100507515

Systems Programming Assessment 2 Log

Minor Note:
I’ve primarily used American spelling of words i.e. colour -> color. as it is more inline with a lot of code found nowadays.

Provided user programs:
All user programs were prefixed with ‘a’ to put them at the top of my file explorer so they’re easier to find. There are other user programs however these are the only ones added to the Makefile but they should show off all implemented functionality.

asnake
A simple game of snake using vm12 and fill rect. Only runs upon user input since I couldn’t find a “check if any input pressed” functionality and it didn’t seem relevant enough to spend time on implementing. Uses WASD for movement and ESC to quit.

acircle
Shows off circle drawing/filling functionality in both vm12 and vm13 modes.

ademo
A slideshow-like program showing off fill rect, moveto, draw line, set pen color + select pen, draw circle and fill circle in vm13 before showing fill rect, select pen and fill circle in vm12. Uses getch to go through.

astage4
A copy of your stage4 example code showing off moveto, drawline, set pen color and select pen in vm13, all being buffered between beginpaint and endpaint. It then does the same, minus set pen color, in vm12.

apixel

Shows off drawing a grid of pixels via setpixel in vm12 and vm13.

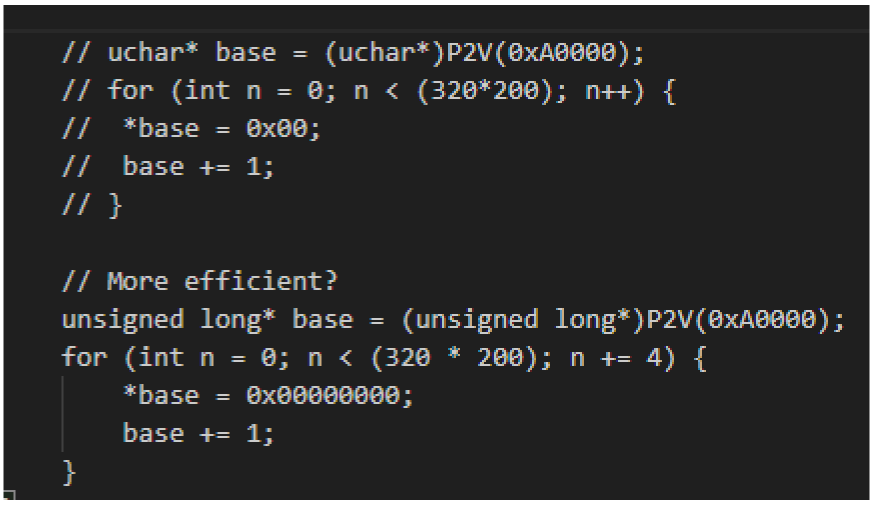
Used Files

Used Files	Usage
acircle.c ademo.c asnake.c astage4.c apixel.c	User programs used in makefile
amulti.c astage1.c astage3.c astage5.c shutdown.c	Unused
user.h	Added all new kernel system calls and ugraphics.c functions.
ugraphics.c	All code graphics functionality for interacting with kernel, buffering frames and some helper functions.
graphics.c	Clear, get and set screen functionality and request/release hdc.
gensyscalls.pl	Added new System calls
graphics_defs.h	Lots of graphics definitions and structs used by ugraphics.c. Has two definitions for Debugging that can be uncommented to display debugging information.
defs.h	Adds a graphics init and clear vm12 function.
rect.h	Rect struct.
sysproc.c	Shutdown exercise code.
vga.c	Added vm12 clear screen function and added getvideomode system call.
Makefile	Added ugraphics.o to ULIB and added new user programs and removed some existing user programs.
main.c	Added reference to graphics.c graphics_init function.

02/12/23

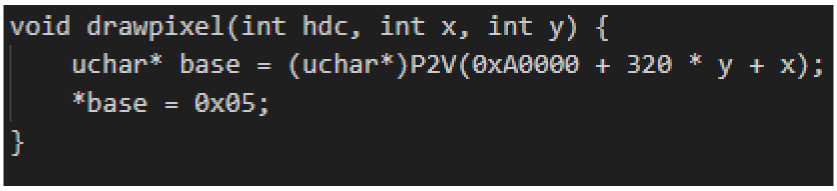
I started out by implementing the clear screen function. I didn't start taking screenshots until later, so I don't have any of doing that although I have previous versions of my code I've taken snippets from.

I did a simple implementation using a for loop. Thinking about what Wayne said in a lecture about their being a faster method, I thought perhaps that changing from a loop using char to a loop using long may have been faster since it's more inline with pointer size and can set more at once. I couldn't really benchmark anything however and it was probably a trial amount of time anyway.



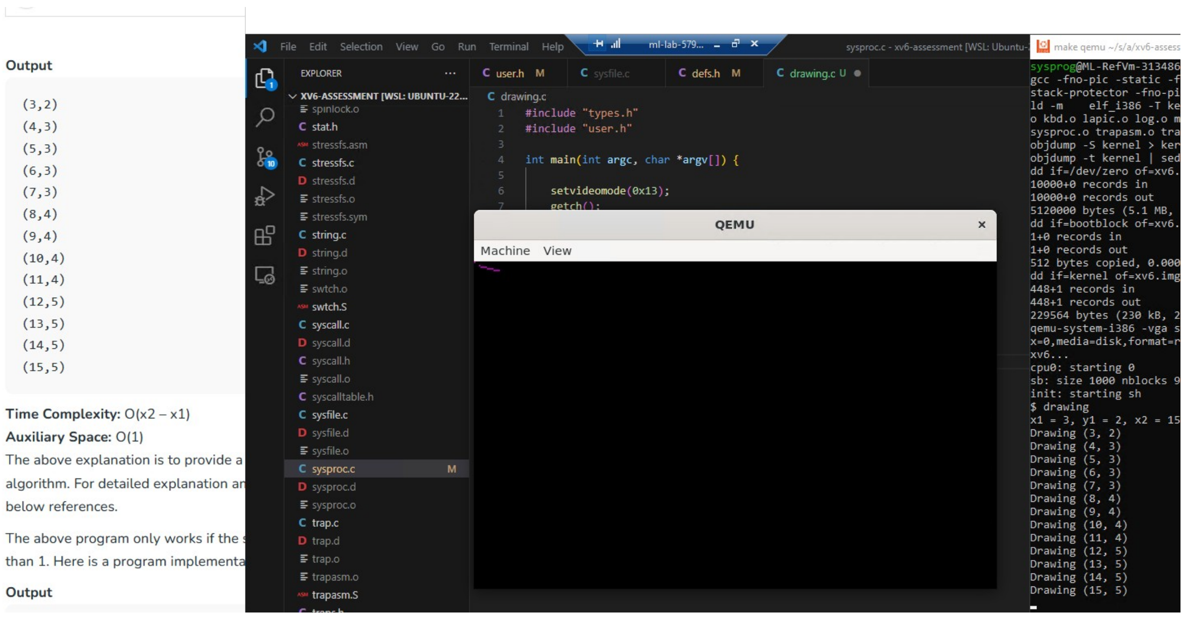
```c
    // uchar* base = (uchar*)P2V(0xA0000);
    // for (int n = 0; n < (320*200); n++) {
    //   *base = 0x00;
    //   base += 1;
    // }

    // More efficient?
    unsigned long* base = (unsigned long*)P2V(0xA0000);
    for (int n = 0; n < (320 * 200); n += 4) {
        *base = 0x00000000;
        base += 1;
    }
```

Adding the set pixel function was next and was quite simple. Again, I don't have screenshots of this. I didn't initially add clamping to the coordinates as it didn't cross my mind at the time.



```c
void drawpixel(int hdc, int x, int y) {
    uchar* base = (uchar*)P2V(0xA0000 + 320 * y + x);
    *base = 0x05;
}
```

Moving onto drawline, I moved through a few implementations before settling on one as different implementations didn't draw as I expected. I used cprintf frequently to confirm the results I was seeing which is a reoccurring theme amongst my testing.



Output

```
(3,2)
(4,3)
(5,3)
(6,3)
(7,3)
(8,4)
(9,4)
(10,4)
(11,4)
(12,5)
(13,5)
(14,5)
(15,5)
```

Time Complexity: O(x2 – x1)
Auxiliary Space: O(1)
The above explanation is to provide a
algorithm. For detailed explanation ar
below references.

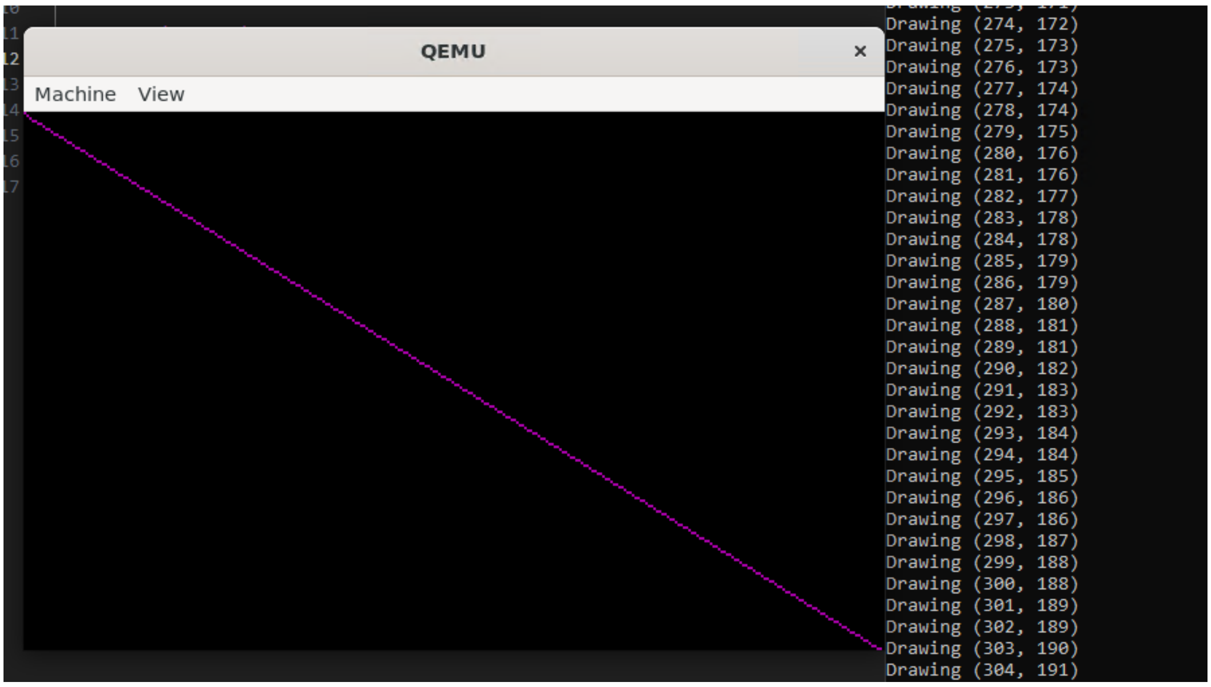
The above program only works if the s
than 1. Here is a program implementa

Output

drawing.c

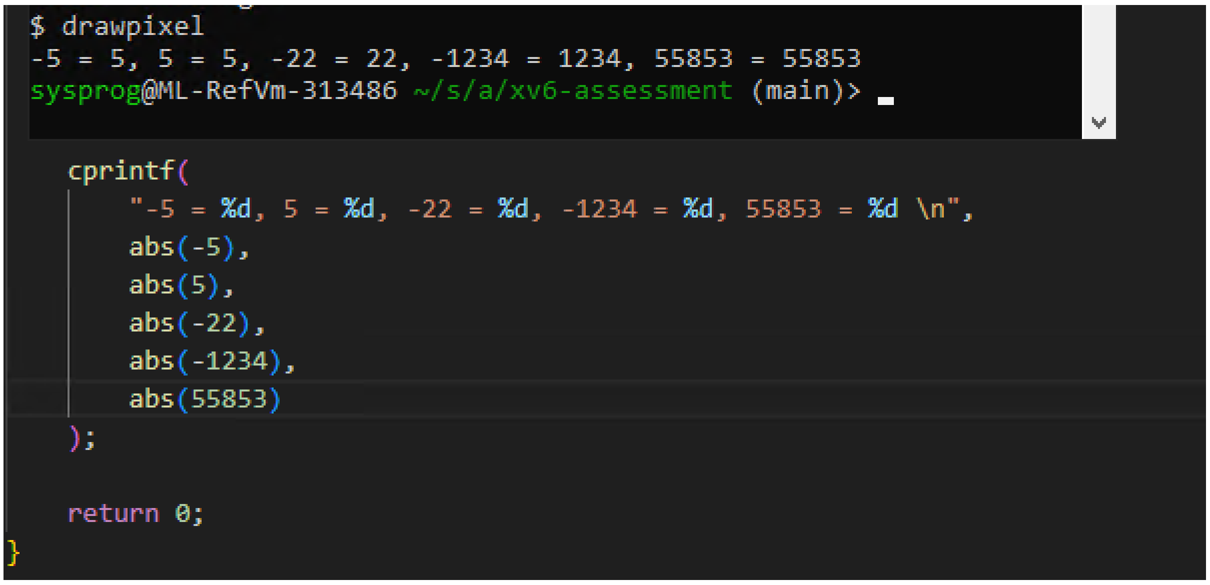
```c
#include "types.h"
#include "user.h"

int main(int argc, char *argv[]) {

    setvideomode(0x13);
    getch();
```

```
sysprog@ML-RefVm-313486
gcc -fno-pic -static -f
stack-protector -fno-pi
ld -m    elf_i386 -T ke
o kbd.o lapic.o log.o m
sysproc.o trapasm.o tra
objdump -S kernel > ker
objdump -t kernel | sed
dd if=/dev/zero of=xv6.
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB,
dd if=bootblock of=xv6.
1+0 records in
1+0 records out
512 bytes copied, 0.000
dd if=kernel of=xv6.img
448+1 records in
448+1 records out
229564 bytes (230 kB, 2
qemu-system-i386 -vga s
x=0,media=disk,format=r
xv6...
cpu0: starting 0
sb: size 1000 nblocks 9
init: starting sh
$ drawing
x1 = 3, y1 = 2, x2 = 15
Drawing (3, 2)
Drawing (4, 3)
Drawing (5, 3)
Drawing (6, 3)
Drawing (7, 3)
Drawing (8, 4)
Drawing (9, 4)
Drawing (10, 4)
Drawing (11, 4)
Drawing (12, 5)
Drawing (13, 5)
Drawing (14, 5)
Drawing (15, 5)
```



```
Drawing (274, 172)
Drawing (275, 173)
Drawing (276, 173)
Drawing (277, 174)
Drawing (278, 174)
Drawing (279, 175)
Drawing (280, 176)
Drawing (281, 176)
Drawing (282, 177)
Drawing (283, 178)
Drawing (284, 178)
Drawing (285, 179)
Drawing (286, 179)
Drawing (287, 180)
Drawing (288, 181)
Drawing (289, 181)
Drawing (290, 182)
Drawing (291, 183)
Drawing (292, 183)
Drawing (293, 184)
Drawing (294, 184)
Drawing (295, 185)
Drawing (296, 186)
Drawing (297, 186)
Drawing (298, 187)
Drawing (299, 188)
Drawing (300, 188)
Drawing (301, 189)
Drawing (302, 189)
Drawing (303, 190)
Drawing (304, 191)
```

I needed an abs function for a different implementation, so I added one and used cprintf to make sure it was working.

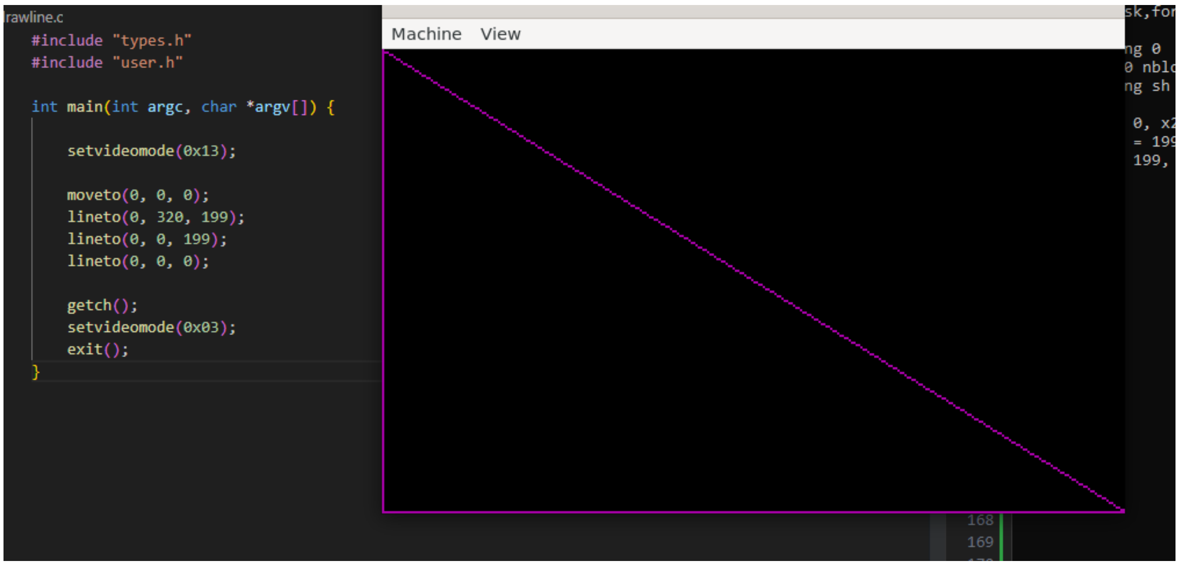


```
$ drawpixel
-5 = 5, 5 = 5, -22 = 22, -1234 = 1234, 55853 = 55853
sysprog@ML-RefVm-313486 ~/s/a/xv6-assessment (main)>
```

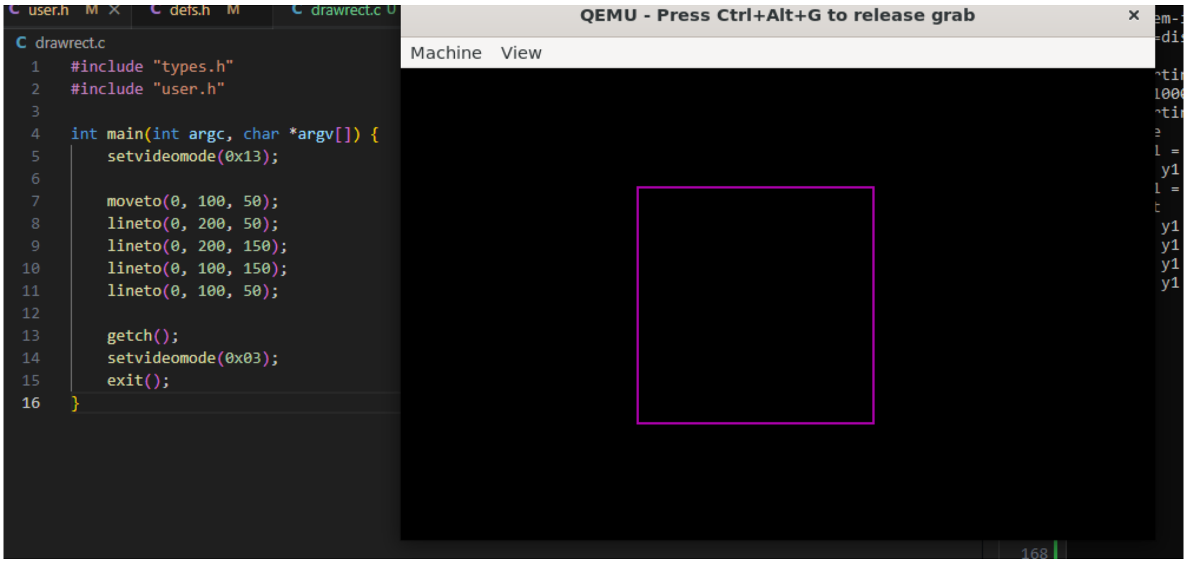
```
    cprintf(
        "-5 = %d, 5 = %d, -22 = %d, -1234 = %d, 55853 = %d \n",
        abs(-5),
        abs(5),
        abs(-22),
        abs(-1234),
        abs(55853)
    );

    return 0;
}
```

03/12/23

Eventually, I settled on a working implementation that worked in all directions. I also implemented moveto, storing all relevant data inside a struct in kernel space in graphics.c.

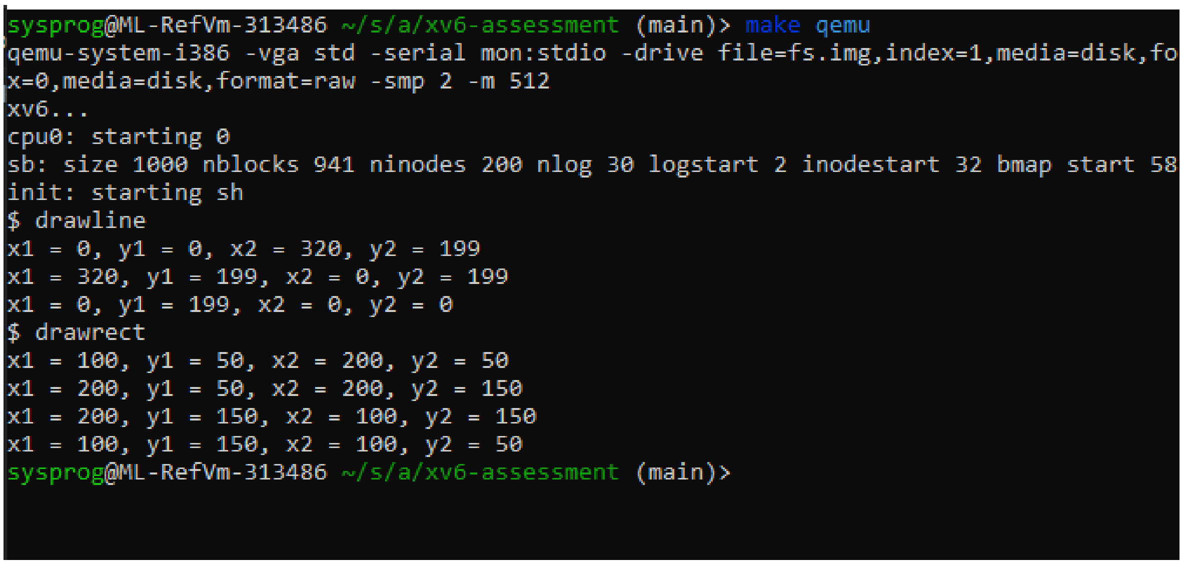


```
#include "types.h"
#include "user.h"

int main(int argc, char *argv[]) {

    setvideomode(0x13);

    moveto(0, 0, 0);
    lineto(0, 320, 199);
    lineto(0, 0, 199);
    lineto(0, 0, 0);

    getch();
    setvideomode(0x03);
    exit();
}
```



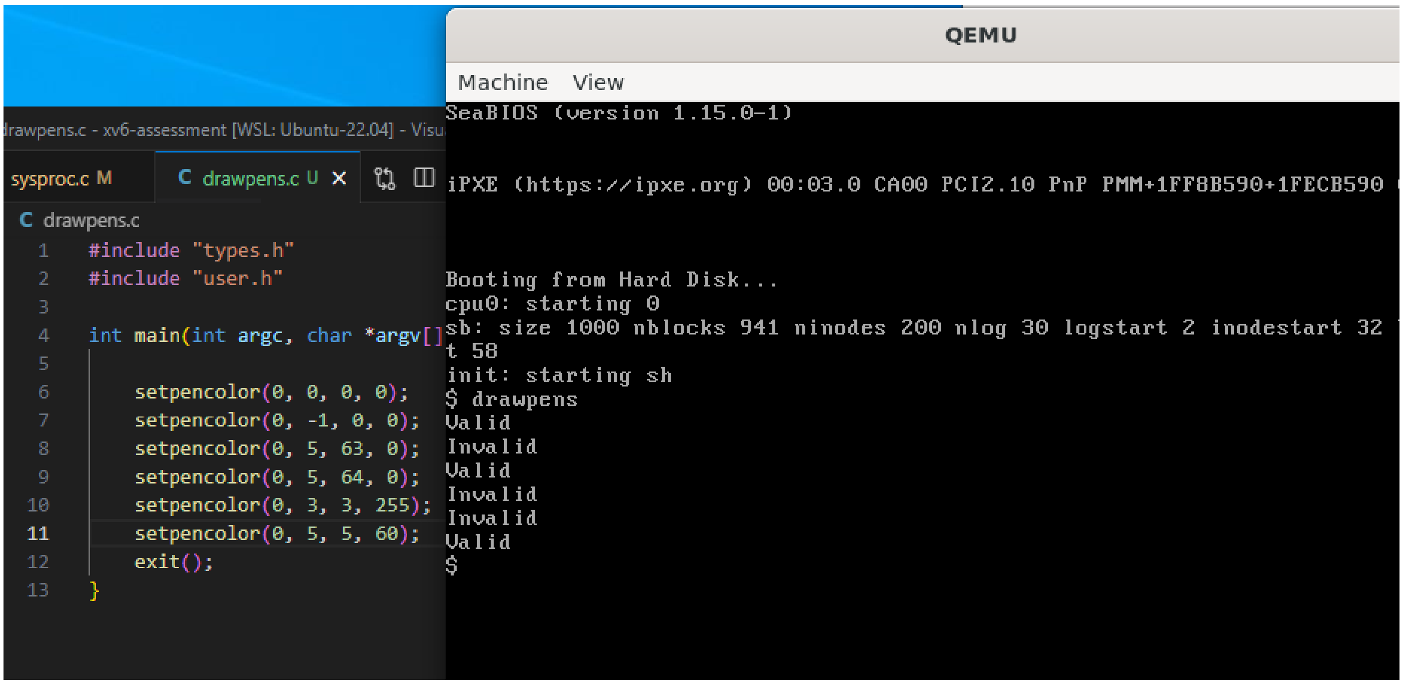
```
#include "types.h"
#include "user.h"

int main(int argc, char *argv[]) {
    setvideomode(0x13);

    moveto(0, 100, 50);
    lineto(0, 200, 50);
    lineto(0, 200, 150);
    lineto(0, 100, 150);
    lineto(0, 100, 50);

    getch();
    setvideomode(0x03);
    exit();
}
```

Output from the above screenshots.



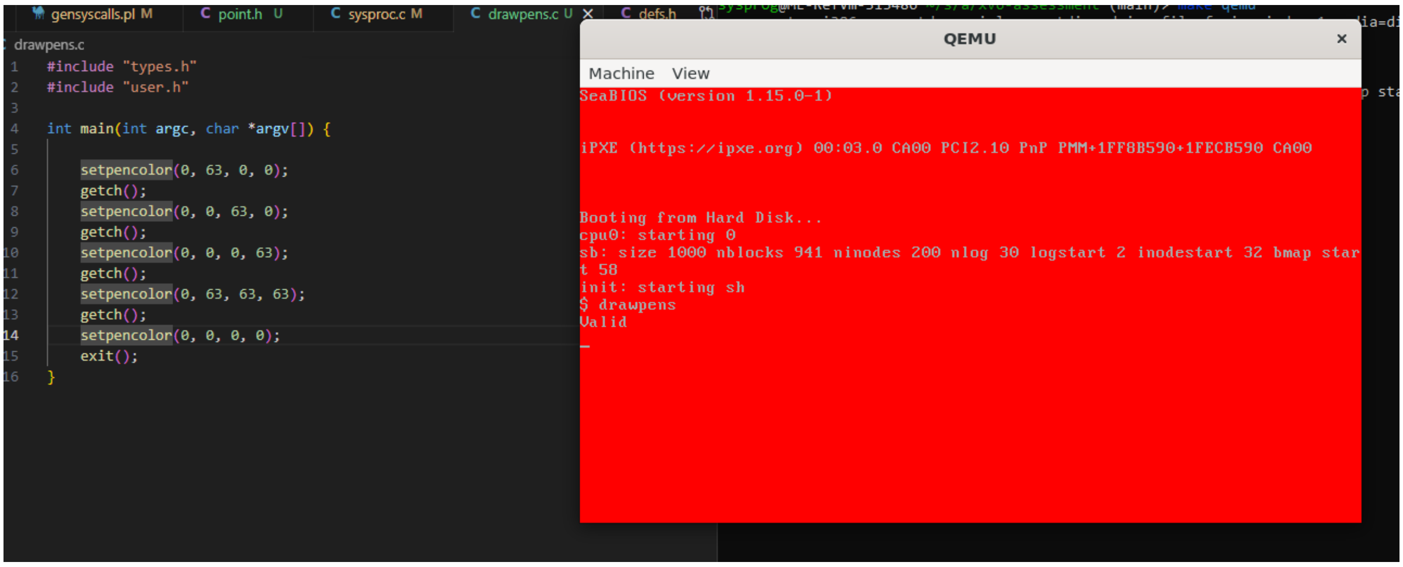
```
sysprog@ML-RefVm-313486 ~/s/a/xv6-assessment (main)> make qemu
qemu-system-i386 -vga std -serial mon:stdio -drive file=fs.img,index=1,media=disk,fo
x=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ drawline
x1 = 0, y1 = 0, x2 = 320, y2 = 199
x1 = 320, y1 = 199, x2 = 0, y2 = 199
x1 = 0, y1 = 199, x2 = 0, y2 = 0
$ drawrect
x1 = 100, y1 = 50, x2 = 200, y2 = 50
x1 = 200, y1 = 50, x2 = 200, y2 = 150
x1 = 200, y1 = 150, x2 = 100, y2 = 150
x1 = 100, y1 = 150, x2 = 100, y2 = 50
sysprog@ML-RefVm-313486 ~/s/a/xv6-assessment (main)>
```

I added the stage 2 setpencolor functionality next, initially making so that pen colors that were out of bounds returned an error instead of being clamped. I tested this with print statements first before actually changing the colors of anything.



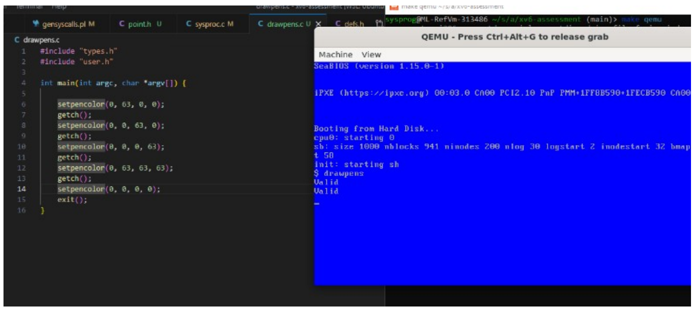
```c
#include "types.h"
#include "user.h"

int main(int argc, char *argv[]

    setpencolor(0, 0, 0, 0);
    setpencolor(0, -1, 0, 0);
    setpencolor(0, 5, 63, 0);
    setpencolor(0, 5, 64, 0);
    setpencolor(0, 3, 3, 255);
    setpencolor(0, 5, 5, 60);
    exit();
}
```

```
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590


Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32
t 58
init: starting sh
$ drawpens
Valid
Invalid
Valid
Invalid
Invalid
Valid
$
```



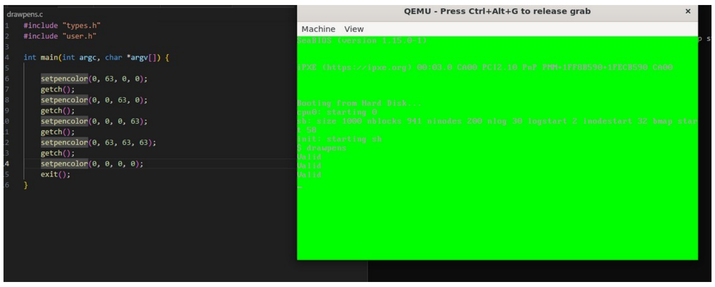
```c
#include "types.h"
#include "user.h"

int main(int argc, char *argv[]) {

    setpencolor(0, 63, 0, 0);
    getch();
    setpencolor(0, 0, 63, 0);
    getch();
    setpencolor(0, 0, 0, 63);
    getch();
    setpencolor(0, 63, 63, 63);
    getch();
    setpencolor(0, 0, 0, 0);
    exit();
}
```

```
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00


Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ drawpens
Valid
_
```



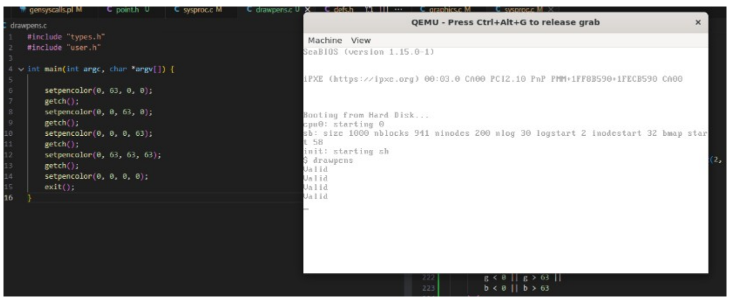
```c
#include "types.h"
#include "user.h"

int main(int argc, char *argv[]) {

    setpencolor(0, 63, 0, 0);
    getch();
    setpencolor(0, 0, 63, 0);
    getch();
    setpencolor(0, 0, 0, 63);
    getch();
    setpencolor(0, 63, 63, 63);
    getch();
    setpencolor(0, 0, 0, 0);
    exit();
}
```

```
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00


Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap
t 58
init: starting sh
$ drawpens
Valid
Valid
_
```



```c
#include "types.h"
#include "user.h"

int main(int argc, char *argv[]) {

    setpencolor(0, 63, 0, 0);
    getch();
    setpencolor(0, 0, 63, 0);
    getch();
    setpencolor(0, 0, 0, 63);
    getch();
    setpencolor(0, 63, 63, 63);
    getch();
    setpencolor(0, 0, 0, 0);
    exit();
}
```

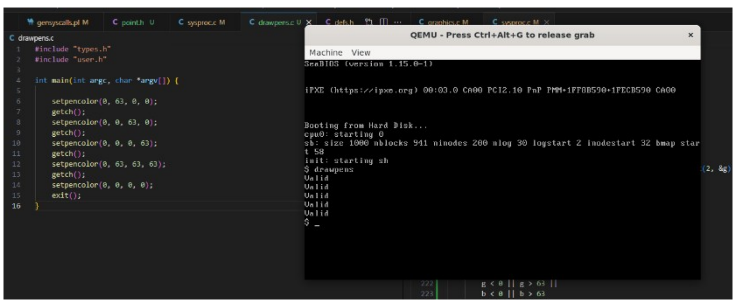
```
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00


Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ drawpens
Valid
Valid
Valid
_
```



```c
#include "types.h"
#include "user.h"

int main(int argc, char *argv[]) {

    setpencolor(0, 63, 0, 0);
    getch();
    setpencolor(0, 0, 63, 0);
    getch();
    setpencolor(0, 0, 0, 63);
    getch();
    setpencolor(0, 63, 63, 63);
    getch();
    setpencolor(0, 0, 0, 0);
    exit();
}
```

```
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00


Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ drawpens
Valid
Valid
Valid
Valid
_
```

```
g < 0 || g > 63 ||
b < 0 || b > 63
```

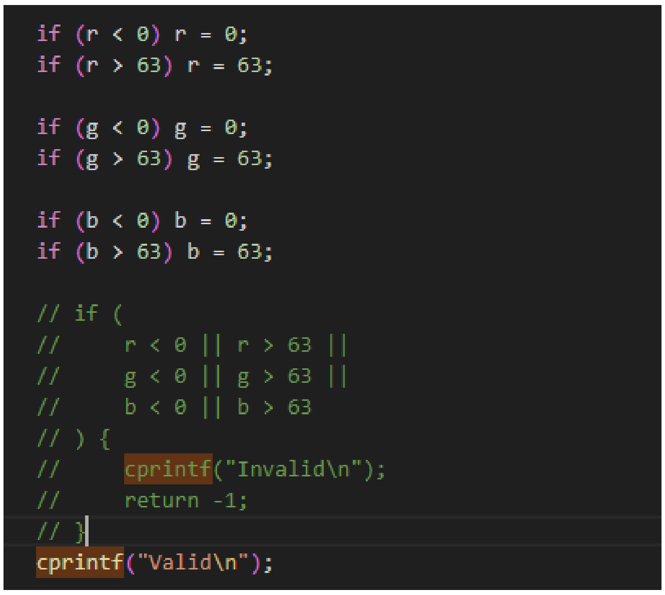


```c
#include "types.h"
#include "user.h"

int main(int argc, char *argv[]) {

    setpencolor(0, 63, 0, 0);
    getch();
    setpencolor(0, 0, 63, 0);
    getch();
    setpencolor(0, 0, 0, 63);
    getch();
    setpencolor(0, 63, 63, 63);
    getch();
    setpencolor(0, 0, 0, 0);
    exit();
}
```

```
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00


Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ drawpens
Valid
Valid
Valid
Valid
Valid
$ _
```

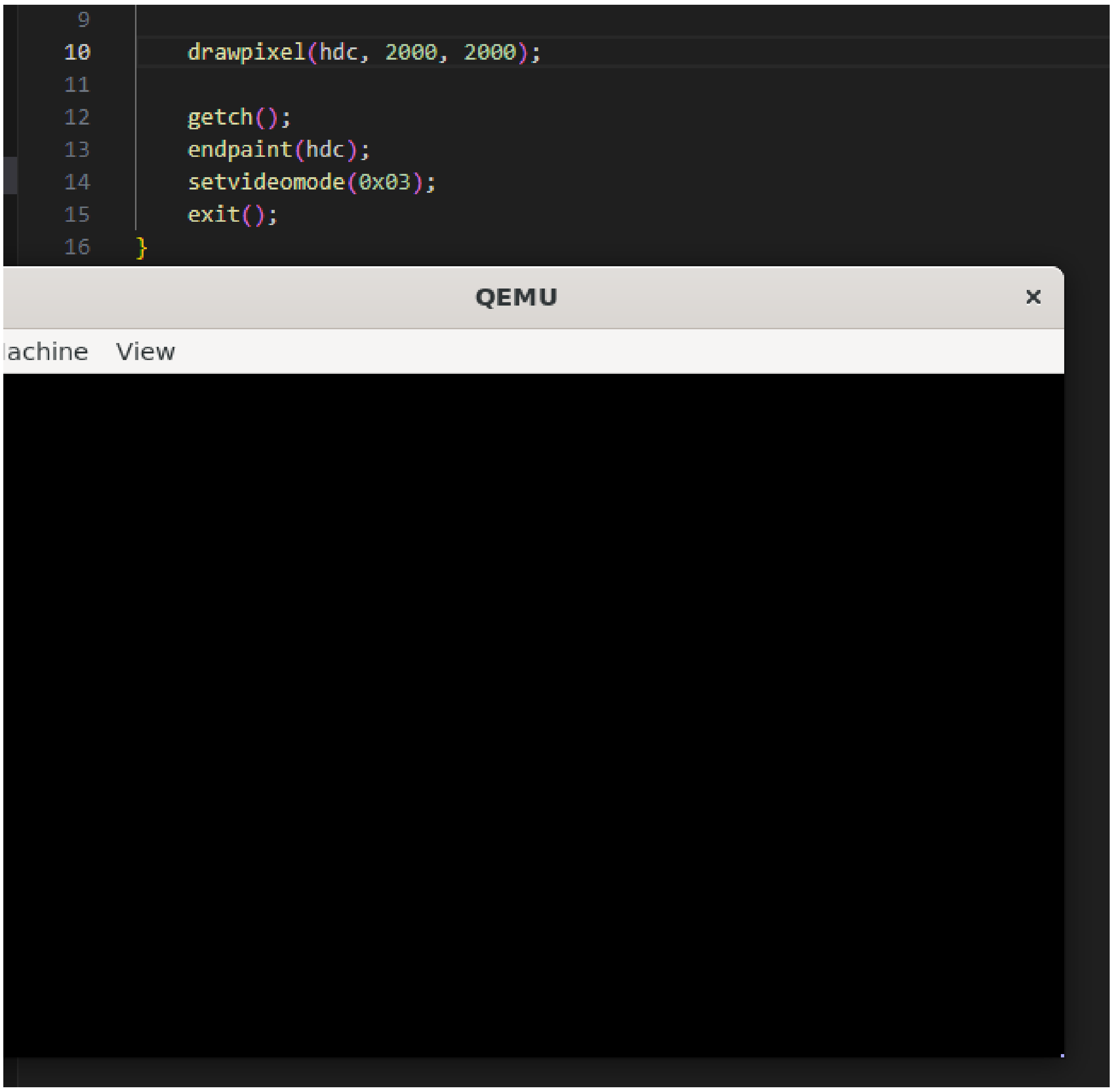
```
g < 0 || g > 63 ||
b < 0 || b > 63
```

I later changed the set pen color to only reject invalid indexes and accept invalid colors by clamping them. I considered using bitwise and but rejected it for now (I did change it at a later date however).



```
if (r < 0) r = 0;
if (r > 63) r = 63;

if (g < 0) g = 0;
if (g > 63) g = 63;

if (b < 0) b = 0;
if (b > 63) b = 63;

// if (
//     r < 0 || r > 63 ||
//     g < 0 || g > 63 ||
//     b < 0 || b > 63
// ) {
//     cprintf("Invalid\n");
//     return -1;
// }
cprintf("Valid\n");
```

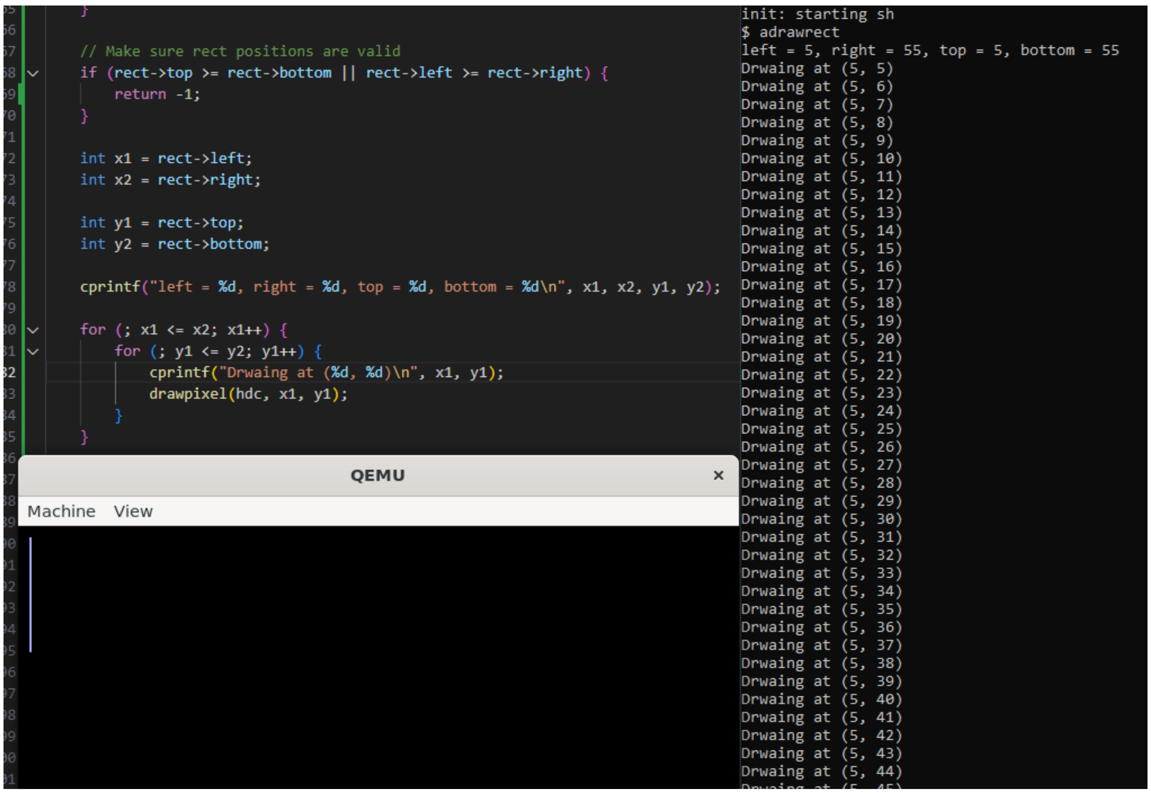
Added clamping to drawpixel (now setpixel) to make sure it was only displaying in bounds.



```

    drawpixel(hdc, 2000, 2000);

    getch();
    endpaint(hdc);
    setvideomode(0x03);
    exit();
}
```

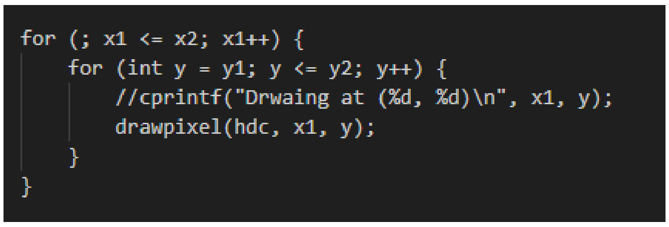
I started work on the fillrect function which was also straightforward once I figured out how to implement the rect struct which I saw in other files should have it's own head file when used across the program. The function is a simple for loop that loops over the coordinates and draws a pixel.

My implementation didn't work initially as I tried to reduce variables used by not using variables in the for loops, however I need a y variable in my for loop as it needed to be reset/looped over. The image below should have been drawing a rectangle. After changing that, it worked fine.

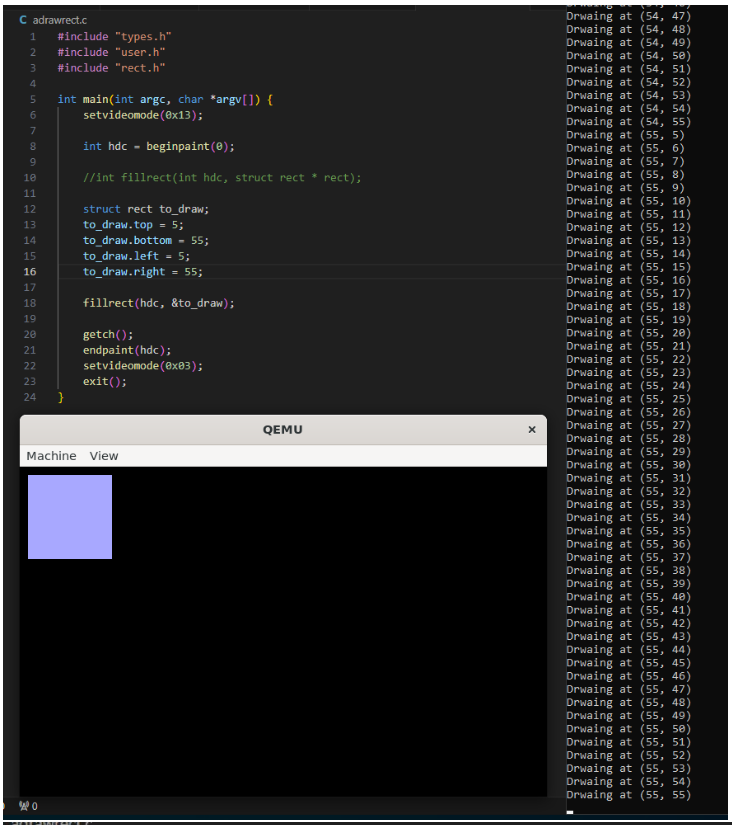


```c
    }

    // Make sure rect positions are valid
    if (rect->top >= rect->bottom || rect->left >= rect->right) {
        return -1;
    }

    int x1 = rect->left;
    int x2 = rect->right;

    int y1 = rect->top;
    int y2 = rect->bottom;

    cprintf("left = %d, right = %d, top = %d, bottom = %d\n", x1, x2, y1, y2);

    for (; x1 <= x2; x1++) {
        for (; y1 <= y2; y1++) {
            cprintf("Drwaing at (%d, %d)\n", x1, y1);
            drawpixel(hdc, x1, y1);
        }
    }
```

```
init: starting sh
$ adrawrect
left = 5, right = 55, top = 5, bottom = 55
Drwaing at (5, 5)
Drwaing at (5, 6)
Drwaing at (5, 7)
Drwaing at (5, 8)
Drwaing at (5, 9)
Drwaing at (5, 10)
Drwaing at (5, 11)
Drwaing at (5, 12)
Drwaing at (5, 13)
Drwaing at (5, 14)
Drwaing at (5, 15)
Drwaing at (5, 16)
Drwaing at (5, 17)
Drwaing at (5, 18)
Drwaing at (5, 19)
Drwaing at (5, 20)
Drwaing at (5, 21)
Drwaing at (5, 22)
Drwaing at (5, 23)
Drwaing at (5, 24)
Drwaing at (5, 25)
Drwaing at (5, 26)
Drwaing at (5, 27)
Drwaing at (5, 28)
Drwaing at (5, 29)
Drwaing at (5, 30)
Drwaing at (5, 31)
Drwaing at (5, 32)
Drwaing at (5, 33)
Drwaing at (5, 34)
Drwaing at (5, 35)
Drwaing at (5, 36)
Drwaing at (5, 37)
Drwaing at (5, 38)
Drwaing at (5, 39)
Drwaing at (5, 40)
Drwaing at (5, 41)
Drwaing at (5, 42)
Drwaing at (5, 43)
Drwaing at (5, 44)
```

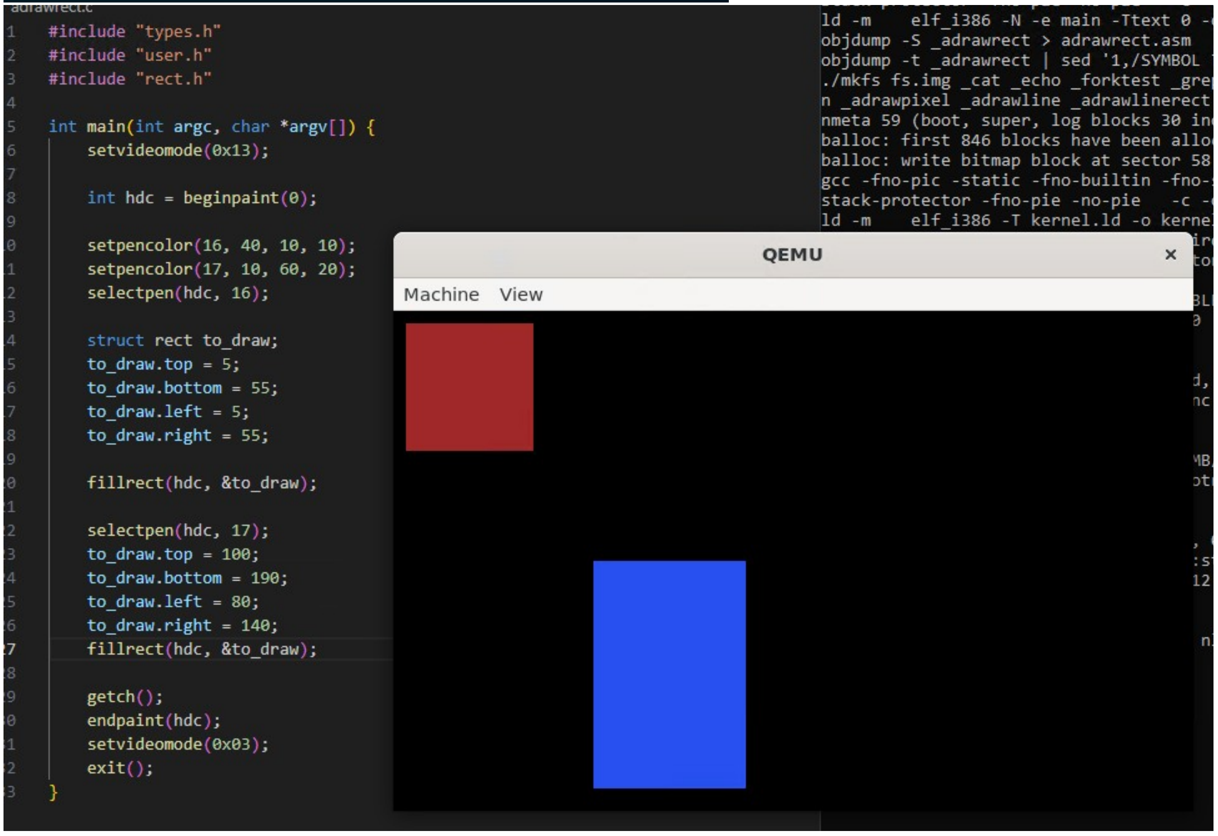


```c
for (; x1 <= x2; x1++) {
    for (int y = y1; y <= y2; y++) {
        //cprintf("Drwaing at (%d, %d)\n", x1, y);
        drawpixel(hdc, x1, y);
    }
}
```



```c
#include "types.h"
#include "user.h"
#include "rect.h"

int main(int argc, char *argv[]) {
    setvideomode(0x13);

    int hdc = beginpaint(0);

    //int fillrect(int hdc, struct rect * rect);

    struct rect to_draw;
    to_draw.top = 5;
    to_draw.bottom = 55;
    to_draw.left = 5;
    to_draw.right = 55;

    fillrect(hdc, &to_draw);

    getch();
    endpaint(hdc);
    setvideomode(0x03);
    exit();
}
```

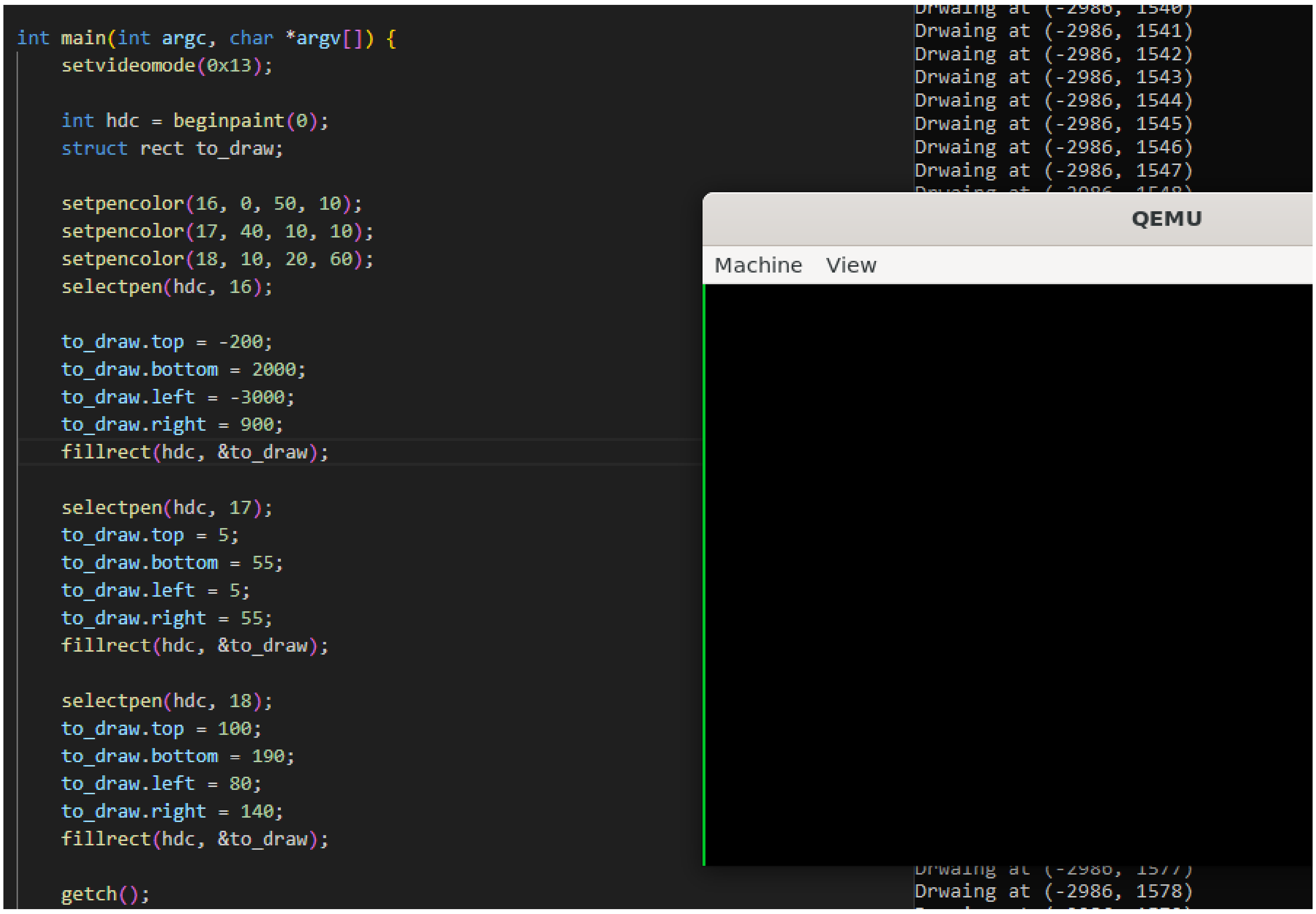
```
Drwaing at (54, 47)
Drwaing at (54, 48)
Drwaing at (54, 49)
Drwaing at (54, 50)
Drwaing at (54, 51)
Drwaing at (54, 52)
Drwaing at (54, 53)
Drwaing at (54, 54)
Drwaing at (54, 55)
Drwaing at (55, 5)
Drwaing at (55, 6)
Drwaing at (55, 7)
Drwaing at (55, 8)
Drwaing at (55, 9)
Drwaing at (55, 10)
Drwaing at (55, 11)
Drwaing at (55, 12)
Drwaing at (55, 13)
Drwaing at (55, 14)
Drwaing at (55, 15)
Drwaing at (55, 16)
Drwaing at (55, 17)
Drwaing at (55, 18)
Drwaing at (55, 19)
Drwaing at (55, 20)
Drwaing at (55, 21)
Drwaing at (55, 22)
Drwaing at (55, 23)
Drwaing at (55, 24)
Drwaing at (55, 25)
Drwaing at (55, 26)
Drwaing at (55, 27)
Drwaing at (55, 28)
Drwaing at (55, 29)
Drwaing at (55, 30)
Drwaing at (55, 31)
Drwaing at (55, 32)
Drwaing at (55, 33)
Drwaing at (55, 34)
Drwaing at (55, 35)
Drwaing at (55, 36)
Drwaing at (55, 37)
Drwaing at (55, 38)
Drwaing at (55, 39)
Drwaing at (55, 40)
Drwaing at (55, 41)
Drwaing at (55, 42)
Drwaing at (55, 43)
Drwaing at (55, 44)
Drwaing at (55, 45)
Drwaing at (55, 46)
Drwaing at (55, 47)
Drwaing at (55, 48)
Drwaing at (55, 49)
Drwaing at (55, 50)
Drwaing at (55, 51)
Drwaing at (55, 52)
Drwaing at (55, 53)
Drwaing at (55, 54)
Drwaing at (55, 55)
```



```c
#include "types.h"
#include "user.h"
#include "rect.h"

int main(int argc, char *argv[]) {
    setvideomode(0x13);

    int hdc = beginpaint(0);

    setpencolor(16, 40, 10, 10);
    setpencolor(17, 10, 60, 20);
    selectpen(hdc, 16);

    struct rect to_draw;
    to_draw.top = 5;
    to_draw.bottom = 55;
    to_draw.left = 5;
    to_draw.right = 55;

    fillrect(hdc, &to_draw);

    selectpen(hdc, 17);
    to_draw.top = 100;
    to_draw.bottom = 190;
    to_draw.left = 80;
    to_draw.right = 140;
    fillrect(hdc, &to_draw);

    getch();
    endpaint(hdc);
    setvideomode(0x03);
    exit();
}
```

```
ld -m    elf_i386 -N -e main -Ttext 0 -
objdump -S _adrawrect > adrawrect.asm
objdump -t _adrawrect | sed '1,/SYMBOL
./mkfs fs.img _cat _echo _forktest _gre
n _adrawpixel _adrawline _adrawlinerect
nmeta 59 (boot, super, log blocks 30 in
balloc: first 846 blocks have been allo
balloc: write bitmap block at sector 58
gcc -fno-pic -static -fno-builtin -fno-
stack-protector -fno-pie -no-pie   -c -
ld -m    elf_i386 -T kernel.ld -o kerne
```

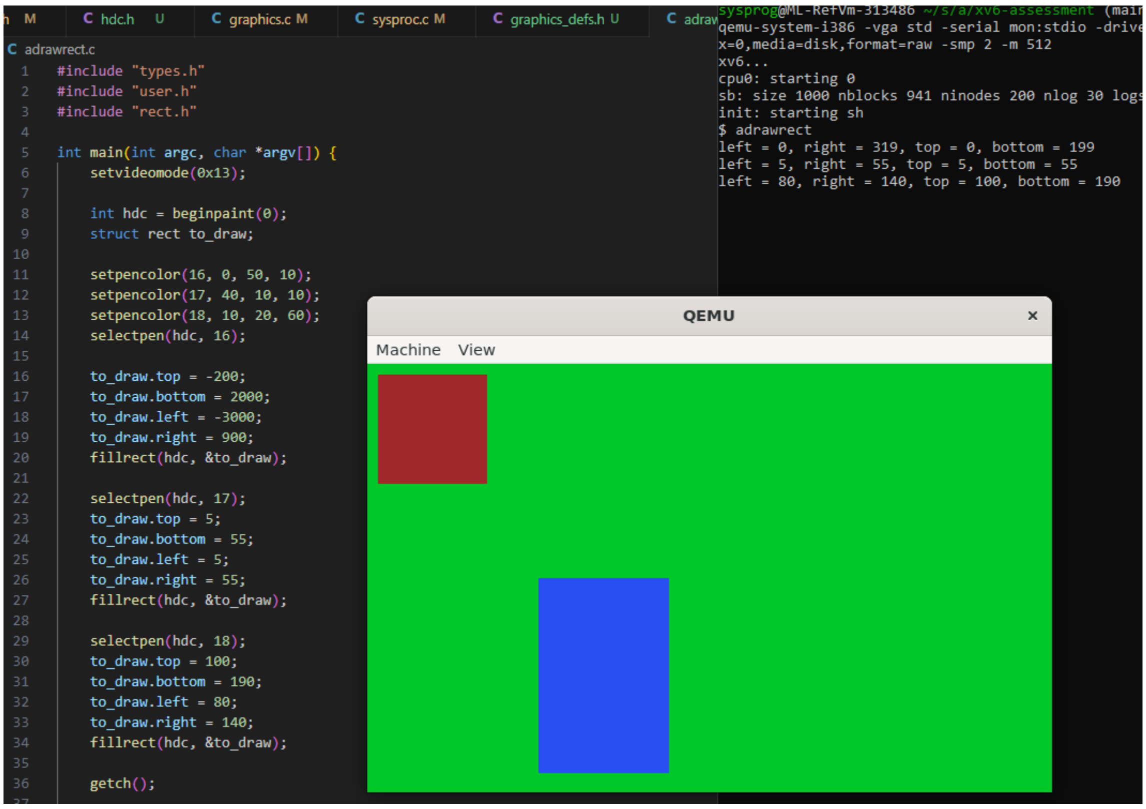
I'd added checks to make sure left couldn't be bigger than right and same with up/down however I didn't initially clamp variables. I found that using the bounds below halted my program as it spent so much time setting pixels off the screen. After deciding to clamp these values, it was much faster.



```
int main(int argc, char *argv[]) {
    setvideomode(0x13);

    int hdc = beginpaint(0);
    struct rect to_draw;

    setpencolor(16, 0, 50, 10);
    setpencolor(17, 40, 10, 10);
    setpencolor(18, 10, 20, 60);
    selectpen(hdc, 16);

    to_draw.top = -200;
    to_draw.bottom = 2000;
    to_draw.left = -3000;
    to_draw.right = 900;
    fillrect(hdc, &to_draw);

    selectpen(hdc, 17);
    to_draw.top = 5;
    to_draw.bottom = 55;
    to_draw.left = 5;
    to_draw.right = 55;
    fillrect(hdc, &to_draw);

    selectpen(hdc, 18);
    to_draw.top = 100;
    to_draw.bottom = 190;
    to_draw.left = 80;
    to_draw.right = 140;
    fillrect(hdc, &to_draw);

    getch();
```

```
Drwaing at (-2986, 1540)
Drwaing at (-2986, 1541)
Drwaing at (-2986, 1542)
Drwaing at (-2986, 1543)
Drwaing at (-2986, 1544)
Drwaing at (-2986, 1545)
Drwaing at (-2986, 1546)
Drwaing at (-2986, 1547)
Drwaing at (-2986, 1548)
Drwaing at (-2986, 1577)
Drwaing at (-2986, 1578)
```

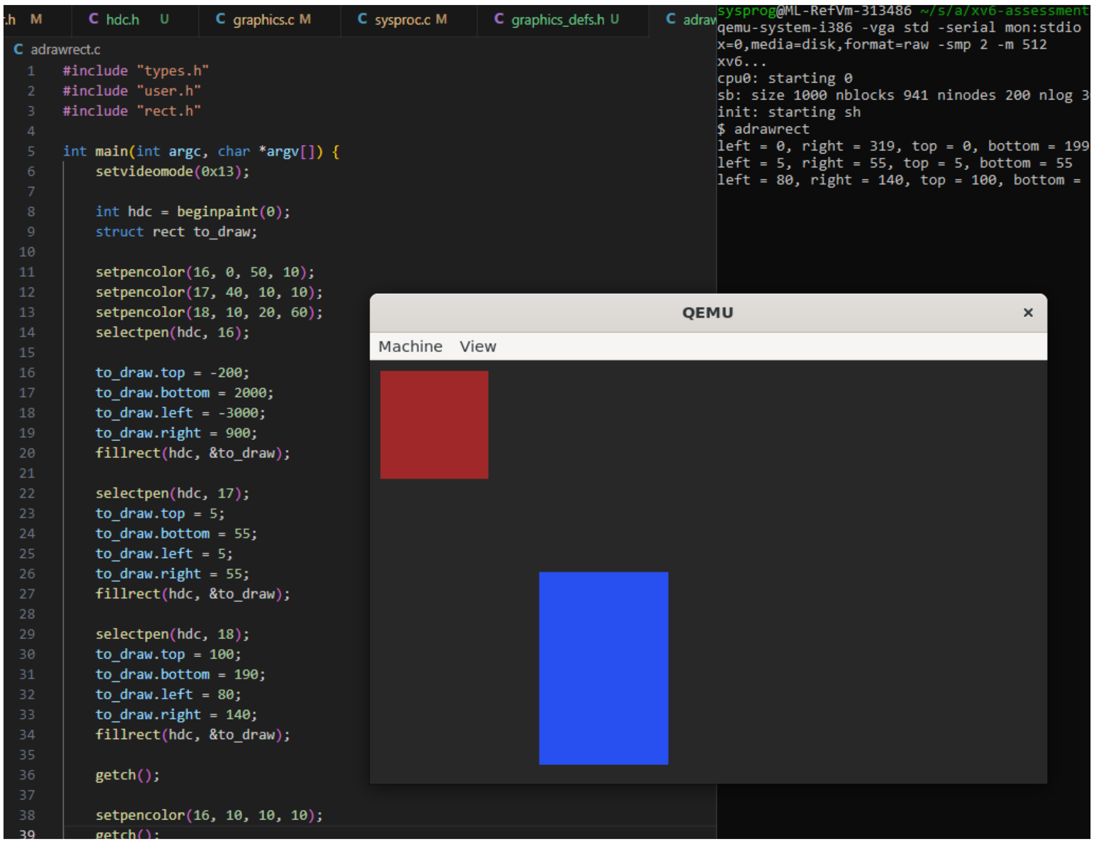
Values on the right are clamped compared to the values on the right.



```c
#include "types.h"
#include "user.h"
#include "rect.h"

int main(int argc, char *argv[]) {
    setvideomode(0x13);

    int hdc = beginpaint(0);
    struct rect to_draw;

    setpencolor(16, 0, 50, 10);
    setpencolor(17, 40, 10, 10);
    setpencolor(18, 10, 20, 60);
    selectpen(hdc, 16);

    to_draw.top = -200;
    to_draw.bottom = 2000;
    to_draw.left = -3000;
    to_draw.right = 900;
    fillrect(hdc, &to_draw);

    selectpen(hdc, 17);
    to_draw.top = 5;
    to_draw.bottom = 55;
    to_draw.left = 5;
    to_draw.right = 55;
    fillrect(hdc, &to_draw);

    selectpen(hdc, 18);
    to_draw.top = 100;
    to_draw.bottom = 190;
    to_draw.left = 80;
    to_draw.right = 140;
    fillrect(hdc, &to_draw);

    getch();
```

```
sysprog@ML-RefVm-313486 ~/s/a/xv6-assessment (mair
qemu-system-i386 -vga std -serial mon:stdio -drive
x=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logs
init: starting sh
$ adrawrect
left = 0, right = 319, top = 0, bottom = 199
left = 5, right = 55, top = 5, bottom = 55
left = 80, right = 140, top = 100, bottom = 190
```

Testing set pen color again on existing pixels.

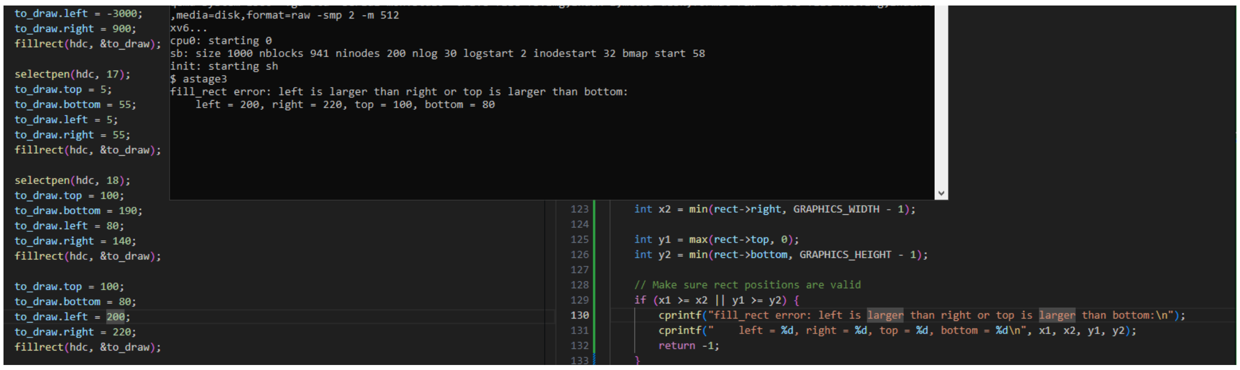


```c
#include "types.h"
#include "user.h"
#include "rect.h"

int main(int argc, char *argv[]) {
    setvideomode(0x13);

    int hdc = beginpaint(0);
    struct rect to_draw;

    setpencolor(16, 0, 50, 10);
    setpencolor(17, 40, 10, 10);
    setpencolor(18, 10, 20, 60);
    selectpen(hdc, 16);

    to_draw.top = -200;
    to_draw.bottom = 2000;
    to_draw.left = -3000;
    to_draw.right = 900;
    fillrect(hdc, &to_draw);

    selectpen(hdc, 17);
    to_draw.top = 5;
    to_draw.bottom = 55;
    to_draw.left = 5;
    to_draw.right = 55;
    fillrect(hdc, &to_draw);

    selectpen(hdc, 18);
    to_draw.top = 100;
    to_draw.bottom = 190;
    to_draw.left = 80;
    to_draw.right = 140;
    fillrect(hdc, &to_draw);

    getch();

    setpencolor(16, 10, 10, 10);
    getch();
```

```
sysprog@ML-RefVm-313486 ~/s/a/xv6-assessment
qemu-system-i386 -vga std -serial mon:stdio
x=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 3
init: starting sh
$ adrawrect
left = 0, right = 319, top = 0, bottom = 199
left = 5, right = 55, top = 5, bottom = 55
left = 80, right = 140, top = 100, bottom =
```

I started on the hdc functionality slightly earlier, as seen in previous screenshots, as I saw it as being something that could change a lot of my existing code. As such, implemented stage3 by changing my struct in graphics.c to have two arrays, one for the hdc data and another for whether they were in use or not. I don't have many screenshots of this as there wasn't a whole lot to show off at the time.

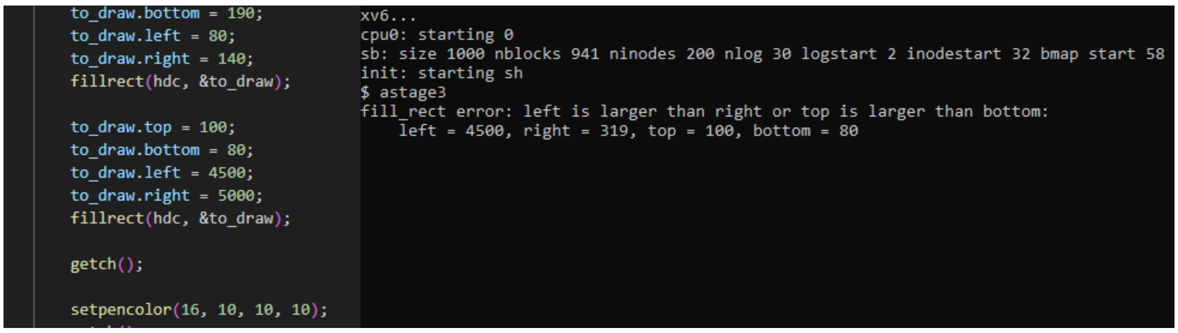
Checking rects can't draw invalid shapes.



```
to_draw.left = -3000;
to_draw.right = 900;
fillrect(hdc, &to_draw);

selectpen(hdc, 17);
to_draw.top = 5;
to_draw.bottom = 55;
to_draw.left = 5;
to_draw.right = 55;
fillrect(hdc, &to_draw);

selectpen(hdc, 18);
to_draw.top = 100;
to_draw.bottom = 190;
to_draw.left = 80;
to_draw.right = 140;
fillrect(hdc, &to_draw);

to_draw.top = 100;
to_draw.bottom = 80;
to_draw.left = 200;
to_draw.right = 220;
fillrect(hdc, &to_draw);
```

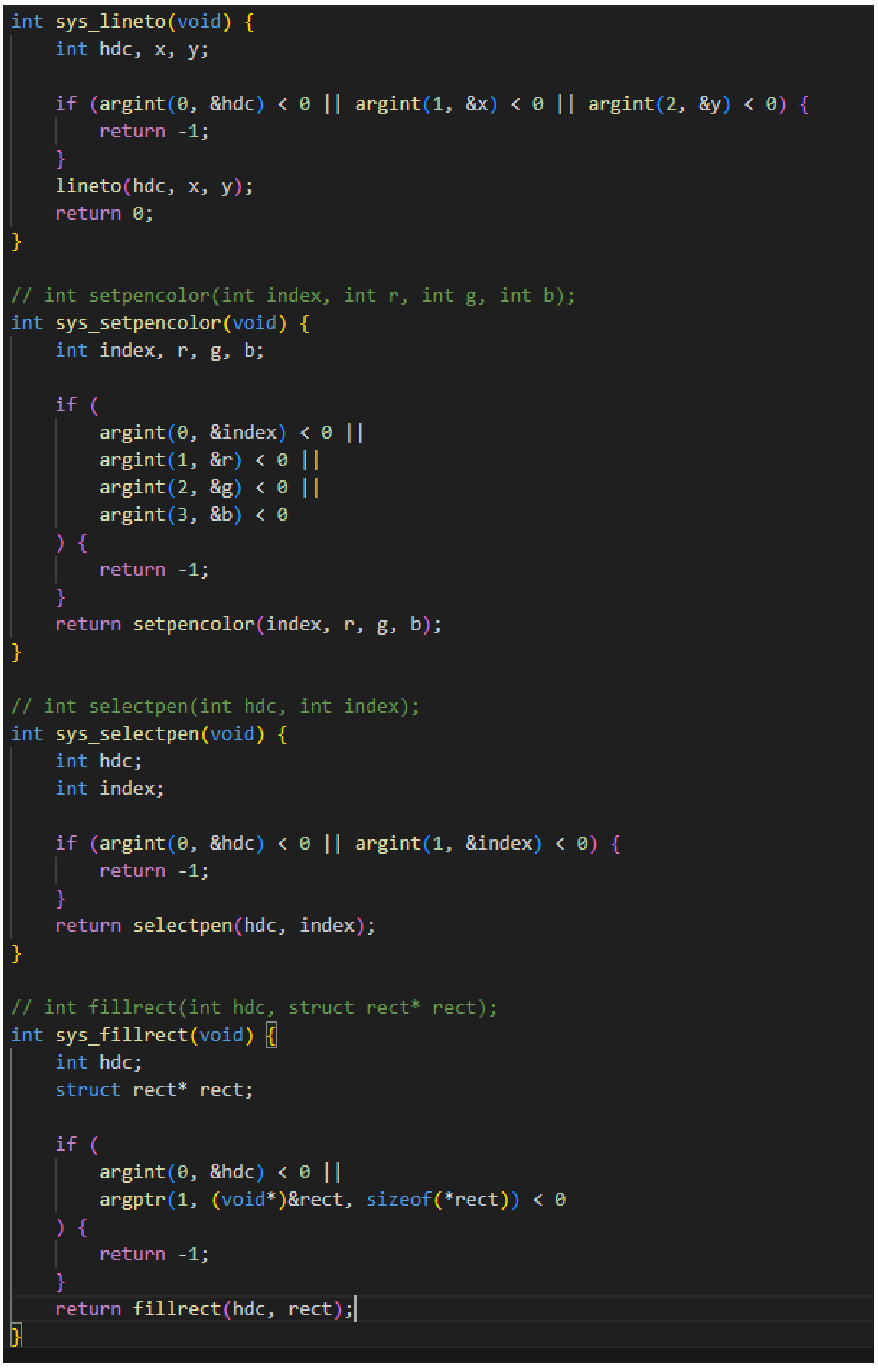
```
,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ astage3
fill_rect error: left is larger than right or top is larger than bottom:
    left = 200, right = 220, top = 100, bottom = 80
```

```
    int x2 = min(rect->right, GRAPHICS_WIDTH - 1);

    int y1 = max(rect->top, 0);
    int y2 = min(rect->bottom, GRAPHICS_HEIGHT - 1);

    // Make sure rect positions are valid
    if (x1 >= x2 || y1 >= y2) {
        cprintf("fill_rect error: left is larger than right or top is larger than bottom:\n");
        cprintf("    left = %d, right = %d, top = %d, bottom = %d\n", x1, x2, y1, y2);
        return -1;
    }
```



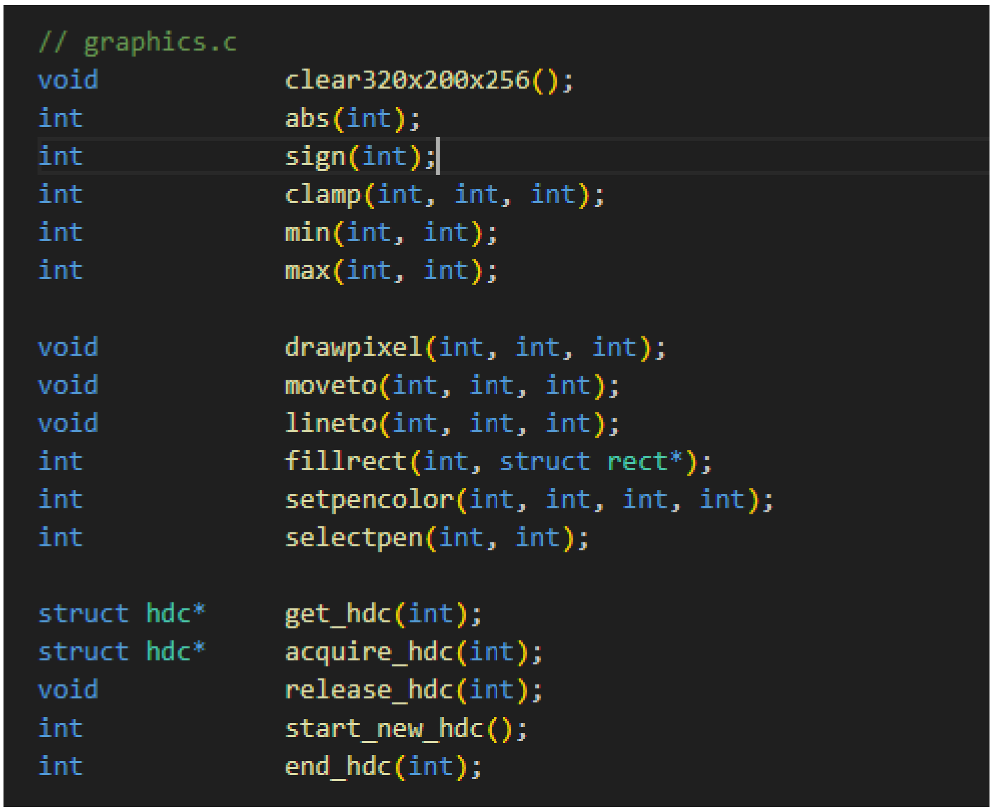
```
    to_draw.bottom = 190;
    to_draw.left = 80;
    to_draw.right = 140;
    fillrect(hdc, &to_draw);

    to_draw.top = 100;
    to_draw.bottom = 80;
    to_draw.left = 4500;
    to_draw.right = 5000;
    fillrect(hdc, &to_draw);

    getch();

    setpencolor(16, 10, 10, 10);
```

```
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ astage3
fill_rect error: left is larger than right or top is larger than bottom:
    left = 4500, right = 319, top = 100, bottom = 80
```

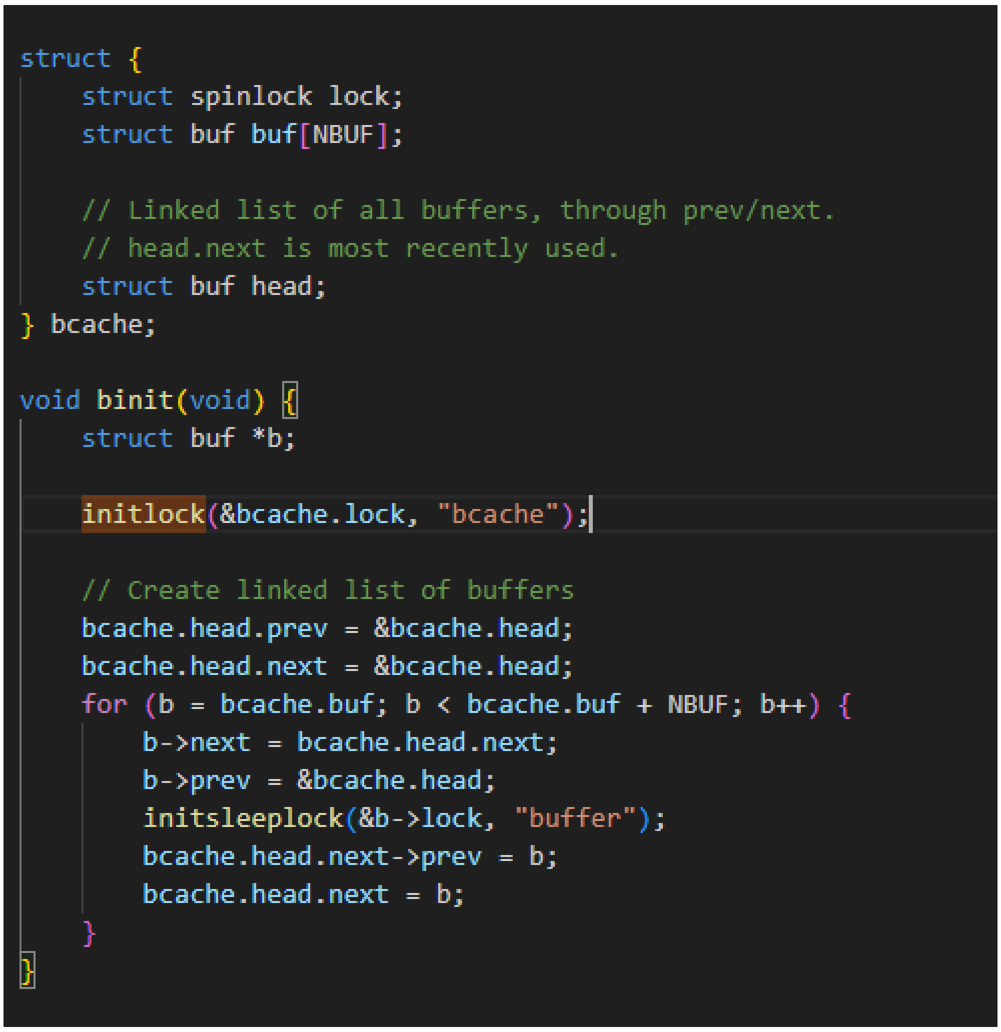


```
int sys_lineto(void) {
    int hdc, x, y;

    if (argint(0, &hdc) < 0 || argint(1, &x) < 0 || argint(2, &y) < 0) {
        return -1;
    }
    lineto(hdc, x, y);
    return 0;
}

// int setpencolor(int index, int r, int g, int b);
int sys_setpencolor(void) {
    int index, r, g, b;

    if (
        argint(0, &index) < 0 ||
        argint(1, &r) < 0 ||
        argint(2, &g) < 0 ||
        argint(3, &b) < 0
    ) {
        return -1;
    }
    return setpencolor(index, r, g, b);
}

// int selectpen(int hdc, int index);
int sys_selectpen(void) {
    int hdc;
    int index;

    if (argint(0, &hdc) < 0 || argint(1, &index) < 0) {
        return -1;
    }
    return selectpen(hdc, index);
}

// int fillrect(int hdc, struct rect* rect);
int sys_fillrect(void) {
    int hdc;
    struct rect* rect;

    if (
        argint(0, &hdc) < 0 ||
        argptr(1, (void*)&rect, sizeof(*rect)) < 0
    ) {
        return -1;
    }
    return fillrect(hdc, rect);
}
```

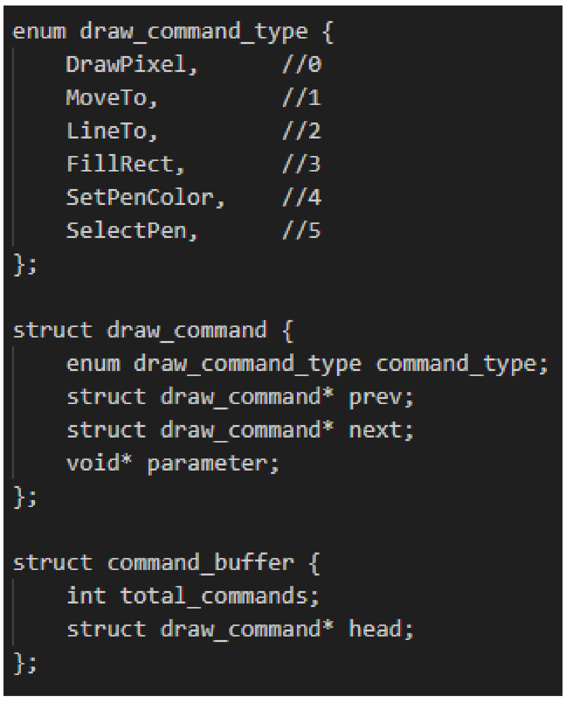
In preparation for queueing calls, I started to move inner functionality out of the system calls and into their own functions that could then be used by whatever need them, making sure to add them to defs.h.


```
// graphics.c
void            clear320x200x256();
int             abs(int);
int             sign(int);
int             clamp(int, int, int);
int             min(int, int);
int             max(int, int);

void            drawpixel(int, int, int);
void            moveto(int, int, int);
void            lineto(int, int, int);
int             fillrect(int, struct rect*);
int             setpencolor(int, int, int, int);
int             selectpen(int, int);

struct hdc*     get_hdc(int);
struct hdc*     acquire_hdc(int);
void            release_hdc(int);
int             start_new_hdc();
int             end_hdc(int);
```

For stage 4, I wanted to use linked lists as I'd seen the implementation of them in xv6 already and it seemed like a good idea.


```
struct {
    struct spinlock lock;
    struct buf buf[NBUF];

    // Linked list of all buffers, through prev/next.
    // head.next is most recently used.
    struct buf head;
} bcache;

void binit(void) {
    struct buf *b;

    initlock(&bcache.lock, "bcache");

    // Create linked list of buffers
    bcache.head.prev = &bcache.head;
    bcache.head.next = &bcache.head;
    for (b = bcache.buf; b < bcache.buf + NBUF; b++) {
        b->next = bcache.head.next;
        b->prev = &bcache.head;
        initsleeplock(&b->lock, "buffer");
        bcache.head.next->prev = b;
        bcache.head.next = b;
    }
}
```

05/12/23

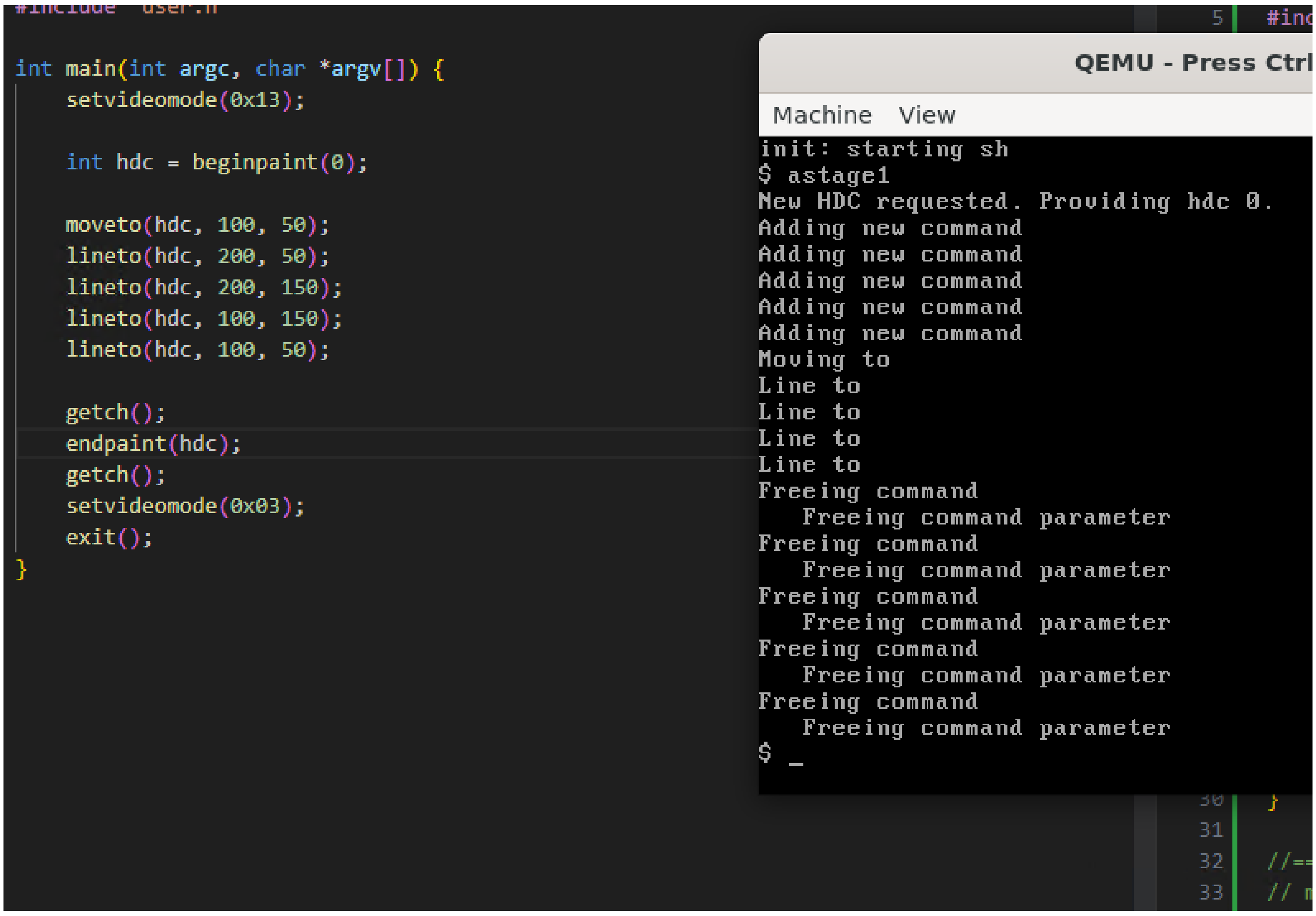


```c
enum draw_command_type {
    DrawPixel,      //0
    MoveTo,         //1
    LineTo,         //2
    FillRect,       //3
    SetPenColor,    //4
    SelectPen,      //5
};

struct draw_command {
    enum draw_command_type command_type;
    struct draw_command* prev;
    struct draw_command* next;
    void* parameter;
};

struct command_buffer {
    int total_commands;
    struct draw_command* head;
};
```

Started working on a linked list system for queueing commands. Calling begin paint and end paint would initialize and clear/submit the lists as needed. The system worked as I expected out the box with minimal changes needed.

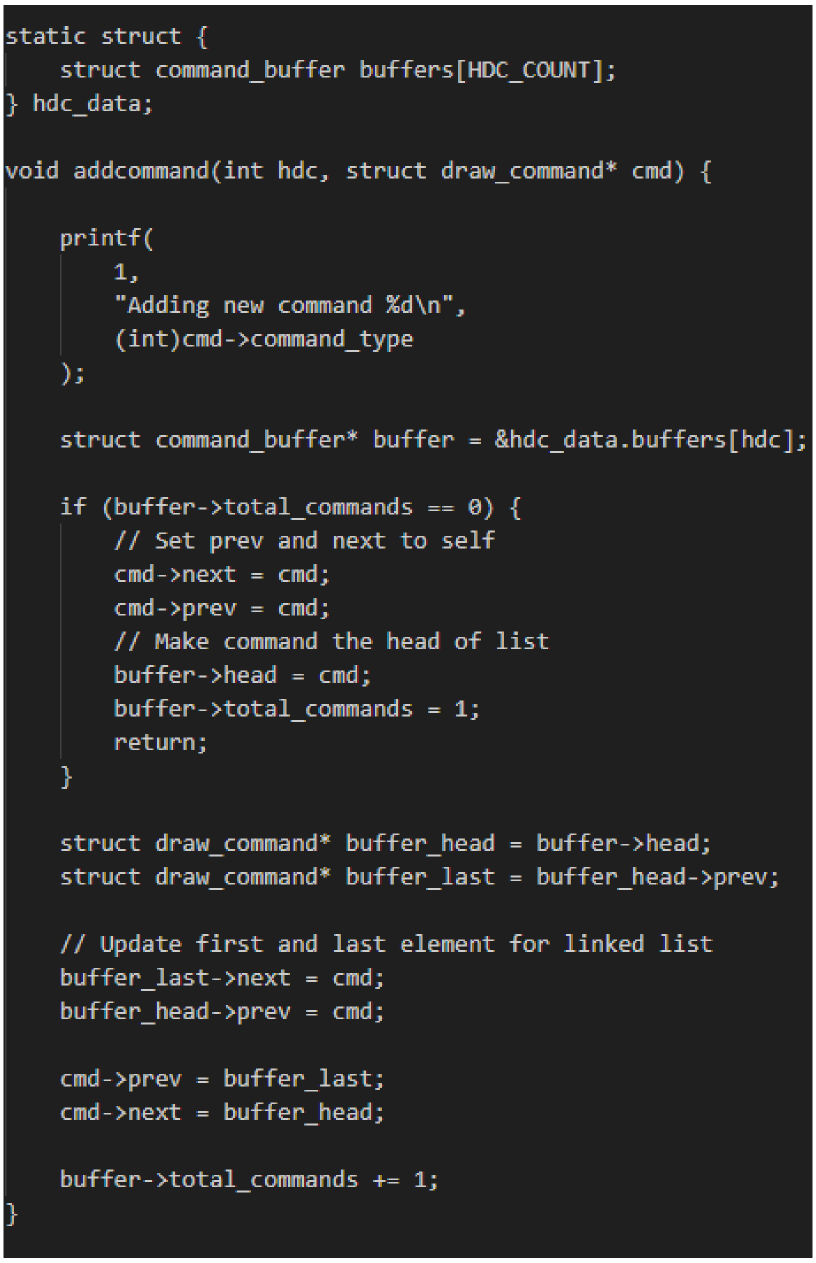
I started off with just outputting what the command submitted was in the kernel space when endpaint was called to make sure the correct commands were making their way in and in the right order before connecting them to the actual functions.

For all of this to work, I was using the malloc function to allocate draw_commands and again for their parameters which were then freed on endpaint. I could then just pass a reference to the command buffer through to the kernel on endpaint.



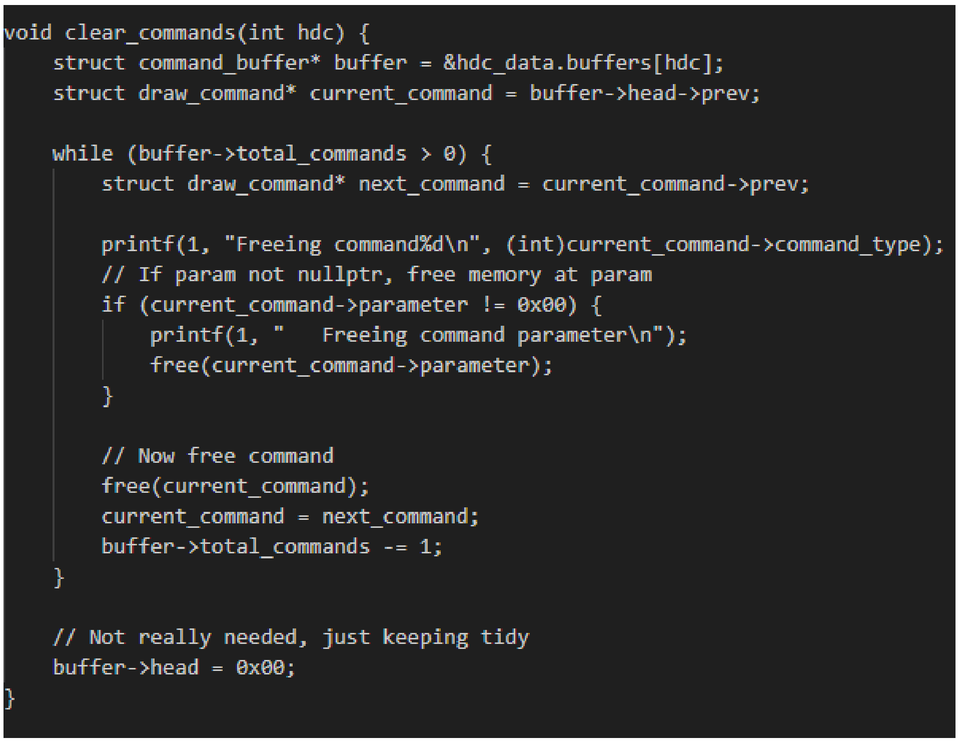
```c
#include "user.h"

int main(int argc, char *argv[]) {
    setvideomode(0x13);

    int hdc = beginpaint(0);

    moveto(hdc, 100, 50);
    lineto(hdc, 200, 50);
    lineto(hdc, 200, 150);
    lineto(hdc, 100, 150);
    lineto(hdc, 100, 50);

    getch();
    endpaint(hdc);
    getch();
    setvideomode(0x03);
    exit();
}
```

```
init: starting sh
$ astage1
New HDC requested. Providing hdc 0.
Adding new command
Adding new command
Adding new command
Adding new command
Adding new command
Moving to
Line to
Line to
Line to
Line to
Freeing command
    Freeing command parameter
Freeing command
    Freeing command parameter
Freeing command
    Freeing command parameter
Freeing command
    Freeing command parameter
Freeing command
    Freeing command parameter
$ _
```



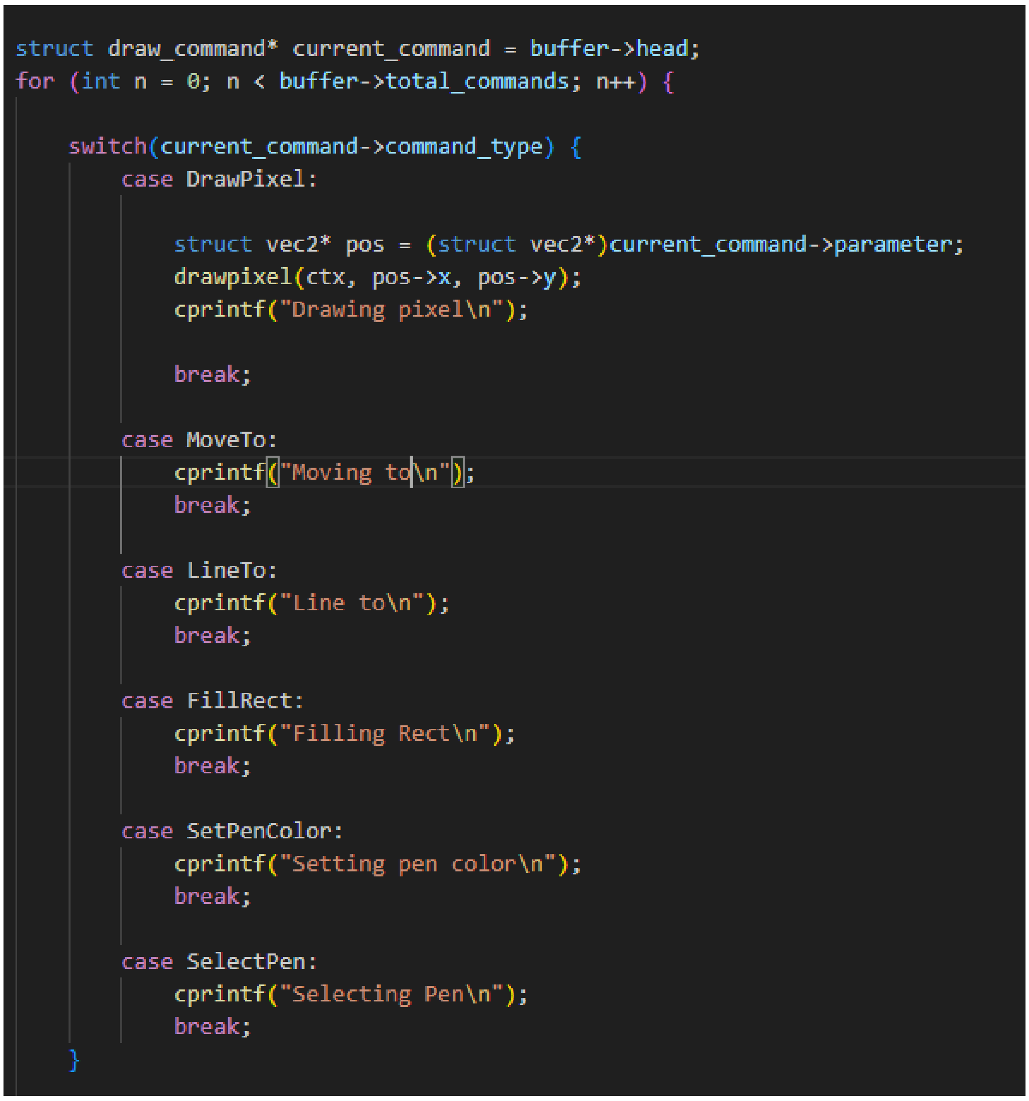
```c
static struct {
    struct command_buffer buffers[HDC_COUNT];
} hdc_data;

void addcommand(int hdc, struct draw_command* cmd) {

    printf(
        1,
        "Adding new command %d\n",
        (int)cmd->command_type
    );

    struct command_buffer* buffer = &hdc_data.buffers[hdc];

    if (buffer->total_commands == 0) {
        // Set prev and next to self
        cmd->next = cmd;
        cmd->prev = cmd;
        // Make command the head of list
        buffer->head = cmd;
        buffer->total_commands = 1;
        return;
    }

    struct draw_command* buffer_head = buffer->head;
    struct draw_command* buffer_last = buffer_head->prev;

    // Update first and last element for linked list
    buffer_last->next = cmd;
    buffer_head->prev = cmd;

    cmd->prev = buffer_last;
    cmd->next = buffer_head;

    buffer->total_commands += 1;
}
```

This is the inner functionality of adding and clearing commands. Commands are cleared in reverse order which is demonstrated later on.



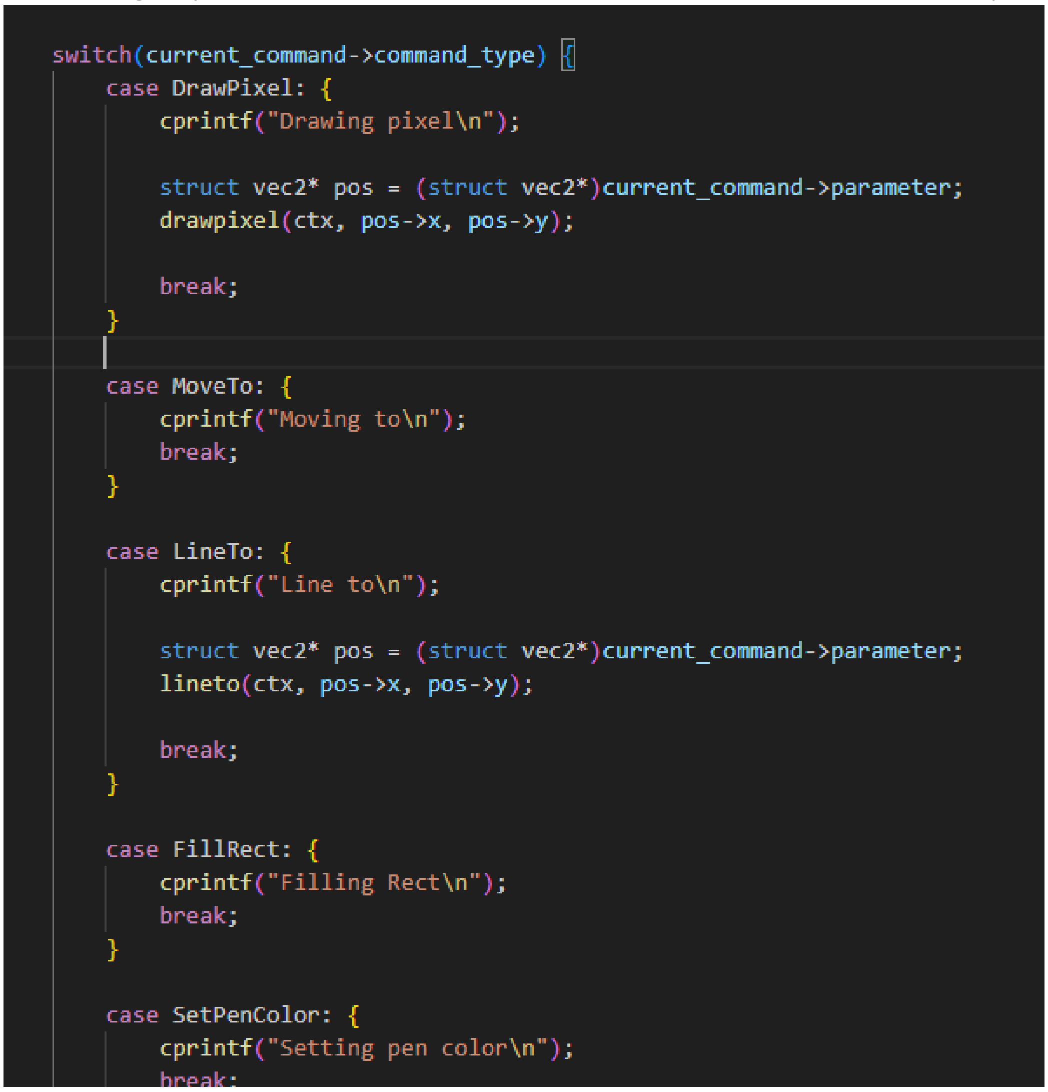
```c
void clear_commands(int hdc) {
    struct command_buffer* buffer = &hdc_data.buffers[hdc];
    struct draw_command* current_command = buffer->head->prev;

    while (buffer->total_commands > 0) {
        struct draw_command* next_command = current_command->prev;

        printf(1, "Freeing command%d\n", (int)current_command->command_type);
        // If param not nullptr, free memory at param
        if (current_command->parameter != 0x00) {
            printf(1, "   Freeing command parameter\n");
            free(current_command->parameter);
        }

        // Now free command
        free(current_command);
        current_command = next_command;
        buffer->total_commands -= 1;
    }

    // Not really needed, just keeping tidy
    buffer->head = 0x00;
}
```

Only draw pixel implemented.

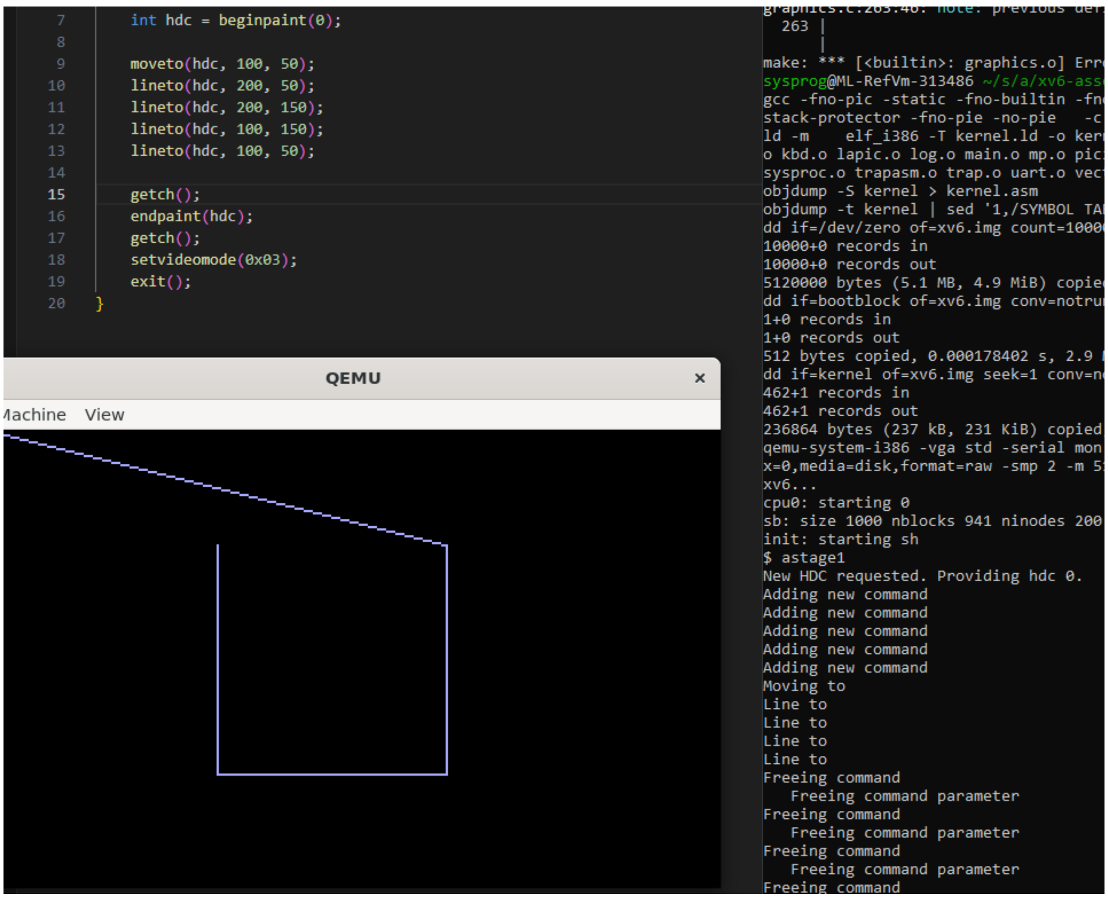


```c
struct draw_command* current_command = buffer->head;
for (int n = 0; n < buffer->total_commands; n++) {

    switch(current_command->command_type) {
        case DrawPixel:

            struct vec2* pos = (struct vec2*)current_command->parameter;
            drawpixel(ctx, pos->x, pos->y);
            cprintf("Drawing pixel\n");

            break;

        case MoveTo:
            cprintf("Moving to\n");
            break;

        case LineTo:
            cprintf("Line to\n");
            break;

        case FillRect:
            cprintf("Filling Rect\n");
            break;

        case SetPenColor:
            cprintf("Setting pen color\n");
            break;

        case SelectPen:
            cprintf("Selecting Pen\n");
            break;
    }
```

Now adding only DrawPixel and Line to which shows us a rect which starts in the top left despite calling moveto.



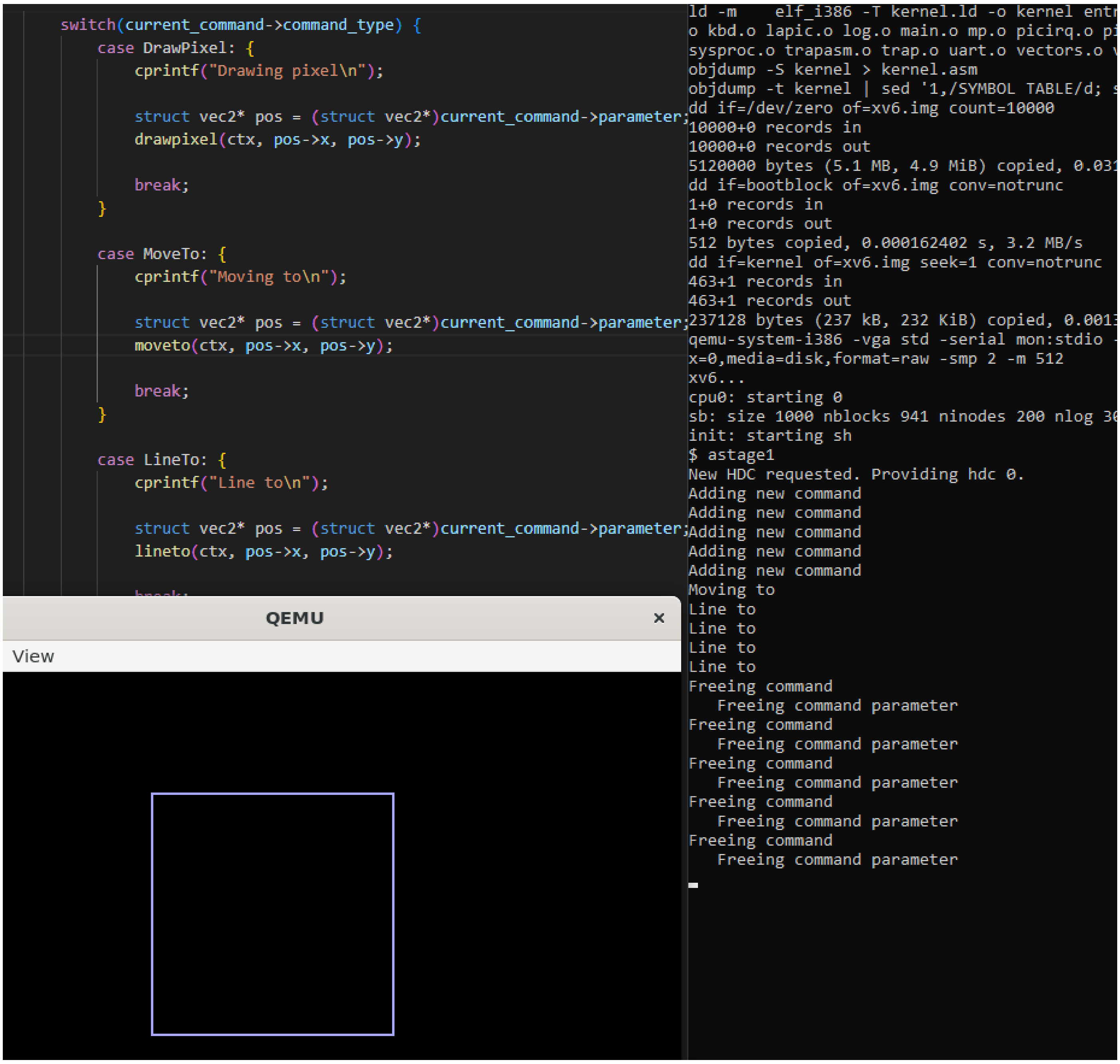
```c
switch(current_command->command_type) {
    case DrawPixel: {
        cprintf("Drawing pixel\n");

        struct vec2* pos = (struct vec2*)current_command->parameter;
        drawpixel(ctx, pos->x, pos->y);

        break;
    }

    case MoveTo: {
        cprintf("Moving to\n");
        break;
    }

    case LineTo: {
        cprintf("Line to\n");

        struct vec2* pos = (struct vec2*)current_command->parameter;
        lineto(ctx, pos->x, pos->y);

        break;
    }

    case FillRect: {
        cprintf("Filling Rect\n");
        break;
    }

    case SetPenColor: {
        cprintf("Setting pen color\n");
        break;
```



```c
    int hdc = beginpaint(0);

    moveto(hdc, 100, 50);
    lineto(hdc, 200, 50);
    lineto(hdc, 200, 150);
    lineto(hdc, 100, 150);
    lineto(hdc, 100, 50);

    getch();
    endpaint(hdc);
    getch();
    setvideomode(0x03);
    exit();
}
```

```
graphics.c:263:46: note: previous def
  263 |
      |
make: *** [<builtin>: graphics.o] Err
sysprog@ML-RefVm-313486 ~/s/a/xv6-ass
gcc -fno-pic -static -fno-builtin -fn
stack-protector -fno-pie -no-pie   -c
ld -m    elf_i386 -T kernel.ld -o ker
o kbd.o lapic.o log.o main.o mp.o pic
sysproc.o trapasm.o trap.o uart.o vec
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TA
dd if=/dev/zero of=xv6.img count=1000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copie
dd if=bootblock of=xv6.img conv=notru
1+0 records in
1+0 records out
512 bytes copied, 0.000178402 s, 2.9
dd if=kernel of=xv6.img seek=1 conv=n
462+1 records in
462+1 records out
236864 bytes (237 kB, 231 KiB) copied
qemu-system-i386 -vga std -serial mon
x=0,media=disk,format=raw -smp 2 -m 5
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200
init: starting sh
$ astage1
New HDC requested. Providing hdc 0.
Adding new command
Adding new command
Adding new command
Adding new command
Adding new command
Moving to
Line to
Line to
Line to
Line to
Freeing command
   Freeing command parameter
Freeing command
   Freeing command parameter
Freeing command
   Freeing command parameter
Freeing command
```

And then again after adding moveto.

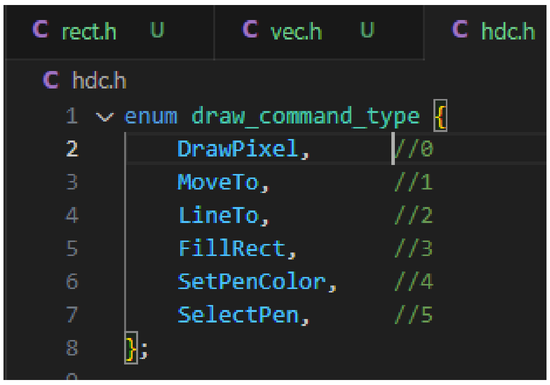


```
switch(current_command->command_type) {
    case DrawPixel: {
        cprintf("Drawing pixel\n");

        struct vec2* pos = (struct vec2*)current_command->parameter;
        drawpixel(ctx, pos->x, pos->y);

        break;
    }

    case MoveTo: {
        cprintf("Moving to\n");

        struct vec2* pos = (struct vec2*)current_command->parameter;
        moveto(ctx, pos->x, pos->y);

        break;
    }

    case LineTo: {
        cprintf("Line to\n");

        struct vec2* pos = (struct vec2*)current_command->parameter;
        lineto(ctx, pos->x, pos->y);

        break;
```

```
ld -m    elf_i386 -T kernel.ld -o kernel entr
o kbd.o lapic.o log.o main.o mp.o picirq.o pi
sysproc.o trapasm.o trap.o uart.o vectors.o v
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.031
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000162402 s, 3.2 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
463+1 records in
463+1 records out
237128 bytes (237 kB, 232 KiB) copied, 0.0013
qemu-system-i386 -vga std -serial mon:stdio -
x=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30
init: starting sh
$ astage1
New HDC requested. Providing hdc 0.
Adding new command
Adding new command
Adding new command
Adding new command
Adding new command
Moving to
Line to
Line to
Line to
Line to
Freeing command
   Freeing command parameter
Freeing command
   Freeing command parameter
Freeing command
   Freeing command parameter
Freeing command
   Freeing command parameter
Freeing command
   Freeing command parameter
```

08/12/23

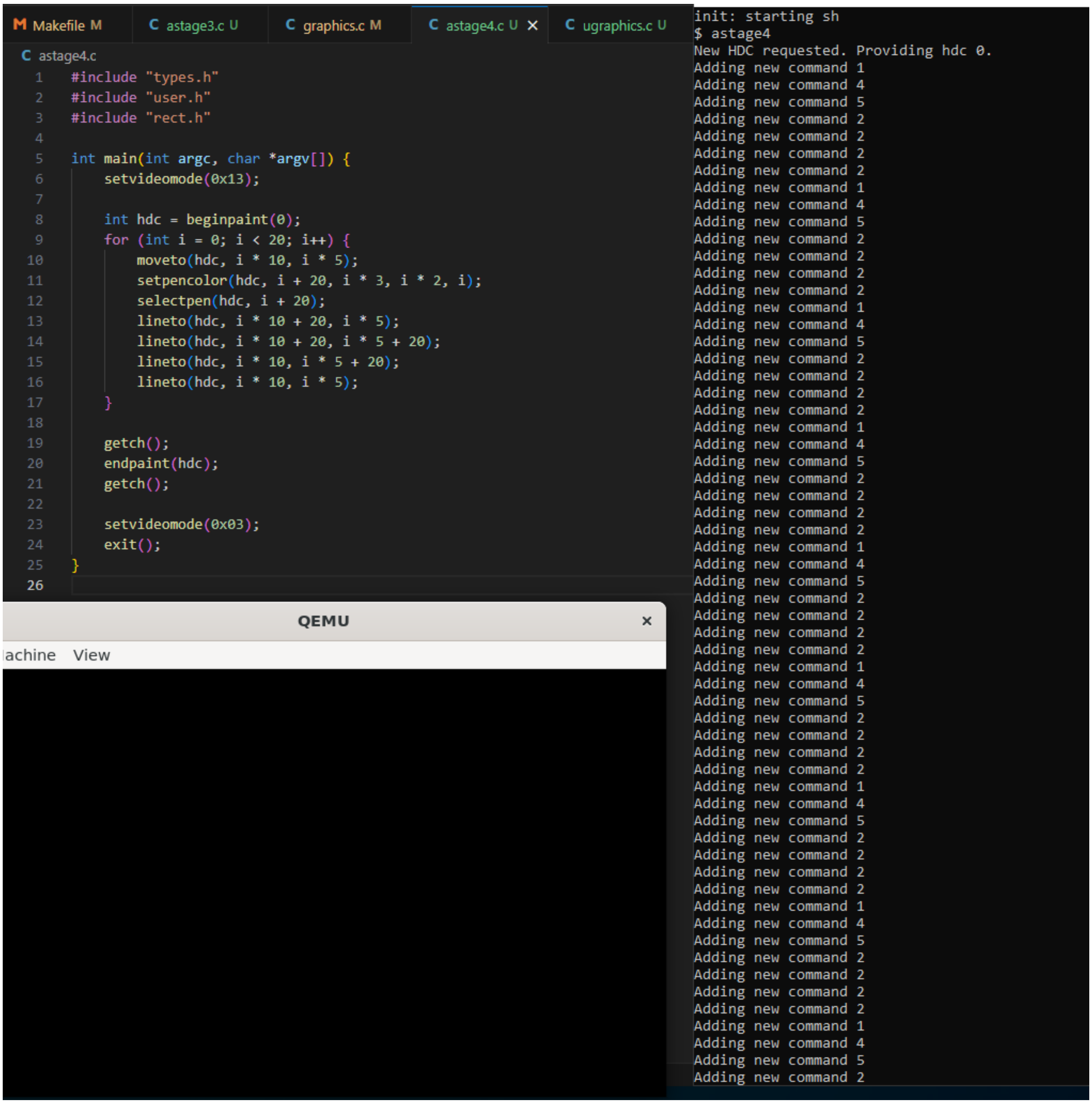
I implemented your stage4 example to test my code with. I needed to implement setpencolor and selectpen with the new system for these to work. I also added more print commands to display what command type was added before we call endpaint and also displayed what type of command was being freed afterwards as well which can be seen in the following screenshots.



```c
enum draw_command_type {
    DrawPixel,      //0
    MoveTo,         //1
    LineTo,         //2
    FillRect,       //3
    SetPenColor,    //4
    SelectPen,      //5
};
```

One of the main problems I have with this being able to queue setpencolor commands requires that the function takes a hdc parameter which in the brief, it doesn't. This is a huge problem as without this, there is no way to queue this command to be executed in bulk later. For now, I opted to just give setpencolor a hdc parameter despite it not having one in the brief. In future, this is no longer how this is implemented.

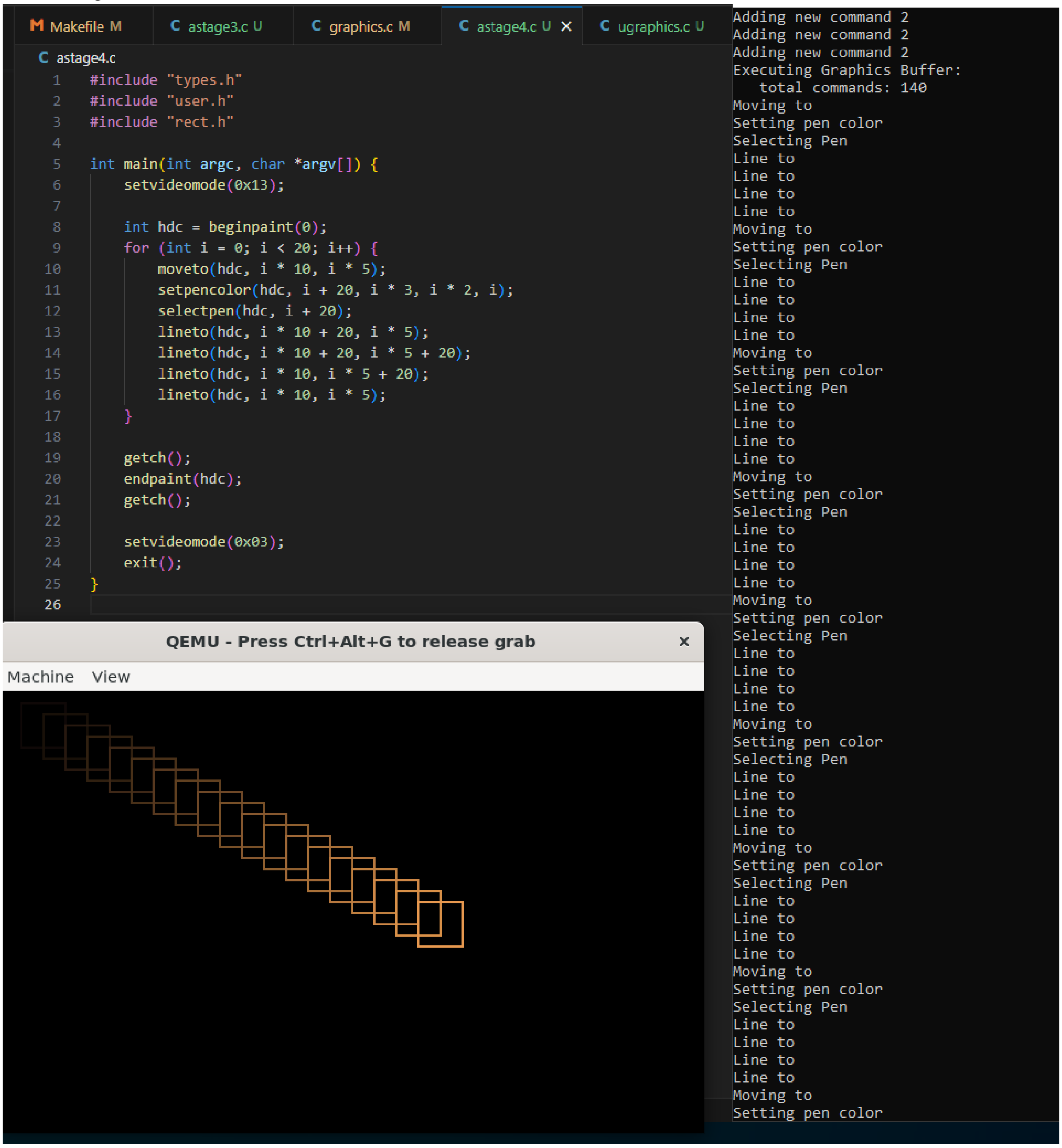
Before first getch:



```c
#include "types.h"
#include "user.h"
#include "rect.h"

int main(int argc, char *argv[]) {
    setvideomode(0x13);

    int hdc = beginpaint(0);
    for (int i = 0; i < 20; i++) {
        moveto(hdc, i * 10, i * 5);
        setpencolor(hdc, i + 20, i * 3, i * 2, i);
        selectpen(hdc, i + 20);
        lineto(hdc, i * 10 + 20, i * 5);
        lineto(hdc, i * 10 + 20, i * 5 + 20);
        lineto(hdc, i * 10, i * 5 + 20);
        lineto(hdc, i * 10, i * 5);
    }

    getch();
    endpaint(hdc);
    getch();

    setvideomode(0x03);
    exit();
}
```

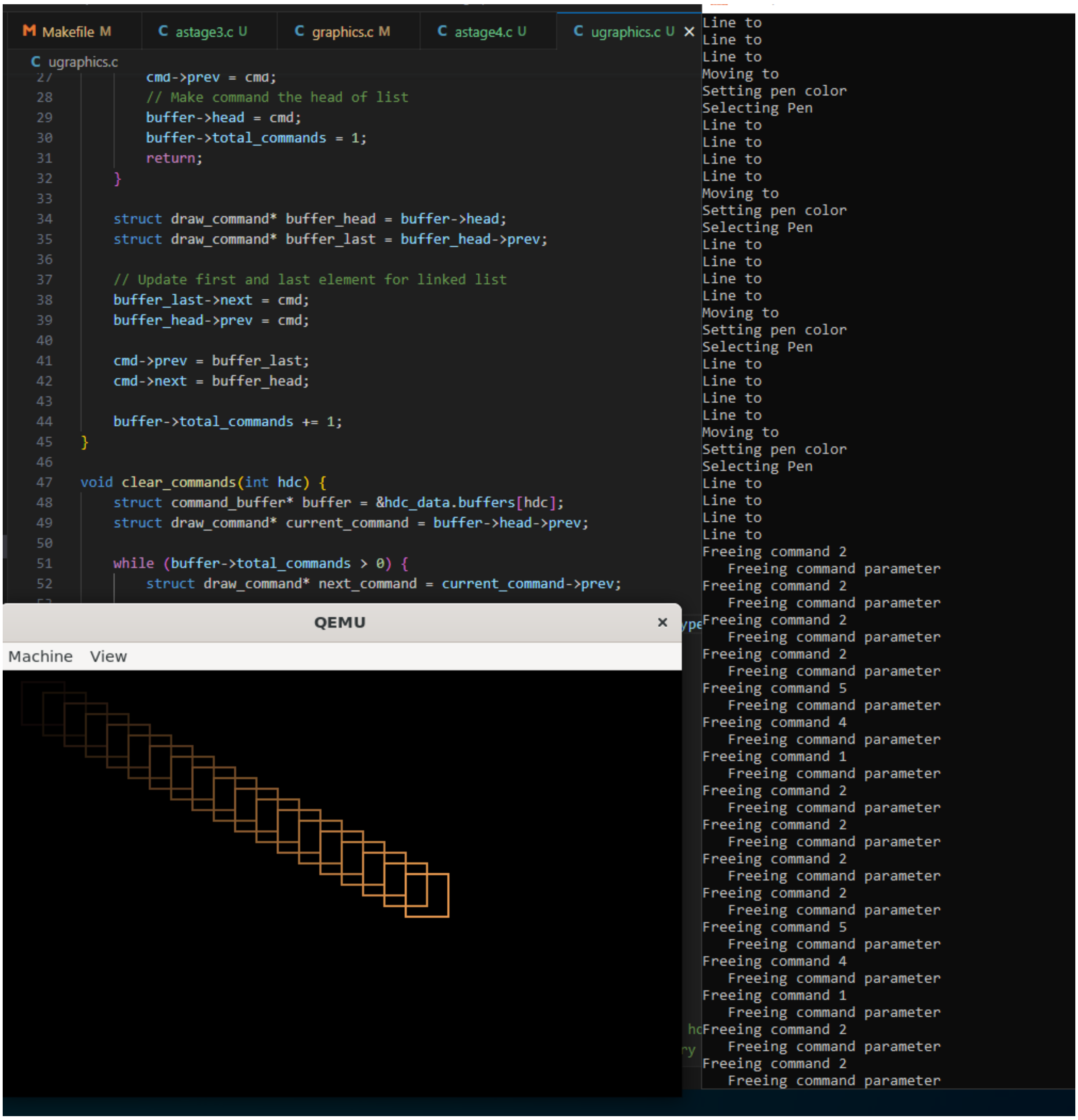
```
init: starting sh
$ astage4
New HDC requested. Providing hdc 0.
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
```

After first getch



```c
#include "types.h"
#include "user.h"
#include "rect.h"

int main(int argc, char *argv[]) {
    setvideomode(0x13);

    int hdc = beginpaint(0);
    for (int i = 0; i < 20; i++) {
        moveto(hdc, i * 10, i * 5);
        setpencolor(hdc, i + 20, i * 3, i * 2, i);
        selectpen(hdc, i + 20);
        lineto(hdc, i * 10 + 20, i * 5);
        lineto(hdc, i * 10 + 20, i * 5 + 20);
        lineto(hdc, i * 10, i * 5 + 20);
        lineto(hdc, i * 10, i * 5);
    }

    getch();
    endpaint(hdc);
    getch();

    setvideomode(0x03);
    exit();
}

```

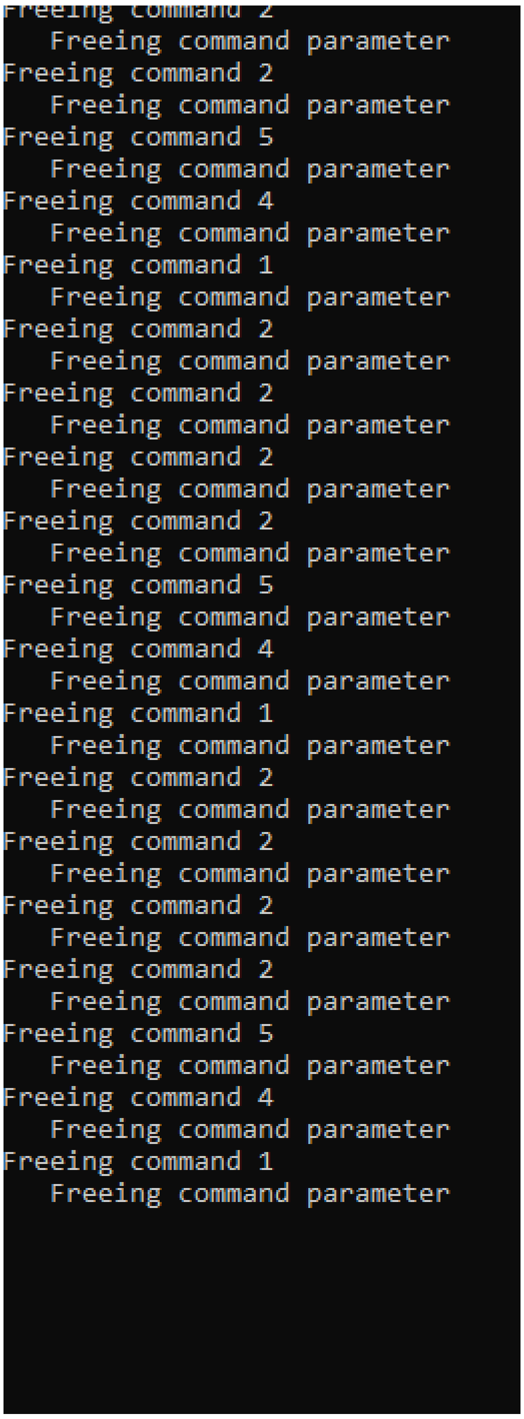
```
Adding new command 2
Adding new command 2
Adding new command 2
Executing Graphics Buffer:
    total commands: 140
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Moving to
Setting pen color
```



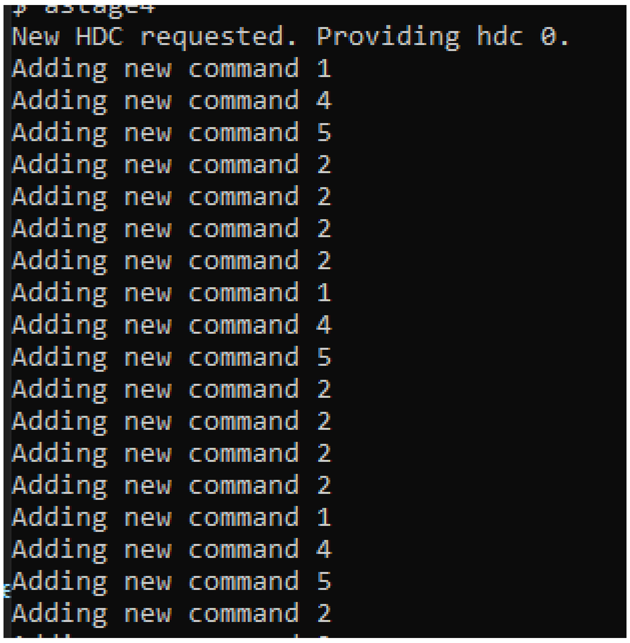
```c
        cmd->prev = cmd;
        // Make command the head of list
        buffer->head = cmd;
        buffer->total_commands = 1;
        return;
    }

    struct draw_command* buffer_head = buffer->head;
    struct draw_command* buffer_last = buffer_head->prev;

    // Update first and last element for linked list
    buffer_last->next = cmd;
    buffer_head->prev = cmd;

    cmd->prev = buffer_last;
    cmd->next = buffer_head;

    buffer->total_commands += 1;
}

void clear_commands(int hdc) {
    struct command_buffer* buffer = &hdc_data.buffers[hdc];
    struct draw_command* current_command = buffer->head->prev;

    while (buffer->total_commands > 0) {
        struct draw_command* next_command = current_command->prev;
```

```
Line to
Line to
Line to
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Moving to
Setting pen color
Selecting Pen
Line to
Line to
Line to
Line to
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 5
   Freeing command parameter
Freeing command 4
   Freeing command parameter
Freeing command 1
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 5
   Freeing command parameter
Freeing command 4
   Freeing command parameter
Freeing command 1
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
```

The commands are freed in reverse order. This is confirmed by looking at the top and bottom of the output, before and after endpaint.

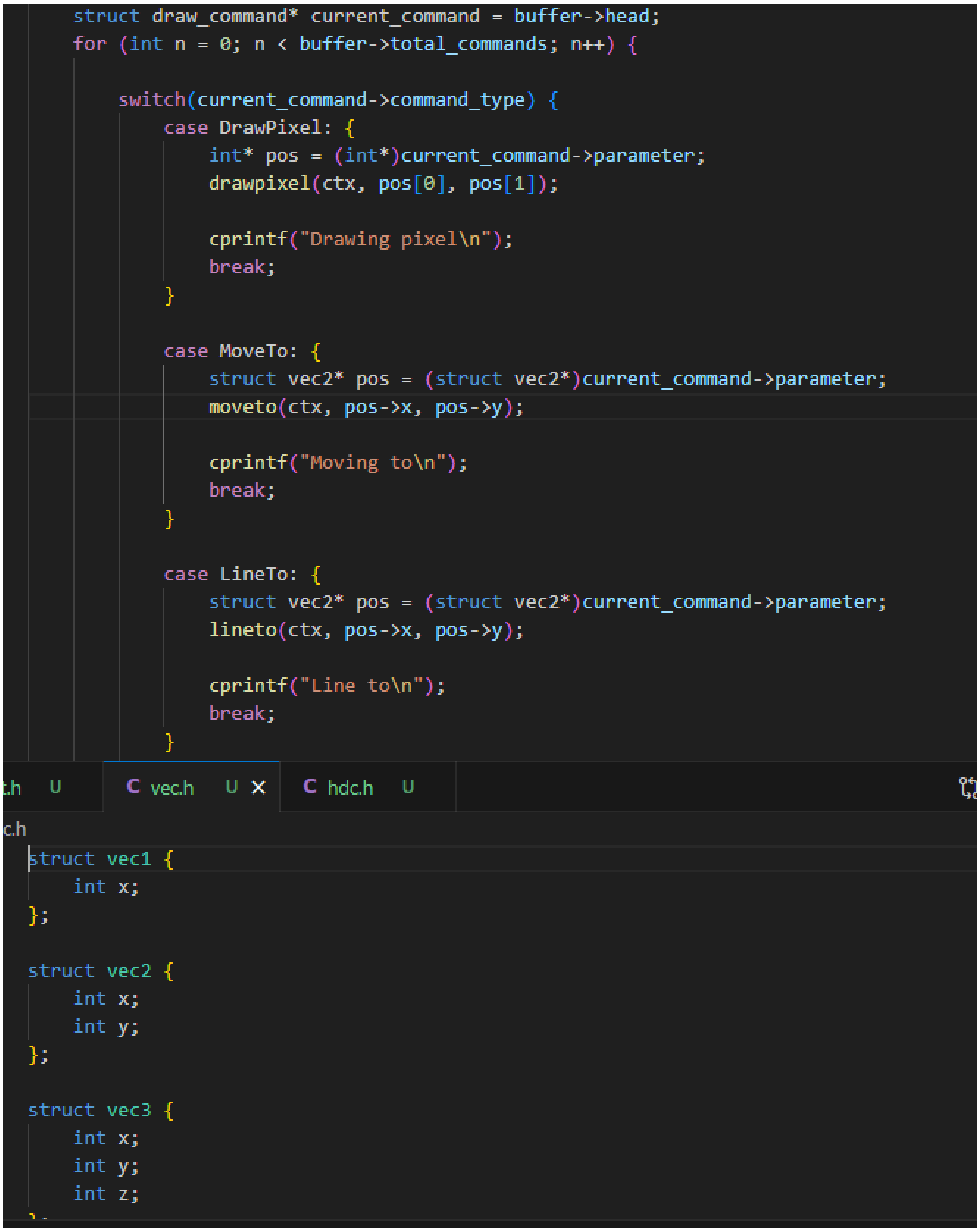


```
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 5
   Freeing command parameter
Freeing command 4
   Freeing command parameter
Freeing command 1
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 5
   Freeing command parameter
Freeing command 4
   Freeing command parameter
Freeing command 1
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 2
   Freeing command parameter
Freeing command 5
   Freeing command parameter
Freeing command 4
   Freeing command parameter
Freeing command 1
   Freeing command parameter
```



```
$ astage4
New HDC requested. Providing hdc 0.
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 2
Adding new command 1
Adding new command 4
Adding new command 5
Adding new command 2
```

I was previously using lots of vec structs to store this info. I kept needing to add more types to accommodate different commands so eventually ditched them in favour of int* arrays. See DrawPixel in the screenshot below. It makes things a little harder to debug but makes creation a lot easier and removes the need for lots of different structs if ints are all that's needed.

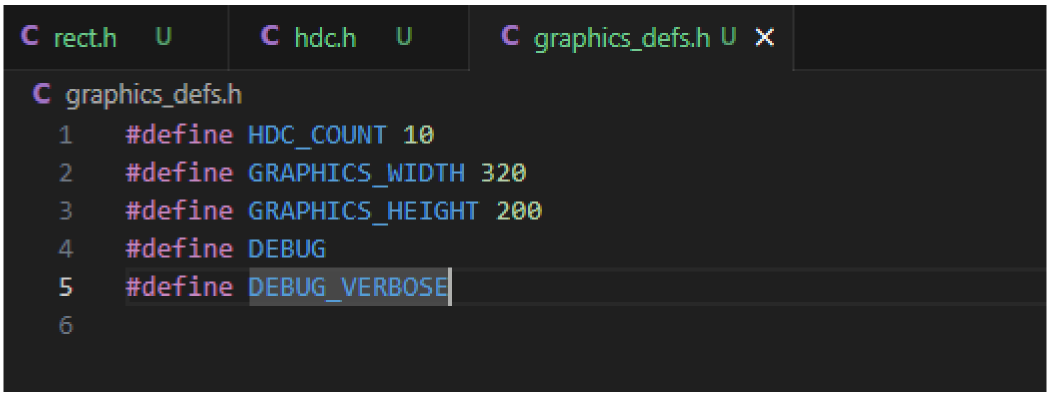


```c
    struct draw_command* current_command = buffer->head;
    for (int n = 0; n < buffer->total_commands; n++) {

        switch(current_command->command_type) {
            case DrawPixel: {
                int* pos = (int*)current_command->parameter;
                drawpixel(ctx, pos[0], pos[1]);

                cprintf("Drawing pixel\n");
                break;
            }

            case MoveTo: {
                struct vec2* pos = (struct vec2*)current_command->parameter;
                moveto(ctx, pos->x, pos->y);

                cprintf("Moving to\n");
                break;
            }

            case LineTo: {
                struct vec2* pos = (struct vec2*)current_command->parameter;
                lineto(ctx, pos->x, pos->y);

                cprintf("Line to\n");
                break;
            }
```

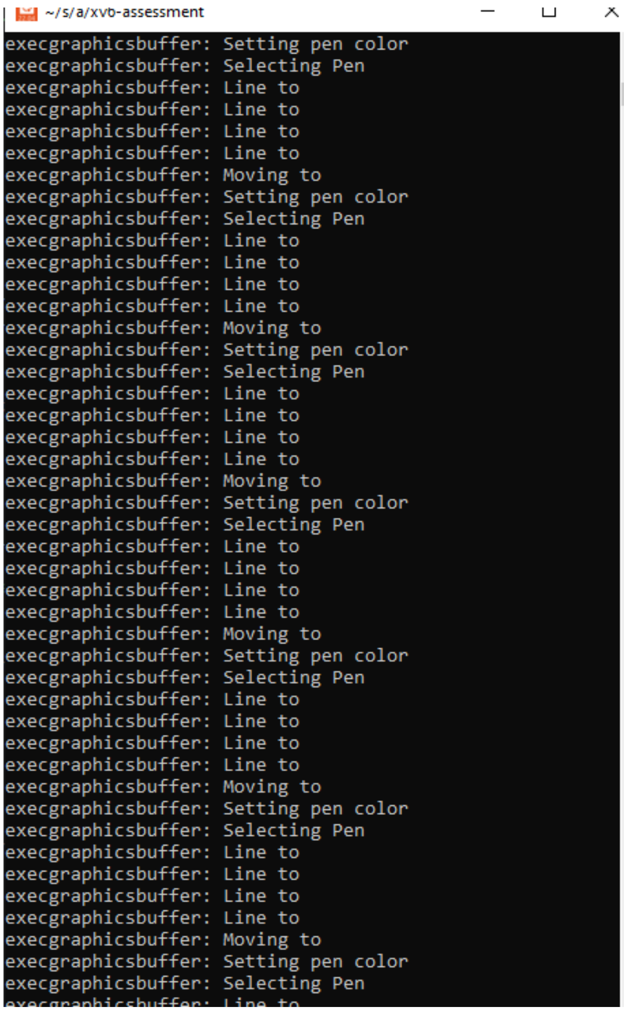
```c
struct vec1 {
    int x;
};

struct vec2 {
    int x;
    int y;
};

struct vec3 {
    int x;
    int y;
    int z;
```

I wanted to add better debugging as I didn't always want lots of text being thrown at the terminal. As such I added two debugging levels as definitions that could be commented/uncommented to change whether debugging functionality was compiled. I went through my code an updated most of the print statements appropriately.

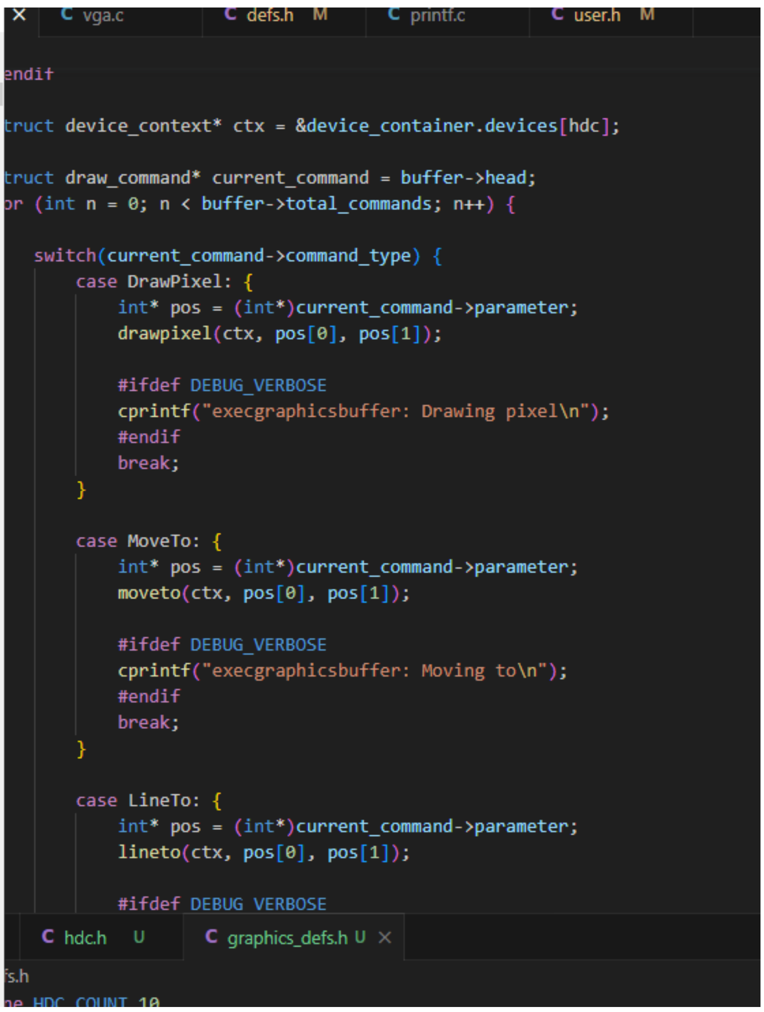
This is running the stage4 code with both debug and debug_verbose enabled:



```c
#define HDC_COUNT 10
#define GRAPHICS_WIDTH 320
#define GRAPHICS_HEIGHT 200
#define DEBUG
#define DEBUG_VERBOSE

```

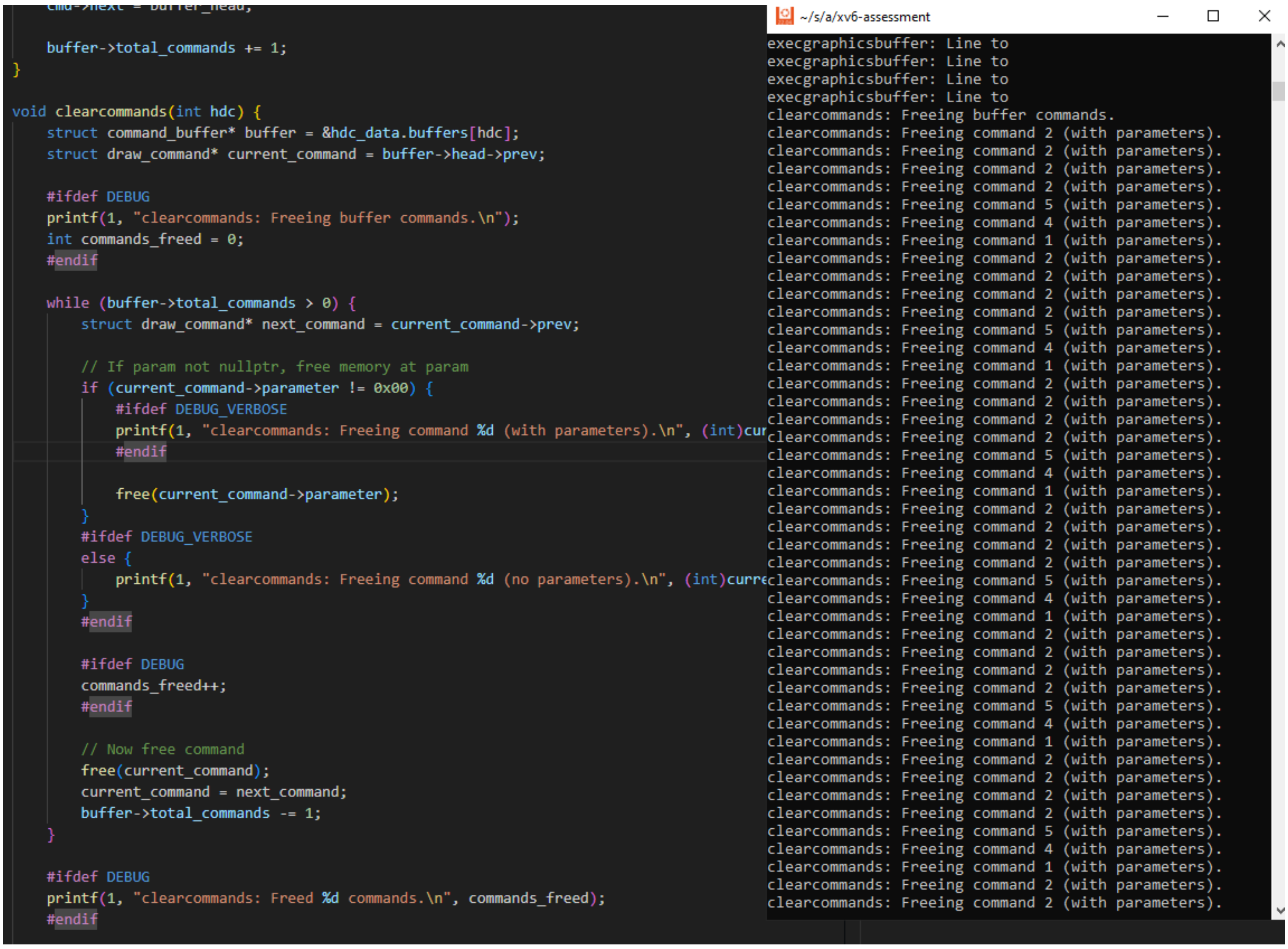


```
execgraphicsbuffer: Setting pen color
execgraphicsbuffer: Selecting Pen
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Moving to
execgraphicsbuffer: Setting pen color
execgraphicsbuffer: Selecting Pen
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Moving to
execgraphicsbuffer: Setting pen color
execgraphicsbuffer: Selecting Pen
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Moving to
execgraphicsbuffer: Setting pen color
execgraphicsbuffer: Selecting Pen
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Moving to
execgraphicsbuffer: Setting pen color
execgraphicsbuffer: Selecting Pen
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Moving to
execgraphicsbuffer: Setting pen color
execgraphicsbuffer: Selecting Pen
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Moving to
execgraphicsbuffer: Setting pen color
execgraphicsbuffer: Selecting Pen
execgraphicsbuffer: Line to
```



```c
endif

truct device_context* ctx = &device_container.devices[hdc];

truct draw_command* current_command = buffer->head;
or (int n = 0; n < buffer->total_commands; n++) {

    switch(current_command->command_type) {
        case DrawPixel: {
            int* pos = (int*)current_command->parameter;
            drawpixel(ctx, pos[0], pos[1]);

            #ifdef DEBUG_VERBOSE
            cprintf("execgraphicsbuffer: Drawing pixel\n");
            #endif
            break;
        }

        case MoveTo: {
            int* pos = (int*)current_command->parameter;
            moveto(ctx, pos[0], pos[1]);

            #ifdef DEBUG_VERBOSE
            cprintf("execgraphicsbuffer: Moving to\n");
            #endif
            break;
        }

        case LineTo: {
            int* pos = (int*)current_command->parameter;
            lineto(ctx, pos[0], pos[1]);

            #ifdef DEBUG_VERBOSE
```

```c
ne HDC_COUNT 10
```

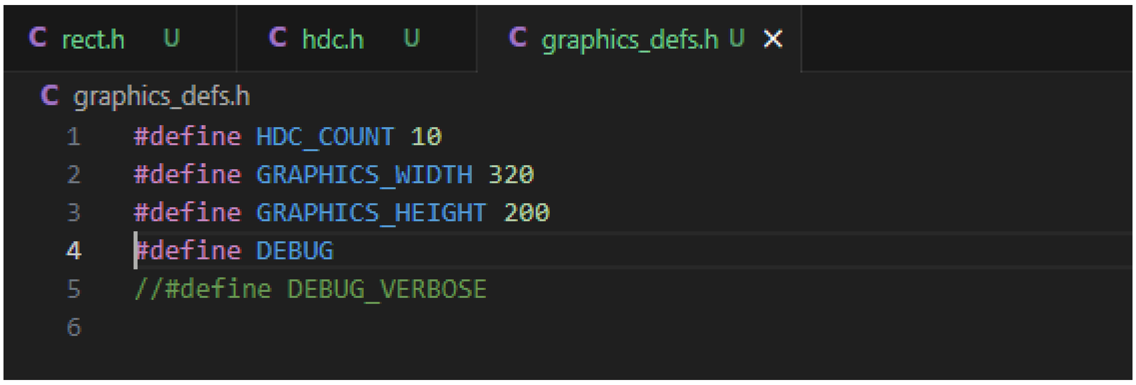


```c
    cmd->next = buffer_head;

    buffer->total_commands += 1;
}

void clearcommands(int hdc) {
    struct command_buffer* buffer = &hdc_data.buffers[hdc];
    struct draw_command* current_command = buffer->head->prev;

    #ifdef DEBUG
    printf(1, "clearcommands: Freeing buffer commands.\n");
    int commands_freed = 0;
    #endif

    while (buffer->total_commands > 0) {
        struct draw_command* next_command = current_command->prev;

        // If param not nullptr, free memory at param
        if (current_command->parameter != 0x00) {
            #ifdef DEBUG_VERBOSE
            printf(1, "clearcommands: Freeing command %d (with parameters).\n", (int)cur
            #endif

            free(current_command->parameter);
        }
        #ifdef DEBUG_VERBOSE
        else {
            printf(1, "clearcommands: Freeing command %d (no parameters).\n", (int)curre
        }
        #endif

        #ifdef DEBUG
        commands_freed++;
        #endif

        // Now free command
        free(current_command);
        current_command = next_command;
        buffer->total_commands -= 1;
    }

    #ifdef DEBUG
    printf(1, "clearcommands: Freed %d commands.\n", commands_freed);
    #endif
```

~/s/a/xv6-assessment

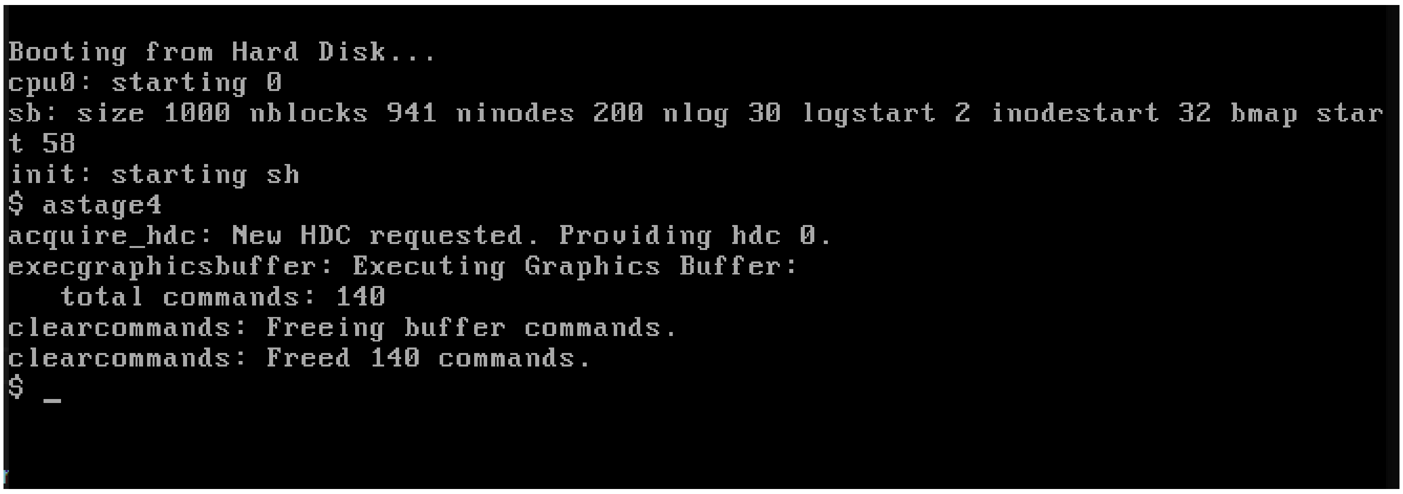
```
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
execgraphicsbuffer: Line to
clearcommands: Freeing buffer commands.
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 5 (with parameters).
clearcommands: Freeing command 4 (with parameters).
clearcommands: Freeing command 1 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 5 (with parameters).
clearcommands: Freeing command 4 (with parameters).
clearcommands: Freeing command 1 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 5 (with parameters).
clearcommands: Freeing command 4 (with parameters).
clearcommands: Freeing command 1 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 5 (with parameters).
clearcommands: Freeing command 4 (with parameters).
clearcommands: Freeing command 1 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 5 (with parameters).
clearcommands: Freeing command 4 (with parameters).
clearcommands: Freeing command 1 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 5 (with parameters).
clearcommands: Freeing command 4 (with parameters).
clearcommands: Freeing command 1 (with parameters).
clearcommands: Freeing command 2 (with parameters).
clearcommands: Freeing command 2 (with parameters).
```

This is with only debug enabled:



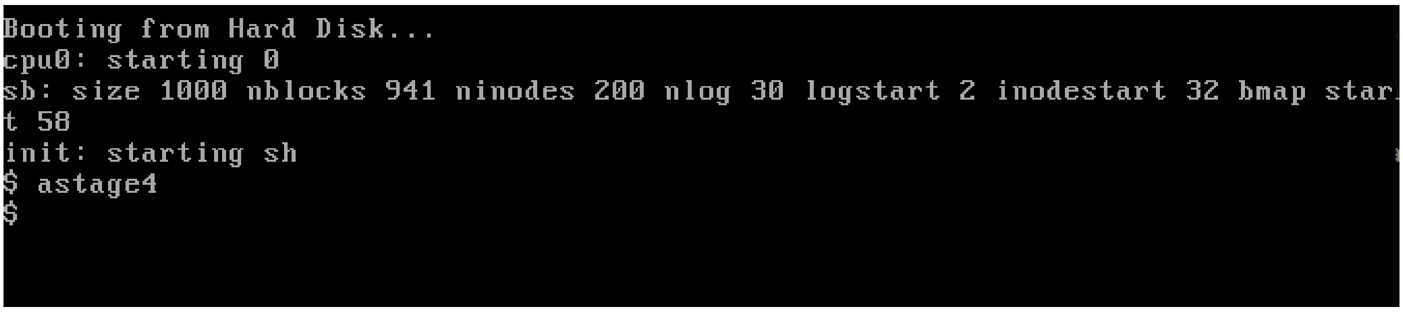
graphics_defs.h

```c
#define HDC_COUNT 10
#define GRAPHICS_WIDTH 320
#define GRAPHICS_HEIGHT 200
#define DEBUG
//#define DEBUG_VERBOSE
```

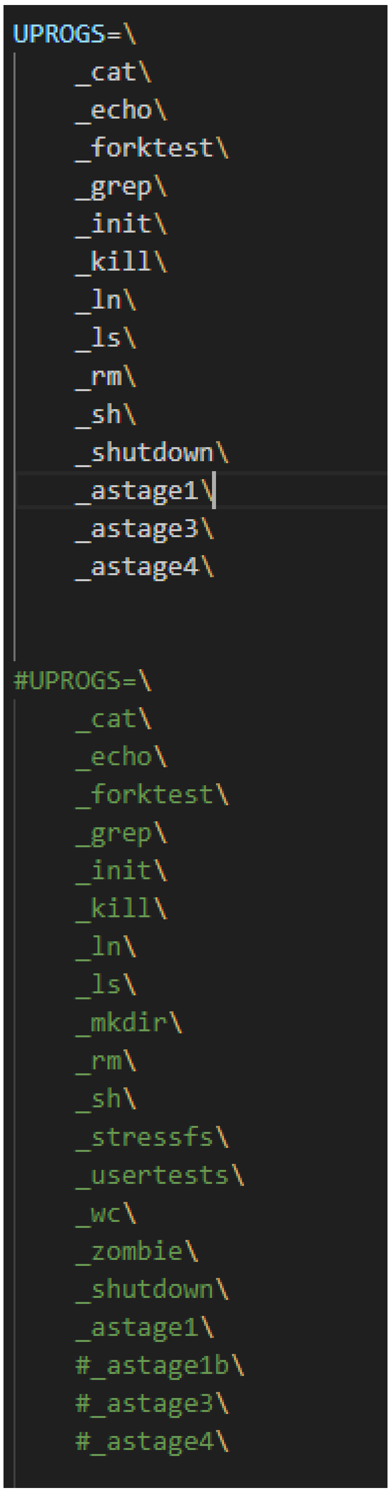


```
Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ astage4
acquire_hdc: New HDC requested. Providing hdc 0.
execgraphicsbuffer: Executing Graphics Buffer:
    total commands: 140
clearcommands: Freeing buffer commands.
clearcommands: Freed 140 commands.
$ _
```

And finally with neither enabled:



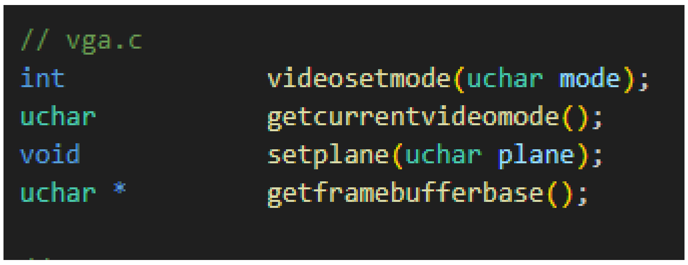
```
Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ astage4
$
```



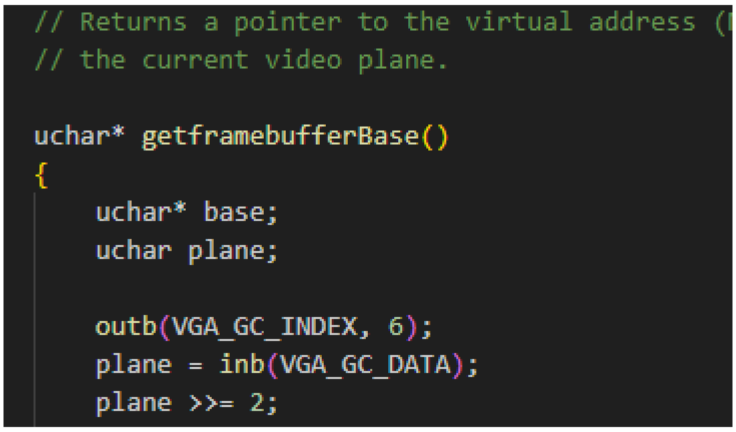
```
UPROGS=\
	_cat\
	_echo\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_rm\
	_sh\
	_shutdown\
	_astage1\
	_astage3\
	_astage4\


#UPROGS=\
	_cat\
	_echo\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_zombie\
	_shutdown\
	_astage1\
	#_astage1b\
	#_astage3\
	#_astage4\
```

At this point I was having problems compiling xv6 as it wouldn’t always compile, giving me errors regarding an assert to do with blocks used/available. I was informed that removing user programs would resolve this for now, so I removed a bunch of the existing ones that weren’t in use:

Moving onto stage 5, I started by changing the function in vga.c as it didn't match the definition in defs.h

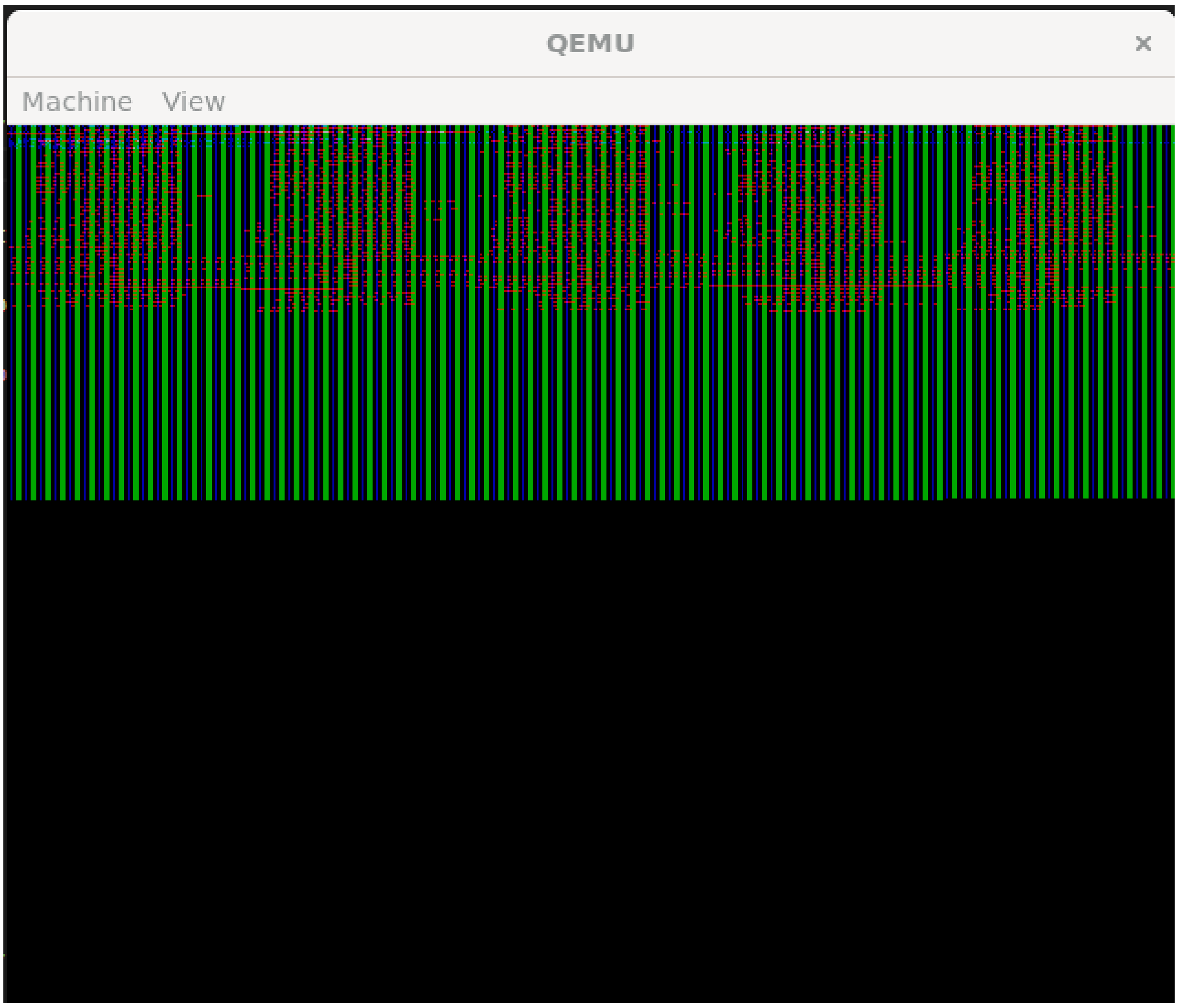


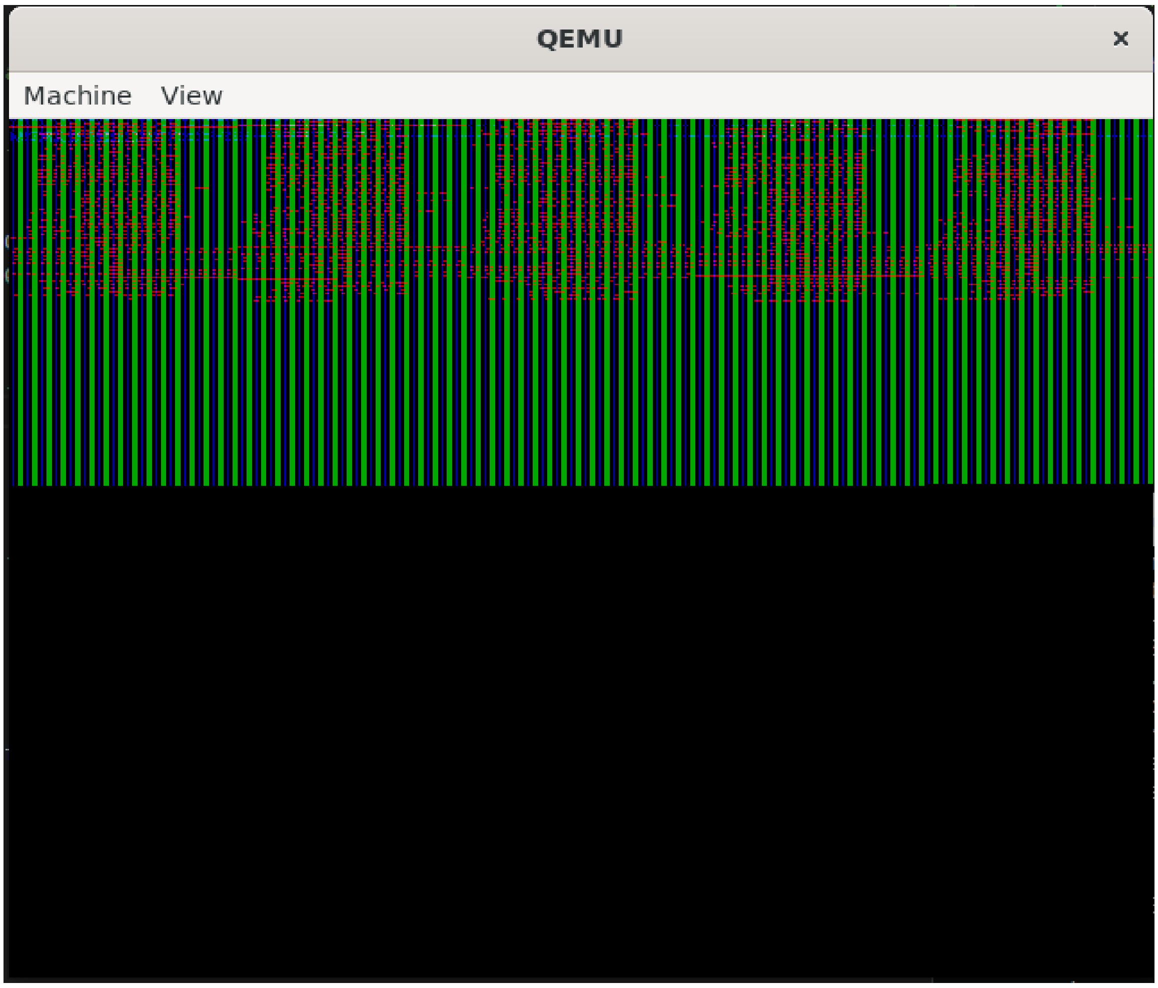
```
// vga.c
int             videosetmode(uchar mode);
uchar           getcurrentvideomode();
void            setplane(uchar plane);
uchar *         getframebufferbase();
```

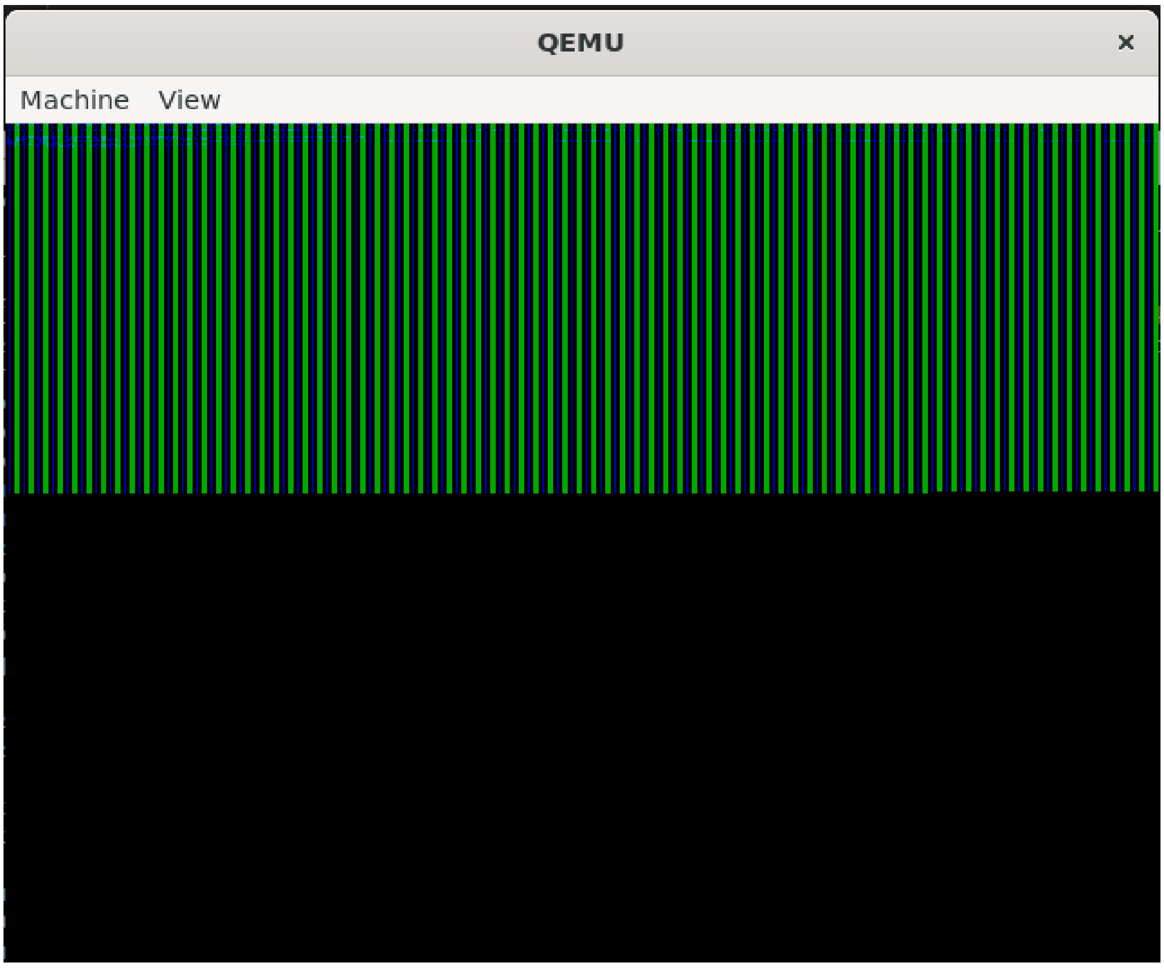


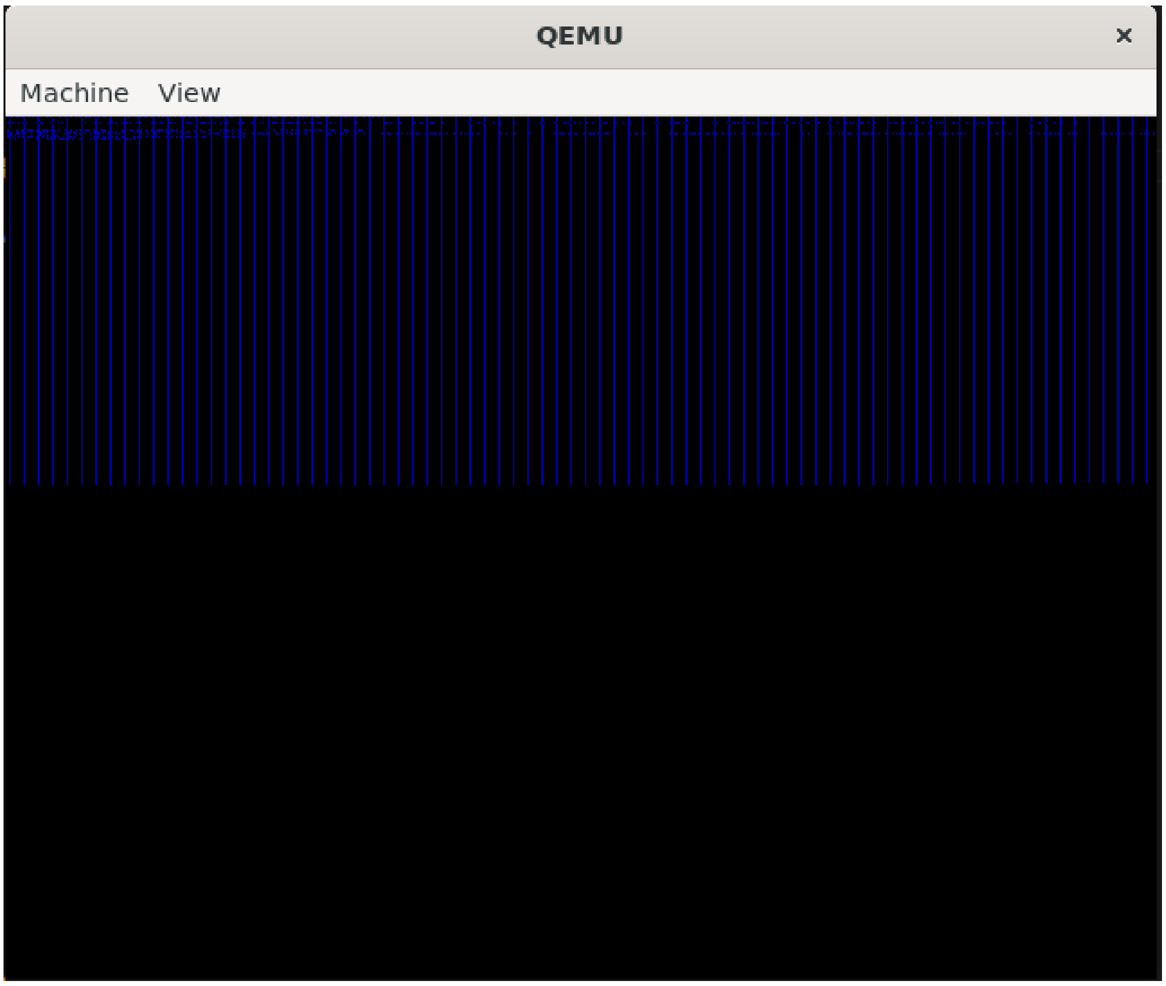
```
 // Returns a pointer to the virtual address (
 // the current video plane.

 uchar* getframebufferBase()
 {
     uchar* base;
     uchar plane;

     outb(VGA_GC_INDEX, 6);
     plane = inb(VGA_GC_DATA);
     plane >>= 2;
```

After that, I experimented with which bits went with what when working with the planes. I used a print statement in clear640x400x16 to crash the program to inspect and take screenshots at every plane. I started by clearing plane 3 -> 0 and then doing the same in reverse 0->3:

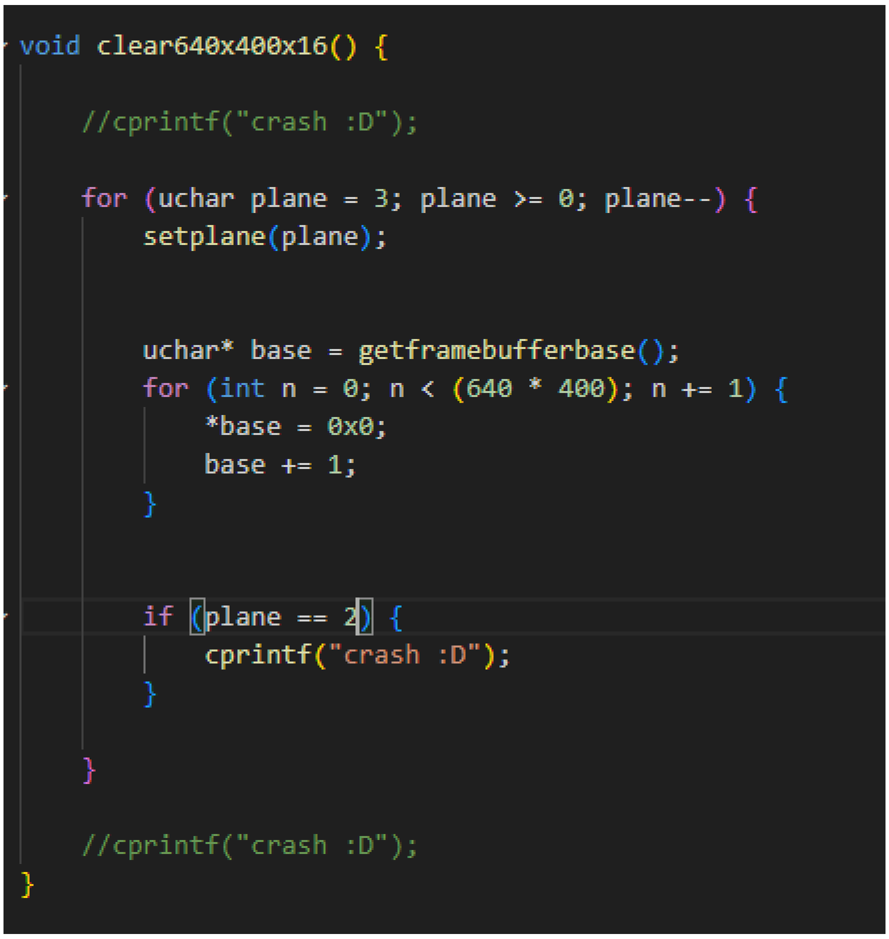



QEMU
Machine
View


QEMU
Machine
View

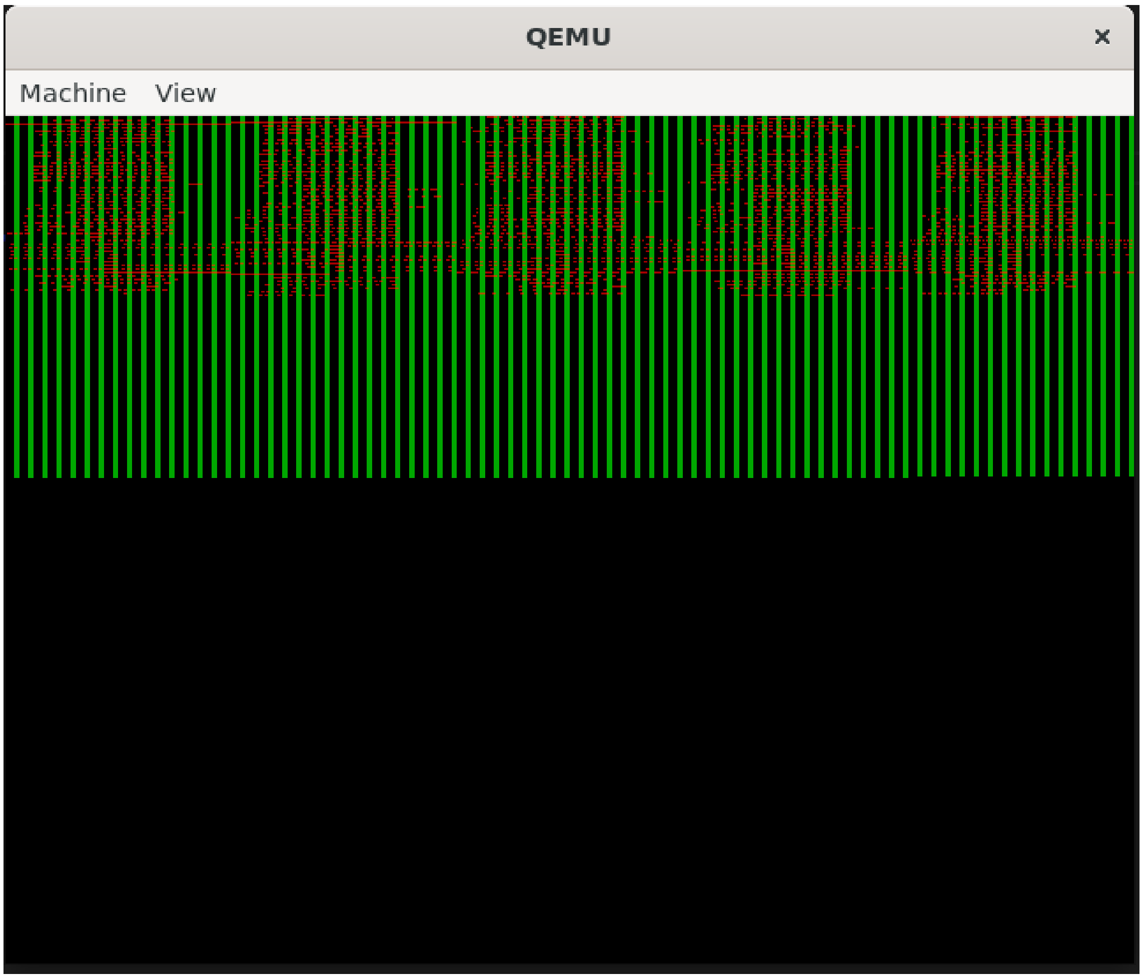

QEMU
Machine
View

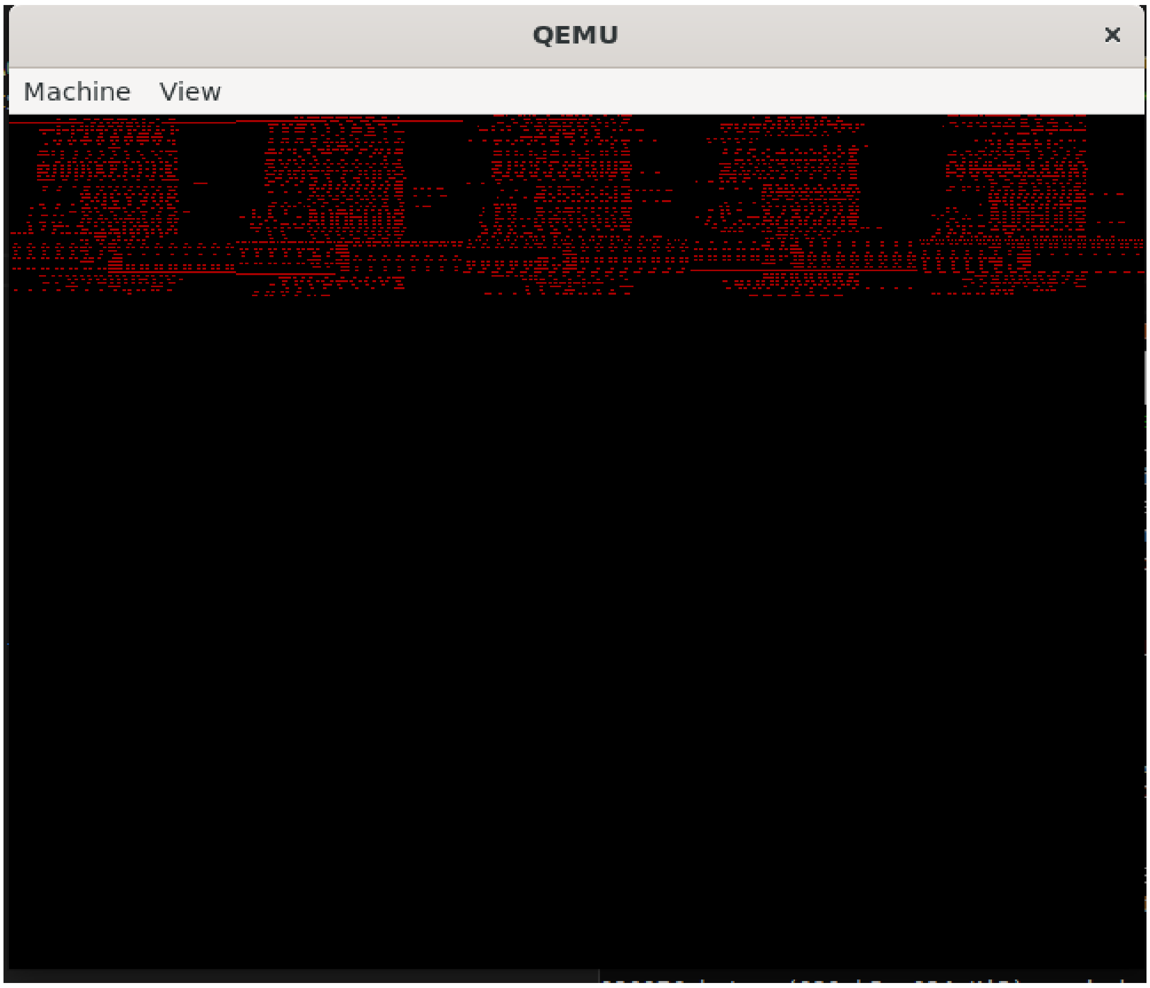
QEMU
Machine
View

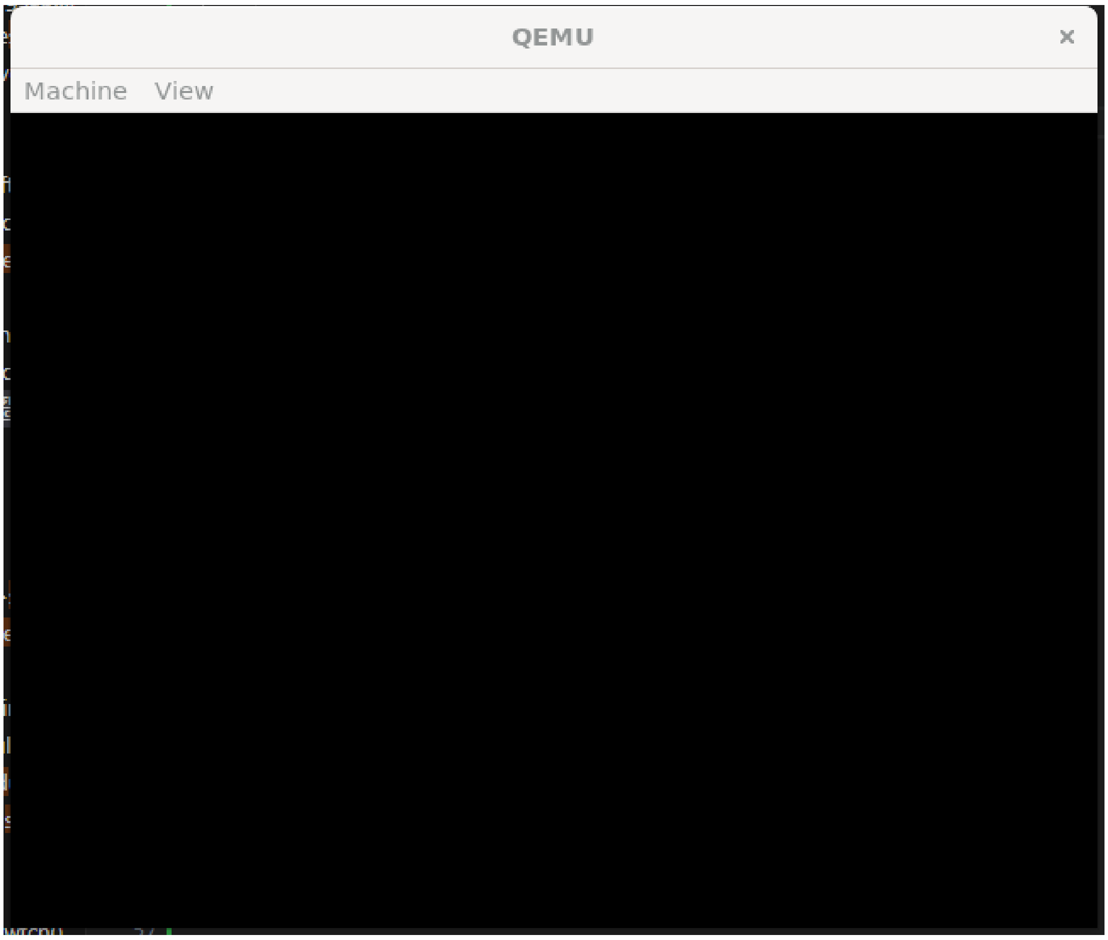


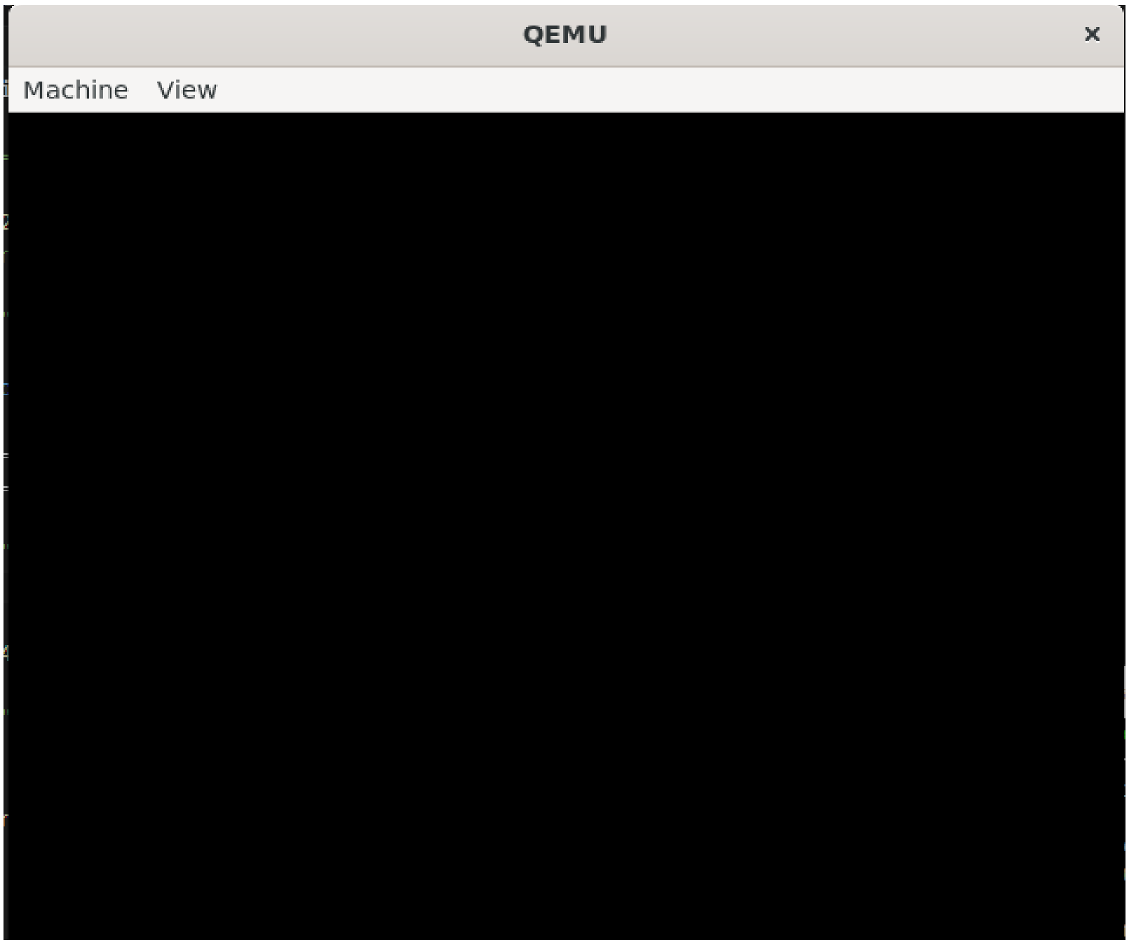
```
void clear640x400x16() {

    //cprintf("crash :D");

    for (uchar plane = 3; plane >= 0; plane--) {
        setplane(plane);


        uchar* base = getframebufferbase();
        for (int n = 0; n < (640 * 400); n += 1) {
            *base = 0x0;
            base += 1;
        }


        if (plane == 2) {
            cprintf("crash :D");
        }

    }

    //cprintf("crash :D");
}
```

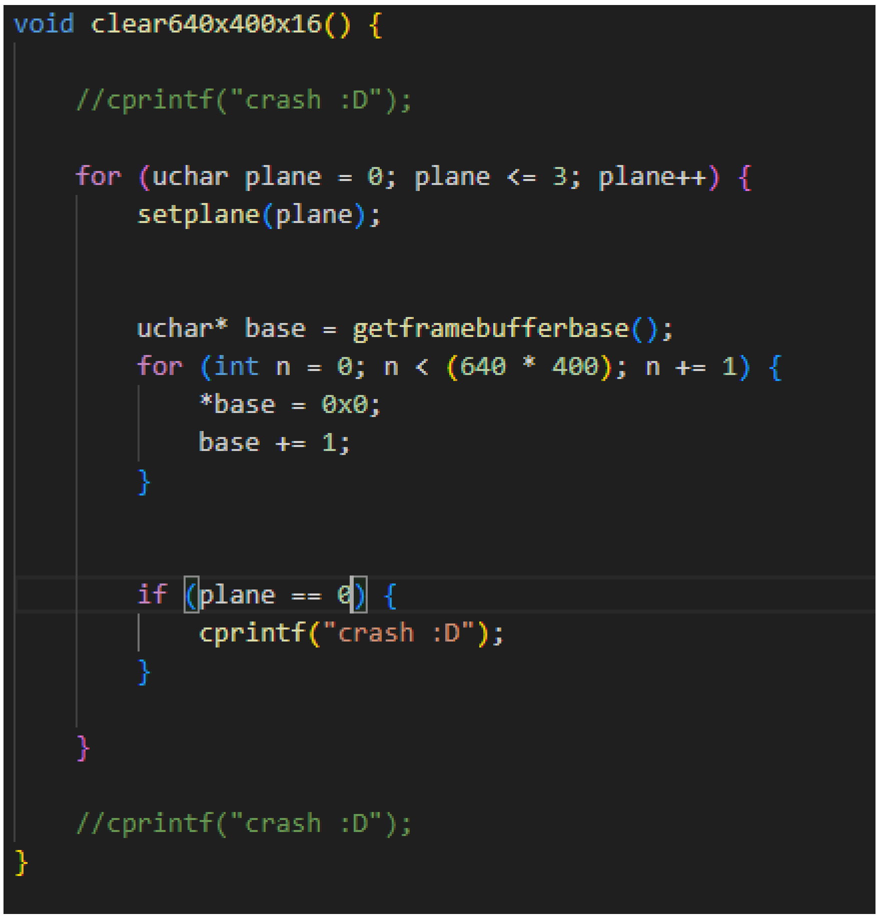
And then in reverse:




QEMU
Machine
View


QEMU
Machine
View





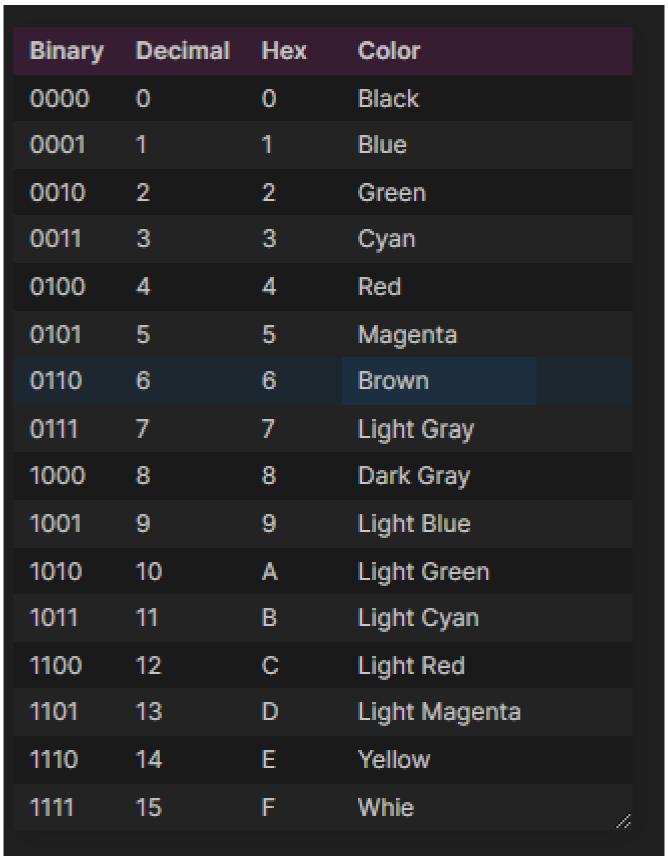
```
void clear640x400x16() {

    //cprintf("crash :D");

    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);


        uchar* base = getframebufferbase();
        for (int n = 0; n < (640 * 400); n += 1) {
            *base = 0x0;
            base += 1;
        }


        if (plane == 0) {
            cprintf("crash :D");
        }

    }

    //cprintf("crash :D");
}
```

I spent a while inspecting these images as I was more confused than I should have been about which the significant bit was as stepping through these frames and figuring out which colors turned what colors when zeroing planes didn't add up with what I was expecting/seeing.

Consulting a color table I wrote up with the colors in different formats, I tried to figure out.

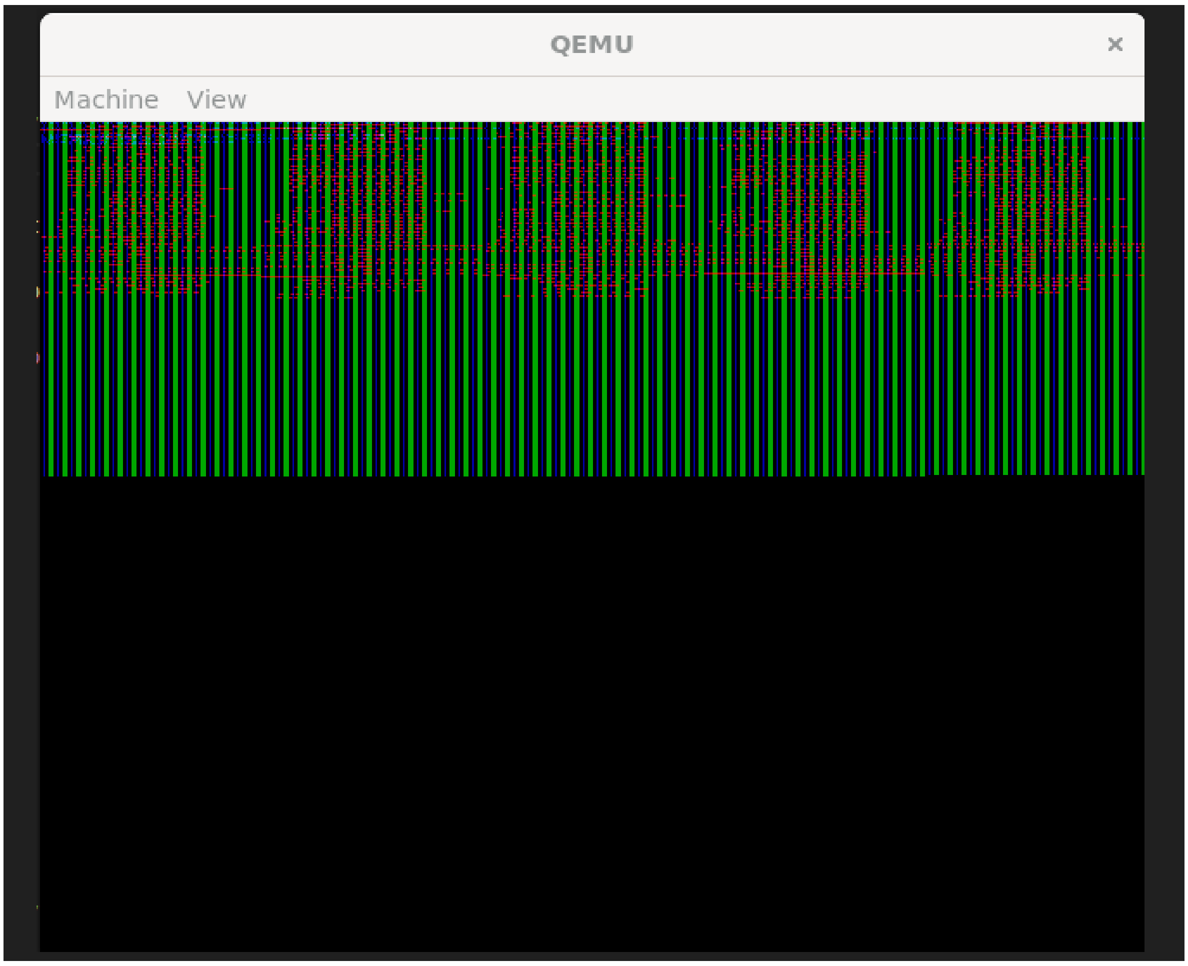
On the next page, I've reshown the image of the screen before clearing and then clearing plane 0. Looking at the difference, I saw that the green in the second image looked different so I assumed that it was going from light green to green which is 1010 to 0010. This could also happen with light red, going from 1100 to 0100 however it doesn't look like it changes.

This doesn't then make sense when clearing plane 1 because green disappears when it should be clearing plane 2 which does that.

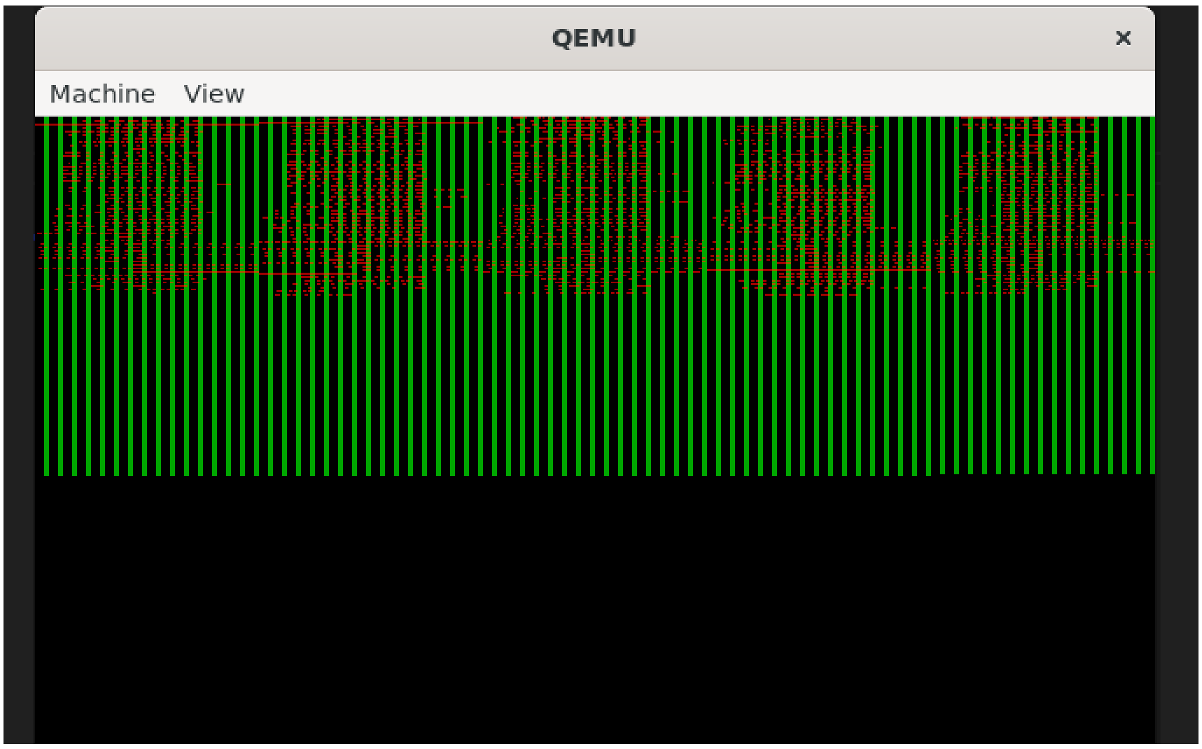


Binary	Decimal	Hex	Color
0000	0	0	Black
0001	1	1	Blue
0010	2	2	Green
0011	3	3	Cyan
0100	4	4	Red
0101	5	5	Magenta
0110	6	6	Brown
0111	7	7	Light Gray
1000	8	8	Dark Gray
1001	9	9	Light Blue
1010	10	A	Light Green
1011	11	B	Light Cyan
1100	12	C	Light Red
1101	13	D	Light Magenta
1110	14	E	Yellow
1111	15	F	Whie

Unchanged

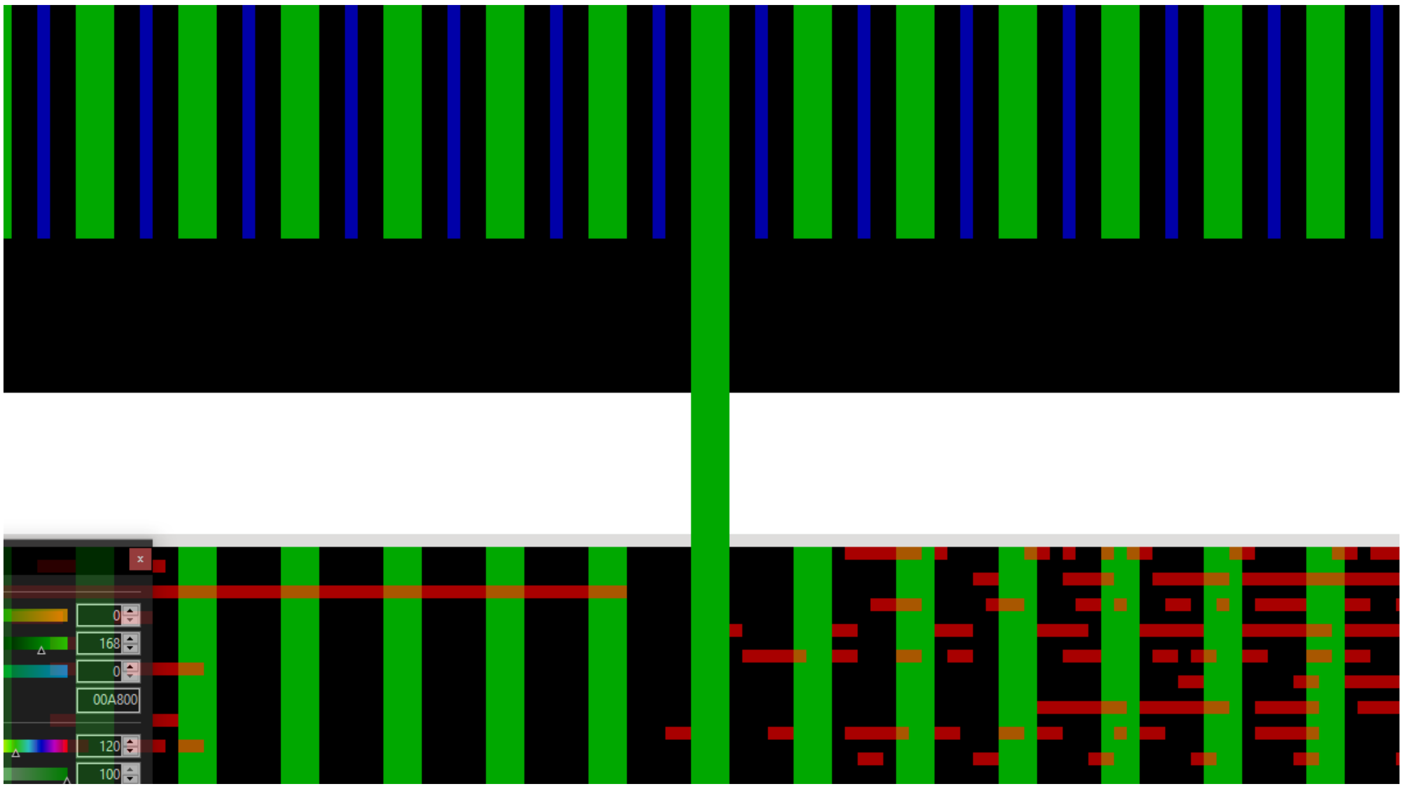


Plane 0 cleared



In hindsight, this shouldn't have been so confusing. Initial inspection looks like the green is much lighter and taking that information, my brain saw what it expected to see and not much else.

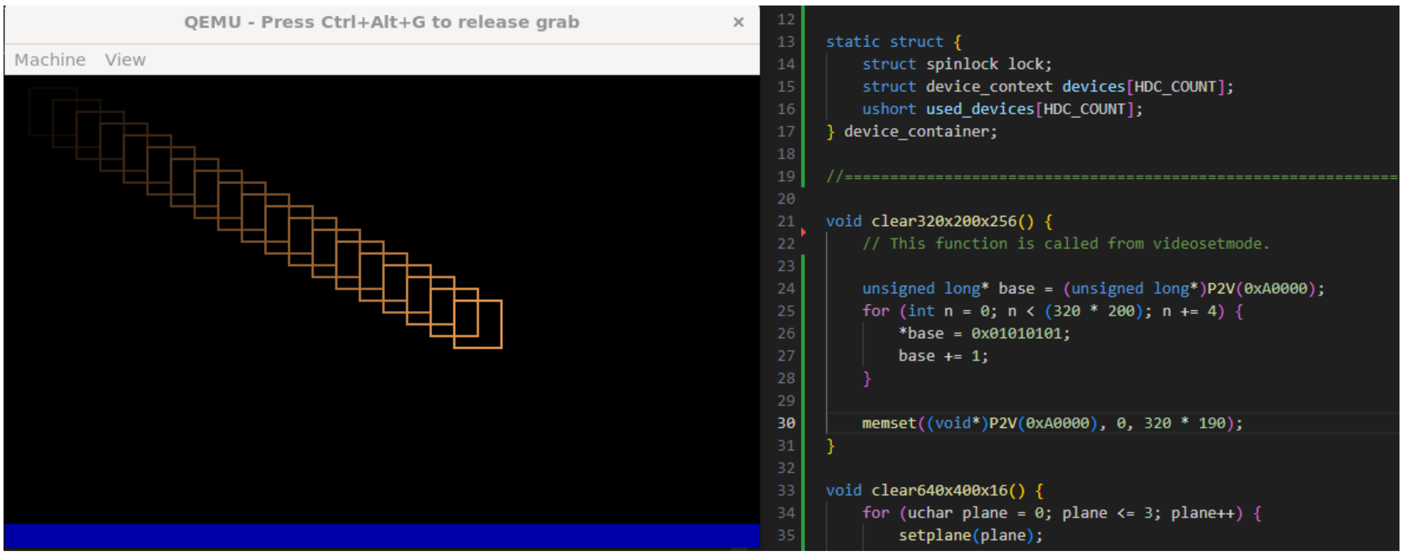
When taken into paint.net, it becomes apparent that the green is the exact same shade however the blue has been removed which makes the green seem so much more vibrant than before:



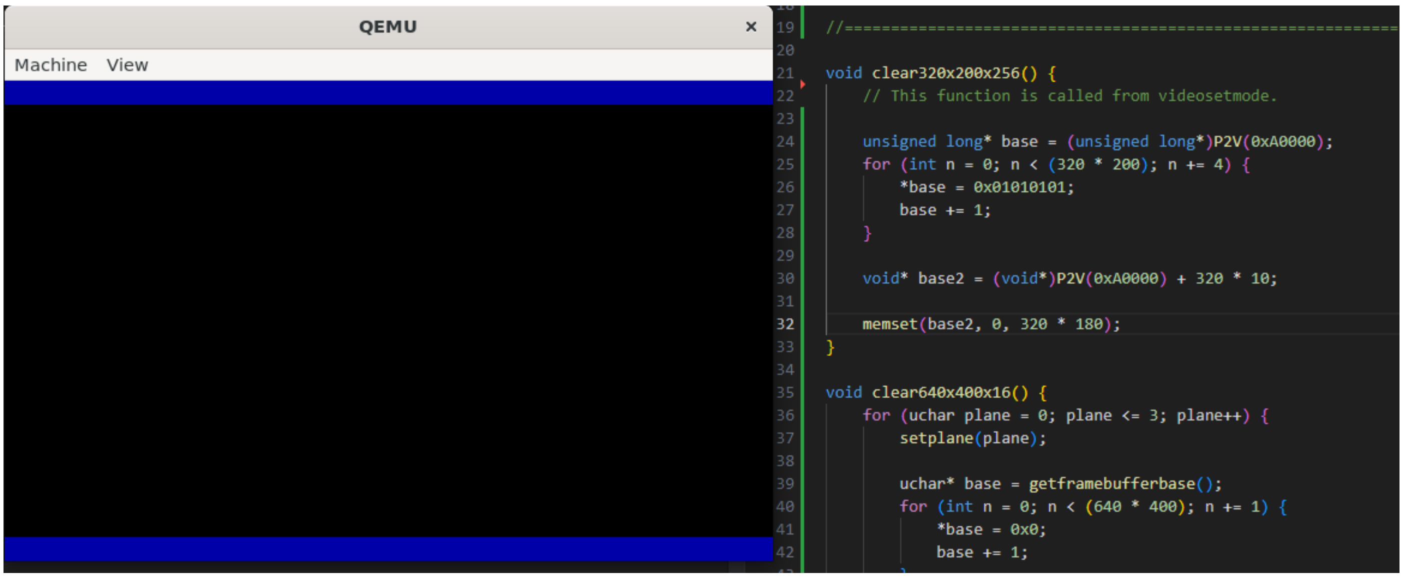
This also means that we're not doing 1010 -> 0010, we're actually doing 0010 -> 0010 which is why when clearing the second bit, we lose the green because it's 0010 -> 0000. I thought we were going left to right instead of right to left and I clearly don't understand the meaning of "the most significant bit".

This now makes sense as we progress to the third bit as red is removed next, its value of 0100 going to 0000.

In the interest of trying to make this more efficient, I've finally looked into how to use the memset function I've seen. I've applied this to the 0x13 video mode first just to see how/if it worked and it looks like it has. The following code sets the background to blue and then uses memset to clear it. I've done this to see how much it clears and if it actually works. In the first image, I'm only clearing 320 * 190 instead of 320 * 200 showing the blue bar at the bottom. The second image is similar except offset at the start to show blue at the top and bottom. This tells me that it works as I expected and I can just use it.

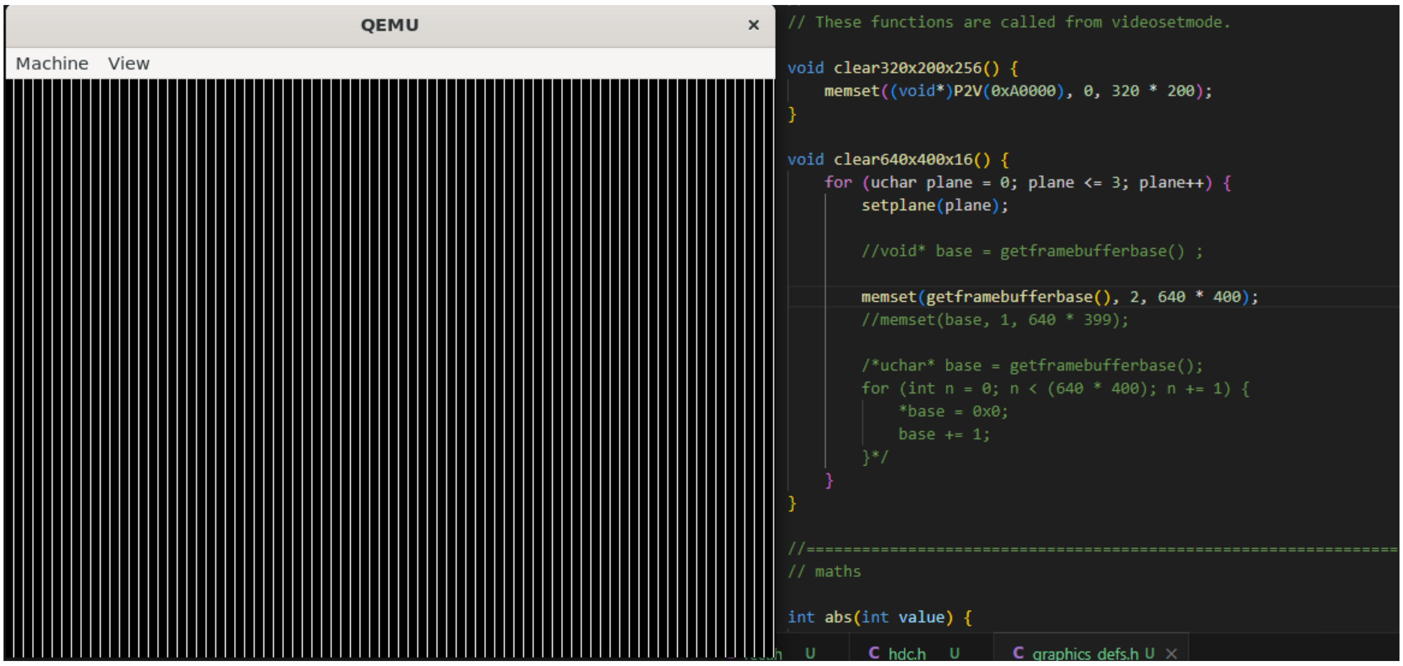


```c

static struct {
    struct spinlock lock;
    struct device_context devices[HDC_COUNT];
    ushort used_devices[HDC_COUNT];
} device_container;

//==================================================================

void clear320x200x256() {
    // This function is called from videosetmode.

    unsigned long* base = (unsigned long*)P2V(0xA0000);
    for (int n = 0; n < (320 * 200); n += 4) {
        *base = 0x01010101;
        base += 1;
    }

    memset((void*)P2V(0xA0000), 0, 320 * 190);
}

void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);
```



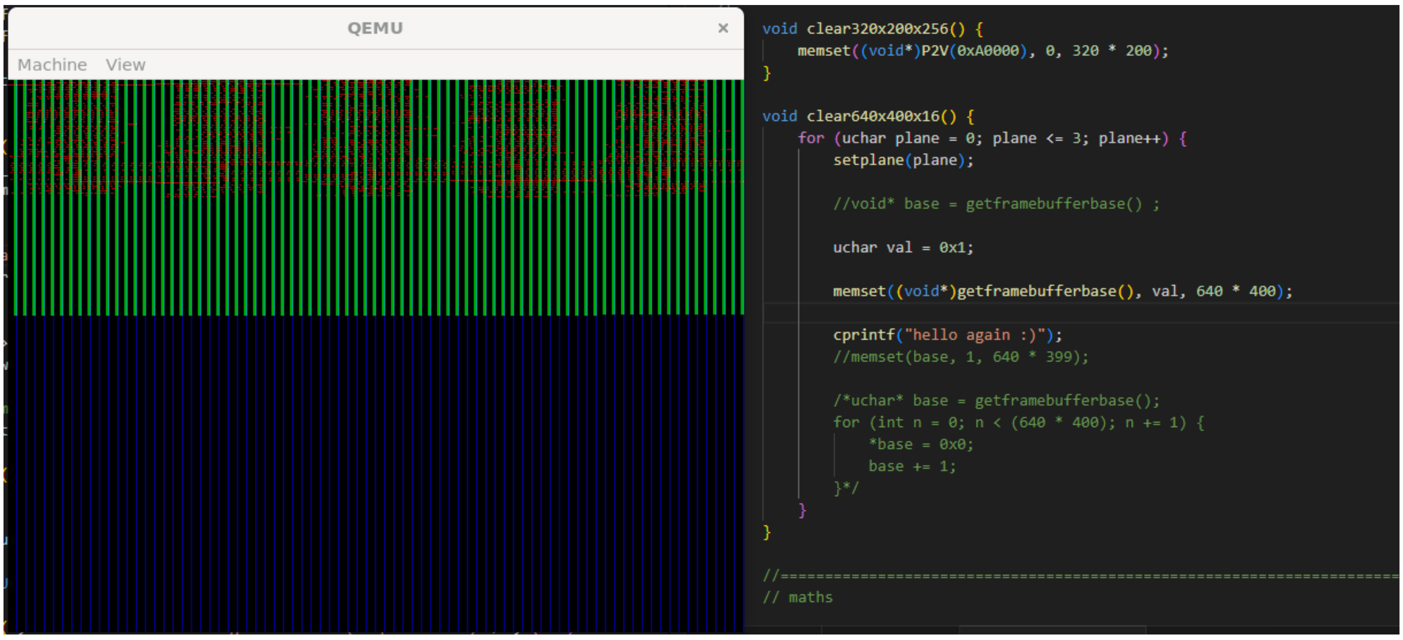
```
//==================================================================
void clear320x200x256() {
    // This function is called from videosetmode.

    unsigned long* base = (unsigned long*)P2V(0xA0000);
    for (int n = 0; n < (320 * 200); n += 4) {
        *base = 0x01010101;
        base += 1;
    }

    void* base2 = (void*)P2V(0xA0000) + 320 * 10;

    memset(base2, 0, 320 * 180);
}

void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);

        uchar* base = getframebufferbase();
        for (int n = 0; n < (640 * 400); n += 1) {
            *base = 0x0;
            base += 1;
```

Applying this to vm12 however gives me strange results:



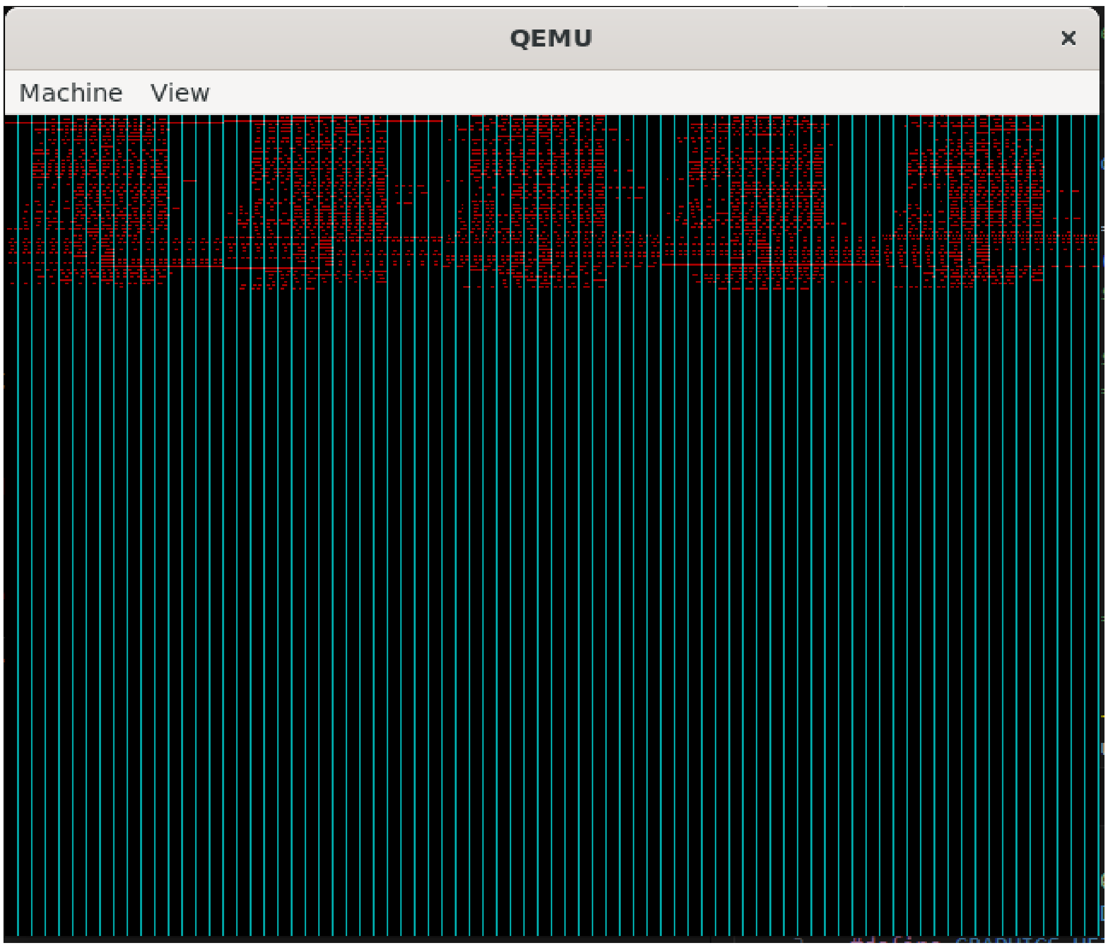
```
// These functions are called from videosetmode.

void clear320x200x256() {
    memset((void*)P2V(0xA0000), 0, 320 * 200);
}

void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);

        //void* base = getframebufferbase() ;

        memset(getframebufferbase(), 2, 640 * 400);
        //memset(base, 1, 640 * 399);

        /*uchar* base = getframebufferbase();
        for (int n = 0; n < (640 * 400); n += 1) {
            *base = 0x0;
            base += 1;
        }*/
    }
}

//==================================================================
// maths

int abs(int value) {
```

Plane0

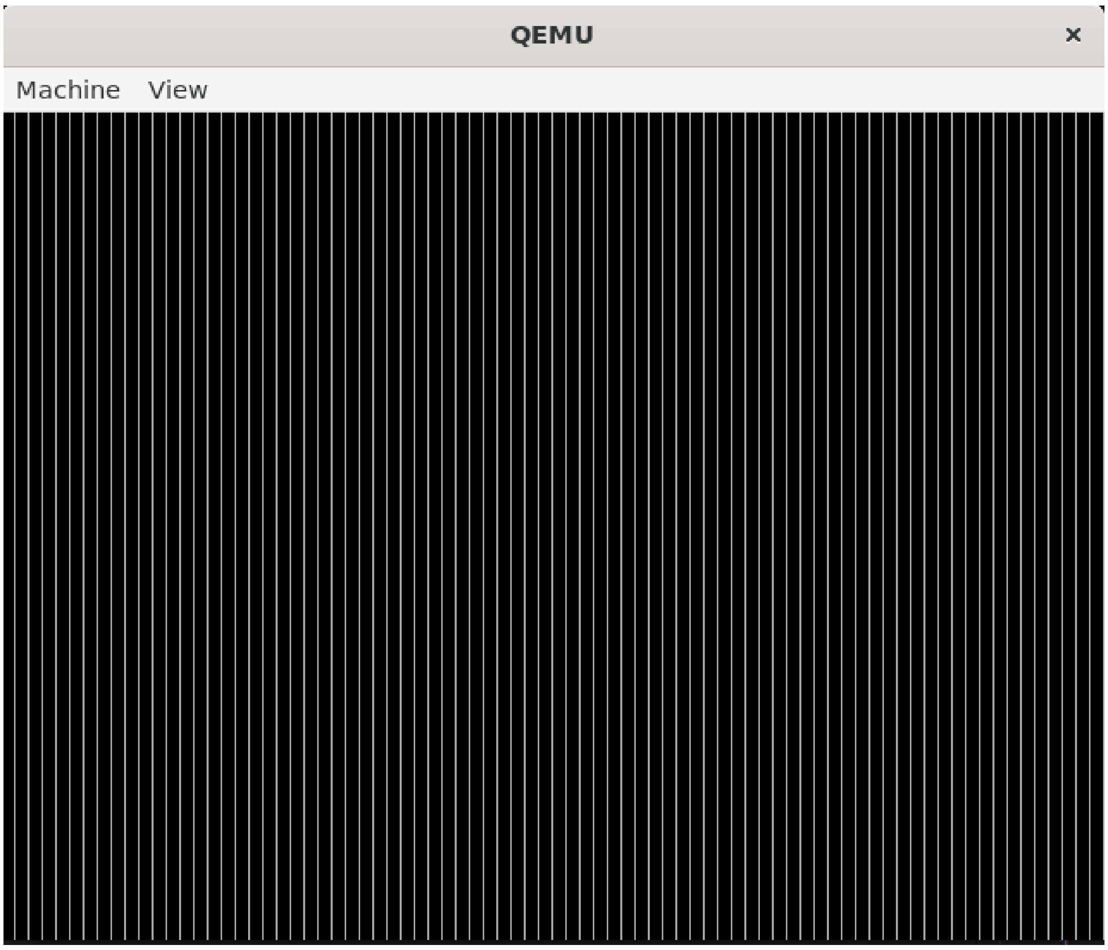


```
void clear320x200x256() {
    memset((void*)P2V(0xA0000), 0, 320 * 200);
}

void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);

        //void* base = getframebufferbase() ;

        uchar val = 0x1;

        memset((void*)getframebufferbase(), val, 640 * 400);

        cprintf("hello again :)");
        //memset(base, 1, 640 * 399);

        /*uchar* base = getframebufferbase();
        for (int n = 0; n < (640 * 400); n += 1) {
            *base = 0x0;
            base += 1;
        }*/
    }
}

//==================================================================
// maths
```

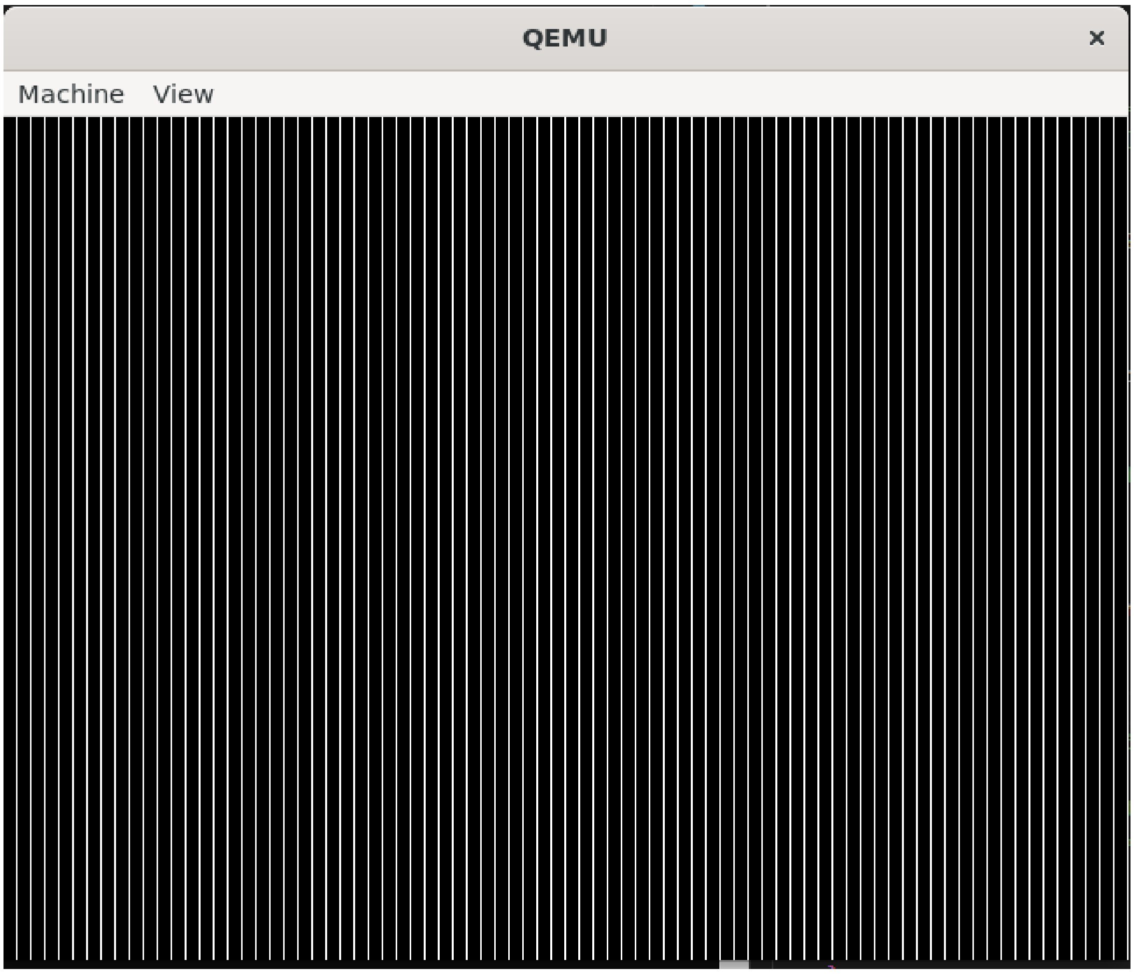
Plane 1



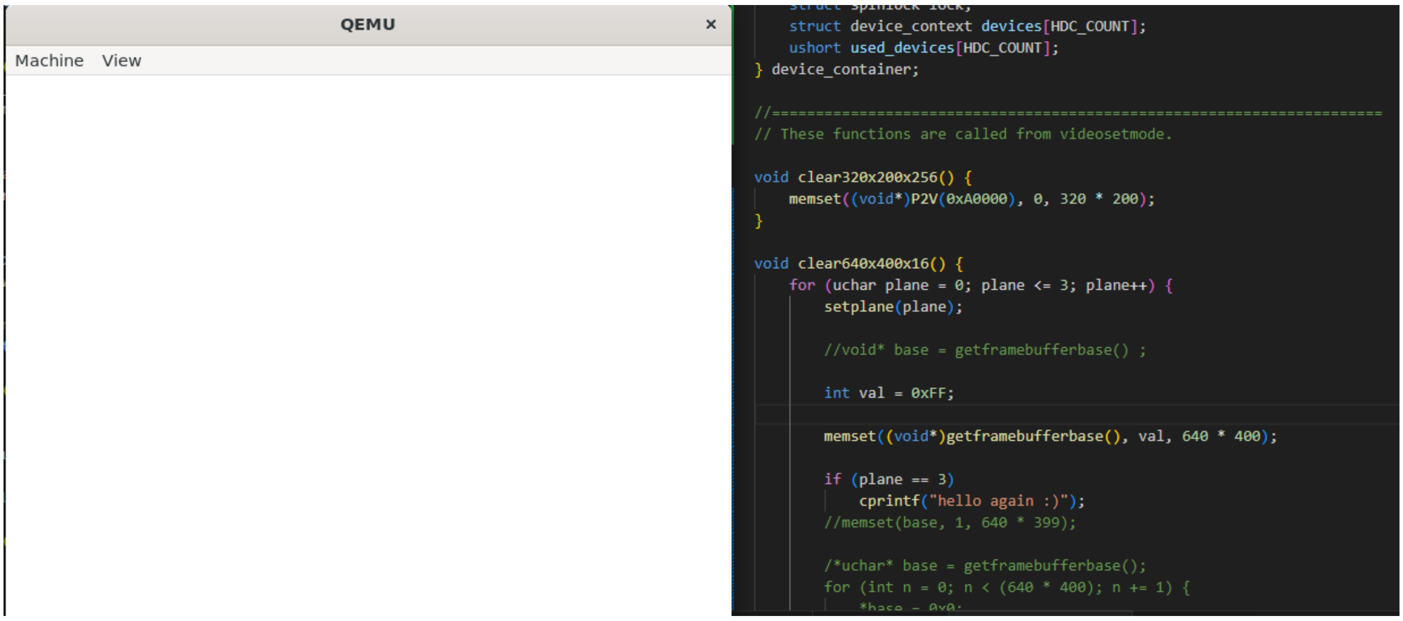
Plane 2



Plane 3

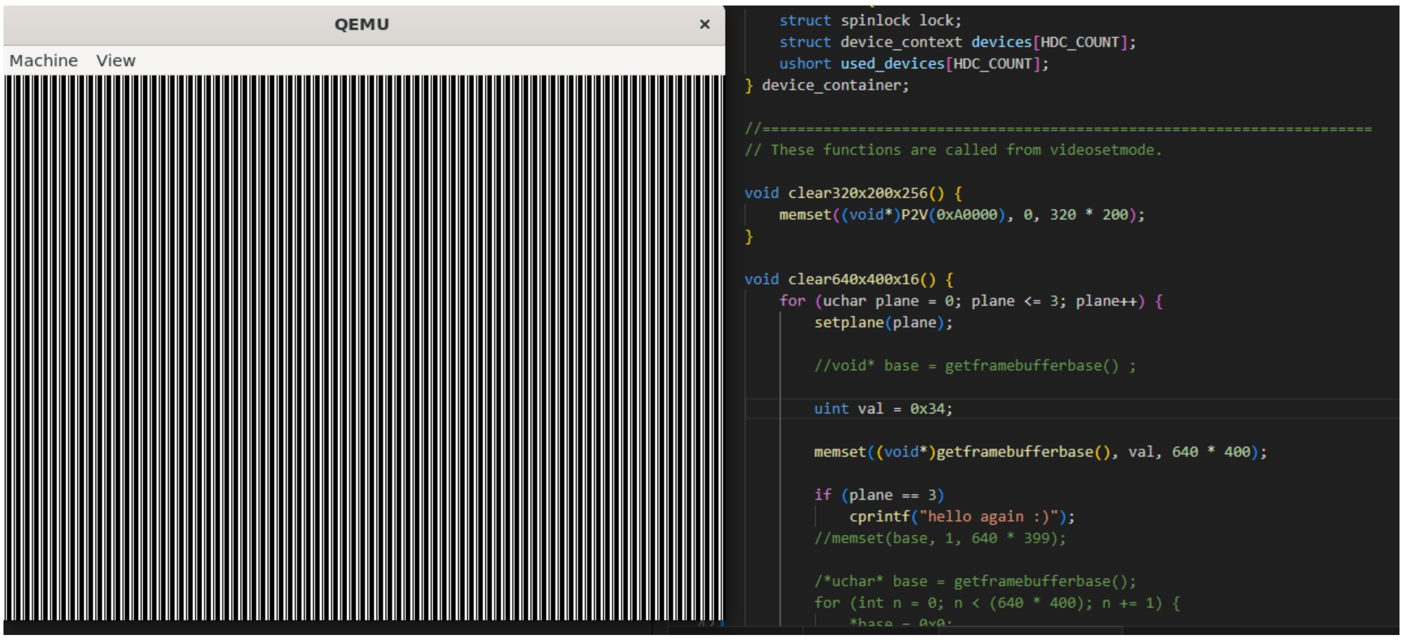


Of course, memset was given a character before which isn't 4 bytes but rather a byte so I should have been doing this:



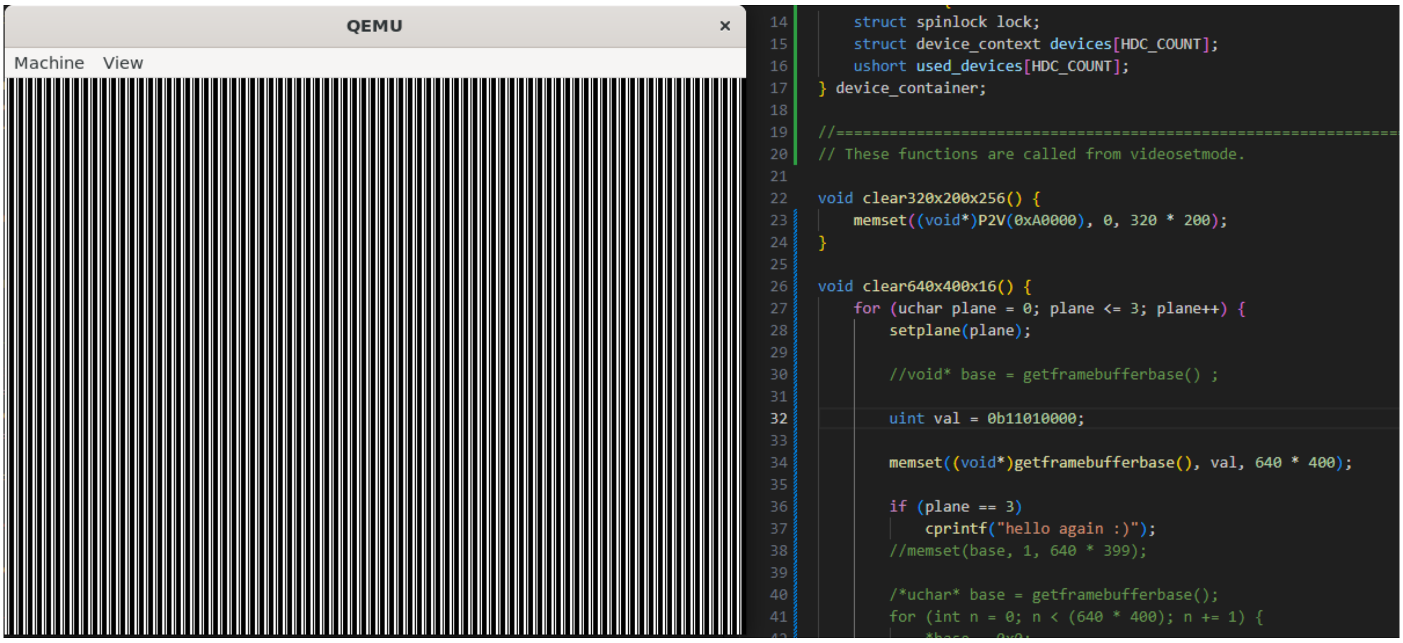
```
    struct spinlock lock;
    struct device_context devices[HDC_COUNT];
    ushort used_devices[HDC_COUNT];
} device_container;

//========================================================================
// These functions are called from videosetmode.

void clear320x200x256() {
    memset((void*)P2V(0xA0000), 0, 320 * 200);
}

void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);

        //void* base = getframebufferbase() ;

        int val = 0xFF;

        memset((void*)getframebufferbase(), val, 640 * 400);

        if (plane == 3)
            cprintf("hello again :)");
        //memset(base, 1, 640 * 399);

        /*uchar* base = getframebufferbase();
        for (int n = 0; n < (640 * 400); n += 1) {
            *base = 0x0;
```

I anticipated that setting to something like 0x34 would then give me a mix of cyan and red however It returns an ugly barcode.



```
    struct spinlock lock;
    struct device_context devices[HDC_COUNT];
    ushort used_devices[HDC_COUNT];
} device_container;

//======================================================================
// These functions are called from videosetmode.

void clear320x200x256() {
    memset((void*)P2V(0xA0000), 0, 320 * 200);
}

void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);

        //void* base = getframebufferbase() ;

        uint val = 0x34;

        memset((void*)getframebufferbase(), val, 640 * 400);

        if (plane == 3)
            cprintf("hello again :)");
        //memset(base, 1, 640 * 399);

        /*uchar* base = getframebufferbase();
        for (int n = 0; n < (640 * 400); n += 1) {
```

Of course, again I am forgetting that each plane it responsible for a bit of the color and the colors aren't sequential which is why the colors are either white or black as I'm setting the values to either 1 or 0 for the same position across all planes which either makes white or black. I can see this from the following with a binary value. The binary value being 11010000 which should be 2 pixels of white, 1 pixel of black, another pixel of white and then all black which is what we then see:



```
    struct spinlock lock;
    struct device_context devices[HDC_COUNT];
    ushort used_devices[HDC_COUNT];
} device_container;

//======================================================================
// These functions are called from videosetmode.

void clear320x200x256() {
    memset((void*)P2V(0xA0000), 0, 320 * 200);
}

void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);

        //void* base = getframebufferbase() ;

        uint val = 0b11010000;

        memset((void*)getframebufferbase(), val, 640 * 400);

        if (plane == 3)
            cprintf("hello again :)");
        //memset(base, 1, 640 * 399);

        /*uchar* base = getframebufferbase();
        for (int n = 0; n < (640 * 400); n += 1) {
```

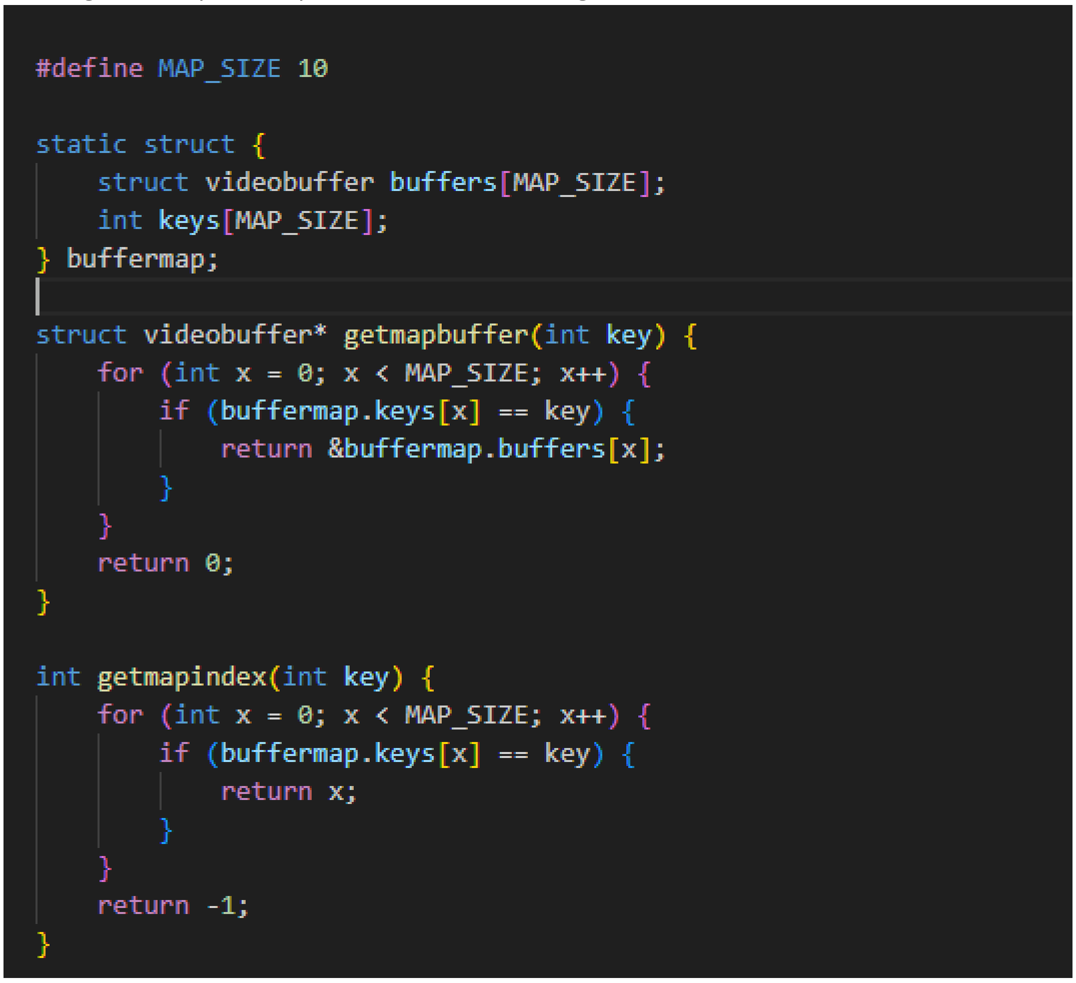
Overall, what this has told me is that setting a clear color for this graphics mode is a bit more complicated than how it seems initially. Not much more complicated but a little more.

09/12/23

Moving into stage5, I want to change to frame buffering however I realize I'm going to have to ditch or change the majority of my code to add this.

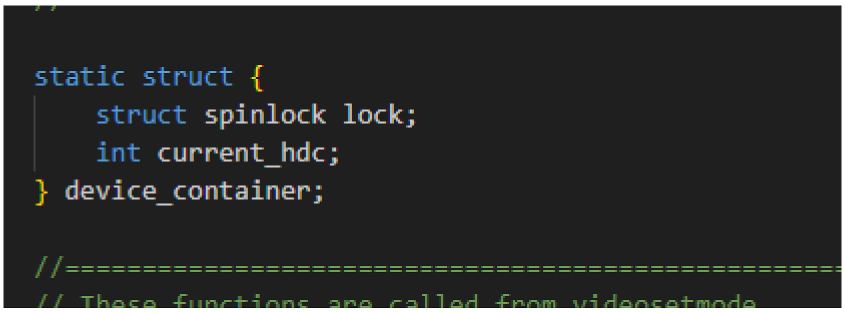
For a while, since hdc was just an id and didn't actually store any useful information in the kernel anymore, it felt like it could just be an id that whenever I needed a new one I could just de previous_hdc += 1. Doing this would need me to change the implementation in ugraphics.c to use something like a map/dictionary.

Looking into simple c maps, I added the following:



```
#define MAP_SIZE 10

static struct {
    struct videobuffer buffers[MAP_SIZE];
    int keys[MAP_SIZE];
} buffermap;

struct videobuffer* getmapbuffer(int key) {
    for (int x = 0; x < MAP_SIZE; x++) {
        if (buffermap.keys[x] == key) {
            return &buffermap.buffers[x];
        }
    }
    return 0;
}

int getmapindex(int key) {
    for (int x = 0; x < MAP_SIZE; x++) {
        if (buffermap.keys[x] == key) {
            return x;
        }
    }
    return -1;
}
```

And then the struct in graphics.c in kernel space could just be this:



```
static struct {
    struct spinlock lock;
    int current_hdc;
} device_container;

//=================================================
// These functions are called from videosetmode
```

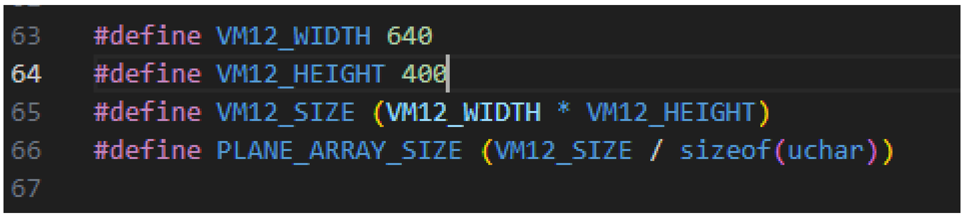
Unfortunately, simple map implementations couldn't remove/recycle previously used keys and I didn't want to spend a huge amount of time figuring this out so I went back to the old system which didn't really feel right but it worked as well as it really needed to.

Also, changing from a command queue system to a screen buffer approach means that I can no longer queue commands which means that my temporary solution for setpencolor having a hdc field is no longer appropriate. Given its awkward, I've opted to keep setpencolor as a system call as any other approach requires just as many, if not more, system calls and a bit of extra code which defeats the point of doing it any other way.

14/12/23

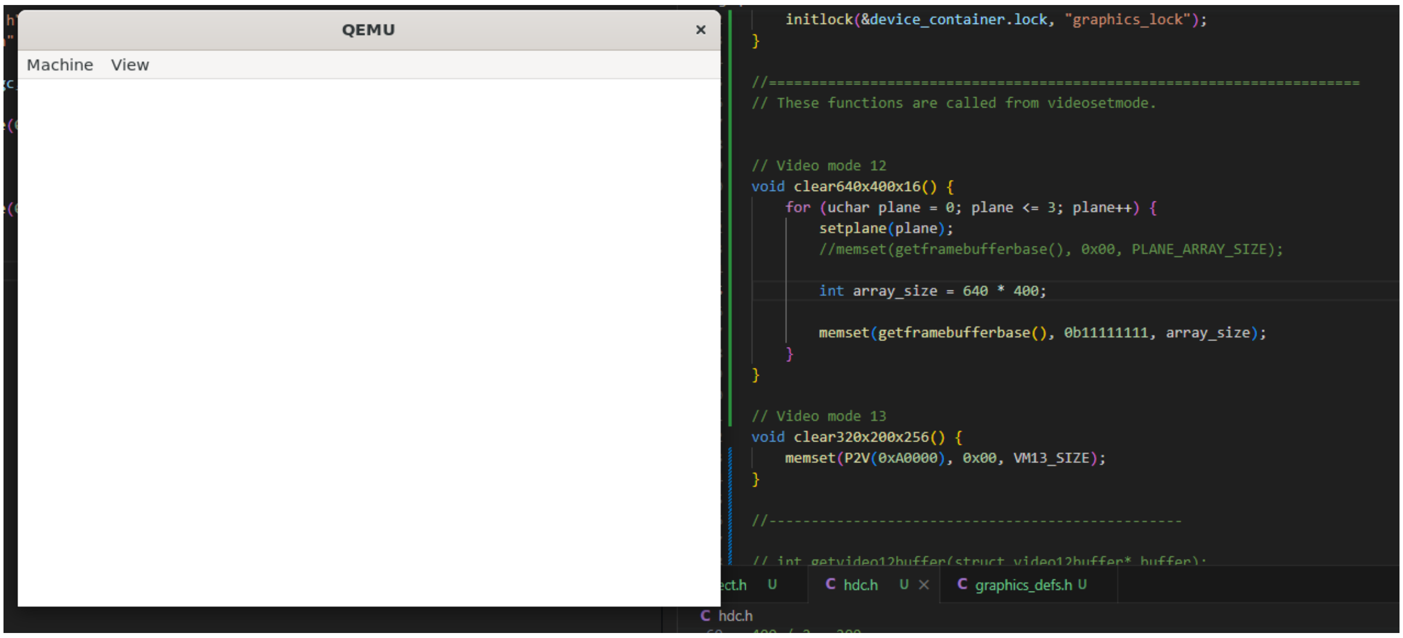
Working on the vm12 mode was a bit of a nightmare at first. Working out how to use an array of 640 * 400 which is in bytes, making it 640 * 400 * 8 when we actually want to work in bits requiring 640 * 400 * 1, was harder to wrap my head around than it felt like it should have been.

After doing some of the maths, I initially came to these definitions.

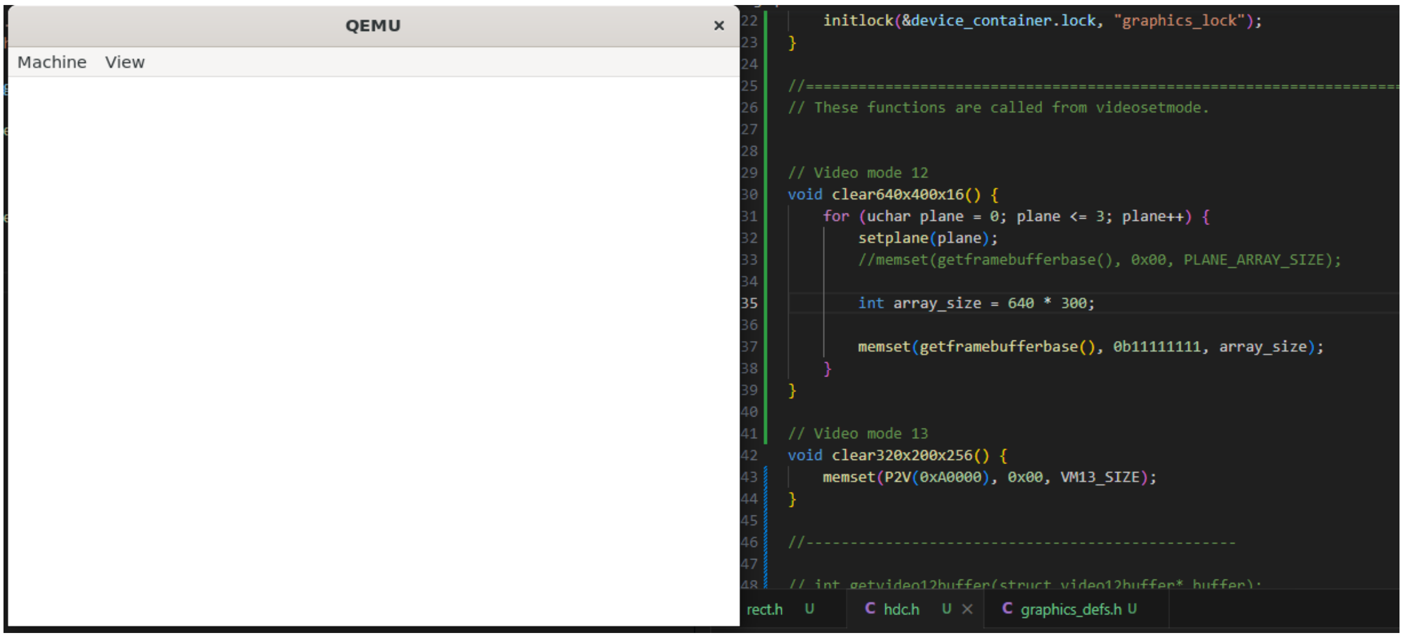


```c
#define VM12_WIDTH 640
#define VM12_HEIGHT 400
#define VM12_SIZE (VM12_WIDTH * VM12_HEIGHT)
#define PLANE_ARRAY_SIZE (VM12_SIZE / sizeof(uchar))
```

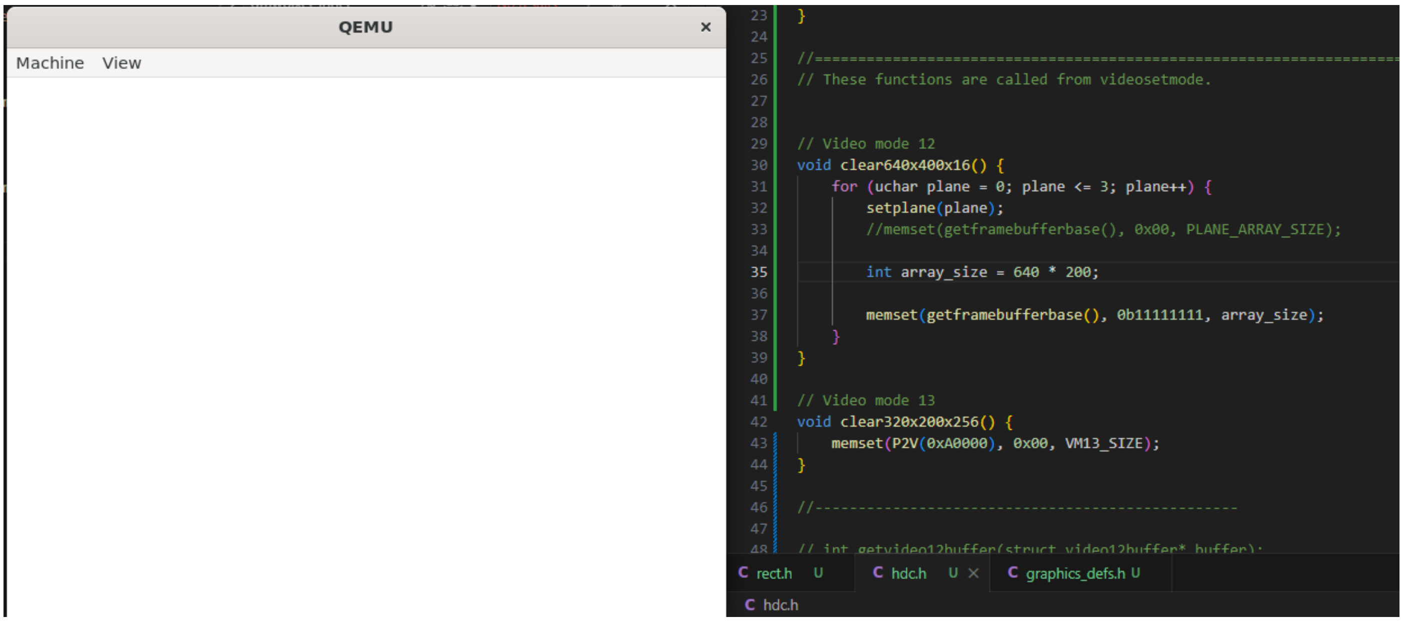
Following this thread, I realized that changing the array size variable to be smaller and smaller wasn't changing how much of the screen was being cleared. Going from 640 * 400 -> 640 * 59 before I saw any change.



```c
    initlock(&device_container.lock, "graphics_lock");
}

//==========================================================================
// These functions are called from videosetmode.


// Video mode 12
void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);
        //memset(getframebufferbase(), 0x00, PLANE_ARRAY_SIZE);

        int array_size = 640 * 400;

        memset(getframebufferbase(), 0b11111111, array_size);
    }
}

// Video mode 13
void clear320x200x256() {
    memset(P2V(0xA0000), 0x00, VM13_SIZE);
}

//------------------------------------------------

// int getvideo12buffer(struct video12buffer* buffer);
```

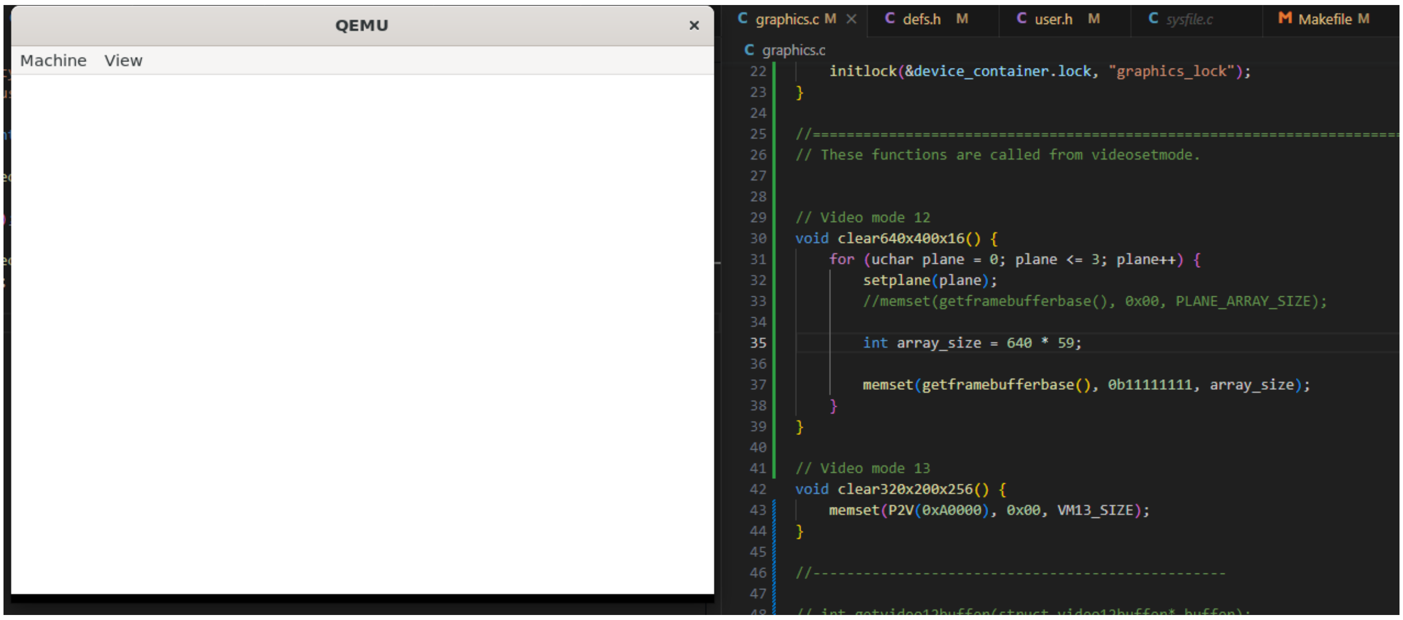


```c
    initlock(&device_container.lock, "graphics_lock");
}

//==========================================================================
// These functions are called from videosetmode.


// Video mode 12
void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);
        //memset(getframebufferbase(), 0x00, PLANE_ARRAY_SIZE);

        int array_size = 640 * 300;

        memset(getframebufferbase(), 0b11111111, array_size);
    }
}

// Video mode 13
void clear320x200x256() {
    memset(P2V(0xA0000), 0x00, VM13_SIZE);
}

//------------------------------------------------

// int getvideo12buffer(struct video12buffer* buffer);
```



```
}

//==================================================================
// These functions are called from videosetmode.


// Video mode 12
void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);
        //memset(getframebufferbase(), 0x00, PLANE_ARRAY_SIZE);

        int array_size = 640 * 200;

        memset(getframebufferbase(), 0b11111111, array_size);
    }
}

// Video mode 13
void clear320x200x256() {
    memset(P2V(0xA0000), 0x00, VM13_SIZE);
}

//-------------------------------------------------

// int getvideo12buffer(struct video12buffer* buffer);
```

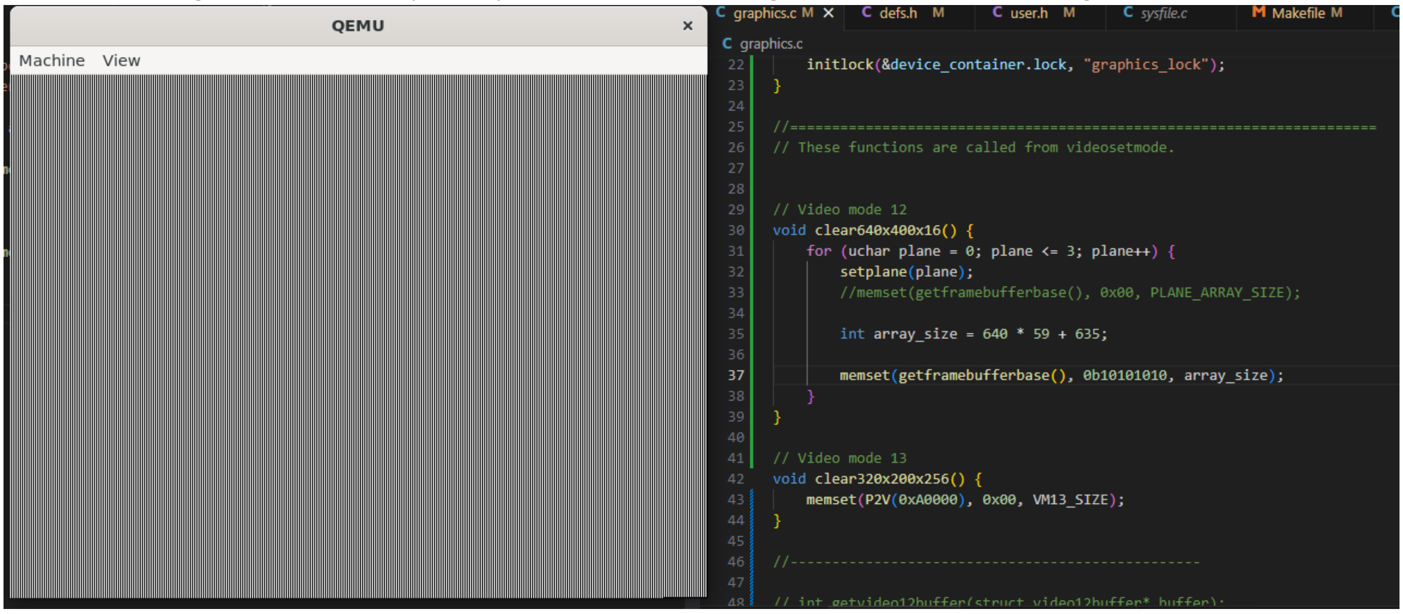
Here we can see a black line at the bottom of the screen



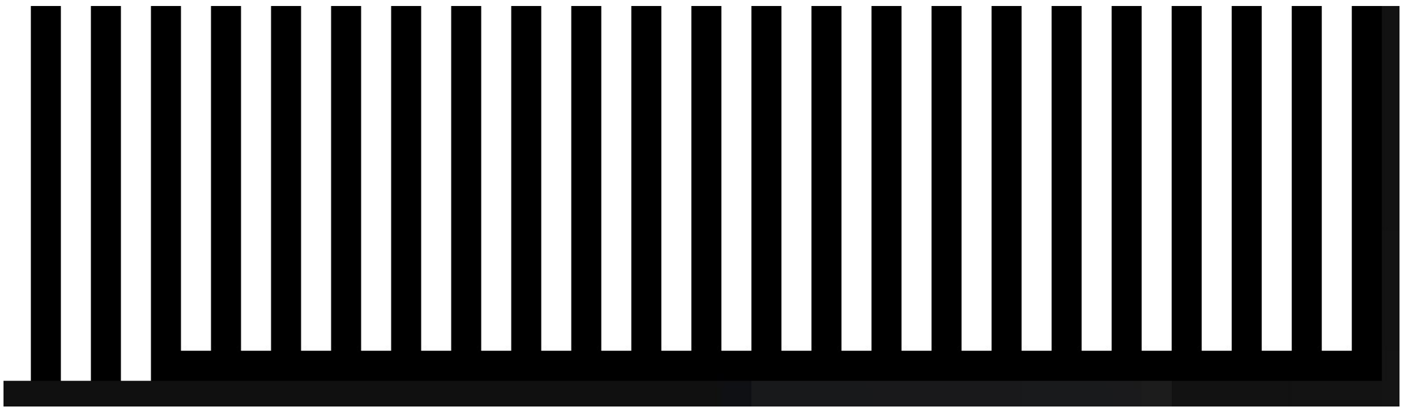
```
    initlock(&device_container.lock, "graphics_lock");
}

//==================================================================
// These functions are called from videosetmode.


// Video mode 12
void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);
        //memset(getframebufferbase(), 0x00, PLANE_ARRAY_SIZE);

        int array_size = 640 * 59;

        memset(getframebufferbase(), 0b11111111, array_size);
    }
}

// Video mode 13
void clear320x200x256() {
    memset(P2V(0xA0000), 0x00, VM13_SIZE);
}

//-------------------------------------------------

// int getvideo12buffer(struct video12buffer* buffer);
```

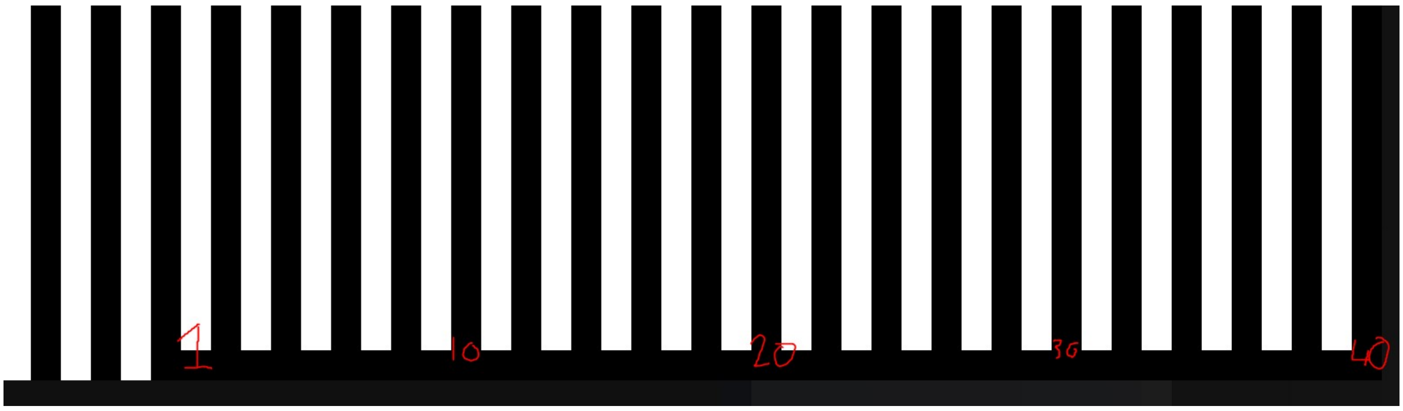
My thinking was 4 arrays of 640 * 400 bits which is 256000. These bits are then stored in chunks of 8 which is 256000 / 8 = 32000. So, we have 32000 bytes which we need to write. However, trial and error indicates that 640 * 59 bytes is when we start to not cover the entire screen which is 37760 bytes.

Here I'm clearing 640 * 59 + 635 bytes of pixels. At the bottom right, there is a small missing section.



```c
    initlock(&device_container.lock, "graphics_lock");
}

//=====================================================================
// These functions are called from videosetmode.


// Video mode 12
void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);
        //memset(getframebufferbase(), 0x00, PLANE_ARRAY_SIZE);

        int array_size = 640 * 59 + 635;

        memset(getframebufferbase(), 0b10101010, array_size);
    }
}

// Video mode 13
void clear320x200x256() {
    memset(P2V(0xA0000), 0x00, VM13_SIZE);
}

//-----------------------------------------------

// int getvideo12buffer(struct video12buffer* buffer);
```

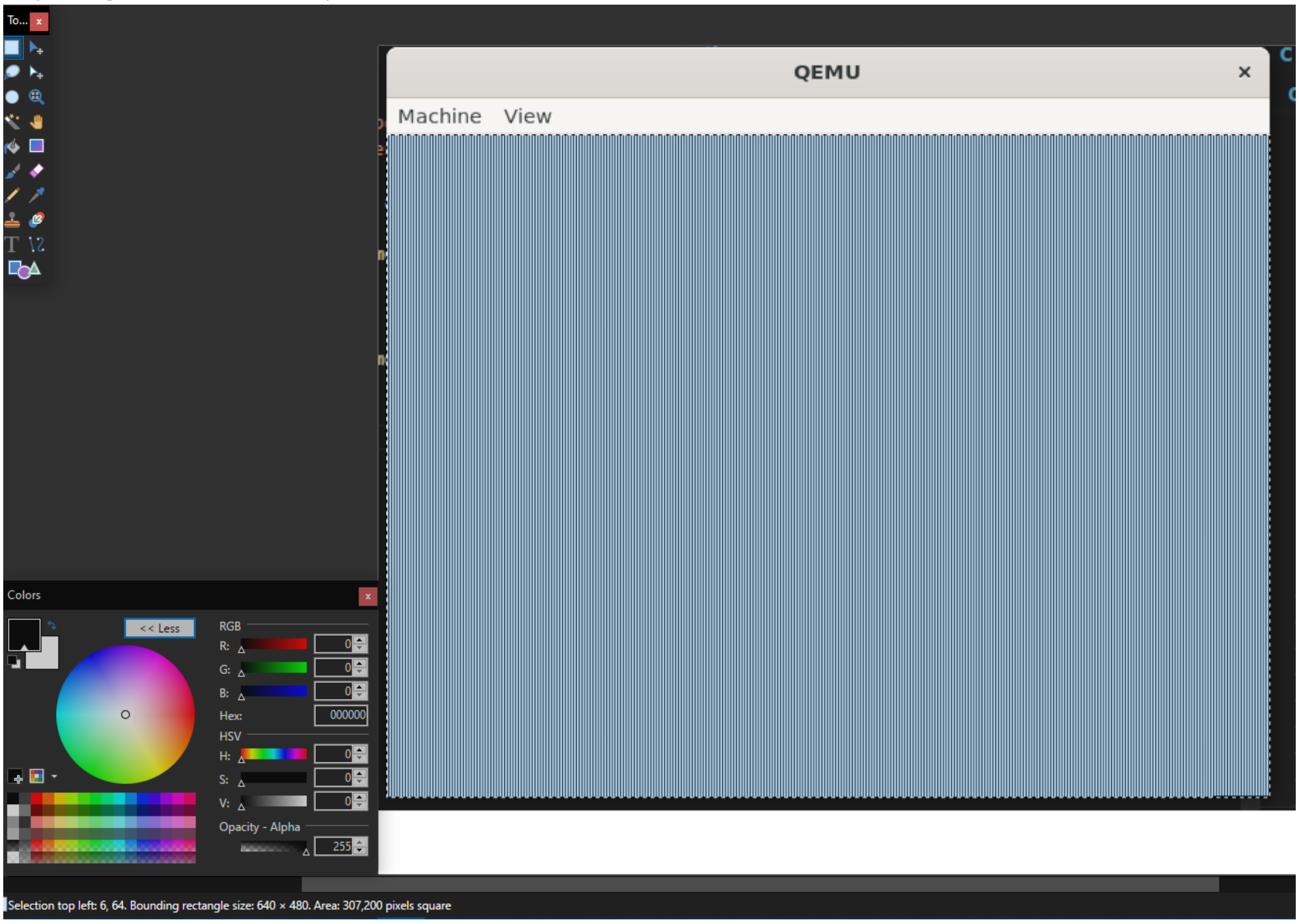




This shows that we're missing 40 bits which is 5 bytes which means that each plane is in fact 640 * 60 bytes long which is 38400 bytes.

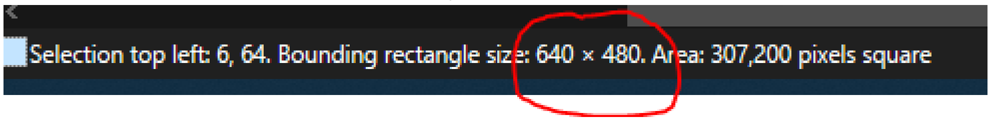
(640 / 8) * 400 = 32000
640 * (400/8) = 32000
(640 * 400) / 8 = 32000

Inspecting the dimensions in paint.net…



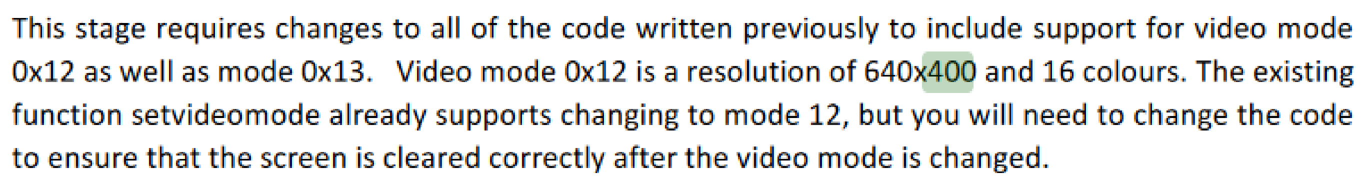
Selection top left: 6, 64. Bounding rectangle size: 640 × 480. Area: 307,200 pixels square

I find that I've 80 more rows than I expected to have.



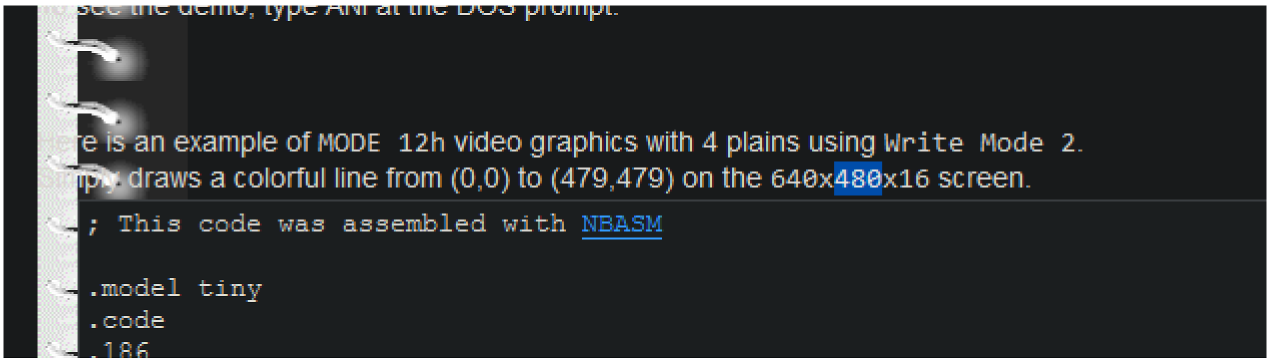
Selection top left: 6, 64. Bounding rectangle size: 640 × 480. Area: 307,200 pixels square

After looking on the internet, I've found multiple sources referring to the dimensions of video mode 12h as 640x480 and not 640x400 as mentioned in the brief.



This stage requires changes to all of the code written previously to include support for video mode 0x12 as well as mode 0x13. Video mode 0x12 is a resolution of 640x400 and 16 colours. The existing function setvideomode already supports changing to mode 12, but you will need to change the code to ensure that the screen is cleared correctly after the video mode is changed.

https://www.fysnet.net/modex.htm

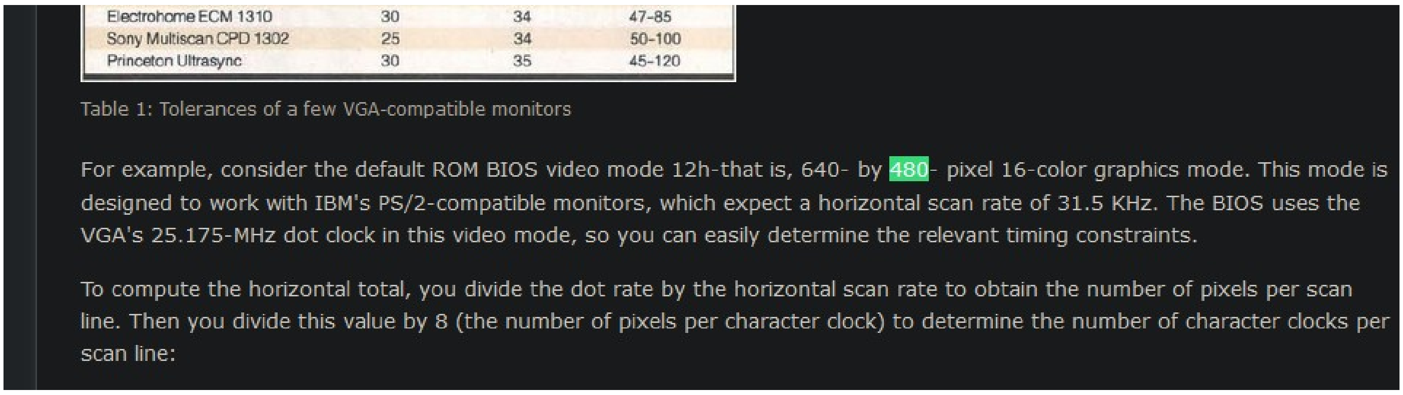


see the demo, type ANI at the DOS prompt.

e is an example of MODE 12h video graphics with 4 plains using Write Mode 2.
mply draws a colorful line from (0,0) to (479,479) on the 640x480x16 screen.

```
; This code was assembled with NBASM

.model tiny
.code
.186
```

https://www.ardent-tool.com/video/VGA_Video_Modes.html



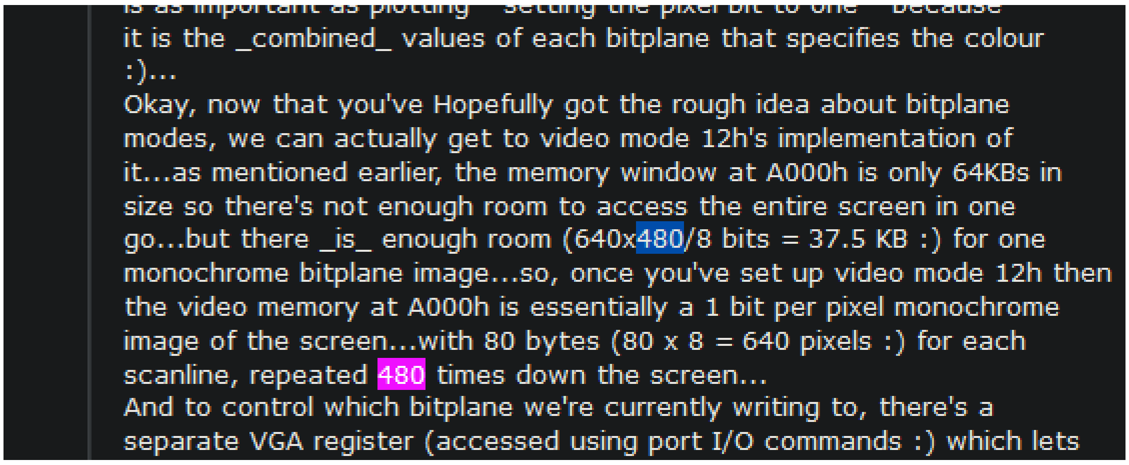
Electrohome ECM 1310	30	34	47-85
Sony Multiscan CPD 1302	25	34	50-100
Princeton Ultrasync	30	35	45-120

Table 1: Tolerances of a few VGA-compatible monitors

For example, consider the default ROM BIOS video mode 12h-that is, 640- by 480- pixel 16-color graphics mode. This mode is designed to work with IBM's PS/2-compatible monitors, which expect a horizontal scan rate of 31.5 KHz. The BIOS uses the VGA's 25.175-MHz dot clock in this video mode, so you can easily determine the relevant timing constraints.

To compute the horizontal total, you divide the dot rate by the horizontal scan rate to obtain the number of pixels per scan line. Then you divide this value by 8 (the number of pixels per character clock) to determine the number of character clocks per scan line:

http://computer-programming-forum.com/45-asm/5a88e2d82140a8ef.htm

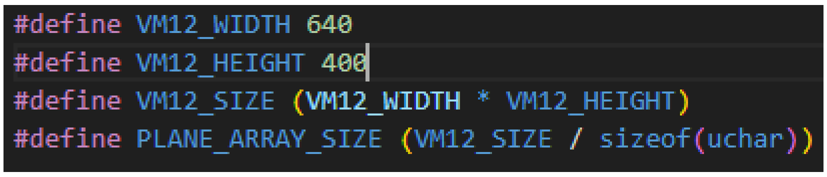


is as important as plotting "setting the pixel bit to one" because it is the _combined_ values of each bitplane that specifies the colour :)...
Okay, now that you've Hopefully got the rough idea about bitplane modes, we can actually get to video mode 12h's implementation of it...as mentioned earlier, the memory window at A000h is only 64KBs in size so there's not enough room to access the entire screen in one go...but there _is_ enough room (640x480/8 bits = 37.5 KB :) for one monochrome bitplane image...so, once you've set up video mode 12h then the video memory at A000h is essentially a 1 bit per pixel monochrome image of the screen...with 80 bytes (80 x 8 = 640 pixels :) for each scanline, repeated 480 times down the screen...
And to control which bitplane we're currently writing to, there's a separate VGA register (accessed using port I/O commands :) which lets

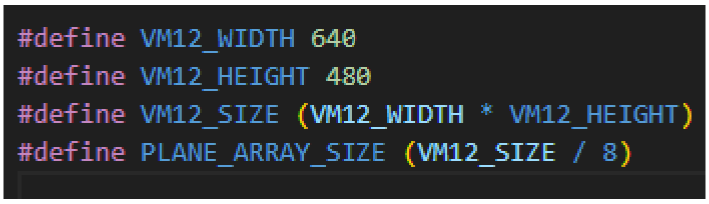
Using this new value, this now lines up with what I was expecting to see:

(640 / 8) * 480 = 38400
640 * (480/8) = 38400
(640 * 480) / 8 = 38400

It turns out my original plane size definition was wrong anyway because sizeof(uchar) is in bytes and not bits.

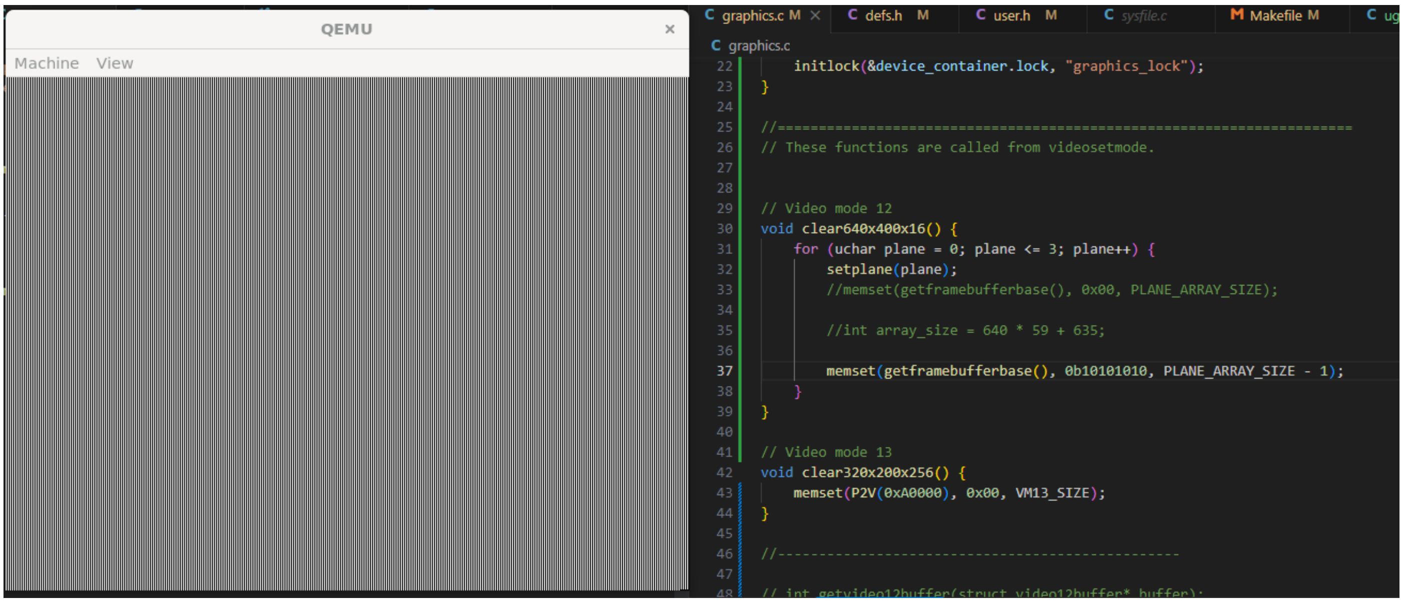


```c
#define VM12_WIDTH 640
#define VM12_HEIGHT 400
#define VM12_SIZE (VM12_WIDTH * VM12_HEIGHT)
#define PLANE_ARRAY_SIZE (VM12_SIZE / sizeof(uchar))
```

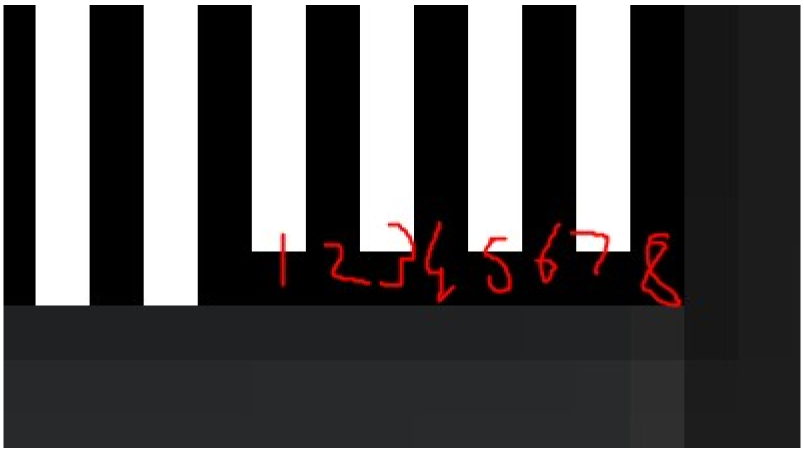


```c
#define VM12_WIDTH 640
#define VM12_HEIGHT 480
#define VM12_SIZE (VM12_WIDTH * VM12_HEIGHT)
#define PLANE_ARRAY_SIZE (VM12_SIZE / 8)
```

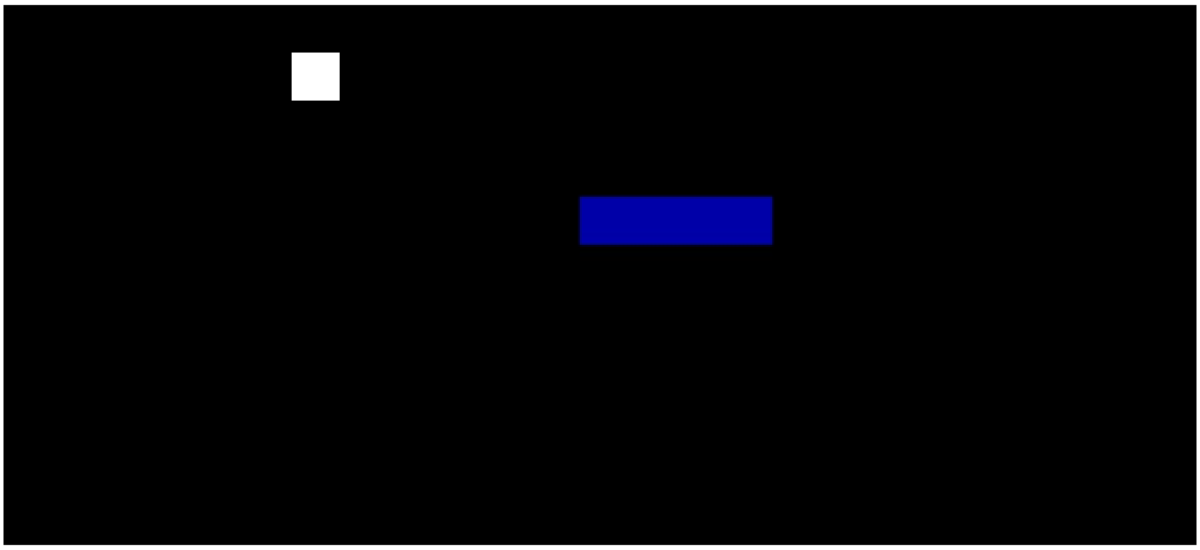
Fixing this definition and then using it in memset but removing 1 byte now works as intended, showing that 8 pixels are missing in the bottom right corner:



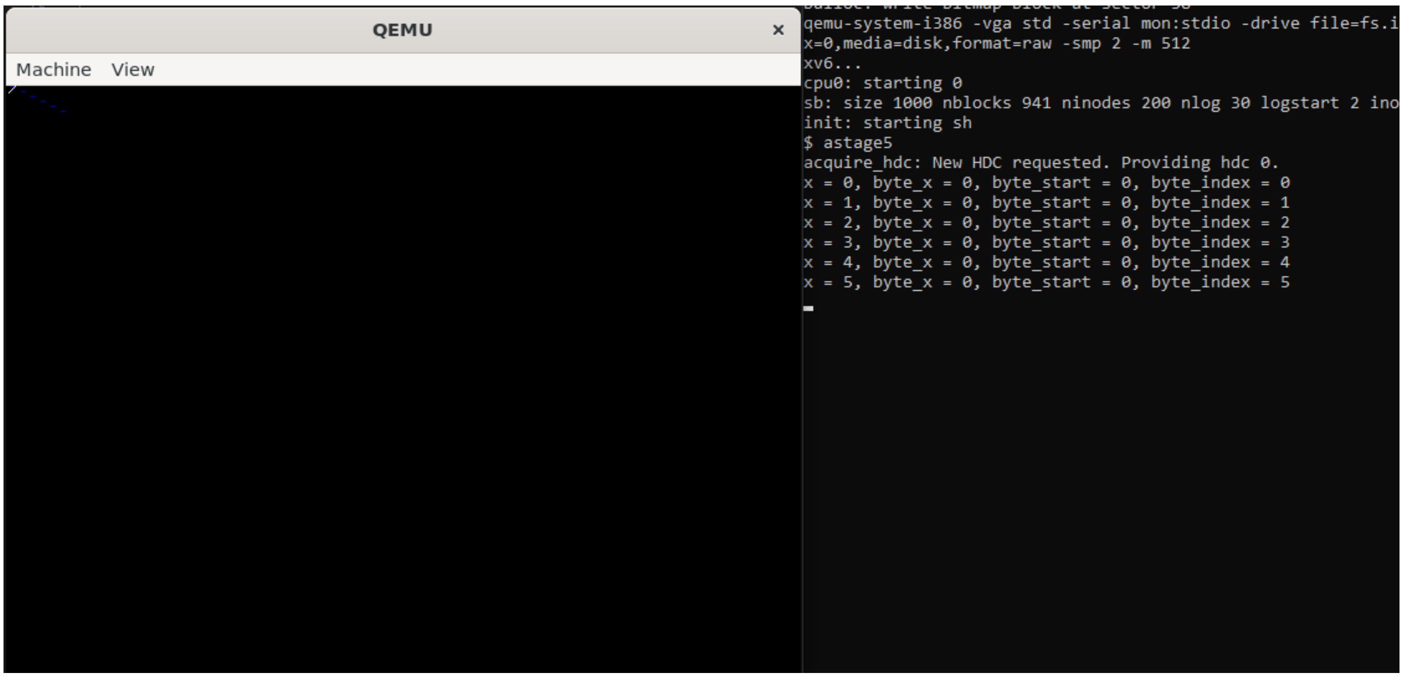
```c
    initlock(&device_container.lock, "graphics_lock");
}

//======================================================================
// These functions are called from videosetmode.


// Video mode 12
void clear640x400x16() {
    for (uchar plane = 0; plane <= 3; plane++) {
        setplane(plane);
        //memset(getframebufferbase(), 0x00, PLANE_ARRAY_SIZE);

        //int array_size = 640 * 59 + 635;

        memset(getframebufferbase(), 0b10101010, PLANE_ARRAY_SIZE - 1);
    }
}

// Video mode 13
void clear320x200x256() {
    memset(P2V(0xA0000), 0x00, VM13_SIZE);
}

//-------------------------------------------------

// int getvideo12buffer(struct video12buffer* buffer);
```



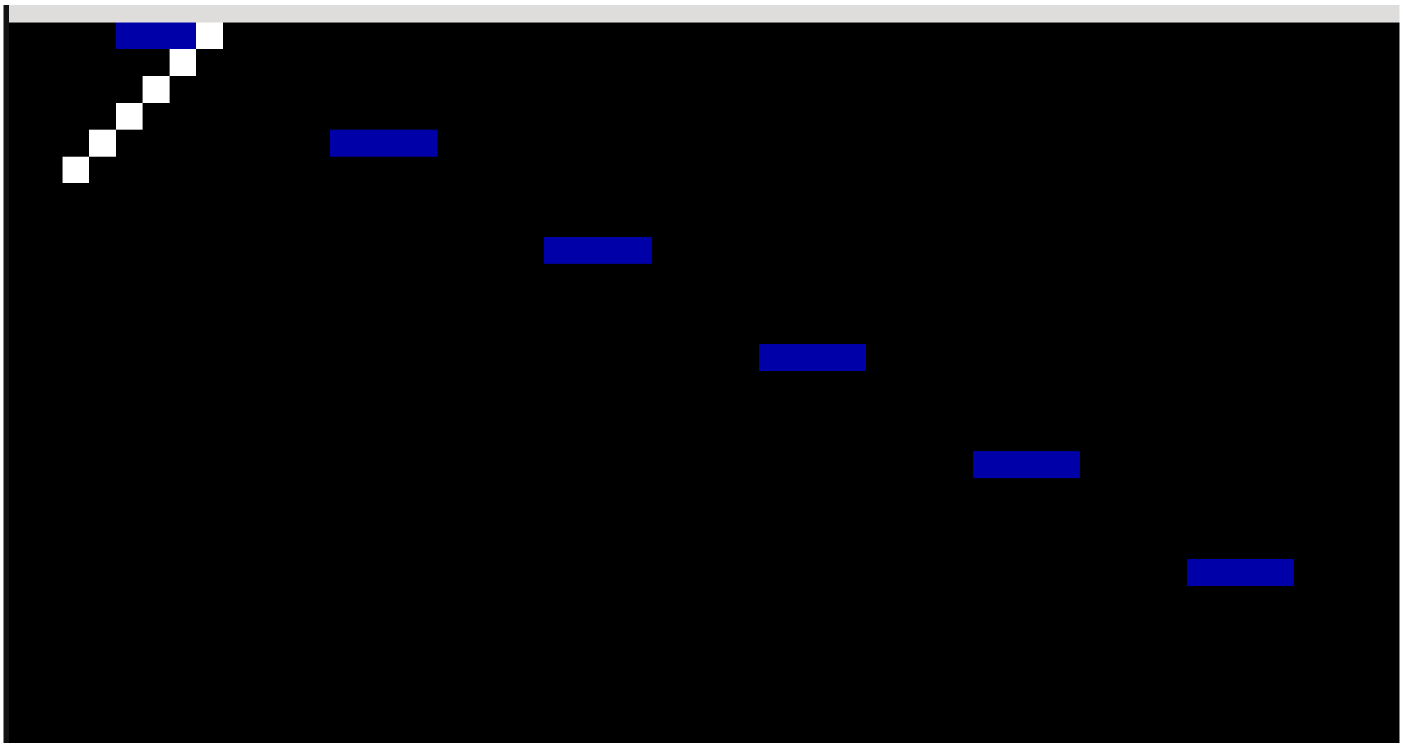
My first implementation of draw pixel for video mode 12h, trying to draw a pixel at 1, 1 gives me this result in the top left corner of my screen.



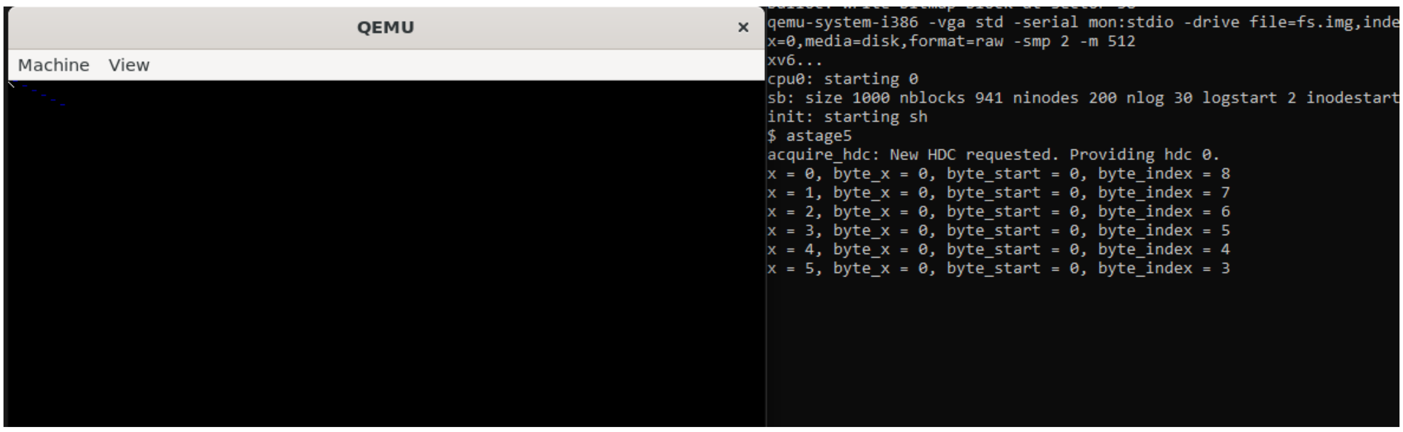
Using drawline from 0,0 to 5,5 gives this result:



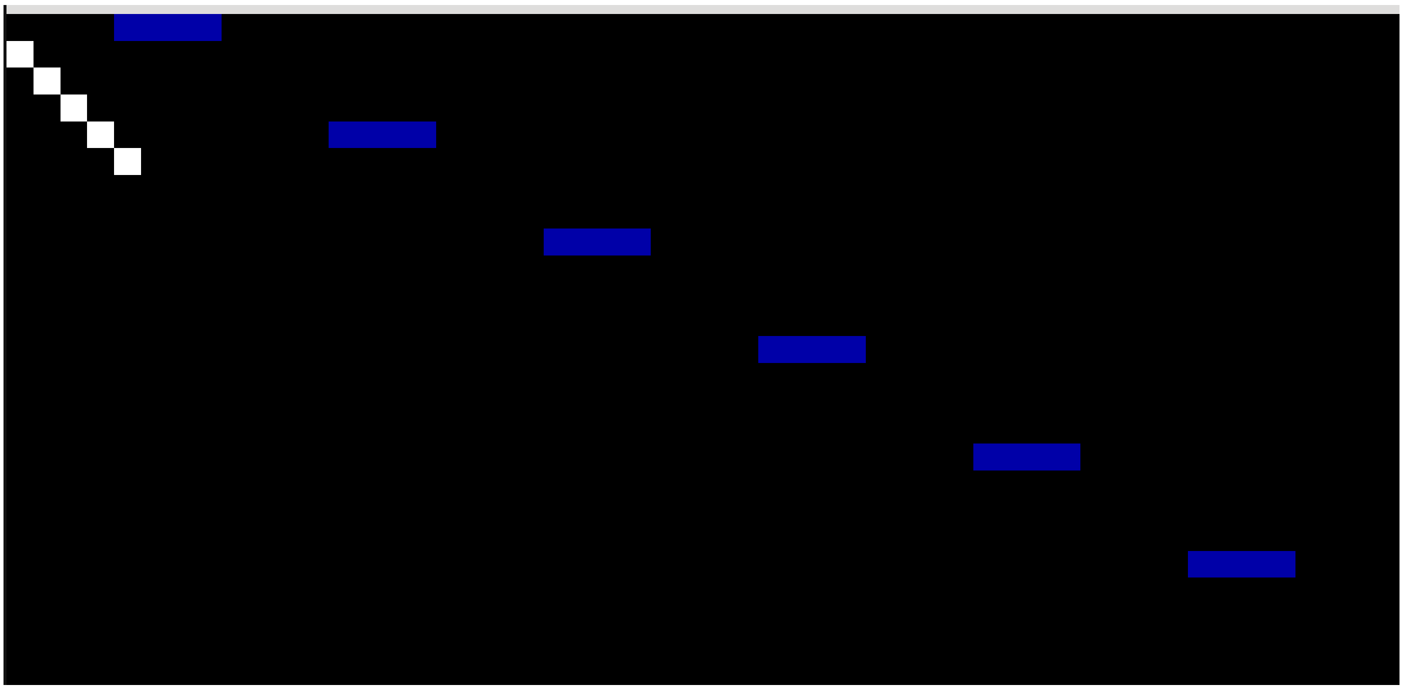
```
balloc: write bitmap block at sector 58
qemu-system-i386 -vga std -serial mon:stdio -drive file=fs.i
x=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 ino
init: starting sh
$ astage5
acquire_hdc: New HDC requested. Providing hdc 0.
x = 0, byte_x = 0, byte_start = 0, byte_index = 0
x = 1, byte_x = 0, byte_start = 0, byte_index = 1
x = 2, byte_x = 0, byte_start = 0, byte_index = 2
x = 3, byte_x = 0, byte_start = 0, byte_index = 3
x = 4, byte_x = 0, byte_start = 0, byte_index = 4
x = 5, byte_x = 0, byte_start = 0, byte_index = 5
```



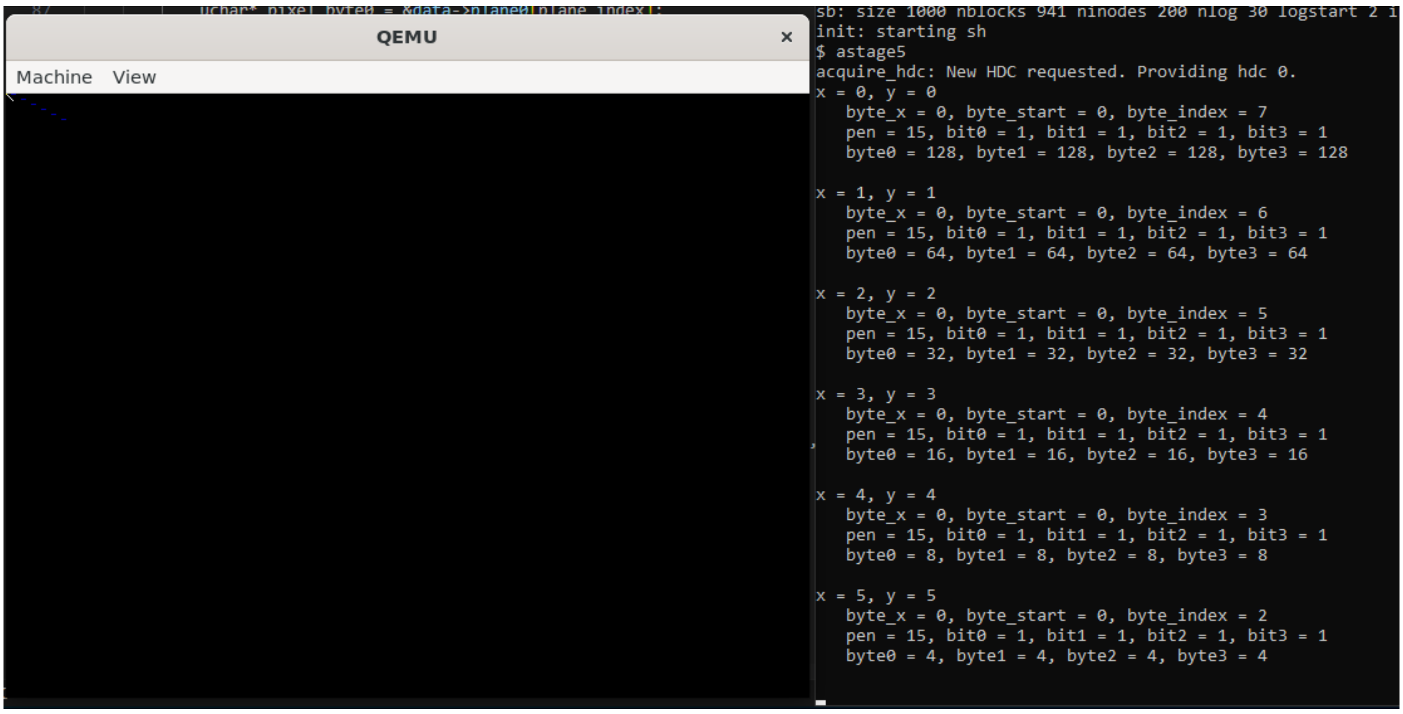
Flipping which bits in each byte we write gives this result:



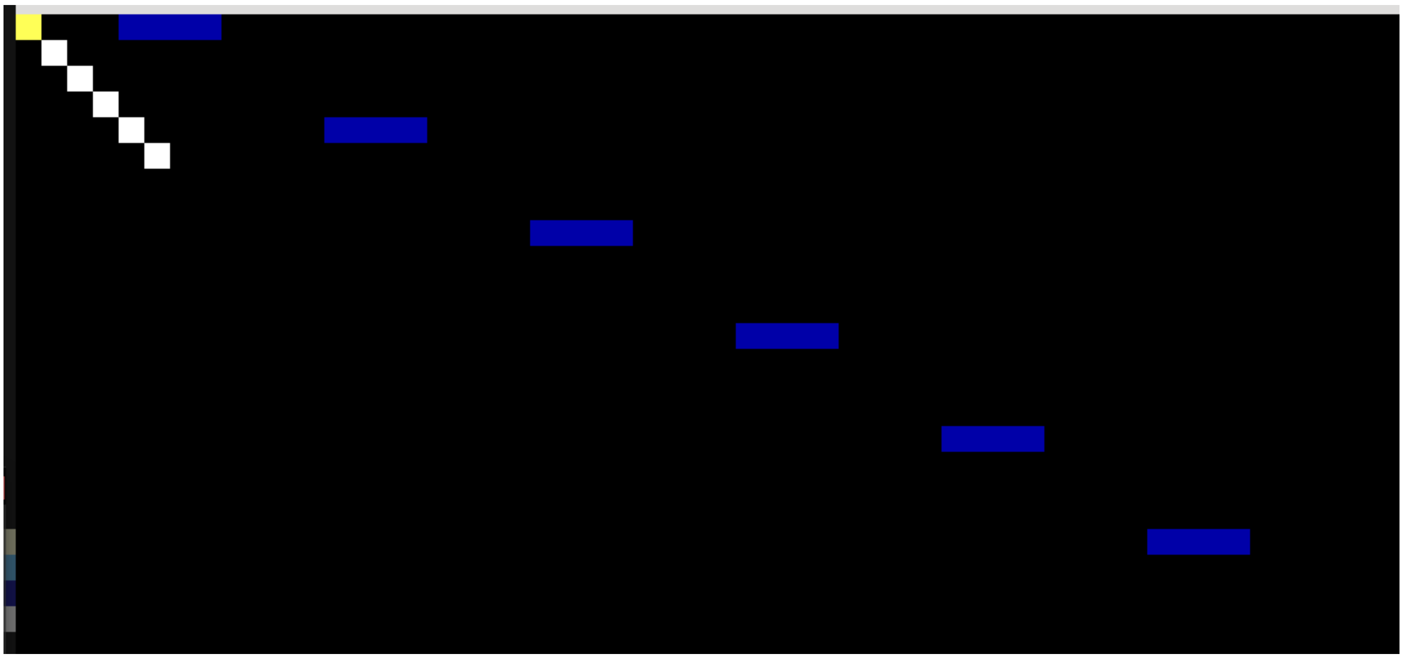
```
balloc: write bitmap block at sector 58
qemu-system-i386 -vga std -serial mon:stdio -drive file=fs.img,inde
x=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart
init: starting sh
$ astage5
acquire_hdc: New HDC requested. Providing hdc 0.
x = 0, byte_x = 0, byte_start = 0, byte_index = 8
x = 1, byte_x = 0, byte_start = 0, byte_index = 7
x = 2, byte_x = 0, byte_start = 0, byte_index = 6
x = 3, byte_x = 0, byte_start = 0, byte_index = 5
x = 4, byte_x = 0, byte_start = 0, byte_index = 4
x = 5, byte_x = 0, byte_start = 0, byte_index = 3
```



After adding more debug, I realized that byte_index should not be 8. It should be 0 to 7. Changing the code to reflect this now makes our line render in the correct place. We still have these blue dashes however and the first pixel is the wrong color.

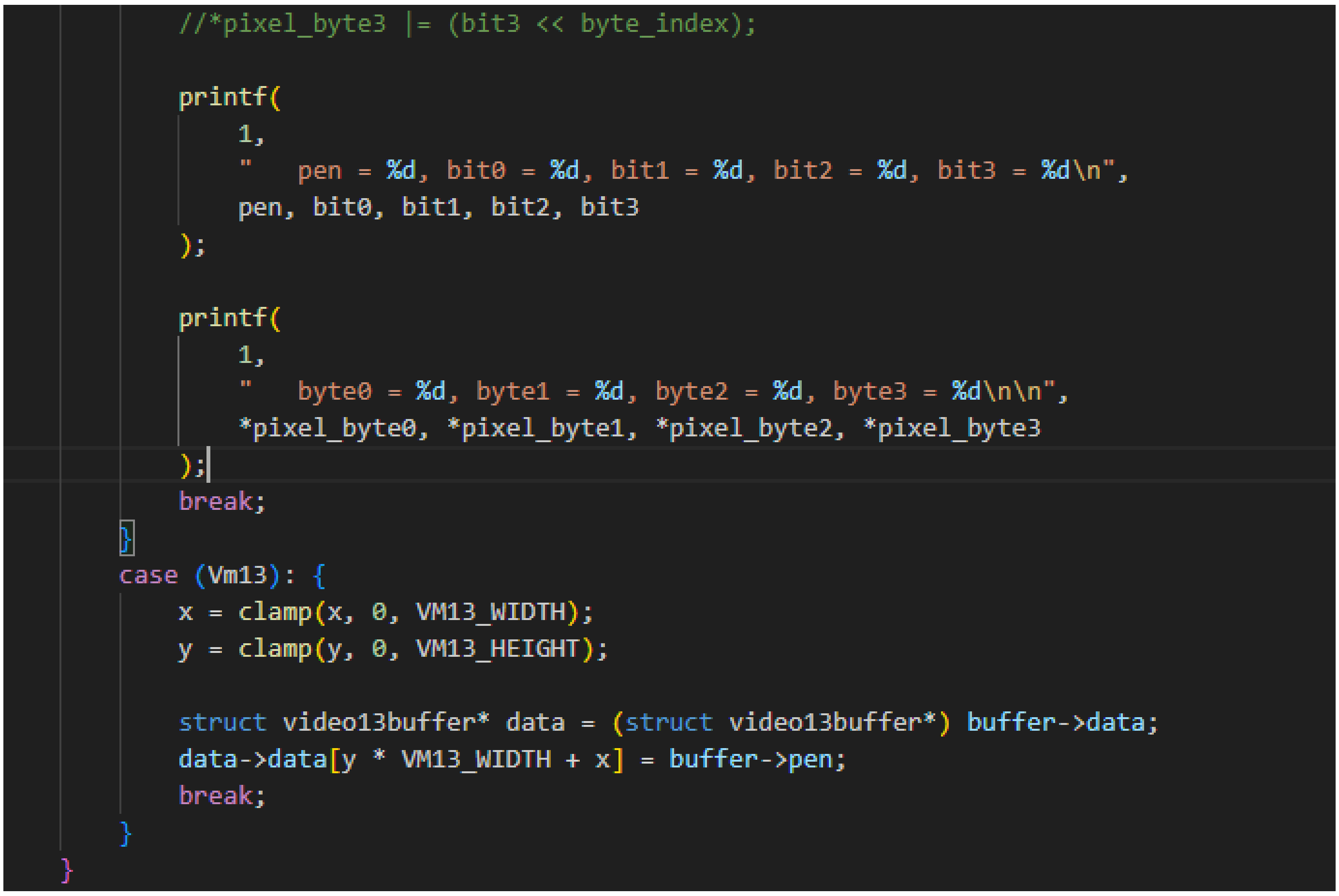


```
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 i
init: starting sh
$ astage5
acquire_hdc: New HDC requested. Providing hdc 0.
x = 0, y = 0
   byte_x = 0, byte_start = 0, byte_index = 7
   pen = 15, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 1
   byte0 = 128, byte1 = 128, byte2 = 128, byte3 = 128

x = 1, y = 1
   byte_x = 0, byte_start = 0, byte_index = 6
   pen = 15, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 1
   byte0 = 64, byte1 = 64, byte2 = 64, byte3 = 64

x = 2, y = 2
   byte_x = 0, byte_start = 0, byte_index = 5
   pen = 15, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 1
   byte0 = 32, byte1 = 32, byte2 = 32, byte3 = 32

x = 3, y = 3
   byte_x = 0, byte_start = 0, byte_index = 4
   pen = 15, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 1
   byte0 = 16, byte1 = 16, byte2 = 16, byte3 = 16

x = 4, y = 4
   byte_x = 0, byte_start = 0, byte_index = 3
   pen = 15, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 1
   byte0 = 8, byte1 = 8, byte2 = 8, byte3 = 8

x = 5, y = 5
   byte_x = 0, byte_start = 0, byte_index = 2
   pen = 15, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 1
   byte0 = 4, byte1 = 4, byte2 = 4, byte3 = 4
```



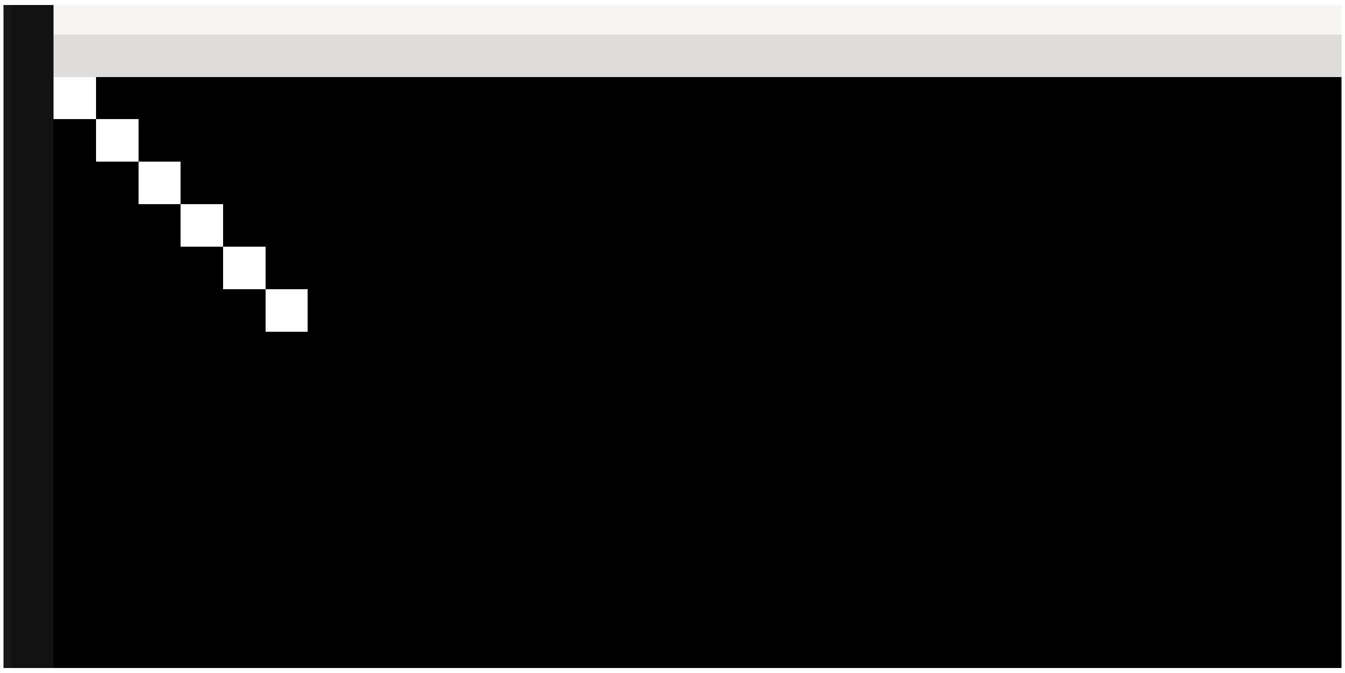
My debug into is saying the first byte has a value of 128.

Turns out the blue lines were because I forgot the break at the end of the a c switch and it was running the video mode 13h code as well.

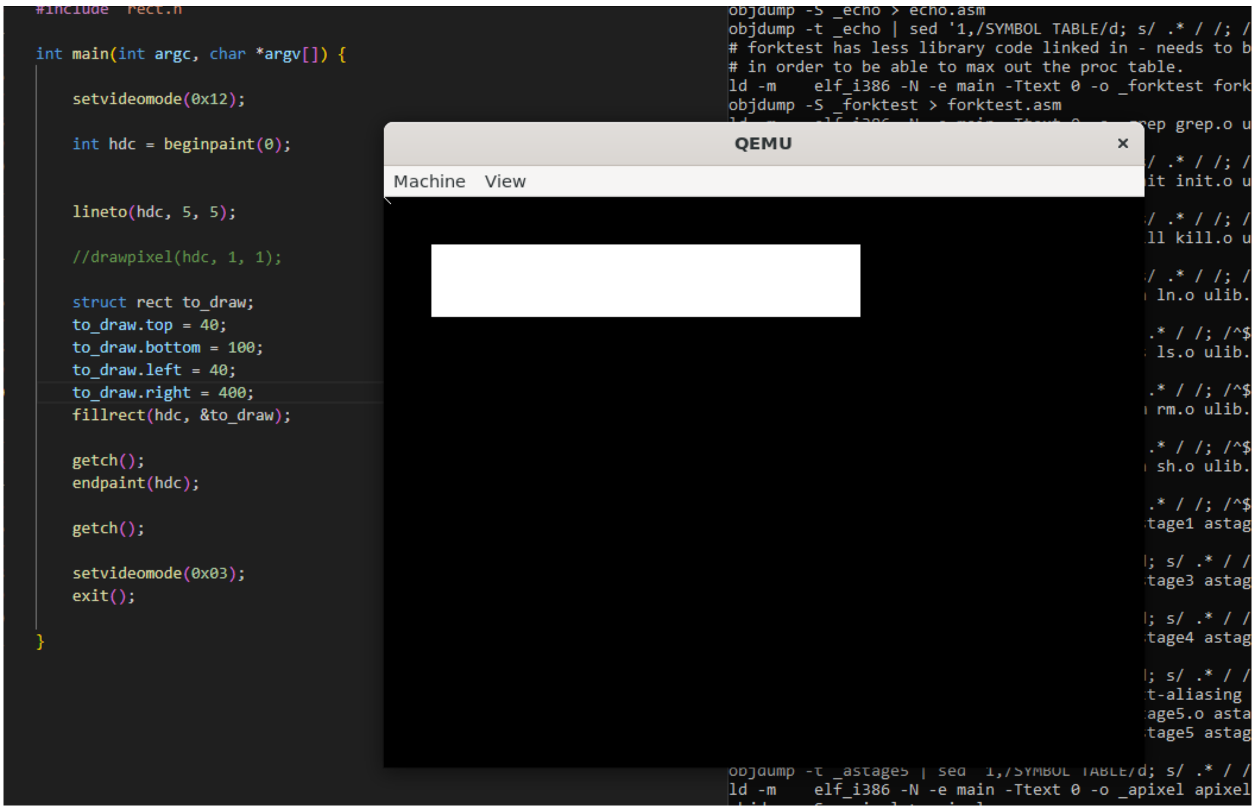


```
            //*pixel_byte3 |= (bit3 << byte_index);

            printf(
                1,
                "    pen = %d, bit0 = %d, bit1 = %d, bit2 = %d, bit3 = %d\n",
                pen, bit0, bit1, bit2, bit3
            );

            printf(
                1,
                "    byte0 = %d, byte1 = %d, byte2 = %d, byte3 = %d\n\n",
                *pixel_byte0, *pixel_byte1, *pixel_byte2, *pixel_byte3
            );
            break;
        }
        case (Vm13): {
            x = clamp(x, 0, VM13_WIDTH);
            y = clamp(y, 0, VM13_HEIGHT);

            struct video13buffer* data = (struct video13buffer*) buffer->data;
            data->data[y * VM13_WIDTH + x] = buffer->pen;
            break;
        }
    }
```

This fixes the blue and yellow problem:



Testing the other functions:

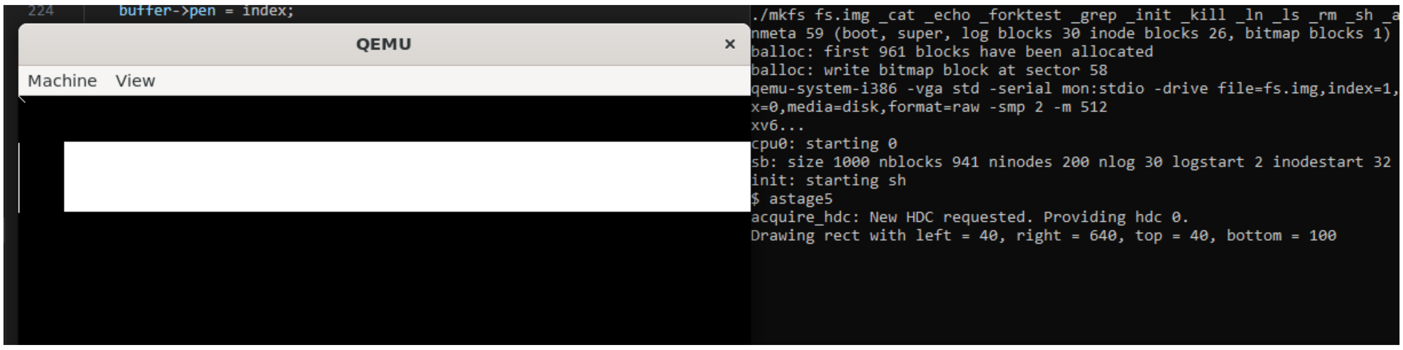


```c
#include "rect.h"

int main(int argc, char *argv[]) {

    setvideomode(0x12);

    int hdc = beginpaint(0);


    lineto(hdc, 5, 5);

    //drawpixel(hdc, 1, 1);

    struct rect to_draw;
    to_draw.top = 40;
    to_draw.bottom = 100;
    to_draw.left = 40;
    to_draw.right = 400;
    fillrect(hdc, &to_draw);

    getch();
    endpaint(hdc);

    getch();

    setvideomode(0x03);
    exit();

}
```

```
objdump -S _echo > echo.asm
objdump -t _echo | sed '1,/SYMBOL TABLE/d; s/ .* / /; /
# forktest has less library code linked in - needs to b
# in order to be able to max out the proc table.
ld -m    elf_i386 -N -e main -Ttext 0 -o _forktest fork
objdump -S _forktest > forktest.asm
```

```
ld -m    elf_i386 -N -e main -Ttext 0 -o _apixel apixel
```

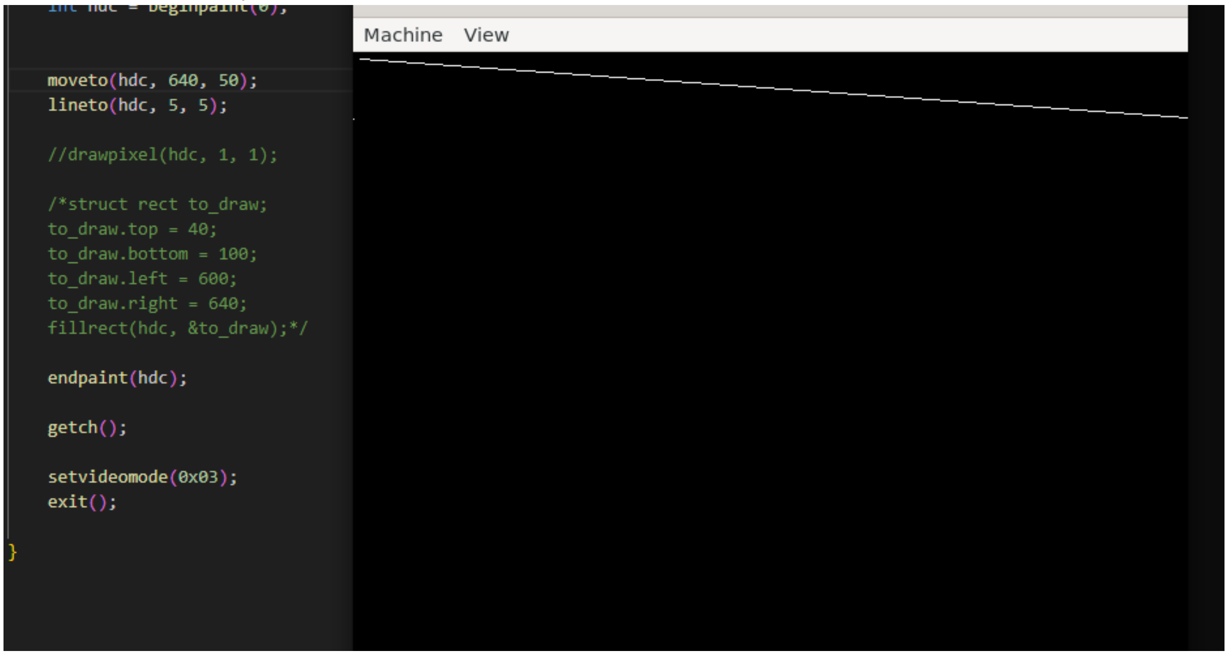
Apparently, I’m not clamping my rects properly.



```
buffer->pen = index;
```

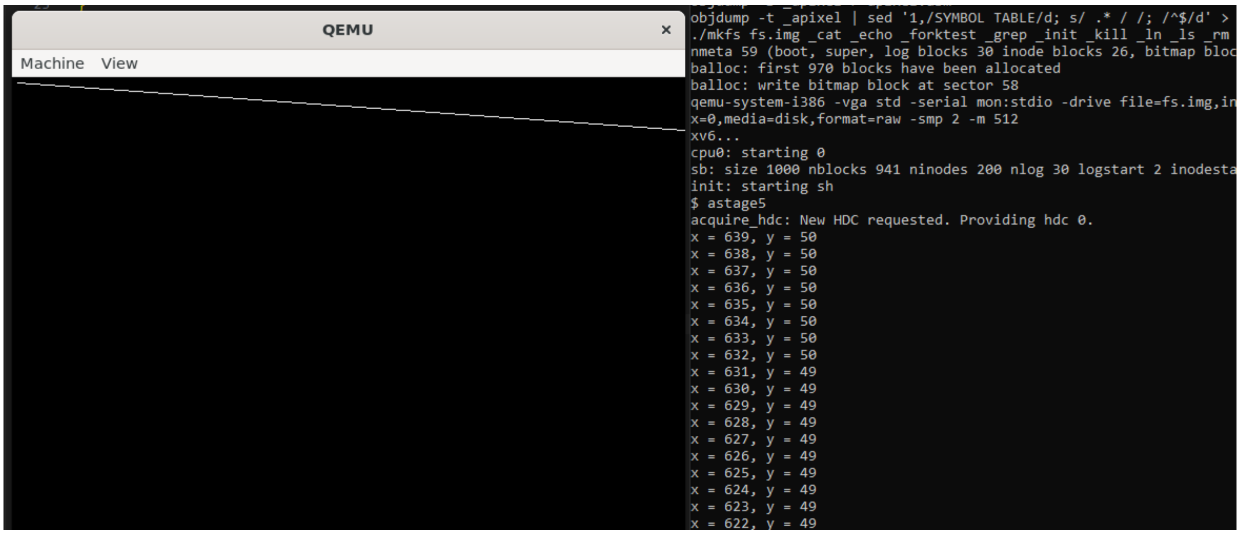
```
./mkfs fs.img _cat _echo _forktest _grep _init _kill _ln _ls _rm _sh _a
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1)
balloc: first 961 blocks have been allocated
balloc: write bitmap block at sector 58
qemu-system-i386 -vga std -serial mon:stdio -drive file=fs.img,index=1,
x=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32
init: starting sh
$ astage5
acquire_hdc: New HDC requested. Providing hdc 0.
Drawing rect with left = 40, right = 640, top = 40, bottom = 100
```

And the same for my lines as well.



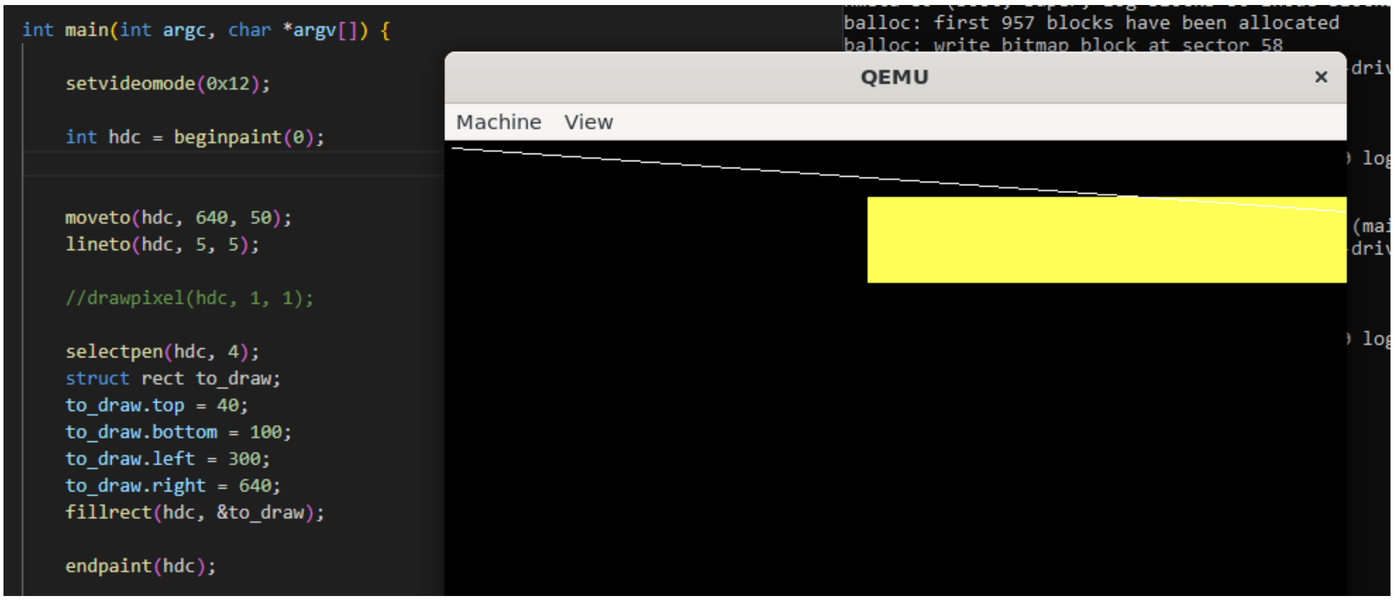
```
    int hdc = beginpaint(0);


    moveto(hdc, 640, 50);
    lineto(hdc, 5, 5);

    //drawpixel(hdc, 1, 1);

    /*struct rect to_draw;
    to_draw.top = 40;
    to_draw.bottom = 100;
    to_draw.left = 600;
    to_draw.right = 640;
    fillrect(hdc, &to_draw);*/

    endpaint(hdc);

    getch();

    setvideomode(0x03);
    exit();

}
```

It turns out I wasn't subtracting 1 from the width and heights when clamping. Doing so solved this issue. Same applies for vm13.



```
objdump -t _apixel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' >
./mkfs fs.img _cat _echo _forktest _grep _init _kill _ln _ls _rm
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap bloc
balloc: first 970 blocks have been allocated
balloc: write bitmap block at sector 58
qemu-system-i386 -vga std -serial mon:stdio -drive file=fs.img,in
x=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodesta
init: starting sh
$ astage5
acquire_hdc: New HDC requested. Providing hdc 0.
x = 639, y = 50
x = 638, y = 50
x = 637, y = 50
x = 636, y = 50
x = 635, y = 50
x = 634, y = 50
x = 633, y = 50
x = 632, y = 50
x = 631, y = 49
x = 630, y = 49
x = 629, y = 49
x = 628, y = 49
x = 627, y = 49
x = 626, y = 49
x = 625, y = 49
x = 624, y = 49
x = 623, y = 49
x = 622, y = 49
```

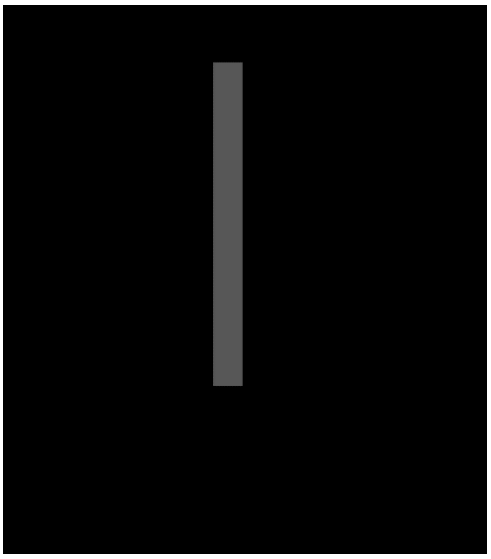
Now testing color, it seems that the wrong color was displayed. To my knowledge, color 4 should be red whilst this rectangle is yellow.



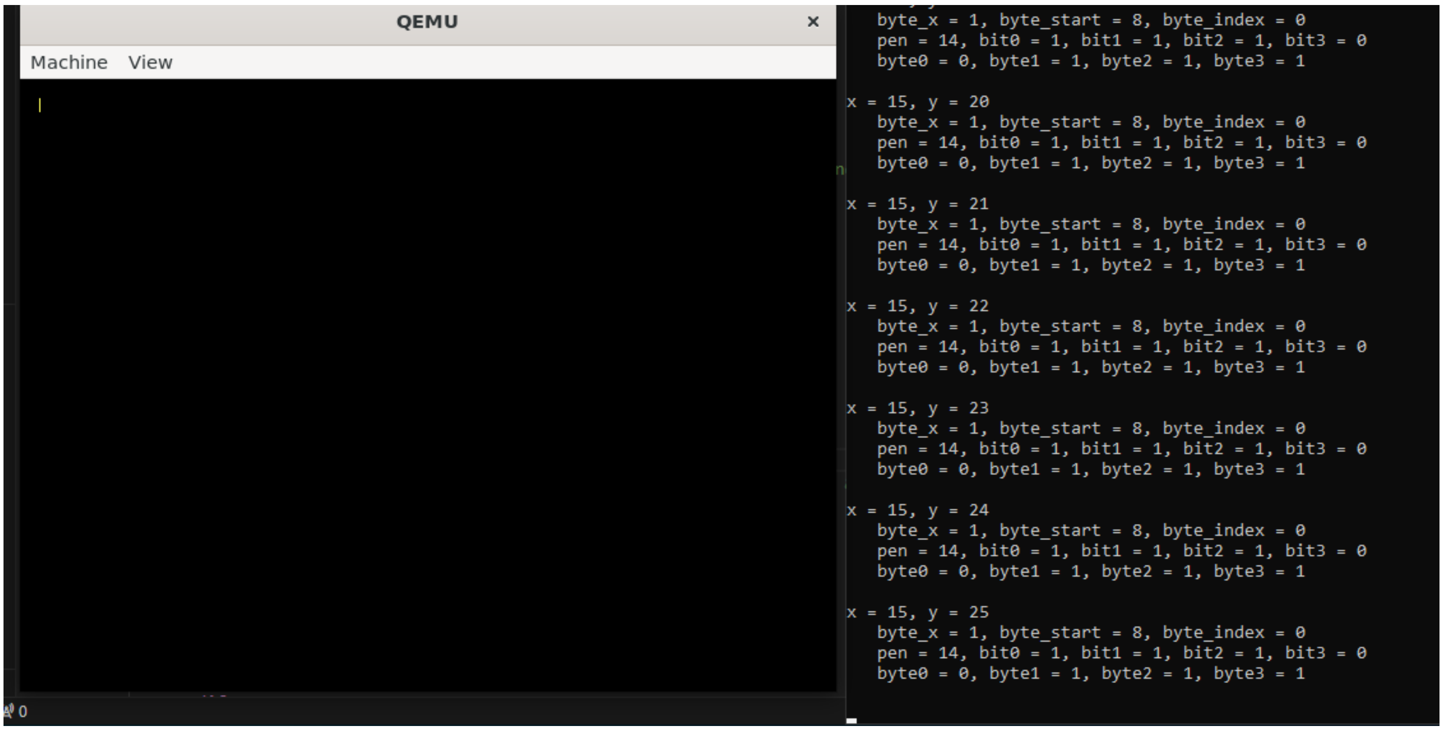
```
int main(int argc, char *argv[]) {

    setvideomode(0x12);

    int hdc = beginpaint(0);


    moveto(hdc, 640, 50);
    lineto(hdc, 5, 5);

    //drawpixel(hdc, 1, 1);

    selectpen(hdc, 4);
    struct rect to_draw;
    to_draw.top = 40;
    to_draw.bottom = 100;
    to_draw.left = 300;
    to_draw.right = 640;
    fillrect(hdc, &to_draw);

    endpaint(hdc);
```

```
balloc: first 957 blocks have been allocated
balloc: write bitmap block at sector 58
```

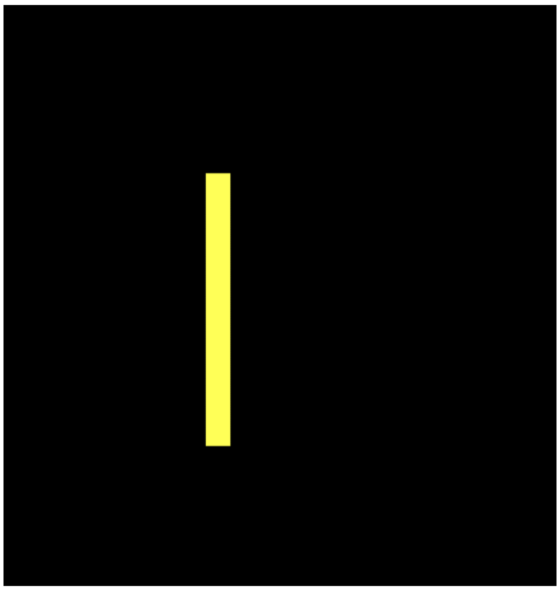
Drawing a line that shouldn't be gray:



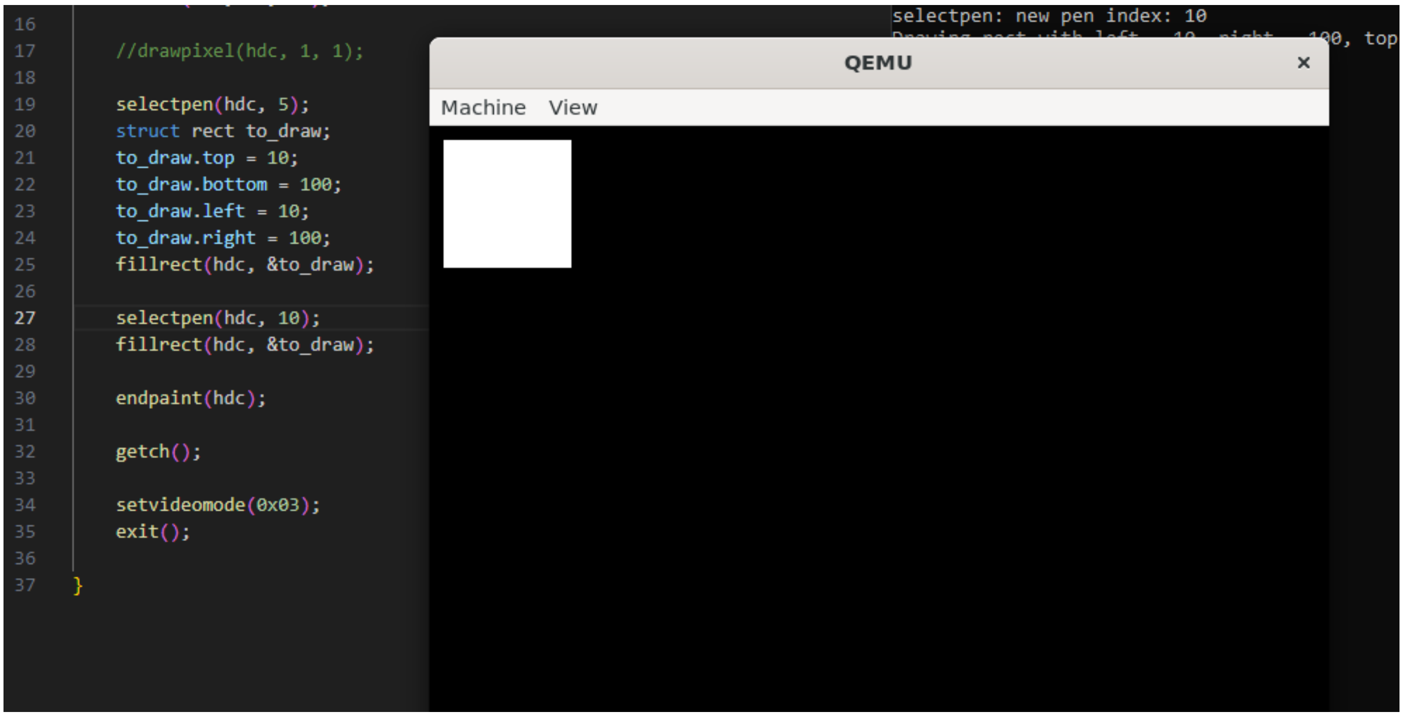
Turns out I was using && instead of & for masking the bits from the pen. As such, it was just returning 1 for everything. Changing this now masks the bits correctly. Looking at the debug info below, the line beginning "pen = 14" shows what each for the 4 bits are equal to which is the correct color code for yellow: 0, 1, 1, 1 (going from plane 0 -> 3).



```
   byte_x = 1, byte_start = 8, byte_index = 0
   pen = 14, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 0
   byte0 = 0, byte1 = 1, byte2 = 1, byte3 = 1

x = 15, y = 20
   byte_x = 1, byte_start = 8, byte_index = 0
   pen = 14, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 0
   byte0 = 0, byte1 = 1, byte2 = 1, byte3 = 1

x = 15, y = 21
   byte_x = 1, byte_start = 8, byte_index = 0
   pen = 14, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 0
   byte0 = 0, byte1 = 1, byte2 = 1, byte3 = 1

x = 15, y = 22
   byte_x = 1, byte_start = 8, byte_index = 0
   pen = 14, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 0
   byte0 = 0, byte1 = 1, byte2 = 1, byte3 = 1

x = 15, y = 23
   byte_x = 1, byte_start = 8, byte_index = 0
   pen = 14, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 0
   byte0 = 0, byte1 = 1, byte2 = 1, byte3 = 1

x = 15, y = 24
   byte_x = 1, byte_start = 8, byte_index = 0
   pen = 14, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 0
   byte0 = 0, byte1 = 1, byte2 = 1, byte3 = 1

x = 15, y = 25
   byte_x = 1, byte_start = 8, byte_index = 0
   pen = 14, bit0 = 1, bit1 = 1, bit2 = 1, bit3 = 0
   byte0 = 0, byte1 = 1, byte2 = 1, byte3 = 1
```



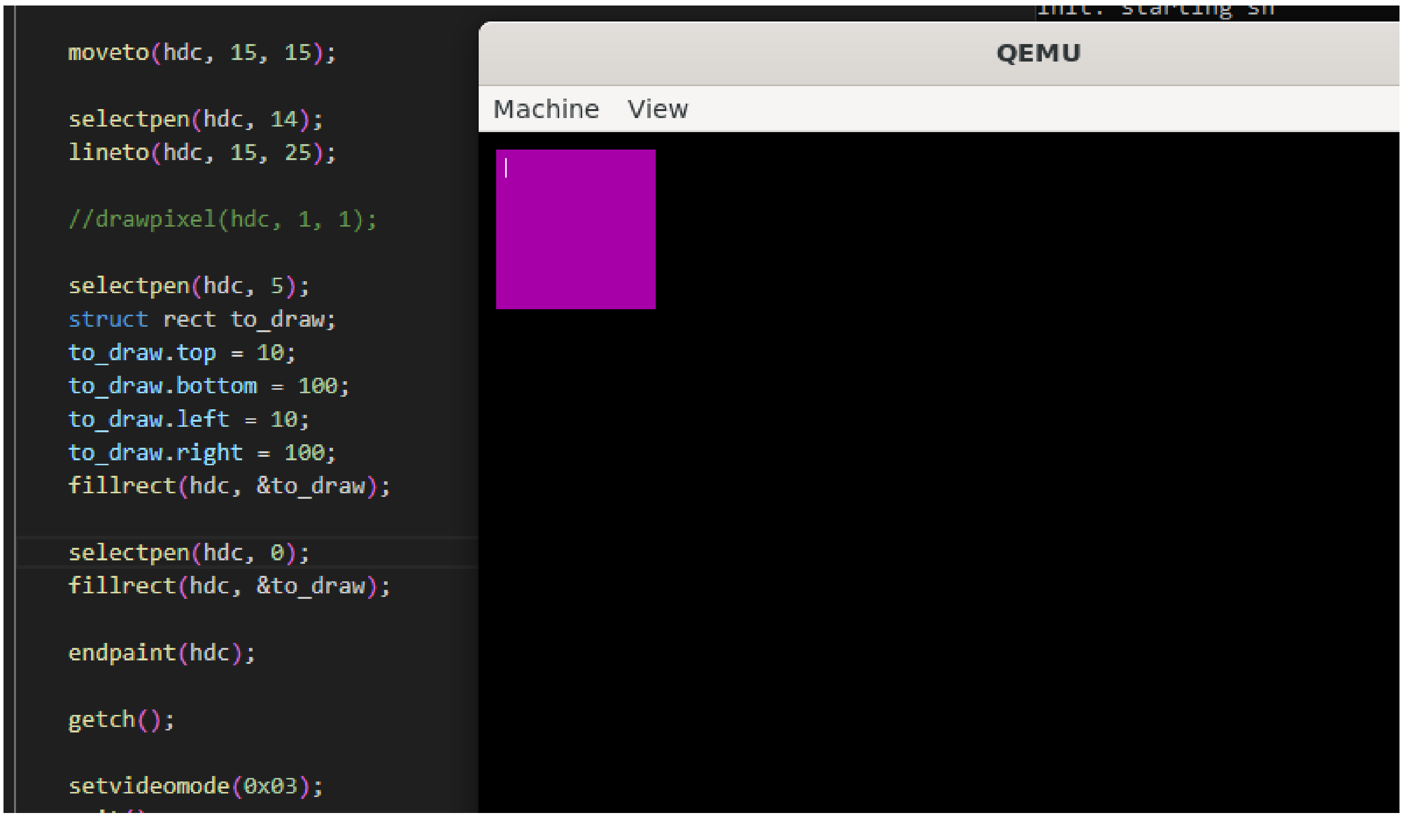
I had a feeling that the set bit functionality was more of an addition than a set and as seen here, setting the same rect twice, once as pen color 5 (0101) and again as pen color 10 (1010), it makes white (1111).



```
    //drawpixel(hdc, 1, 1);

    selectpen(hdc, 5);
    struct rect to_draw;
    to_draw.top = 10;
    to_draw.bottom = 100;
    to_draw.left = 10;
    to_draw.right = 100;
    fillrect(hdc, &to_draw);

    selectpen(hdc, 10);
    fillrect(hdc, &to_draw);

    endpaint(hdc);

    getch();

    setvideomode(0x03);
    exit();
}
```

```
selectpen: new pen index: 10
```

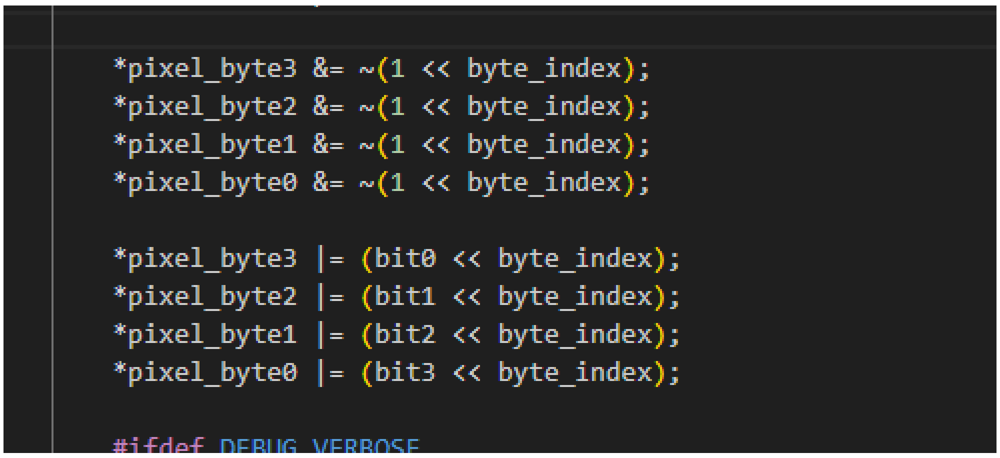
And again, drawing a black rectangle over an existing rectangle doesn't clear it.



```
    moveto(hdc, 15, 15);

    selectpen(hdc, 14);
    lineto(hdc, 15, 25);

    //drawpixel(hdc, 1, 1);

    selectpen(hdc, 5);
    struct rect to_draw;
    to_draw.top = 10;
    to_draw.bottom = 100;
    to_draw.left = 10;
    to_draw.right = 100;
    fillrect(hdc, &to_draw);

    selectpen(hdc, 0);
    fillrect(hdc, &to_draw);

    endpaint(hdc);

    getch();

    setvideomode(0x03);
```

```
init: starting sh
```

Clearing the bit before fixes this issue however my draw pixel section for video mode 12 just feels so verbose. It's so much more complicated than the vm13 and I feel like there's a much simpler solution but I'm not sure yet. For now through, it works and I'm mostly happy with it.

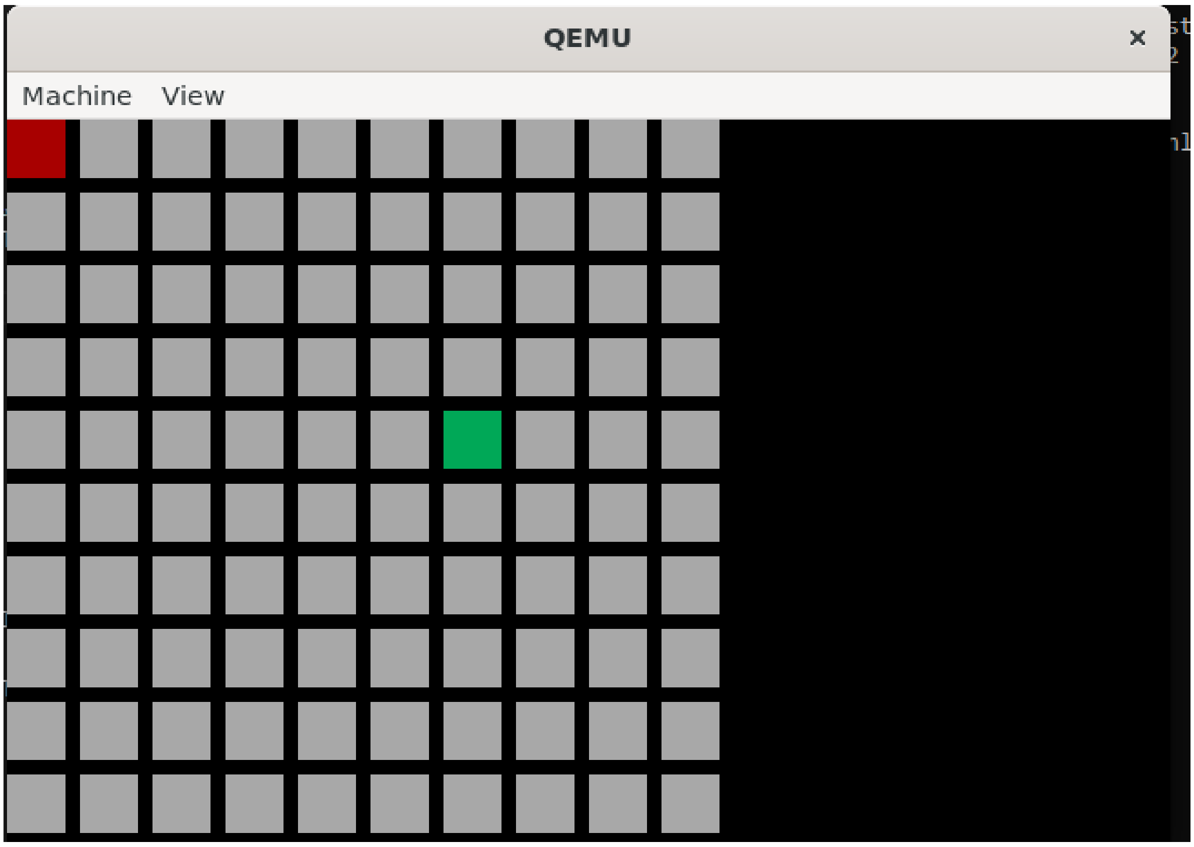


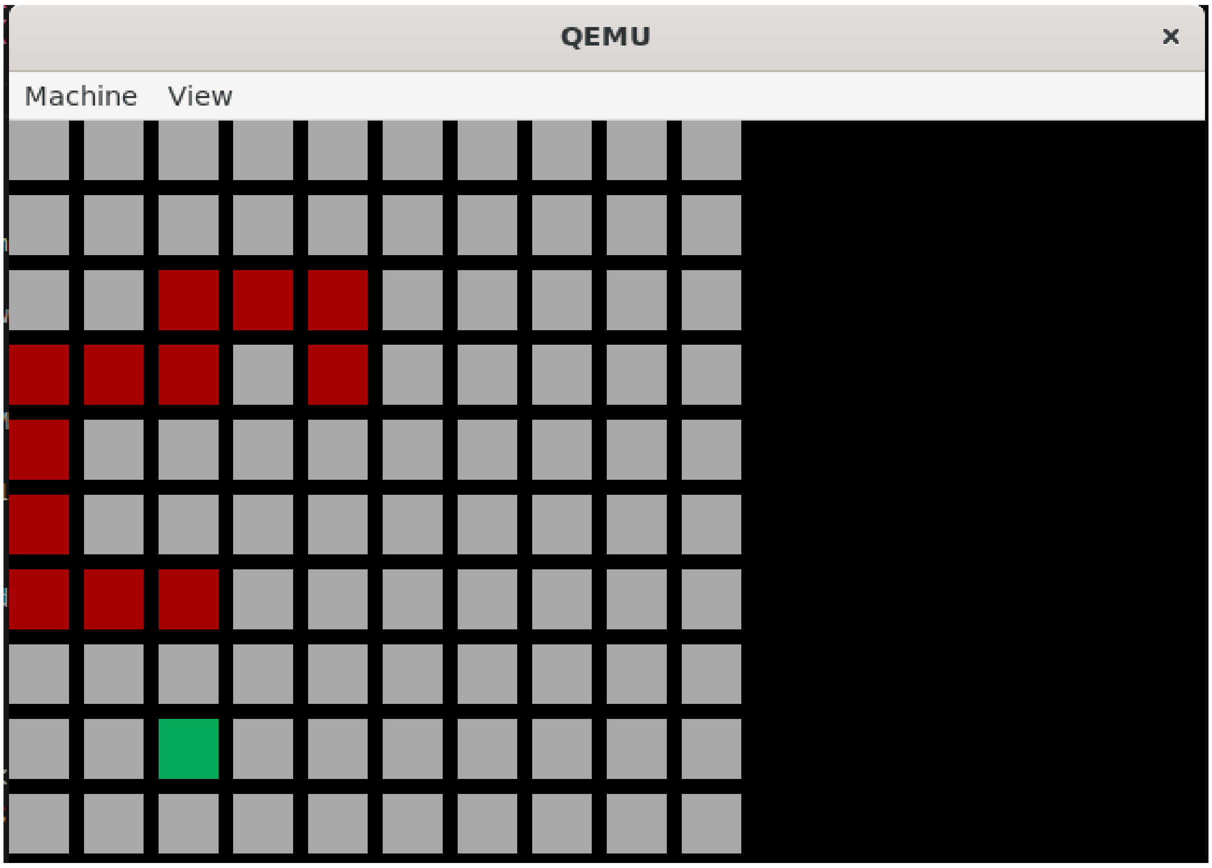
```
    *pixel_byte3 &= ~(1 << byte_index);
    *pixel_byte2 &= ~(1 << byte_index);
    *pixel_byte1 &= ~(1 << byte_index);
    *pixel_byte0 &= ~(1 << byte_index);

    *pixel_byte3 |= (bit0 << byte_index);
    *pixel_byte2 |= (bit1 << byte_index);
    *pixel_byte1 |= (bit2 << byte_index);
    *pixel_byte0 |= (bit3 << byte_index);

    #ifdef DEBUG_VERBOSE
```

15/12/23

As a games programming student, I thought it might be fun to try and do something relating to that in this module. As such, I tried implementing snake in vm12. For the most part it works. It only runs when the user inputs a key as I didn't know how to check for key pressed yes/no and I didn't want to spend a huge amount of time on it if it didn't relate to the graphics functionality of the assessment.

I made a rand function from multiplying a large number by the current system time * a provided seed and then getting the modulus of that result.

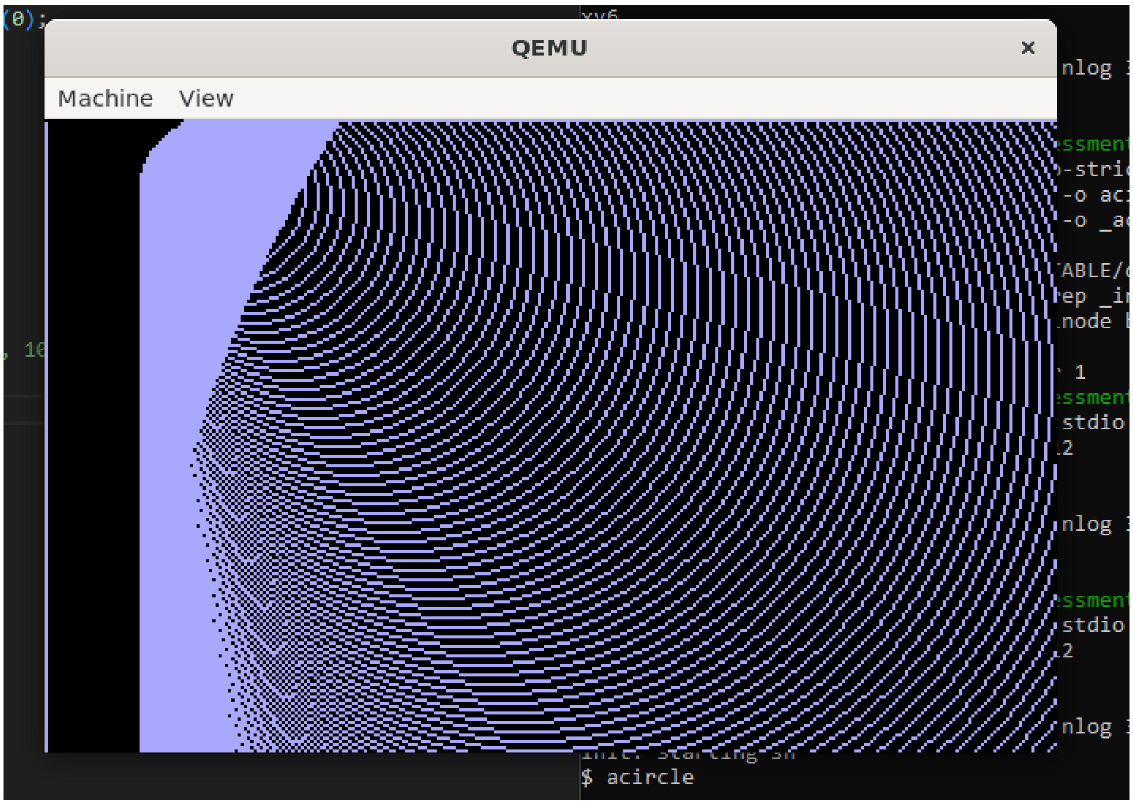




30/12/23

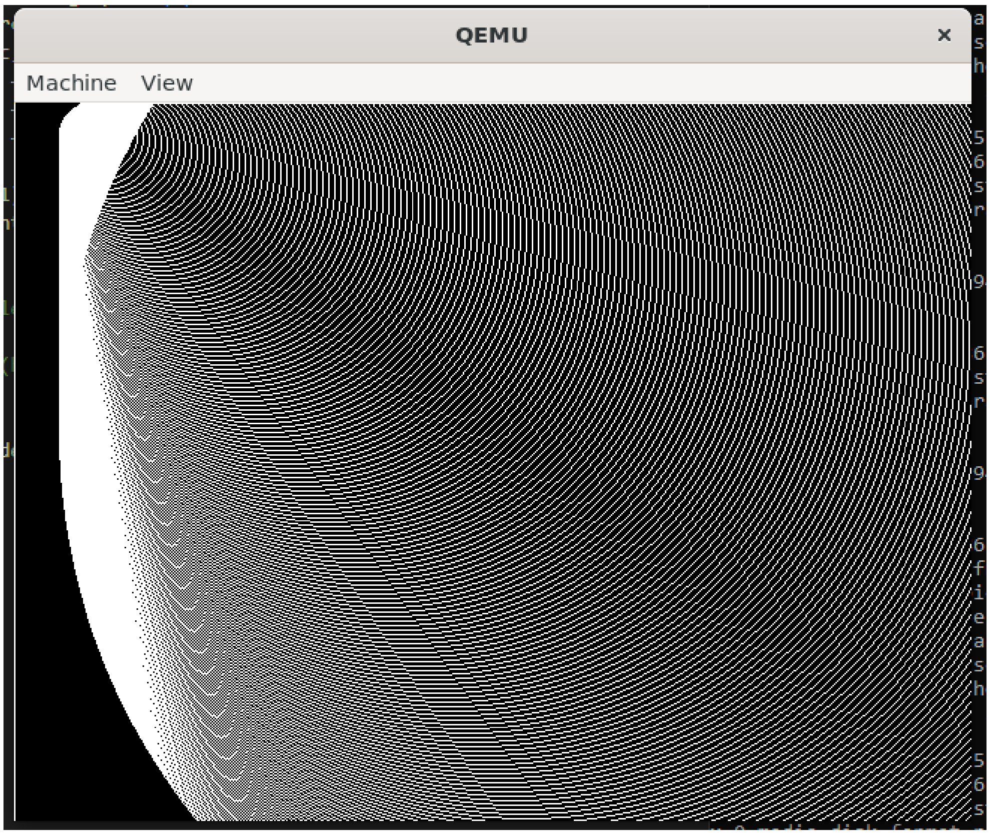
I wanted to add circle drawing functionality as the last big thing. I used Bradenham's circle drawing algorithm as it made sense since the lack of floats.

Vm13

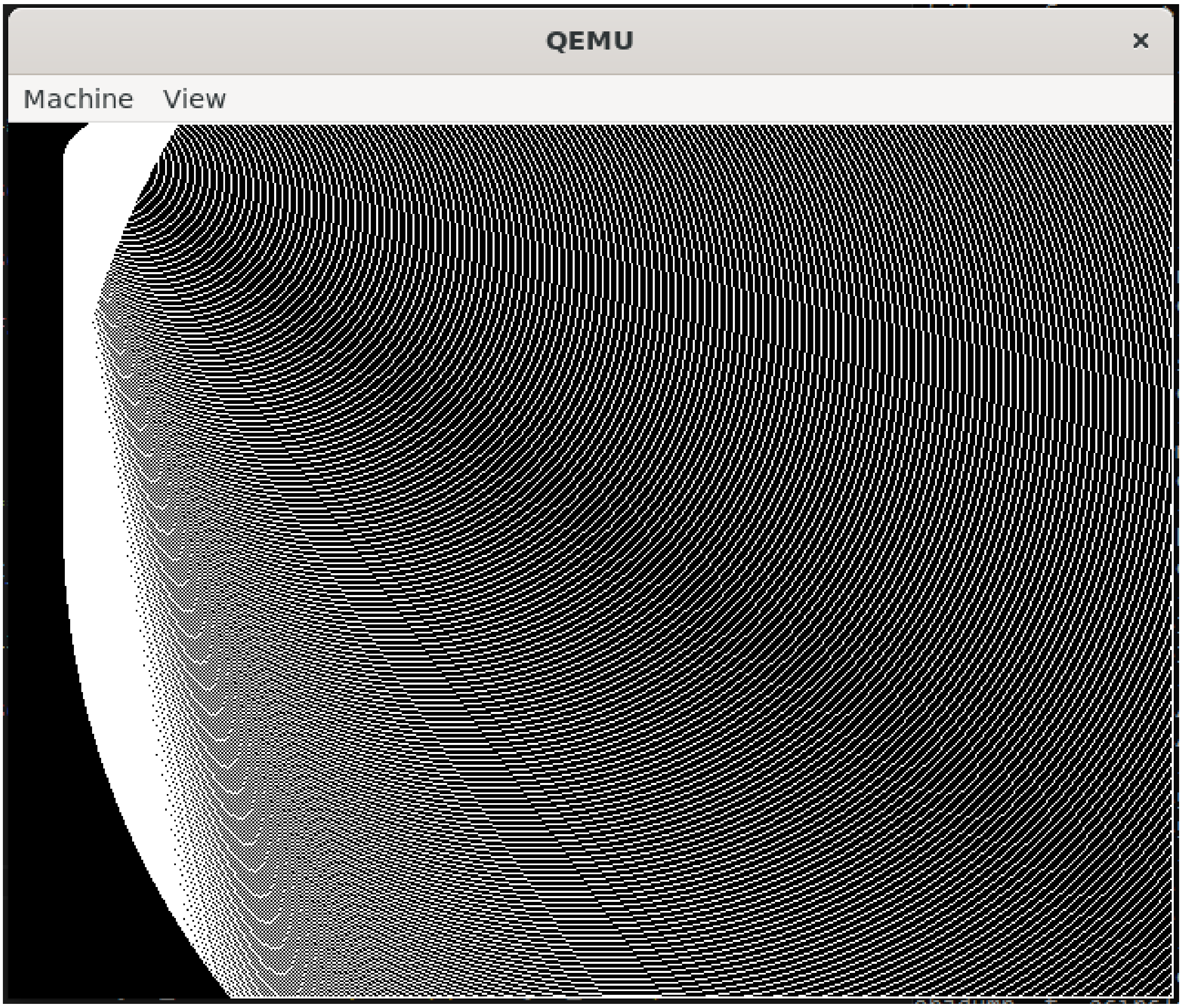


```
$ acircle
```

Vm12



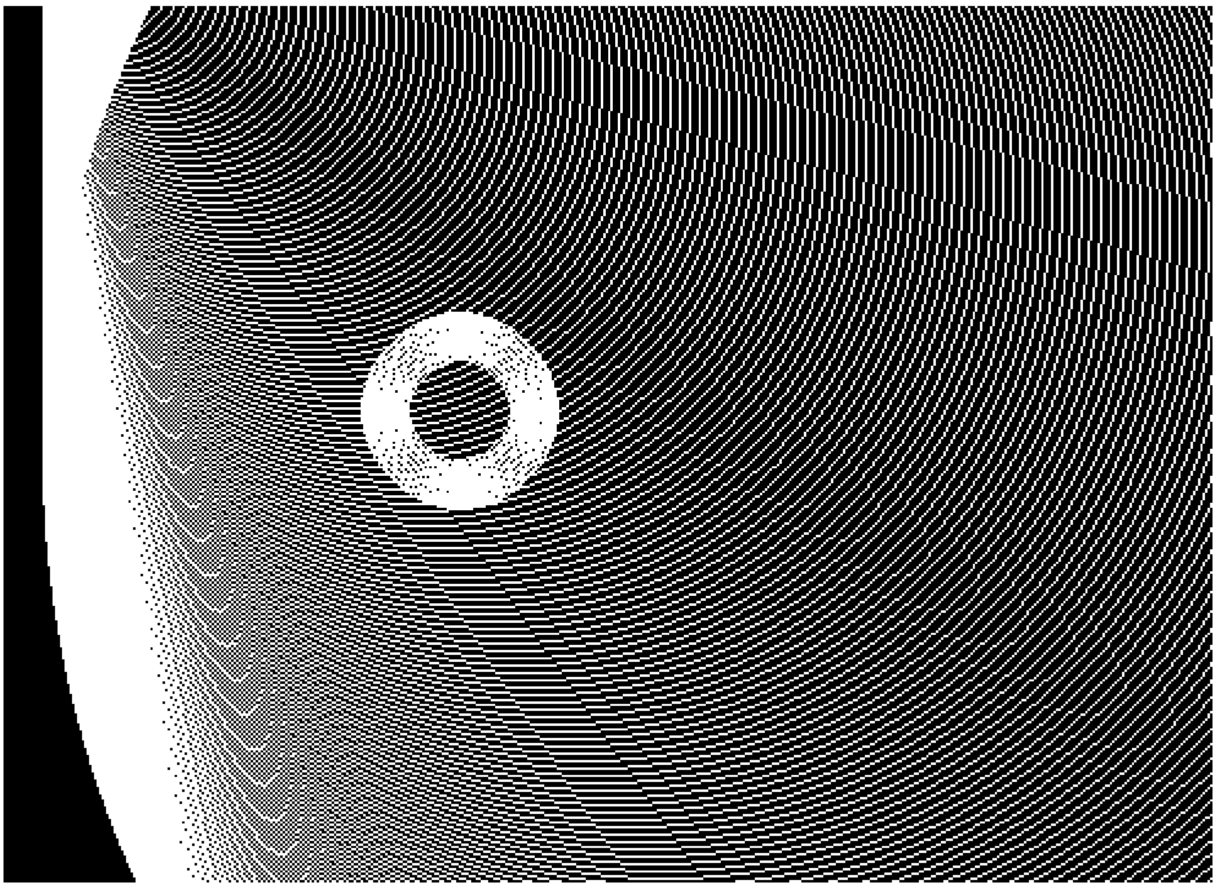
On the above images, apparently at some point, I'd undone the setpixel clamping as pixels were appearing on the left of the screen when they shouldn't have been. Clamping this value properly prevents a wrap around however it also gives me a border where pixels were clamped:



This made me think, why even bother clamping the value. If it's out of bounds, why not just return from the function. Especially in vm12 where the setpixel is much more complex.

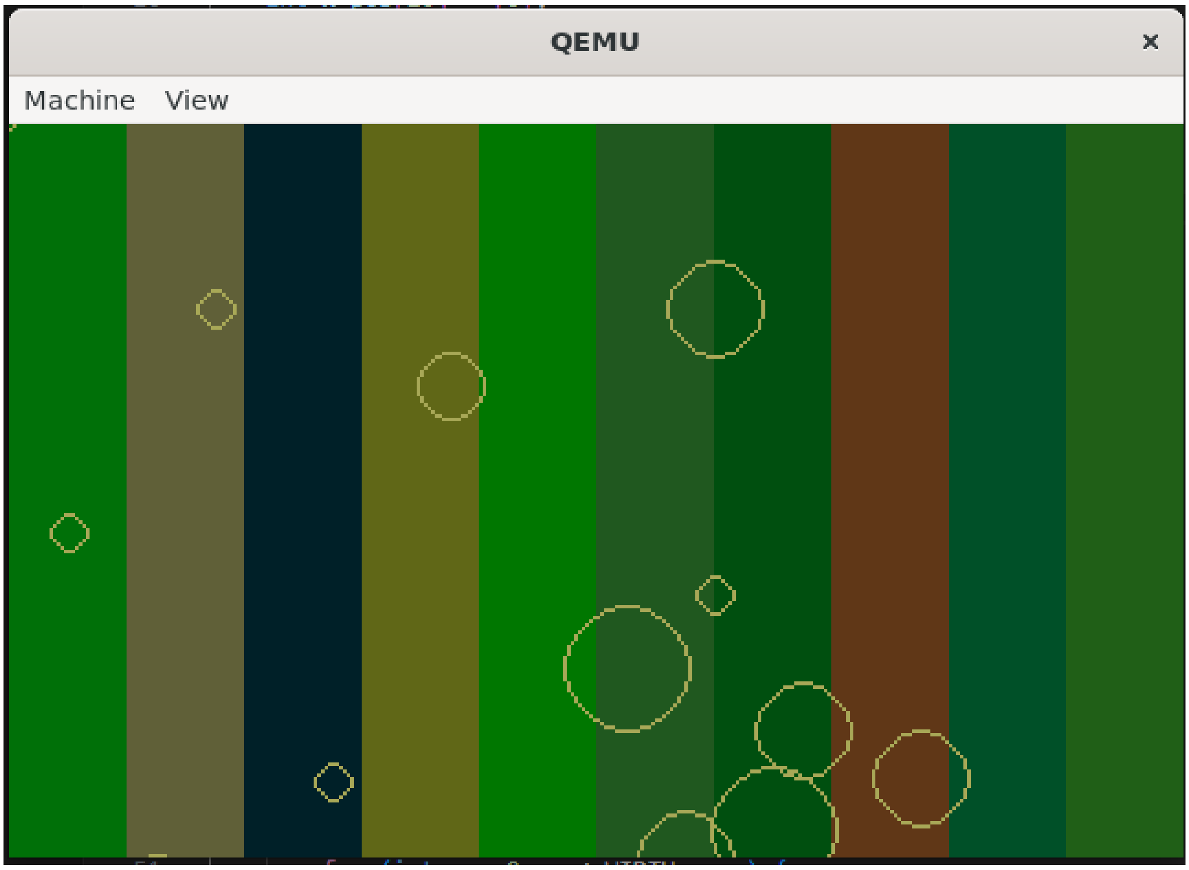
I also implemented the midpoint circle drawing algorithm as the circles drawn with Bradenham's algorithm were more blocky than I wanted.

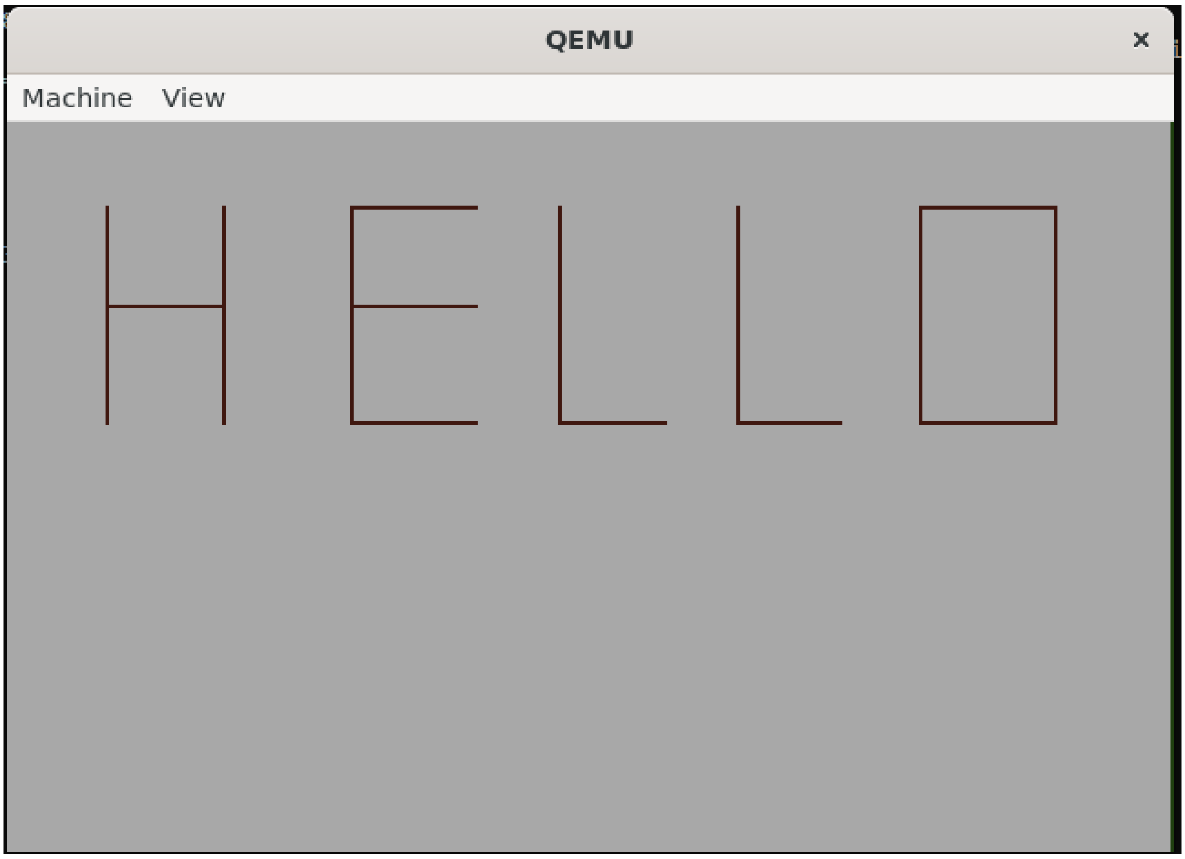
Tried a simple fill circle by calling multiple drawcircle and increasing the radius each time.



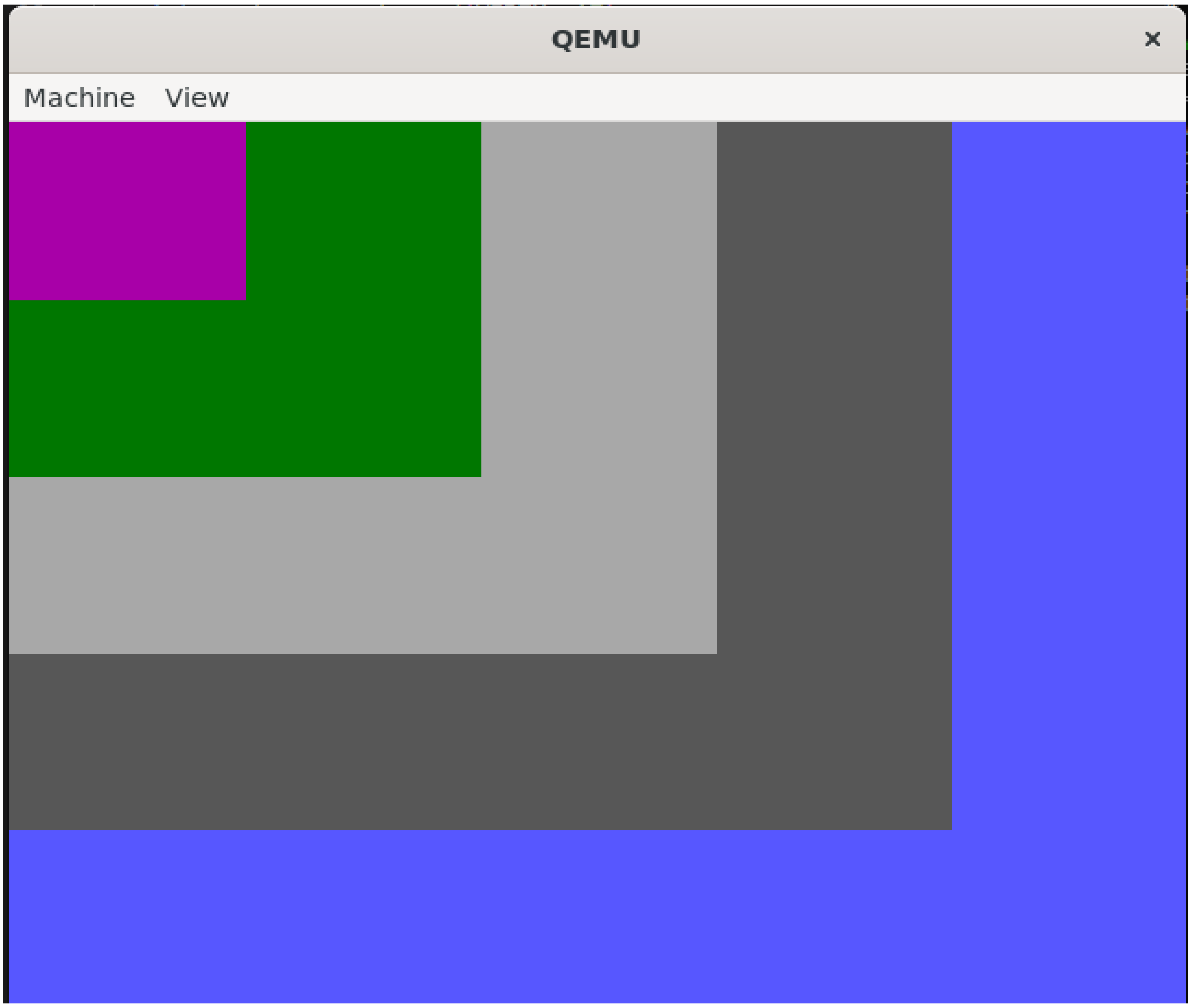
12/01/24

I spent time in the labs putting together my demo programs. I made one called “ademo.c” which is a slideshow of a few different features, first in vm13 and then into vm12, mainly drawing rects, lines, changing and setting pen colors randomly and drawing circles.

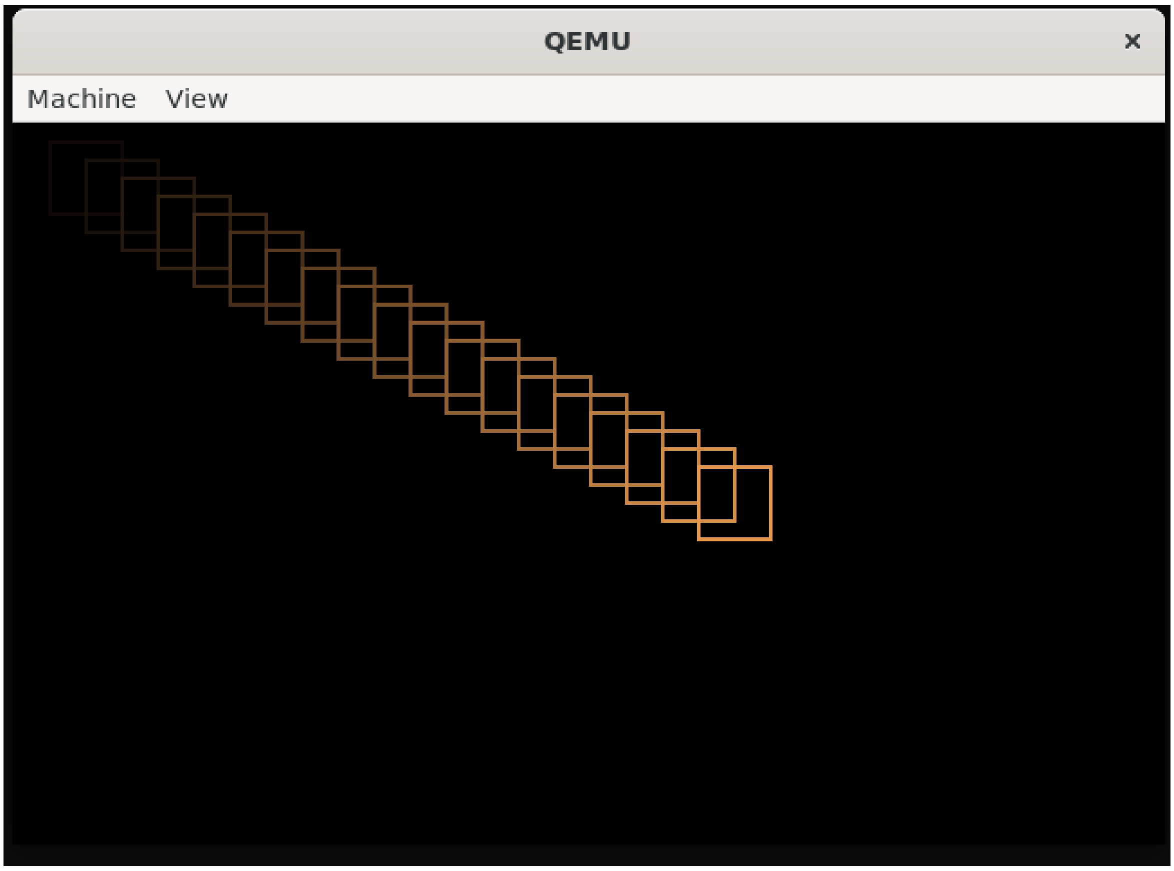


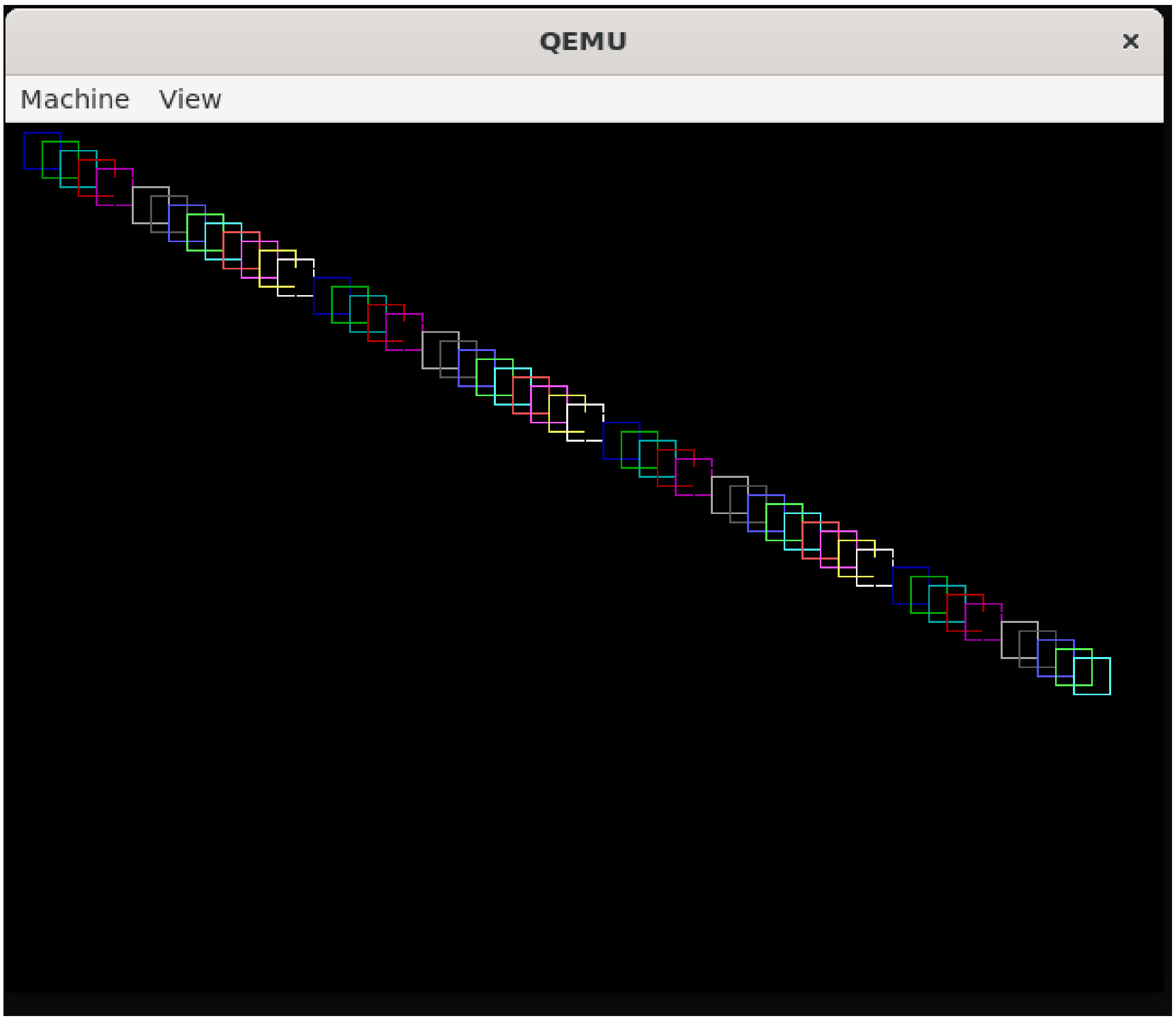


“ademo.c” then goes into vm12 just drawing rects before a circle fadeout.

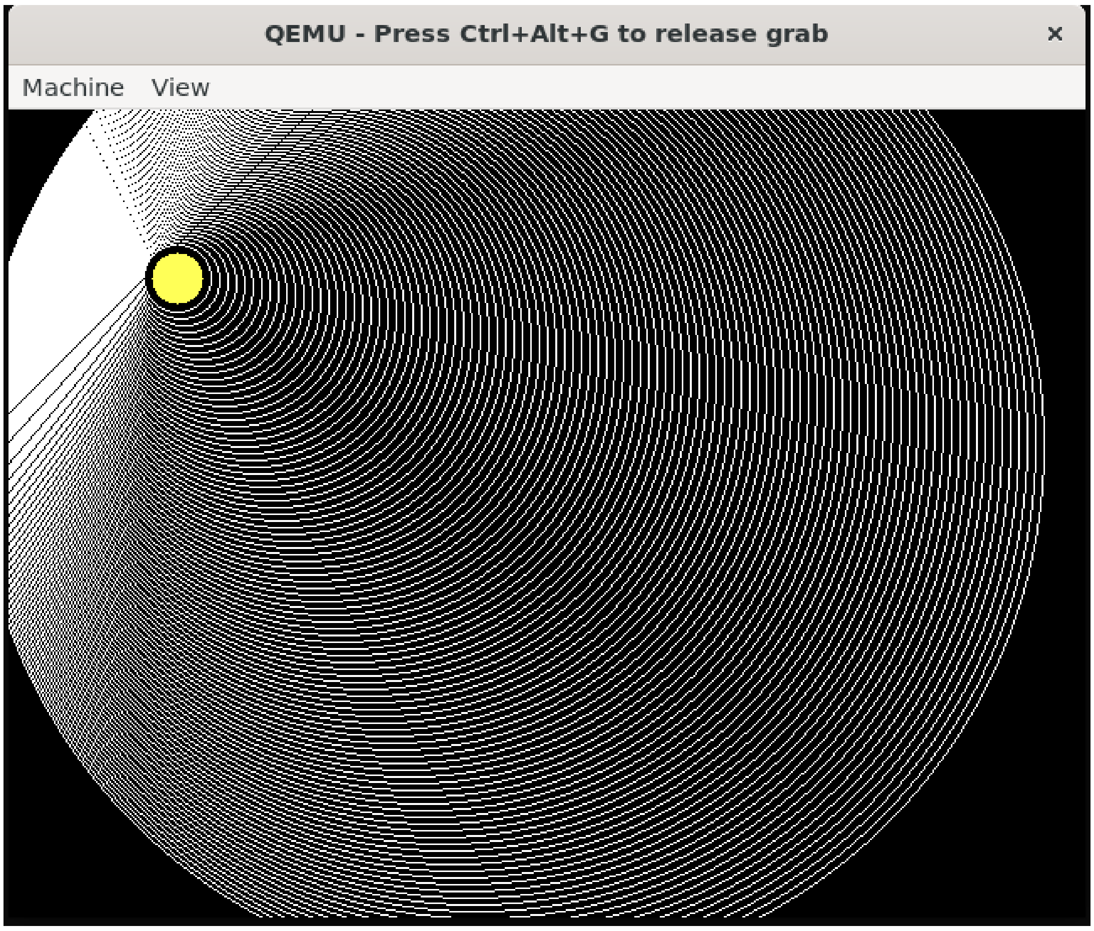


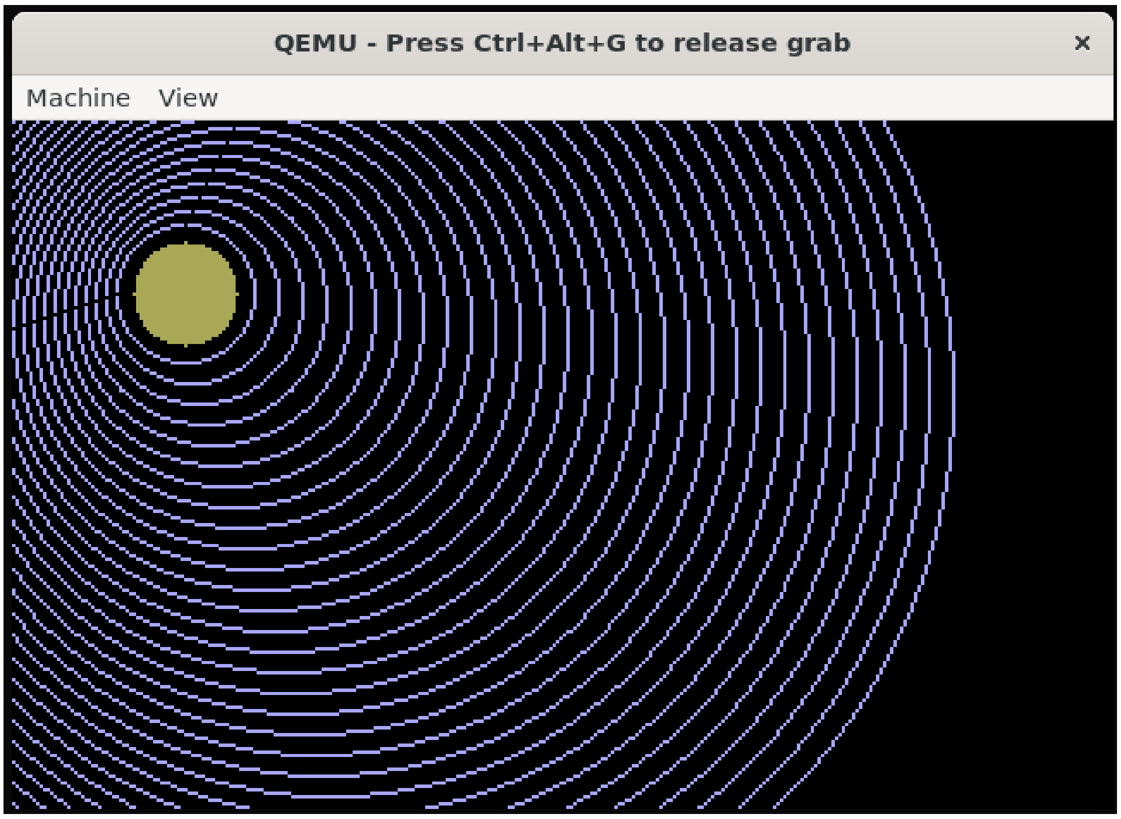
I added ontop of the astage4 program as it shows setting and using pen colors, drawling lines, moveto and batch calling functions which is almost everything I’ve implemented minus vm12 so I added another part afterwards which does the same in vm12, minus the setpencolor stuff.





Finally I expanded on an earlier acircle code to display in both vm12 and vm13. I also added a fill circle function which is shown off here.





At the end of all of this, I never actually used the qemu debugging functionality. I never felt the need to use it as pretty much all of the time, I'm able to use print statements to get all the data I feel is relevant at the time, writing appropriate user programs to run the thing I want to run and get the data I need to figure out where to go next.