Project 3 Report:

Team: C-Food
Team Members: Shamari Feaster

Timeline:

Note: My timeline chronicles my descent into madness as I attempt to save my semester by completing what has proved to be the most arduous thing I have ever had to code. Each day on the timeline lists the things I worked on as well the many issues I had throughout the development. After the timeline is a note that sums up my experience with the project.

List of Issues:

- Scope blow up (extreme scope creep)
- Unknown requirements
- Unfamiliarity with the subject matter
- Still getting used to writing in C
- Sheer size of the code base
- Managing the code base

See the timeline for a more detailed discussion of the issues and how I dealt with them

11-14-12
I started reading the Microsoft specs and start coding the basic boot sector paring algos outlined in there.

11-15-12

Still learning about the system, I haven't written much code yet as I don't really understand how the file system works. I created my boot sector structure and have successfully loaded the boot sector info in there

11-16-12
Started to define my constants and structures. Now that I have the boot sector info I feel a little more confident to start messing around. I'm going to use the same concept of a packed structure to load up directory entries.

11-17-12
After successfully pulling directory entries from the image I'm ready to tackle the FAT. I created some basic FAT entry retrieval functions. I'm now realizing how little experience I have with bitwise programming. Although the task are relatively simple getting use to the idiosyncrasies of this type of programming is making the process pretty slow. After that I worked on some FAT entry writing functions. The true scope of the project is starting to become apparent to me as I now understand how the system functions. This is going to take a while.

11-18-12
After about 8 hours writing FAT entry getters/setters I'm ready to start trying to walk some cluster chains. My first forray into this area is a function that walks the FAT sequentially calculating how many clusters are free. After this I verify that I'm getting correct information by comparing the results with the known size of the disk and they seem to be matching up although ont perfectly. I chalk this up to the fact that I don't know how my OS calculates image size.

11-19-12
After another 8-10 hour session yesterday I feel confident that I can manipulate the FAT correctly. I've started to contemplate the filesystem functions like ls, rm, ect and I realize that I'm going to need some sort of environment to keep track of things like my pwd, pwd cluster, what file are open, what directories I've been to.....the list goes on. I resolved that a environment struct will do the trick for the terminal state but for my files and directories I need another abstraction. For this I created a FILEDESCRIPTOR struct. This is what will populate my open file and directory tables. At this point the code is starting to get unruly, mostly due to the fact that it's hard to properly design a program for which you have no idea of the requirements when you start. This was a long day and I'm honestly getting fatigued from multiple long days but failure is not an option so I soldier on. I decided to go with a circular array for the file tables. My thinking is that by the time it wraps around you don't have use for the oldest files in the table. Once again I can't claim this is the best design but it's what I'm going with

11-20-12
Today was the day for verifying that my file table solution would work and I hit some snags as far as properly allocating the array and making sure it wrapped around correctly. Most of my morning was spent on this. After I got it working I created function to operate on the table and hit a few more snags there but nothing major. A few more functions to create and operate on FILEDESCRIPTOR's and I feel like it's about time to start on the file system function. I figure the easiest ones are going to be ls and cd and they also give me a chance to see how my directory history tables function in the real world. This would prove to take a considerable amount of time as the true nature of file system navigate begins to become apparent. Things you have to keep track of and update, things that have to remain constant, special cases like the fact that the root directory has no '.' or '..' changing how you traverse it. This isn't scope creep, this is scope blowup! That being said I'm not hitting roadblock so my spirits aren't too bad and slowly but surely it's coming together. By the end of the night I've got a working version of ls and cd.

11-21-12
I figure the next best thing to work on is touch. It seems simple enough, just create a directory entry with the file name and pretty much everything else set to 0. Like everything else in this project this necessitated numerous functions and many hours of thought on the best way to accomplish this in a way that wouldn't cause problems in the future, a recurring theme throughout the entire development process. So I wrote functions that create file entries, create special directory file entries, since I figured I'd be doing that a lot. I realized I also need function that walk the pwd and figure out where to place the new entry - not only that if the current

cluster is full I needed to be able to expand the cluster chain and place the new entry in the proper place. Needless to say this is going to take a while

11-22-12
Starting touch has opened my eyes to a whole host of new functionalities that need to be built as well as retooling of old ones. Now I'm really into the design part of the project. I'm caught between trying to keep the code readable , as it has ballooned to well over 1000 lines, and simple. Both have their downsides. If I decompose each function too much it makes problems hard to hunt down because control is jumping all over the place. On the other hand if I decompose too little I risk losing readability.

11-23-12
I decided that I would approach decomposition like this: if the functionality will be repeated then it belongs in a function, however if it is very specific to the function it resides in then it will stay. This is not a perfect solution and truthfully I wonder if my some of my function aren't too meaty but I had to make a choice and this is the one I made. I did end up multi purposing ls() to perform cd and search functionality. I know this breaks with the kernel convention but this It was easier to track down and fix directory traversal errors if they were in the same place and since all three functionalities had involved traversal I grouped them. That being said there we other instances that needed traversal such as finding open directory entry spots that I broke out of ls because it had gotten a little beefy in the parameter department.

11-24-12
I'm feeling really good about my toolset and have finished and tested touch and mkdir, both of which work and do not corrupt my image. Now I decide to get on to the tedious business of error handling what I have. I decide to go with error codes for each function. I did this for 2 reason: each functionality could have multiple things go wrong with it, making simple boolean returns not an option and because I wanted to keep my driver function as clean as I could. I noticed the driver was getting way too dirty with things that really didn't have much business being in a driver so I've started to move them into the function and given the function a way to signal if/ when things went wrong. I also moved on to fopen, and fclose. This is really where I test my file tables.

11-25-12
After another 10-12 hour day I've successfully written and tested fopen, fclose, touch, mkdir. rmdir and rm proved to be a lot more complex than I thought they were going to be. After finishing those I feel like I'm in the home stretch. All I'm left with is what I always suspected would be the real monsters of the project fwrite and fread. fwrite is definitely taking my math and programming skills to task. I spent most of the day planning and writing the code and running through it in my head, too afraid to test it until I was satisfied with what I was seeing on the screen. After 8 or so hours I felt good enough about what I was seeing to compile it, clean out all the inevitable errors that come with writing 8 hours of code without a test and give it w whirl. Surprisingly it's not a total disaster. There were bugs, but because of my now intimate familiarity

with my solution they were pretty easy to iron out.

11-26-12
I figure, correctly, that I can pretty much reuse my fwrite code to do fread and thing go pretty smooth. At this point I have finished all of the required functionality. I am now totally obsessed with this project, I've been coming up with solutions in my dreams on multiple nights and I have a pretty good idea how to tackle the extra credit parts. I work late into the night on path resolving but I run out of steam on resolving special directories in a path. I simply don't have anything left and I still have to break the what is now 2600 line behemoth into manageable and logical file as well as give everything a final, thorough test before the deadline.

11-27-12
I broke everything down into logical components, created my readme and now it testing, testing, testing. We've been given an extra day but this project has taken a real toll on my life and I just want it over with at this point. After at least 40 hours at the keyboard I'm spent.

In Conclusion

This project taxed me, mentally and physically, in way that no other program I have written ever has. I would call it one of my crowning achievements as a coder if it wasn't for the fact that what I made is ultimately pretty useless.

What I really learned was how important knowing what your program needs ahead of time is to creating a well structured program. I guess you won't always know what you program needs, especially if what you're programming is new to you, but it's hard to know if you are going about something the right way when you don't have any idea what is going to be needed down the line. You THINK you know what is needed in the future but the minute you put finger to key the reality of what needs to be done to complete your task grows like a weed. This scope "blow up" was the most frustrating part.

What would I change? A lot. Honestly, I don't feel like we were not prepared for what this program needed to be. There were a TON of design decisions that were hard to make because I felt like I didn't understand what I was building most of the time. Granted my experience is unique because of my lack of a partner and the fact that I will fail the class and not graduate on time with anything less than an 80%. So the pressure on me to grind this out has certainly skewed how I view the experience. That being said, I now have an appreciation for how complex and tedious system level programming is. All in all, I'd be lying if I said I wasn't glad for this semester to come to an end but I have come out of this a better programmer and I guess that's the point, right?