Daniel Casper
12/1/2018

Project 3 Writeup

The purpose of Project 3 was to create a simulation of a file system using the object oriented paradigm. The file system needed to support three allocation methods: contiguous, indexed, and chained. In the contiguous method, blocks for files would be allocated in sequential order (from freely available blocks). Here, only the first and last block needed to be stored in the file table. The indexed method would randomly select blocks to allocate for the file, storing the order of the allocated blocks in the file table. The chained method, however, required that each block contain a point to the next block in the series, where only the initial block would be contained within the file table.

My initial design strategy was to first write the class for the contiguous method, then pull methods out into an abstract file class. After doing so, I planned to use inheritance for the indexed and chained methods, utilizing polymorphism in the main routine and implementing the virtual functions in the respective child classes. The file table would have fixed length entries (a divisor of 512 large enough to contain all file metadata), so as to facilitate deletion from, searching in, and updating the file table. In order to prevent from reading bytes in a block past the length of the original file, the last byte of the file was also maintained in the file table. The free space bitmap was simply to utilize the first 256 bytes of the second block in the file system. Each respective byte would be updated when a file was written or deleted. My original design strategy also assumed that C-style arrays would be easier to work with, and this proved to be somewhat true for the contiguous strategy (since only the starting and ending block needed to be known).

This project proved to be very difficult and time consuming. Had I better anticipated the difficulty of randomly selecting blocks using simple C-style arrays, I would have chosen to use the C++ vector data structure in the chained allocation method. Instead, I adopted the use of vector only when programming the indexed method, forcing me to often use function overloading instead of virtual functions. The vector data structured proved to be easy to work with, and resulted in cleaner code than my attempts to manipulate C-style arrays (in particular, the use of iterables). Had I used the vector data structure initially, I could have better reused the delete file, read, and write back file functions from the original File Structure class. In hindsight, I should have defined the base class first, then proceeded to make its subclasses. This would have saved me hours of development and created a much cleaner structure for the classes.

The end result was that I was able to fully implement the contiguous and indexed strategy (since all information was contained within the file table and consumed in a similar manner). Due to scheduling constraints, however, I was unable to fully implement the chained allocation method. Perhaps if I had more efficiently designed the file system base class and used vectors in the contiguous method, I could have created sufficient time to implement the chained allocation method. Such was not the case.

Through this project, I learned just how difficult it is to maintain a file system. Without the use of data structures, the complexity of managing such a system is near impossible. Further, it was a refreshing revisit to the world object oriented design.