*Systems Programming Assignment 2 Development Diary*

| Date of Session | Time Spent | Development Segment | Notes |
| --- | --- | --- | --- |
| 10/11/2021 | 1 Hour 20 Minutes | Setup | Set up a private git repository for the assignment, using the cloned data from wrippin/xv6-assessment. Then set up an authentication token to allow easy interaction with the repository via the Ubuntu VM command line. Tested the commit/push/pull commands to ensure they all worked as expected on the device – to prevent any future issues arising with these functionalities. |
| 10/11/2021 | 3 Hours 30 Minutes | Stage 1 | Implemented the basic clearing and pixel setting functions. Issues still arise in which the screen is not initially cleaned when the graphics mode is entered, and there are efficiency issues that may become relevant later, however the basic functions required for drawing are present. |
| 12/11/2021 | 2 Hours 50 Minutes | Stage 1 | Implemented the move position function to allow for changing of the graphics cursor position. Additionally implemented the “line to” functionality using the Bresenham line algorithm. These two functions as well as the example commands provided in the specification have been implemented in the “Painter” command, which as of yet takes the following arguments:  Painter [“-e”,”-p”,”-l”]  -e: uses the example square drawing functionality  -p [x] [y]: Plots the points specified in arguments x and y  -l [x0] [y0] [x1] [y1]: Sets the movement position to the arguments provided in x0 and y0, then draws a line to x1 and y1.  As of yet these functions have no error testing built in, nor any hdc usages. |
| 12/11/2021 | 1 Hour 20 Minutes | Stage 1 | Modified the existing arguments for the painter command as such:  Added the -m argument to allow the movement of the cursor towards an arbitrary coordinate  Changed the -l argument to no longer allow movement position to be set beforehand, this function is now expected to be called after the -m function  In addition, changes were made to ensure that the movement values had correct default values, and capacities were applied to user inputs – all inputs have been given a minimum value of 0 and a variable maximum based on the value it represents (319 for x coordinates, 199 for y coordinates). Finally, some text was appended to explain the existing usages in the case of an invalid input, and all functions were given a minimum number of arguments. |
| 16/11/2021 | 3 Hours 40 Minutes | Stage 2 | Implemented all functions required by stage 2 – the creation of pens, the selection of pens and the ability to draw a rectangle. Also spent time reworking old code, including changing the clear function to use a more efficient algorithm. Additionally added a level of error checking for parameters for system calls, with appropriate error messages displayed as a response within the commands that call them.  Pens can be created in any index from 16-255, and must be supplied with a RGB value with a capacity of 63 per colour using the -dp command. The user can then use any pen from 0-255 by using the -gp command with an argument for the index of the pen. The -gp function only checks the boundaries of the index, and does not check if the index has been written to by -dp.  The colour of the graphic produced by an operation is adjusted by the pen currently in use, which by default is set to 15 (the built-in white colour). This system can also be used to overwrite existing pens, though notably not the 0-15 system pens – a precaution put in place to prevent any unforeseen conflicts with pre-existing code or any extensions to the code that may occur in the future.  The fill rectangle system call uses a pointer to a rect structure which stores the top, bottom, left and right of a request – the values of which are automatically set into the range from 0 to Height and 0 to Width respectively. The painter function which calls the rectangle drawing function automatically sorts the 2 x parameters and 2 y parameters into top, bottom, left and right.  This function uses the previously implemented Bresenham line algorithm to draw the requested rectangle (looping through each y coordinate), though notably does not use the cursor nor set the cursor upon completion. |
| 23/11/2021 | 6 Hours 20 Minutes | Stage 3 | Implemented the basic hdc creation mechanics in the system call beginpaint() – attempted to switch the user program to use a different style of input recognition with limited success |
| 23/11/2021 | 2 Hours | Stage 3 | Implemented the input recognition system and tested the usage of beginpaint() and the HDC system. Some additions are still required to finish the stage however the basic foundations for the procedure are present. |
| 27/11/2021 | 1 Hour 30 Minutes | Stage 3 | Added the endpaint function, accessed via the -kHDC command. This system call is only utilised if the user requests it, however they are given a warning if they attempt to exit the painter program space without closing the hdc. This is done to demonstrate the ability of the system to hold onto hdcs and prevent interactions between them – which is best shown through the -gHDC command which allows a user to select any hdc index desired. This is designed as solely an administrative feature that would not be included in a released version of this project, but is a necessary feature to show the working of stage 3. |
| 08/12/2021 | 1 Hour 50 Minutes | Stage 3 | Added locking to beginpaint and endpaint – ensuring that the HDC interactions do not cause issues such as HDCs overwriting one another or two cpus gaining the same HDC (unless this is done via -gHDC, which is for demonstration purposes as previously noted). In addition, consideration was made into the use of locking for the pen interactions, as they both access data that all hdcs can access. After contemplation the decision was made that **Begin paint and end paint should not use locking, and should instead become part of the queue to be implemented in stage 4, thereby having their processes locked during the drawing queue procedure**. If locking was added into the system calls and the procedure was queued, two locks would be needed or deadlock would occur. If these functions were not added to the queue but implemented locking, the possibility would remain for the pen data to be modified before the endpaint queue was called. The only solution is to have the pen functions act inside the queue and therefore its lock but have no lock of its own. |
| 10/12/2021 | 4 Hours | Stage 4 | Began to implement some queueing systems – including adding a new structure to store a function and all its arguments to be processed at the endpaint call. When implemented, this should stop repeated references to system calls, however, as of yet it has outstanding errors that prevent the system from working as expected. The largest current issue is that the procedure args are currently stored as char\*\*, but when provided with a char\*[] value, the strings represented within the array are dropped at the end of the function, resulting in the char\*\* pointing to deallocated memory addresses. This must be fixed soon. |
| 11/12/2021 | 3 Hours | Stage 4 | Changed the approach to arguments for queued functions by switching the system to a structure based system. This means that the values are not dropped as was the case with the char\*\* implementation. However, a new issue has emerged with this implementation in which all queued actions seem to store the same data as the first queued action, likely due to some issue with the creation of new structs. |
| 12/12/2021 | 3 Hours 30 Minutes | Stage 4 | Fixed some issues with the queueing system, now allowing the ability to create pixel requests on demand and execute them at the endpaint call. Next up is the implementation of the ability to manually queue this data, a process that has been started using the writeQueue function, but has outstanding errors that prevent it from meeting its potential. |