COMP 310 – ECSE 427 Winter 2021

Take Home Final Exam Project

Start: April 7, 2021 9:00 EDT, **Due**: April 21, 2021 9:00 EDT

Instructions

- **IMPORTANT:** You **must answer and sign Question 1** for us to grade your Question 2. **If Question 1 is not completed, we will not grade Question 2 even if you submitted a solution.**

- This is an **individual, open-book, closed-internet exam**.
  Therefore, the following sources **are not permitted:**
  - You are *not* permitted to get help from friends or strangers.
  - You are *not* permitted to use any online tools, chat services, or websites like Chegg, Reddit and Stack Overflow, or other such places.
  - You are *not* permitted to use the internet in any way.
  - You *cannot use* course material from any other course.

  You **are permitted** to access the following course materials:
  - myCourses and mimi.
  - You are permitted to use the textbook and your course notes.
  - You are permitted to access the labs, assignments, and lectures of this course.
- There are no extensions for this assignment. You cannot submit a solution to this test after the due date and time.
- Partial marks are awarded, according to the marking scheme we provide.
- We will use MOSS to verify that the code you wrote is unique to you. We will check your code to online resources that have similar code in case you copied from those resources.
- If you have questions, the instructors will be available during these 14 days on Piazza (please post a **private message** – public questions for this test will be deleted).
- The instructors will have office hours on Zoom where you can ask questions. The schedule and Zoom links have been published on myCourses.

Grading

Question 1	…………………………………………………………………………………	YES / NO
Question 2	…………………………………………………………………………………	________ / 40

Question 1 – Academic Integrity

IMPORTANT: Question 2 will not be graded if Question 1 is not signed and submitted.

McGill University values academic integrity. Therefore, all students must understand the meaning and consequences of cheating, plagiarism and other academic offences under the Code of Student Conduct and Disciplinary Procedures (see www.mcgill.ca/students/srr/honest/ for more information, approved by Senate on 29 January 2003).

Please affirm the following statement:

> I have neither given nor received unauthorized aid on this exam and I agree to adhere to the specific Terms and Conditions that govern this exam.
>
> Sign in this box:

chen kua

Please sign in the above box, and then take a picture or scan this page or sign using a PDF signature, and upload this signed page to the final exam Assignment submission box.

Note: The submission box accepts multiple uploads, so you will be able to submit your solution to Question 2 after you upload the signed picture of Question 1. Please do that now.

Question 2 – Programming Assignment

Preliminary Notes

Problems. The exam consists of **4 problems, each worth 10 points:**

- Three problems contained in this document, which are common to all students.
- One randomized problem. You will find the 4th problem in the myCourses Quiz tool entitled Final Exam Project Question.
- We highly recommend attempting to solve the problems in the given order.

Codebase. You have been given a codebase to work from. This codebase is a valid solution for Assignment #4. Please take some time to get accustomed to this codebase, as you will need to modify this given codebase to solve the 4 problems. You are allowed (and encouraged) to reuse any functions and data structures provided in the codebase to implement the new functionality.

Do **not** use your own OS developed in the assignments for this exam.

In this exam you will be asked to modify the given codebase to implement four additional script commands. Testfile scripts for each problem are provided to you in this document (and in the Quiz tool for the 4th problem). You have also been provided special source files and code hooks where you must implement the extra functionality – you will find more details in the problem text.

To compile the codebase, run the provided `./recompile.sh` script.

The following commands should run on mimi with the provided codebase. Compile the codebase and run the commands below (also available in `TESTFILE.txt` provided to you) now:

```
1. mount partition0 10 5
2. write file1 [hello world test]
3. read file1 var
4. print var
5. quit
```

The command on line 4 should display: `hello world test`

IMPORTANT: If the codebase fails to execute, contact Prof. Balmau or Prof. Vybihal immediately.

Submitting your solutions

All solutions (even to the problem in the Quiz tool), must be submitted to the submission box in the Assignment tool. The submission box accepts multiple submissions. We strongly recommend answering this exam in the following order:

1. **Sign and upload Question 1** to the Assignment submission box for this exam (you should have done so already!). **If you do not complete Question 1, we will not grade Question 2.**
2. **Upload the solutions to Problems 1, 2, and 3** presented in this document to the Assignment submission box for this exam. Remember that partial marks are given.
3. After completing step 2, go to the Quiz tool and get your final randomized Problem 4. **When entering the quiz, your randomized question will be assigned to you.** We suggest you take a screenshot of the final question and save the screenshot to your computer.
4. The Quiz tool will present Problem 4 as a true/false question. After reading it and taking the screen shot, **select true, and submit the quiz.**
5. **Press submit on the Quiz tool** to prove to us that you looked at the Problem 4 question. **If you did not press submit, we will not grade your solution to Problem 4.**
6. **Upload the solution to Problem 4** to the submission box in the Assignment tool for this exam.

You have now completed the exam!

Problem 1: Open Command [10 points]

Change the provided codebase so that you **cannot read or write to a file unless you initially open the file**. Trying to read or write to a file without first opening the file will generate an error message. The syntax of your open command is:

```
open_EXAM <file_ID_in_active_file_table> filename
```

- The `file_ID_in_active_file_table` parameter is an integer that corresponds to the array cell index in the active file table where the user wishes to open the file.
- The `filename` parameter is the name of the file that will be opened by the `open_EXAM` command.

You will also need to adjust change the read and write commands. Do not modify the read and write commands that were given to you. Instead, add the following 2 commands:

```
read_EXAM <file_ID_in_active_file_table> var
```

- The `file_ID_in_active_file_table` parameter is an integer that corresponds to the array cell index in the active file table where the file has been opened.
- The `var` parameter is a shell variable into which the contents of the file will be read.

```
write_EXAM <file_ID_in_active_file_table> [alphanumeric strings separated by
spaces]
```

- The `file_ID_in_active_file_table` parameter is an integer that corresponds to the array cell index in the active file table where the file has been opened.
- The `[]` square brackets, as before, contain alphanumeric strings separated by spaces. The contents inside the brackets will be written to the file.

Code hooks in the codebase

You were provided with three function templates in `interpreter.c` called `openCommandEXAM(),` `readCommandEXAM()`, and `writeCommandEXAM().` You have also been provided with two empty files, `DISK_driver_problem1.c` (and .h). Use these three places to implement the `open_EXAM,` `read_EXAM,` and `write_EXAM` functionality. Feel free to expose other functions and data structures already existent in the codebase to your `DISK_driver_problem1` files. You should be able to complete your implementation by only modifying these three locations. However, if you absolutely need to make modifications elsewhere in the codebase, make a note in the README file, explaining why.

Implementation details

- The `open_EXAM` command is used to setup the data structures in the OS and locate the file in the partition.
- When a file is opened the pointer to the file is initialized to point to the beginning of the file.
- To make things easy, the `open_EXAM` command has an index number that refers to the cell of the active file table.

- The `read_EXAM` and `write_EXAM` commands will reference that index number to know which file to access. If the file is not opened, `read_EXAM` and `write_EXAM` will show the error message `“ERROR: Open the file first.”`
- The `read_EXAM` and `write_EXAM` commands **must not look for the file in the partition**, instead they use the data structures in the operating system that were initialized by `open_EXAM`.
- Once a cell has been used to open a file, that call cannot be used again. An error will be generated, `“ERROR: Index number in use.”`, if the script attempts to open the same cell number.
- Apart from these points, the `read_EXAM` and `write_EXAM` commands will behave in the same way as the given `read` and `write` commands.

Testing the code

Test your code with the following test file (also available in TESTFILE_PROB1.txt).

```
1. mount partition1 10 5
2. write_EXAM 0 [hello]
3. open_EXAM 0 file1
4. write_EXAM 0 [hello world test]
5. open_EXAM 0 file2
6. open_EXAM 1 file2
7. read_EXAM 0 var
8. print var
9. quit
```

The command on line 2 should display: `ERROR: Open the file first.`
The command on line 5 should display: `ERROR: Index number in use.`
The command on line 8 should display: `hello world test`

Notes:

- We will not test other corner cases. We will test your solution with the above test file and by looking at your source code to see if it was constructed correctly and whether it respects the expected programming style for this class.
- Your code will also be analyzed by MOSS for cheating.
- Your solution must run on mimi.

Marking breakdown:

- **1 point.** Using provided code hooks and naming
- **3 points.** Correct initialization OS data structures during `open_EXAM.`
- **2 points.** `Read_EXAM` and `write_EXAM` commands syntax correctly uses file index.
- **3 points.** `TESTFILE_PROB1.txt` runs to specification
- Your TA can remove **up to 3 points** from the total if the coding style is not respected.

Problem 2: Close Command [10 points]

Change the provided codebase to **add the close file command to your scripting language.** The syntax of your close command is:

`close_EXAM <file_ID_in_active_file_table>`

- The `file_ID_in_active_file_table` parameter is an integer that corresponds to the array cell index in the active file table where the user has previously opened the file.

Code hooks in the codebase

You were provided with a function template in `interpreter.c` called `closeCommandEXAM().` You have also been provided with two empty files, `DISK_driver_problem2.c` (and .h). Use these three places to implement the `close_EXAM` functionality. Feel free to expose other functions and data structures already existent in the codebase to your `DISK_driver_problem2` files. You should be able to complete your implementation by only modifying these three locations. However, if you absolutely need to make modifications elsewhere in the codebase, make a note in the README file, explaining why.

Implementation details

- The `close_EXAM` command will free memory of any buffers used during opening, reading, writing to a file.
- It will also make available the cell of the active file table data structure you used for Problem 1 to be useable for other opened files.
- The `open_EXAM` command, if used again, can reuse a closed cell.
- The `close_EXAM` command will return an error when it is asked to close a cell that is not in use: "`ERROR: Index number not in use.`"

Testing the code

Test your code with the following test file (also available in TESTFILE_PROB2.txt).

```
1. mount partition2 10 5
2. open_EXAM 0 file1
3. write_EXAM 0 [hello world test]
4. close_EXAM 0
5. close_EXAM 0
6. open_EXAM 0 file1
7. read_EXAM 0 var
8. close_EXAM 0
9. print var
10. quit
```

The command on line 5 should display: `ERROR: Index number not in use.`
The command on line 9 should display: `hello world test`

Notes:

- We will not test other corner cases. We will test your solution with the above test file and by looking at your source code to see if it was constructed correctly and whether it respects the expected programming style for this class.
- Your code will also be analyzed by MOSS for cheating.
- Your solution must run on mimi.

Marking breakdown:

- **2 points.** Correct reset of index.
- **2 points.** Correct freeing of buffers.
- **2 points.** Correct behavior when closing a location that was not open.
- **4 points.** `TESTFILE_PROB2.txt` runs to specification
- Your TA can remove **up to 3 points** from the total if the coding style is not respected.

Problem 3: Seek Command [10 points]

Change the provided codebase to **add the seek command to your scripting language.** The syntax of your seek command is:

`seek_EXAM <file_ID_in_active_file_table> <offset>`

- The `file_ID_in_active_file_table` parameter is an integer that corresponds to the array cell index in the active file table where the user has previously opened the file.
- The `offset` parameter is an integer that can be positive or negative. A positive value moves the file pointer towards the end of the file. A negative value moves the file pointer towards the beginning of the file. The offset is computed from the current position of the file pointer.

Code hooks in the codebase

You were provided with a function template in `interpreter.c` called `seekCommandEXAM().` You have also been provided with two empty files, `DISK_driver_problem3.c` (and .h). Use these three places to implement the `seek_EXAM` functionality. Feel free to expose other functions and data structures already existent in the codebase to your `DISK_driver_problem3` files. You should be able to complete your implementation by only modifying these three locations. However, if you absolutely need to make modifications elsewhere in the codebase, make a note in the README file, explaining why.

Implementation details

- The `seek_EXAM` command moves the file pointer by `offset`, starting from its current position.
- It uses the index number in the active file table to identify which file pointer is being updated.
- `seek_EXAM` does not move the file pointer outside of the file.
- If `seek_EXAM` is used to move beyond the file ending, the OS will stop the pointer at the end of the file. A message is displayed: "`ERROR: Out of bounds. Stopped at end of file.`"
- If the command is used to go before the file beginning, the OS will stop the pointer at the beginning of the file. "`ERROR: Out of bounds. Stopped at start of file.`"
- Note that the `read_EXAM` and `write_EXAM` commands will start from the position in the file where `seek_EXAM` moved the file pointer.

Testing the code

Test your code with the following test file (also available in TESTFILE_PROB3.txt).

```
1. mount partition3 10 5
2. open_EXAM 0 file1
3. write_EXAM 0 [hello world test]
4. close_EXAM 0
5. open_EXAM 0 file1
6. seek_EXAM 0 6
7. read_EXAM 0 var
8. close_EXAM 0
```

```
9.  print var
10. open_EXAM 0 file1
11. seek_EXAM 0 100
12. seek_EXAM 0 -100
13. quit
```

The command on line 9 should display: `world test`
The command on line 11 should display: `ERROR: Out of bounds. Stopped at end of file.`
The command on line 12 should display: `ERROR: Out of bounds. Stopped at start of file.`

Notes:

- We will not test other corner cases. We will test your solution with the above test file and by looking at your source code to see if it was constructed correctly and whether it respects the expected programming style for this class.
- Your code will also be analyzed by MOSS for cheating.
- Your solution must run on mimi.

Marking breakdown:

- **2 points.** Seek uses index correctly.
- **2 points.** File pointer updated correctly.
- **2 points.** File boundary respected.
- **4 points.** `TESTFILE_PROB3.txt` runs to specification
- Your TA can remove **up to 3 points** from the total if the coding style is not respected.