Project 1 for CS222P

Deadline: Week 3 Monday, January 20th, 2020, 11:45 PM on Github

Introduction

In project 1, you will implement a very simple paged file (PF) manager. It builds up the basic file system required for continuing with projects 2,3 and 4. The PF component provides classes and methods for managing files and pages in files. In addition, you will implement the first few operations for a record-based file (RBF) manager (which you will continue working on in project 2) on top of your basic paged file system. This document aims at providing you with the necessary information required to start project 1.

Goal

The goal of project 1 is threefold:

- Get (re)familiar with a C/C++ development environment.
- Implement a simple paged file system.
- Implement a few operations of a record-oriented (a.k.a. tuple-oriented) file system.

The detailed description of the project itself is in Project 1 Description.

Overview of Steps

- Set up the development environment.
- Download and deploy the codebase of Project 1.
- Finish the development of Project 1.

Detailed Instructions

1. Set up the development environment

- **Mac Setup Guide**
- **Windows Setup Guide**
  - We don't recommend you to develop directly under Windows because of potential compatibility issues.
  - A good alternative is to develop under a Linux Virtual Machine, here's guide to install Ubuntu on Windows VirtualBox
  - If you still want to use Windows, here's guide to install MinGW and use eclipse
- **Linux Setup Guide**
- **Use Linux environment to test your code**

You can develop on your own machine with CLion, Eclipse or your favorite IDE and other tools, BUT you should make sure you code will work on a Linux Machine (or a Linux virtual machine). The assistants will be using a Linux machine to test and grade the assignment. To this end, students should follow the standard C library and common C++ libraries, e.g., STL, to implement your project, and make sure your code is GCC/G++ compilable. You can also use C++11 features if you wish. Please make sure that you do not call any platform specific functions, such as Windows-specific APIs in your projects.

Before submitting your code, you **should** test it under a Linux environment. The code that can be compiled on your machine won't always compile and run correctly under Linux.

We will use GradeScope's machine Ubuntu 18.04 with gcc 5.5.0 to grade your code. Optionally, you can use ICS openlab hosts. You can use your **ICS account** to connect to openlab by using "ssh <your ics username>@openlab.ics.uci.edu". For Windows users, you can use Putty.

> You can also use Virtual Machines such as VMware Player , and VirtualBox . To install Linux on VMWare Player, visit http://partnerweb.vmware.com/GOSIG/Ubuntu_18_04_LTS.html . If you would like to use VirtualBox , visit http://www.wikihow.com/Install-Ubuntu-on-VirtualBox .

2. Download and deploy the codebase of Project 1

- Follow the **GitHub Guidance** to accept the Github Assignment, which will create the repo for you.
- Follow the **Codebase Depoly Guidance** to deploy project1 in CLion and run the test cases.

3. Finish the development of Project 1

We have seen the results of running the code in codebase. But, since the implementation of the methods is empty, you cannot manage any files yet. Please finish the implementation in pfm.cc as well as the following methods in rbfm.cc (besides the constructor and destructor): 1) insertRecord. 2) readRecord. 3) printRecord. The remaining methods are not required for part 1 of the project; instead you will implement them as part of part 2 of the project. Please write your own test cases to test your code. You are responsible for anticipating other things that might go wrong that we haven't provided public tests for, just as you would be if you were building a DBMS in an industrial setting.

You may find these functions useful: http://www.cplusplus.com/reference/clibrary/cstdio/

Submission Instructions

We'll take your Github code as submissions. You need to include all the code in your Github repository. You don't have to submit anything on EEE or Canvas.

On the night of the end of the grace period, we will run a script that gets the latest commit ID of the **master branch** in your repository. We'll use the time of the latest commit to determine if you use the Grace Period or not. So if you don't plan to use the Grace Period, don't push anything to the master branch of Github during the Grace Period. After the grace period, you can continue using your repository normally for the next project.

Remember that we will only be checking the code in the **master branch** of your repository. Do not try to cheat, e.g., by modifying the timestamp of a commit to an earlier date. We'll use the time that you push your commits to Github. So be sure to push your changes to Github before the deadline! Such cheating cases will be subject to very bad consequences. To avoid any surprises, we will be publishing the final commit number

that we have recorded for every team. It's your responsibility to check the commit number published by us. If you think we have made a mistake on the commit ID, you need to tell us ASAP.

The following are requirements on your submission. **Points will likely be deducted if they are not followed.**

- **Suppress any debug messages that you put in your code. Only the original messages in the test cases should be printed.**
- Write a short report (template provided as project1_report.txt) to briefly describe the implementation (key design choices) of your paged file and record-based file system layers. Please provide a **text file** rather than a PDF, Word Document, or other non-text format. Do not use RTF format.

- You must make sure your makefile runs properly!
- Please **DO NOT** change the repo directory structure, make sure it stays the same as when you accepted the assignment, with the following directory hierarchy:

    master/ {rbf, makefile.inc, README.md, project1_report.txt, CMakeLists.txt} where the rbf folder includes your source code and the makefile.

    (e.g., master/ {rbf, makefile.inc, README.md, project1_report.txt, CMakeLists.txt})

- **IMPORTANT: Again, you need to make sure that your script works under Linux, as the assistants will use Linux to grade your assignment. Also, suppress any debug messages that you put in your code. Only the original messages in the test cases should be printed.**
- **'Do not modify the makefile of your project.**' When testing, we will use our makefile instead of yours to include private test cases. If you change the makefile, then likely your project cannot compile and you will lose some points.

Testing

Please use the codes inside the codebase. Note that these test cases will be used to grade your project partially, but we also have our own private test cases. This is by no means an exhaustive test suite. You should add more cases to this and test your code thoroughly!

Note: Again, this is not the complete suite we will use to test your codebase -- just a sample of it.

Grading Rubrics

Please see the grading rubric here.

Q & A

- **Q1**: Consider a case where Page 3 of the file is full but Page 2 is partially filled and the user wants to append data? Now, if the size of the data that he or she wants to write is more than the available space on Page 2, what is the expected action to be taken? Do we just fit in whatever data we can and truncate the rest, OR completely disallow the user from making such a write?
  A: AppendPage() always happens to the end of the file, so this scenario can't arise. The number of file bytes affected by each page operation is always PAGE_SIZE. The paged file system layer always deals in pages -- nothing more and nothing less.

- **Q2**: Is it fine if I do the file handling in C++ using the binary mode of read/write?
  A: You should indeed use the binary mode!

- **Q3**: Why is the access specifier of the constructor and destructor of the class PagedFileManager set to be "protected"?
  A: The PagedFileManager is a singleton class, which means only ONE instance of PagedFileManager is allowed. You cannot instantiate the class by calling its constructor. Instead you should get an instance of the class by calling the Instance() function of PagedFileManager. The Instance() function has been implemented for you in pfm.cc. The same applies to the RecordBasedFileManager.

- **Q4**: As for pages, if I understand correctly, the Read/Write/AppendPage functions are operating on these files, and if you want to write the 3rd page (page number: 2) of a file, you'd seek 8K bytes into the file and start writing the data. Is this correct, or am I misunderstanding the concept of pages?
  A:
  - Read reads a page that has to already exist
  - Append adds a new page at the end of the file
  - Write overwrites a page that has to already exist
    To write to the 3rd page of a file, the file should already have at least 2 pages (page numbers: 0,1) that contain valid data. Then you can either append data to 3rd page if it doesn't exist, or overwrite the 3rd page if it already exists. Please do not leave "holes" in files by writing past EOF. We won't allow the case of appending garbage pages to happen.

- **Q5**: Since I need to change the path of codebase in makefile.inc to test the project, do I need to change it back when I submit the zip file?
  A: No, you don't need to change back, but you need to instead make sure the path is **relative** so that the test.sh script will also work on another machine.

- **Q6**: When inserting a tuple, do we only have to consider insertion of the new tuple at the end of the last page? Or do we have to be able to support insertion in whatever free space may exist among all the current pages?
  A: You should first try to insert the record on the last (currently existing) page. If that fails, you should then try to find the first page with sufficient space available (e.g., looking from the beginning of the file). If none exists, then (and only then) should you append a new page to hold the new tuple.

- **Q7**: What's the data format for data being passed to insertRecord?
  A: The API format for insertRecord is as follows: Suppose you have five fields and their types are varchar(20), integer, varchar(20), real, and string. If a record is ("Tom", 25, "UCIrvine", 3.1415, 100), then the format of the record should be: [1 byte for the null-indicators for the fields: bit 00000000] [4 bytes for the length 3] [3 bytes for the string "Tom"] [4 bytes for the integer value 25] [4 bytes for the length 8] [8 bytes for the string "UCIrvine"] [4 bytes for the float value 3.1415] [4 bytes for the integer value 100]. Note that integer and real type fields do not have an associated length value in front of them; this is because each of these types always occupies 4 bytes.

  > The first part of the input contains *n* bytes for passing the null information about each of the incoming record's fields. The value *n* can be calculated by using this formula: ceil(number of fields in a record / 8). For example, in this case, since there are 5 fields, the size of "n" can be calculated by ceil(5/8) = 1. If there are 20 fields, the size will be ceil(20/8) = 3. The left-most bit in the first byte corresponds to the first field. The right-most bit in the first byte corresponds to the eighth field. If there are more than eight fields, the left-most bit in the second byte corresponds to the ninth field and so on.
  >
  > If a field value is NULL, the corresponding bit in the null bit vector will be set to 1. For example, if we have a record ("Tom", 25, NULL, NULL, 100) whose third attribute and fourth attribute are NULL, the first part contains *00110000* as the bit pattern in one byte. The actual byte representation will be: [1 byte for the null-indicators for the fields: 00110000] [4 bytes for the length 3] [3 bytes for the string "Tom"] [4 bytes for the integer value 25] [4 bytes for the integer value 100]. Note that there are no values to represent NULL values in the actual data. You MUST follow this API format!

NOTE: This API data format is just intended for passing data into the insertRecord(). This does not mean that the internal representation of your record should be the same as this format -- in fact, it almost certainly will not be! (On-page record formatting options will be covered in lecture, and your project should make good choices for what it does based on what you learn in class.)

- **Q8**: Can we assume that a record can fit on a page (i.e., the size of a record < the predefined page size)?
  A: Yes. You can assume that a record can fit on a page.

Platform Q & A

- **Q1**: Which gcc version should I use?
  A: The reader(s) will use **GradeScope's AutoTesting** machines' **gcc 5.5.0** to grade your code. Please make sure your code compiles and runs properly on **GradeScope**. You can submit on GradeScope to test as many times as you want before the grace period ends, this is optional but highly recommended.

- **Q2**: Do I have to use a specific OS platform?
  A: You can implement your code on any operating system where g++ works, such as Windows or Linux. However, you cannot use any platform specific APIs. And once again, note that the assistants will use **GradeScope's Linux machine** to grade your code. Please make sure that your code works on Linux!

Last modified on 2020-1-7 上午10:02:28