**CSCE 611**

**Gautham Srinivasan (UIN: 927008557)**

**MP7 Design Document**

**Objective:**

The objective of machine problem 7 is to implement a vanilla file system where the files support sequential access only and design of thread safe system **[Option 1]**.

**Design:**

The file system maintains the metadata of the files and it is responsible for creating and deleting the files. I have used block 0 in the disk to maintain the metadata of the files. The metadata are file descriptor, current position, start block and total blocks. Block 1 contains the list of free blocks which uses bitmap to look for the free blocks. The files can be accessed by creating an object of File class which is used to read and write data to the disk. The File object requests a free block from file system when the dedicated file blocks are exhausted.

**Detailed Implementation File System:**

As said above, the file system maintains the metadata of the files like for example, how many blocks the file contains, in the 0th block and list of free blocks maintained as a bitmap in 1st block. Whenever it creates the File object, by default a free block is assigned.

1. **FileSystem() –** Does nothing much apart from resetting the buffer of 512 bytes.
2. **Mount() -** Assigns disk pointer which is later used to access the disk
3. **Format() –** Writes zero to all the blocks except the 0th and 1st block. We mark these blocks as used in the bitmap of 1st block.
4. **LookupFile() –** Reads the 0th block and checks whether the file descriptor is present. If present, it creates and returns a File() object, else return NULL.
5. **CreateFile() –** Assigns a free block to a file and mark the block as not free.
6. **DeleteFile() –** Resets the blocks assigned to the file.
7. **GetBlocks() –** Gets contiguous free blocks when requested. It checks whether the contiguous blocks are free and returns a start block. It also updates the metadata in 0th and 1st block.
8. **FreeBlocks() –** Mark the contiguous blocks as free which can be used for other files.

**Detailed Implementation Files:**

A file is data which is stored in one or more blocks in the disk. Here, I have maintained contiguous blocks which are used to read and write. The file class maintains the read/write pointer position, read/write blocks. I have used separate variables for read and write to make it simple..

1. **File() –** Initialized its own data variables. The start block and total blocks which are given by the file system is being maintained.
2. **Read() –** Reads the number of bytes requested from the disk and puts them into the buffer. If the read pointer reaches end of the block, it reads from the next block.
3. **Write() –** Write the number of bytes requested to the disk. During write, if we require another block, we request total blocks + 1 and move the content of previous blocks to the new blocks and finally write to the last additional block.
4. **Reset() –** Resets the current pointer to 0
5. **Rewrite() -**  Writes 0 to all the blocks in the file. Frees the existing blocks and gets one block by default.
6. **EOF() –** Returns true if the read pointer reaches the buffer size

**Design of thread safe system [Option 1]:**

As multiple threads can access the disk, we can protect the disk when the user calls read and write API. As we cannot use POSIX mutex or semaphore, I have implemented test and set function which protect the disk when read and write API is called by the user. This function can be made atomic by masking the interrupts. If the mutex is already taken by a thread and another thread tries to lock, then we do not busy wait and make the thread yield the CPU.