Trevor Hodde

CS502 Operating Systems

Z502 OS Design Doc

1. **Functionality and Justification of High Level Design**

Since the first two tests in project 2 involve page faults, the design of the page table system up front was important. Each process was given a pointer to a page table variable. This is so that each process could have access to the data it was working with when a page fault occurs. Page faults occur because the test program tries to directly access the hardware through MEM\_READ and MEM\_WRITE, then the operating system has to handle the fault. The first time a page fault occurs, the page table is allocated and a list of available frames is created. From then on, the frame list is checked each time a page fault occurs and free frames are passed back to the process for use.

The next few tests deal with Disk IO, which is handled by the operating system instead of the hardware, unlike the previous tests. First, the test program initiates a disk request (which is either a read or a write). Then, the OS reviews the status of the disk to see what state it is currently in. If the disk is busy, the program sleeps for 20 clock cycles and this check is repeated until the disk is available. Once the disk is available, the read/write request is made to the hardware and the program is suspended. Once the hardware returns from the disk operation, the process is reactivated. A queue of processes waiting to for read/write operations is maintained as well.

My choice of page replacement algorithm goes as follows: as pages are requested, they are allocated and given to a process for use, however there is a point at which there are no more available frames to write to. At this point, a flag is set which tells the fault\_handler that there are no more frames available and that one needs to be stolen. Then, we need to loop through all the frames and find one that is not in use. This is called the “victim” frame (in my algorithm at least) which is copied to the Shadow Table so we know that it is backed up and can be paged into physical memory later. The victim frame is then written to memory, cleared out and returned to the fault\_handler as a fresh frame. This is how frames are recycled for later use.

1. **Current Test Status**

Currently**, tests 2a through 2d** are fully functional (sometimes 2d has an occasional hiccup, though the output is typically accurate). Then, **tests 2e and 2f** are extremely close to being complete. They both execute to completion, however there are some errors with reading back the data. The rest of the tests are not at a completely stable state, however not much more work would be required to get them finished. Much of my time was spent up front designing the page table functionality and the disk reading and writing functionality, so I am confident that the design is robust enough that completing the final few tests would not be very difficult.

I spent a large amount of time on design so that I would have a plan for moving forward and for handling future tests. For example, I spent a lot of time structuring the page table design so that it would be robust enough to handle difficult tests like test 2e and f. Additionally, disk IO is a very important task, which many of the tests depend on, so I spent a lot of time on these functions making sure various cases worked. This is actually where most of my time went because I found a few bugs in the Z502 code that gave me some very inconsistent output. Once I tracked these bugs down (which took several hours), I had to work around it. I realized that some of these problems were also specific to Mac users, which also caused a problem because I had come so far in this project and I could not simply modify a small amount of code to compile on a regular linux system. However in the end, my disk IO was fairly successful and it was worthwhile even though it took longer than I expected.

1. **Uniqueness of Design**

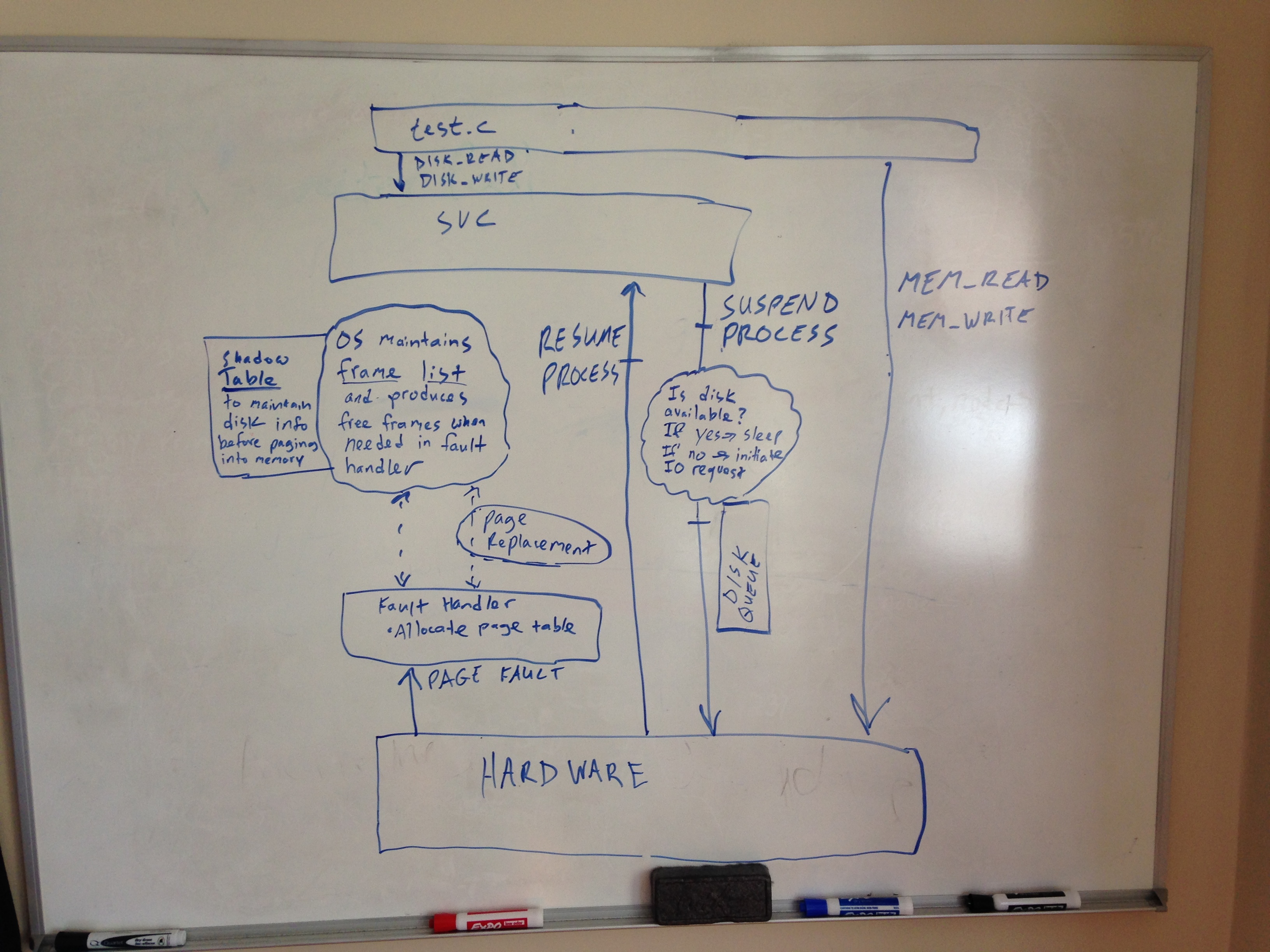
I think I can safely say that I am probably one of the only people to build this project on a Mac. I did not see many comments from students in class about Mac issues, and everyone I spoke to mentioned that they were building the project either on Windows or Ubuntu. This caused me to have to adjust a few small things that involved pthreads, though it really was not anything special.

Probably the most unique thing that I did in this project (though I cannot be completely sure because I haven’t seen other students’ code…) was included a DISK object and a page table object directly on a PCB. This was so that each process could have access to data that it was handling directly. It also allowed me to know whether or not a process was reading or writing data at any time. This was useful for handing interrupts so that data was never lost if an interrupt was thrown during a write operation. I could always access the data directly from the current process.

Another unique aspect of my design is the use of a shadow table. I do not think many students would have opted to take this approach. I used this in my design because I needed a better way to write data when doing page replacement. There were times when data was being lost because I was not sure how to find virtual memory without it being in physical memory already. Though this part of the project does not work fully, it was still helpful to me and I learned a lot more in the process of working on it. It seems to me that many students would have stuck with only their page table design to avoid the extra pain of debugging a shadow table in addition to page tables.

With regard to “special features,” I really did not have time to implement anything extra besides the shadow table, though I really wanted to. I was actually hoping to have time to complete both test2g and test2h as well because the concept of shared memory is interesting to me, but if I had another week, I probably could have gotten to it. However, the most important thing to me was getting the basic functionality working well enough to support later tests, so I thought it was best to work harder on getting the first few tests running well.

1. **High Level Design**

**** See picture below