Documentation of Project 1, CS222

Jianfeng Jia (UCINetID:jianfenj)

Overview

In this project I implement a simple page file(PF) component. This library manage a file list which are created by a file manager. And inside each file, we can manipulate data in page size. The core class of this project is PF_Manager which create, open, close and destroy files while maintain the file list information. The other class PF_FileHandle is straightforward. Once PF_Manager give it a FILE*, it read or write the data from or into that file in page unit. The other feature of PF_Manager is that we could store the filelist created by itself on the disk when it needed. So that we could recover from disk in case of system crashed.

Usage

There are some test in pftest.cc to test whether the system works fine. I just add some append feature into the exist pftest.cc, including appending, reading and writing data.
We can just goto pf/ directory and “make clean && make && ./pftest” and it’s should have some results shown up.

Code Hierarchy

pf/ contains PF* structures
pf/pf.h(.cc) implement of PF_Manager and PF_FileHandle
pf/pf_filenode.h(.cc) implement of PF_FileNode which contains some file properties
pf/pf_filelist.h(.cc) implement of PF_FileList which contains the created file list and store the PF_FileNode inside.
util/ contains some utilities that could be reused.
util/logger.h(.cc) simply a few logger static functions controlling the printing.
util/io.h(.cc) only fileexists function by now.
util/linked_list.h template class of linked_list.h

Design

PF_Manager

CreateFile and DestroyFile

PF_Manager contains a PF_FileList to store the files information which are created by CreateFile method. Each call to CreateFile method will create a non exists file while append a file information to that list, which is actually a linked-list (We will talk it more afterward). Each call to DestroyFile method will destroy the exist file which should be found in that linked-list. Then we delete that file physically, as well delete whose information store in the the filelist.

This filelist is so important that should not be lost in case of some system failure. So at the end of EVERY CreateFile and DestroyFile operation we write this filelist onto disk. The name of this log file is a const char* initialed ".storedfilelist.dat" at the beginning of the construction.

OpenFile and CloseFile

OpenFile and CloseFile is oriented with PF_FileHandle. The challenge is that when it comes to CloseFile, the only parameter is just one PF_FileHandle. Thus the PF_FileHandle must have a member to store that file. I opened the file inside the PF_Manager and transfer that FILE* descriptor to FileHandle.
The other require is that PF_Manager must to know the FILE* to CloseFile. And directly exposed the FILE* pointer is out of control. So I make the PF_Manager the friend class of PF_FileHandle to make sure no other function could touch this file.
The other thing of OpenFile is to check the filename is create by CreateFile. I first check whether the filename passed is inside of the filelist. If it not in, we know that this file is not create by ourselves. So I return -1 to refuse operating on it. Otherwise, we just keep on open.

RecoverfromLog

PF_FileList keeps track on the files that have been created. So if system need to recover from the failure, we could call this method to read from the exist log. The implement is rather simple, I just fread() from the file and reconstruct the whole PF_FileList node by node.

PF_FileList

PF_FileList is simply a linked-list. I wrote a very simple template linked-list in uitl lib and inherit all the method, that is AppendNode, DeleteNode, Contains check. Besides that I add the StoreInto() and ReadFrom() method to restore and to recover. Because it simply implemented as a linked-list, so the operation cost is O(n). Hashtable might be a good choice, but it consume more space and more complicated. I intend to add a search index on it to speed up the searching. However I think it depends on the need of the projects afterwards.

PF_FileNode

PF_FileNode is the node structure inside of PF_FileList. It distinguished by filename. And I add a open times property inside. Each time it opened by OpenFile in PF_FileManager, I will add the open times ++. And when it called by CloseFile I decrease the open time once. We could add more properties as the projects requirements grows.

PF_FileHandle

FileHandle here is very straightforward. I suppose that Read and Write page is the most frequent operation here. Thus I throw all the management things to PF_Manager. So the FileHandle is light and clean. It just fread and fwrite, each by a page.
One requirement is that PF_FileHandle should not be re-assigned. So I checked the FILE* to decide whether it's NULL or not.
The other thing that not clear is that whether we should reuse the handle after PF_Manager CloseFile(&FileHandle). I reserved the possibilities here when the Manager ask for the FILE*, I just give it and reset this handle by ::GetAndResetFile. Once someone ask for FILE*, it is probably out of control by Handle. So I reset FILE* to NULL to stop operate anymore.

TODO:

1. I haven't done the check for the correctness of existing log file.
2. I need to find an efficient index of filelist to speed up the search.
3. More test on FileManager.