Jacob Hessong

MP7 Design Document

Option 1 – Design of thread safe file system

Changes: Format function was changed from being static to avoid complications with accessing variables, which works since there is only one file system used. The kernel file also had to be updated to reflect this, as it contained a static call to the Format function.

In this machine problem, the objective was to implement a basic file system. The design was divided into two primary sections. These two sections were a class for the file system as a whole and one for files. The file system class allows the file class to have the basic structure and functions necessary to work properly.

Starting with the file system, an inode structure was first created which helped with managing the files on the disk. These nodes hold important information corresponding to the file it points to. Because the files are named by their id numbers, this is also in the structure. Further, it maintains an array of all disk blocks that the file uses as well as the size of the file. This helps with determining the end of file location later on. The private variables in this class are a pointer to a simple disk object (simple disk is sufficient for this implementation), a bitmap of free blocks with info for 8 blocks at each index, and the total number of blocks, management blocks, and inodes that are used. The inodes stores the file information and are written on the disk. When the information has to be updated, the inodes are scanned to find the inode with the corresponding id. The information is updated and then the inode is written back onto the disk. The mount function updates the disk pointer to the disk passed to the function. The constructor simply initializes all variables, setting the disk to NULL since it will be mounted later.

Format updates the disk and size of the file system that will be installed on the disk. The local management information is then updated to be used. The bitmap is set to be used for free file allocation where a certain number of blocks are reserved for inodes. The block map is set as 1 for each allocated map in the bitmap that is used for inodes. Then, all blocks on the disk are set to 0 and that is iteratively written on all of the blocks. The lookup function initializes file objects rather than when create file is called. Create file first searches the disk to see if a file with the passed id already exists, and if not, an inode is created for the file and is written to the disk. This allows the information to be retrieved easier when the lookup function is called. It returns a file pointer that has all of its information already initialized, such as the size and block information. The delete file function simply frees the allocated memory for the file and clears the inode data. Lastly, there are some helper functions defined to make the implementation of the require functions simpler. These help with allocating, deleting, and updating the size and data of blocks as well as erasing the content of a block.

The file class declares FileSystem as a friend class, allowing access to private variables of the FileSystem class. The blocks used in this file are also cached in the file class so that the inode does not need to be read again unless more blocks have to be added, saving costly reads. An index and position variable are used to help with read and write operations, with index indicating block number and position indicating the byte index in a block. The file constructor sets all of the private variables of the file. These are all initialized when the file lookup function is called. In the read function, a 512 byte buffer is created and a read operation fills this with information from the current position in the file. If the position exceeds 512, the function moves to the next block before resetting the position to 0. EoF is also checked for each read to

make sure the read does not exceed the end of the file. Write is implemented similarly, but writes to the disk instead and updates the block data and size in the inode when finished. Reset simply resets the pointer to the start of the file instead of wherever it previously was. Rewrite is used to erase the content of a file. This is done by erasing the content of all blocks and freeing them except for the first block (which is still erased, just not freed). This makes sure that the file is still kept with its updated inode information. Finally, the EoF function checks if the current position of the file has not gone past the end of the file by checking the index and position against the size variable.

Design of thread safe file system:

A thread safe file system would need to allow simultaneous read and write operations. This causes a change to the file system design. The file system now needs a lock on the disk resource so that only one thread can execute the actual read/write operation. With a multiprocessor system, the locks implementation has to prevent starvation for other threads, like when the thread dies and keeps the lock. This also has to be implemented in a critical section to control access. When several read or write operations are requested, a cache can be implemented so that the write process can be sped up.