Trevor Hodde

CS502 Operating Systems

Z502 OS Design Doc

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

When execution of the OS begins, a root process is created. This is the process in charge of spawning other processes, and it allows for contexts to be switched or deleted without leaving the program hanging. This process is assigned a pid of 1 so that the OS always knows that the root process has a dedicated value and that it should not be removed. The root process is given to the dispatcher when it is created and from there, all processes are controlled, killed, started, stopped, etc.

Processes are maintained in 3 different Linked Lists. A ready queue, a timer queue, and a general process queue. The process queue holds references to all existing processes when they are alive. The other two queues are managed by the dispatcher and the scheduler. When a process state is set to SLEEPING, it is added to the timer queue for some amount of time (referred to by its *delay* field). Conversely, if a process state is set to READY, it is added to the ready queue. The list file is also responsible for ordering processes based on priority.

The svc function is really where all the work is done. Each system call is checked and handled in this function and it is the main controller for the whole program. This is where system variables are manipulated throughout execution of the test program. The svc function passes required state information for processes and errors either around to other functions in the base class or (eventually) back up to the test class depending on the result of particular operations.

This seems like a good way to set up the OS because it allows a front function (svc) to really handle all of the communication up and down between the tests and the OS. It also handles the setup of the root process separate from the test functions. The root process then works with the dispatcher and the interrupt\_handler to manage the other processes (switching contexts, process creation, process removal, etc.).

1. **Current Test Status**

Currently**, tests 0 through 1i** and **1k** are fully functional. Test 1j is not working at this point, however there is not much more work to do to get it where it needs to be. I still need to implement some of the Message handling logic to get test1j finished.

I spent a good amount of time up front on design so that I would have a plan for moving forward and for handling future cases. For example, I spent a lot of time structuring the dispatcher to handle cases that will be useful for later tests. Additionally, context switching is a very important task, which many of the tests depend on, so I spent a lot of time on this function making sure various cases worked. Finally, a big time sinker was building the linked list. There were a large amount of errors that were discovered as I went along through the tests. I added a recursive function for removing child processes based on the parent pid. This was fairly time consuming as well, however it proved to be very beneficial.

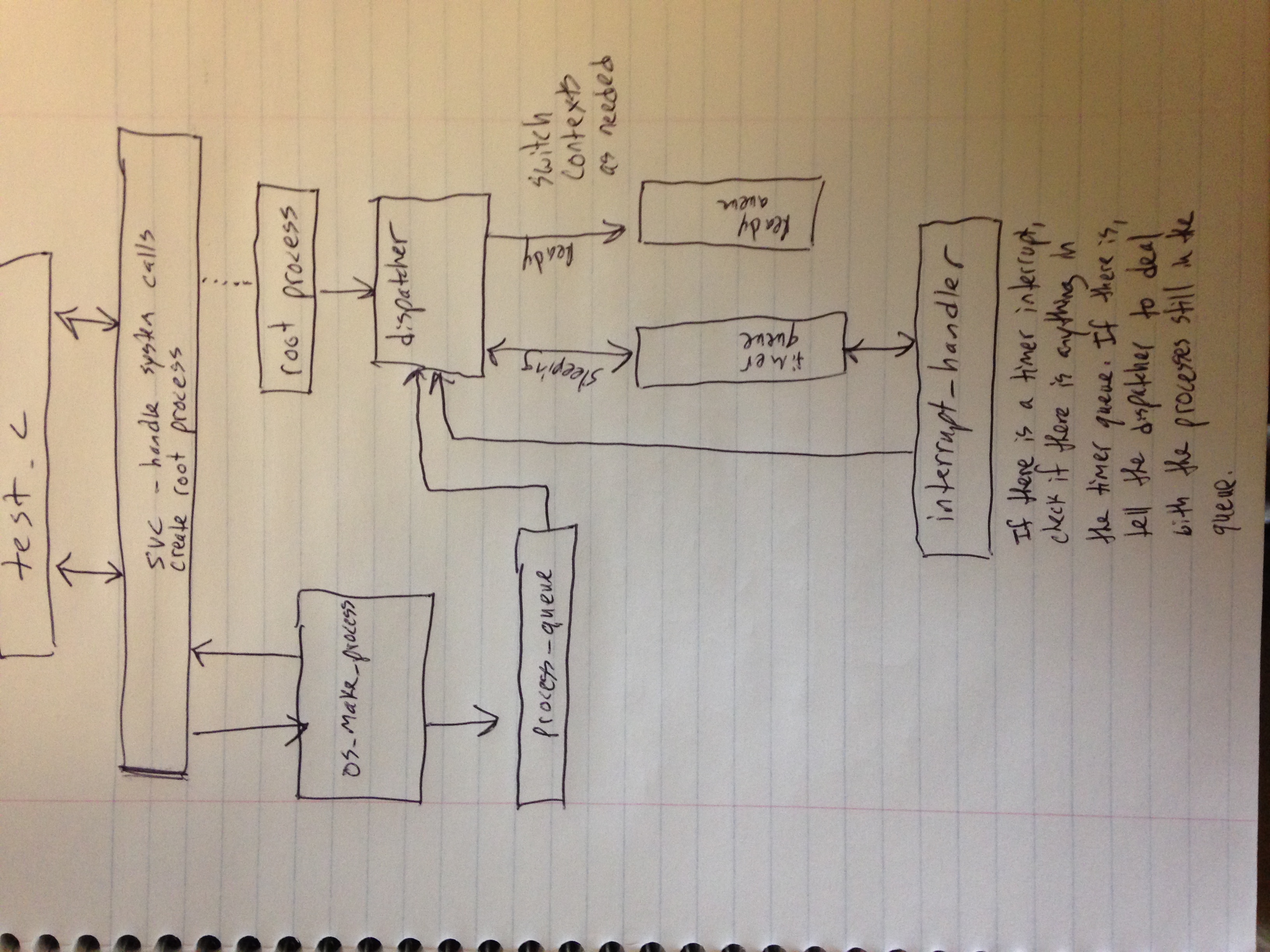
1. **Plan for Moving Forward**

Though there was a lot of overhead in time spent up front on design and implementation of various functions, it was worth the effort. I have moved forward quickly through the last few tests and it seems like my attention to future detail is paying off. I estimate that I will have test1 finished in its entirety by Saturday, October 19. This will give me a few more evenings of coding, which will really help, and it will leave me in good shape for handling test2.

Also, since there are really no “special features” built into the OS yet, I would like to spend some time coming up with something interesting for my OS to do. I spent more time on initial design than on special feature planning so far, however I expect to be on schedule throughout the continuation of the project, so I would like to explore the boundaries of my OS and make some interesting changes eventually.

1. **High Level Design**

See picture below

****