1) Yes. Our implementation has external fragmentation. External fragmentation is when a single file is potentially allocated using non-adjacent sectors in disk. If we create and write 1 byte to 2 files then write 2k bytes to the first file, the first file will allocate sectors 1, 3, 4, 5, and 6. This demonstrates external fragmentation as the file is not allocated using solely adjacent/sequential sectors.

2) Internal fragmentation is the result of a sector being allocated for a file, but the file not using the entirety of that sector. The average number of bytes wasted due to internal fragmentation for a randomly-sized file is bytes. Therefore, the expected wastage for 10 files is 2,540 bytes.

3) Roughly the same as the read/write latency is the biggest issue, and we are not writing large enough blocks of data at a time to result in a noticeable bandwidth change.

4) We can store 31 files in our disk (the free space manager is treated as a hidden file). We can increase this limit by having an expandable directory. The last “file” in the directory could indicate where the next sector of the directory is located, and we could load that sector to continue reading the directory.

5) Yes, we do. If a file is open, any thread can write to it. The write function itself does not place a semaphore on the writing, so therefore the functions writing data must use semaphores to handle the write synchronization. This is because the write function only writes 1 byte. We believed that we would often require writing more than one byte at a time to the file, so we wished to place the semaphores around the redirected printf function to ensure that the data we wished to write was written correctly.