename);
        int ret = copy(src_path, dest_path);
        free(dest_path);
        return ret;
}

void view_marks(char * uid) {
        char * dest_path = malloc(115 * sizeof(char));
        get_base_dir(dest_path);
        strcat(dest_path, "/students/");
        strcat(dest_path, uid);
        strcat(dest_path, "/results/result.txt");

        int dest_fd = lock(dest_path);
        FILE * dest = fdopen(dest_fd, "r");

        if(dest == NULL) {
                printf("No results found for user '%s'\n", uid);
                free(dest_path);
                return;
        }

        int mark;
        char comments[200];
        fscanf(dest, "%d%%\n", &mark);
        fgets(comments, 200, dest);

        char * grade;

        if (mark <= 15)
                grade = "F-";
        else if (mark <= 30)
                grade = "F";
        else if (mark <= 34)
                grade = "F+";
        else if (mark <= 39)
                grade = "E";
        else if (mark <= 43)
                grade = "D-";
        else if (mark <= 46)
                grade = "D";
        else if (mark <= 59)
                grade = "D+";
        else if (mark <= 53)
                grade = "C-";
        else if (mark <= 56)
                grade = "C";
        else if (mark <= 59)
                grade = "C+";
        else if (mark <= 63)
                grade = "B-";
        else if (mark <= 66)
                grade = "B";
        else if (mark <= 69)
                grade = "B+";
        else if (mark <= 79)
                grade = "A-";
        else if (mark <= 89)
                grade = "A";
        else if (mark <= 95)
                grade = "A+";
        else
                grade = "A++";
```

```
        printf("Results for user '%s':\n\tMark: %d%% (%s)\n\tComments: %s\n", uid, mark, grade, comments);

        fclose(dest);
        unlock(dest_fd);
        free(dest_path);
}

int copy(char * src_path, char * dest_path) {
        FILE * dest;
        FILE * src;
        int src_fd;
        int reader;

        src_fd = lock(src_path);
        if(src_fd == -1) {
                return 0;
        }
        src = fdopen(src_fd, "rb");
        dest = fopen(dest_path, "wb");

        printf("Copying.");
        while(1) {
                reader = fgetc(src);
                if(reader == EOF) break;
                fputc((int) reader, dest);
        }
        printf("\n");

        fclose(src);
        fclose(dest);
        unlock(src_fd);
//      free(src_path);
        return 1;
}
```

9.7 user.h

```
/* user.h Copyright (c) Alexander Brown, March 2011 */

/**
 * @brief       Defines operations for a user.
 * @author      Alexander Brown.
 * @version     1.0
 */

#ifndef _USER_H
#define _USER_H

/**
 * Logs in a user.
 * @param uid          The user id of the user.
 * @return          <code>1</code> - If the operation was sucessful.
 */
int login(char * uid);

/**
 * Downloads the assignment.
 * @param dest_path        The path of the file to download to.
 * @return              <code>1</code> - If the operation was sucessful.
 */
int download_assignment(char * dest_path);


int submit_assignment(char * src_path, char * uid);

/**
 * Views the marks and comments for this assignment.
 * @param uid        The User ID of the user.
 */
void view_marks(char * uid);

/**
 * Copies the source file to the destination file.
 * @param src_path        The path to the source file.
 * @param dest_path        The path to the destination file.
 * @return              <code>1</code> - If the operation was sucessful.
 */
int copy(char * src_path, char * dest_path);

#endif /* _USER_H */
```

9.8 utils.c

```
/*
 * utils.c copyright (c) Alexander Brown, March 2011
 */


#include <stdlib.h>
#include <stdio.h>

#include "utils.h"


void get_base_dir(char * base_dir) {
	FILE * file = fopen("config", "r");
	if(file == NULL) {
		printf("config file not found.");
		exit(1);
	}
	fscanf(file, "%s\n", base_dir);
	fclose(file);
}
```

9.9 utils.h

```
/**
 * @author Alexander Brown.
 */

#ifndef _UTILS_H
#define _UTILS_G

void get_base_dir(char * base_dir);

#endif /* _UTILS_H */
```