100517323 – Systems Programming – Assignment 2

Implementation Log

|  |  |  |
| --- | --- | --- |
| **Date** | **Time Spent** | **Description** |
| 14th November 2021 | 5 hours | I started implementing the system call for set pixel. The system call takes the parameters in, but I cannot seem to put the hex value 0x0F or number 15 into the memory location. The program keeps throwing errors. First I tried to use pointer to the memory location 0xa0000 then using an offset variable by doing the following:  ‘memoryVar’[offset] = 0x0F;  This kept throwing errors. I then tried to use memcpy to copy the value into the address but couldn’t seem to figure out how to get it working. Decided to give up after spending so long on it and not getting anywhere. Will give it another try when my mind in clearer. |
| 16th November 2021 | 1 hour 30 mins | The problem I has last time was because I was not mapping physical memory to virtual. We was told in the lecture to look in console.c to find the solution to do this. I was missing P2V(0xa0000) when pointing to that address.  After finding the solution I was able to complete setpixel and then I completed the clear system call also. |
| 18th November 2021 | 1 hour 30 mins | I moved onto moveto and completed this relatively easily. I added x and y variables outside of the system calls in the graphics.c class to complete this so the system calls share them instead of creating new instances. I made another system call called print to test out moveto and it all worked fine.  I moved onto lineto but only initialized it because I had forgot to put sys\_ in front of it and couldn’t find the error. |
| 21st November 2021 | 5 hours | I started and completed the drawline function. I used Bresenhams line drawing algorithm to implement this system call. I used <https://medium.com/geekculture/bresenhams-line-drawing-algorithm-2e0e953901b3> to help understand how it works. |
| 25th November 2021 | 2 hours | I implemented the first two methods of the second stage, setcolour and select pen. I moved onto the fill rect method but struggle to understand how the pointer to the rectangle should be handled, I will come back to it next time. |
| 28th November 2021 | 2 hours | I finished the fillrect system call. I didn’t quite understand how to pass and retrieve the rect structure so I looked in user.h to see if any other system call used a structure. A structure called stat is used in one so I tried to look to see how that was implemented but I couldn’t find it in the code so I searched it on google. I found this website:  <https://stackoverflow.com/questions/53383938/pass-struct-to-xv6-system-call>  I used this to help me implement fillrect. I then create a new c file called stage to demonstrate the system calls implemented in stage 2. |
| 30th November 2021 | 2 hours | I went through all of my code which I had implemented to make sure I had got it all correct. |
| 7th December 2021 | 2 hours | I started working on task 3. The first thing I did was read file.c as there was a similar implementation of a locking mechanism in there. Once I understood how the lock in file.c worked I moved onto creating a structure for a window. I created the structure in a file called window.h. The window structure will hold all the parameters needed for program to draw to the screen without being interrupted by a different program. I put 4 int varibales in it: x, y, index and available. Index is the selected pen index and available will be set to 0 if the window is free and 1 if its already taken.  Next I created a structure in graphics.c called windows which contains a spinlock structure and an array of window structures of size 20.  I had global variables in graphics.c for x, y, index and hdc. I deleted these as the window structure will now store x, y and index. And I declared hdc in each separate function. After deleting x, y and index I had quite a few errors as they were used a lot. I had to replace them with the window using the hdc value as an index into the array of windows.  My code changed from x = “value” to windows.window[hdc].x = “value” |
| 9th December 2021 | 2 hours | I started implementing the beginpaint and endpaint system calls. The begin paint system call is almost the same as the filealloc function in file.c other than its for a window instead of a file. So that was quite easy to implement. Begin paint just finds the first window which its available parameter is set to 0, sets its to 1 and then returns the index of that window for use as a hdc in userspace.  The endpaint function was easy as it just takes a hdc as a parameter and then uses it as an index into the window array to set the available value back to 0 and then returns.  In file.c there was a function initializing the lock for that class, but I didn’t quite understand what it was doing or how it worked so I left this until I could get some guidance in class. |
| 10th December 2021 | 2 hours | I went to the class and got some guidance on initializing the lock. After getting a better understanding of what the function in file.c was doing I added a function called initGraphicsLock in graphics.c which did the same as fileinit in file.c other than it used the graphics lock. I also added the definition in defs.h so it can be used in main.c. I also called the function from main.c in its main function so when the OS starts it automatically initializes the graphics lock.  After initializing the graphics lock I starting using acquire and release on the system calls I had implemented already. I put acquire(&windows.lock) just after initializing the varibales of system calls and release(&windows.lock) just before the return call. The reason I put the acquire lock after initializing the variables is because I wanted the system call to return -1 and end if any variable failed to load as acquiring and releasing the lock every time this happened would have been a waste of cpu time. (I updated my locks on the 16th of December and explained why in the implementation log for that day.)  After putting all of the locks in place I created a new user program to test to see if it all worked.  The user program is called stage3 and it forks a child process which draws 10 red squares in a diagonal line. The parent process draws 10 yellow squares in a diagonal line but starts where the child process finishes.  At first my program kept freezing. I realized it was to do with my beginpaint system call. In the loop to find an available window I was returning before my call to release the lock, so the lock stayed held.  After fixing the problem above I was getting another problem which was called printing zombie to the console. After some searching I realized I was because I was not putting wait() after the parent process in user space. After adding wait() the program ran as I wanted it to, displaying 20 squares, 10 red and 10 yellow. |
| 14th December 2021 | 2 hours | Looked back over my code from the first three stages to try and make some bits cleaner. My user program for stage3 was the main piece I worked on. Before my code had a fork and a loop inside the child and parent fork. Now stage three has a fork, but inside the child it calls a program called stage4 using exec. |
| 16th December 2021 | 3 hours | I tried to start implementing stage 4 but struggle to understand how to start it so I went back to my locks to see if they could be improved.  I changed my locks so they get acquired only when shared resources could be being used. Most of the locks are now acquired when a system call is going set a pixel on the screen and released after. This will mean two system calls cannot write the same pixel at the same time.  I also put locks in the setpencolour function around the code where outb is used. This is so two processes cannot output bytes to the same ports at the same time.  I kept the locks in beginpaint and endpaint because they are using shared resources when checking if a window is available or when making a window available.  Finally, I removed my locks from moveto and selectpen because they are not using any shared resources. They are using hdc as an index into the window array. Only one process can have any hdc at a time so no other process will be able to access the indexed window.  After dealing with my locks, I added some setpixel calls and fill rect calls to stage 3. I also added another fork. This allows me to showcase all my work in a single user program. |

My test file name is called ‘stage3’. This demonstrates all system calls up and including stage 3’s.