Lab2: Programación VGA

Sistemas Operativos 2020

FaMAF - UNC

- Versión 2016: Matías Molina, Nicolás Wolovick.
- Versión 2017, 2018: Matías Molina.
- Versión 2019: Pablo Ventura, Matías Molina.
- Versión 2020: Ignacio Moretti, Pablo Ventura.
- Versión 2021: Ignacio Moretti, Pablo Ventura.

Plazo

Tres semanas, desde el martes 22/09 al martes 13/10 a las 15:59:59 hs.

Objetivos Generales

- Manejar un dispositivo.
- Familiarizarse con el código de `xv6`.
- Programar en *userspace* y *kernelspace*.
- Manejar el ciclo edición-compilación-ejecución en `xv6`.
- Utilizar `git` para el ciclo de modificación de `xv6`.

Objetivos Particulares

- Agregar syscalls
  - Cambiar el modo (gráfico/texto).
  - Pintar un pixel.
- Utilizar puertos para acceder a registros.

Forma de evaluación y entrega

Este laboratorio se entregará con *commits* subidos al repositorio git de Bitbucket `lab2grupoXX` que tienen asignado y deberá incluir un archivo `README.md` en formato markdown con el informe sobre el proceso de desarrollo del lab.

- La organización del git deberá ser:
  - Dentro del repositorio poner la implementación dentro de un **directorio** `xv6`.
  - El informe `README.md` deberá ir en la **raíz** del repositorio.
- Implementación funcional en el repositorio respetando a rajatabla el estilo de código de `xv6`.

- Informe en formato markdown mostrando el proceso de desarrollo.
  - ¡No hay que repetir el enunciado!
- Deadline: **lunes 4 de Octubre** a las 23:59 (GMT-3).

Antes de comenzar

- Obtener la última versión de `xv6` clonando el repositorio https://github.com/mit-pdos/xv6-public.
- Instalar `qemu`, en particular `qemu-system-i386`.
- `make qemu` dentro del código de `xv6` ejecutará el sistema.

Programación VGA

El término VGA proviene de "Video Graphics Array" y es un adaptador de pantalla introducido por IBM en 1988. Básicamente se encarga de procesar la información desde la CPU hacia un monitor para que sea humanamente entendible. Pueden encontrar bastante introducción en la Wikipedia.

¿Qué necesitamos?

- Entender números en binario y hexadecimal.
- Entender el acceso a memoria mapeada y puertos I/O.
- Conocer un poco VGA (manual de referencia?).

Modos VGA:

Se pueden especificar dos maneras disintas de mostrar los datos en pantalla (y describe, en realidad, cómo es el acceso a memoria):

- modo *texto (0x30 mode)* (80x25) y
- modo *gráfico (0x13 mode)* (320x200).

En el primero, se asume que la información que estamos mostrando corresponde a caracteres (fuentes) y no a píxeles como es en el segundo caso. En particular, el modo gráfico asume que cada celda de memoria corresponde a un píxel, es decir, si modificamos un valor en la matriz que representa la pantalla, entonces estamos cambiándole el color al pixel. Para el modo texto se utilizan dos buffers distintos, uno para definir cómo es el carácter y otro para indicar como mostrarlo por pantalla (abajo hay algo más de información).

Doom (320x200)

Nota: En realidad hay más modos, no solo 320x200 y 80x25. Pero nos vamos a centrar en estos dos.

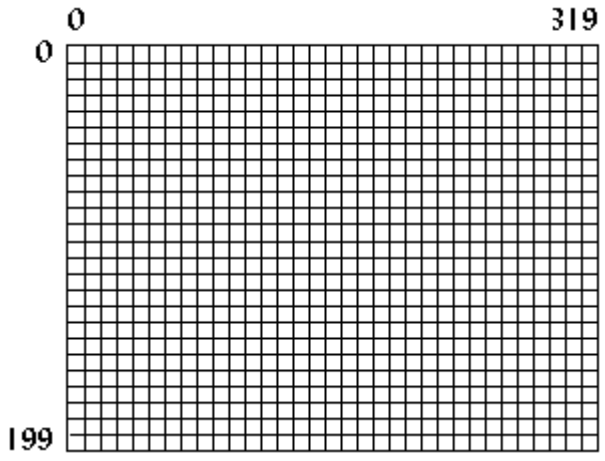
Standard VGA:

La memoria está dentro del dispositivo en el rango de direcciones `0x000A0000 - 0x000BFFFF`. Todas estas direcciones están mapeadas con las *direcciones físicas* de la memoria principal, por lo que si accedemos a éstas direcciones físicas, entonces estamos accediendo a las del dispositivo.

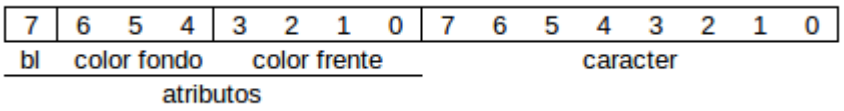
El modo `0x13` se corresponde con el modo gráfico con una resolución 320x200 (64000 bytes). Podemos acceder a una dirección de memoria específica a partir de la dirección `0xA0000` y como el espacio de memoria es continuo podemos calcular (linealmente) la dirección de memoria que se corresponde con cada píxel (x,y):

```
uchar *VGA = 0xA0000;
offset = 320*y + x;
VGA[offset] = color;
```

Teniendo en cuenta que el origen (0,0) se encuentra en la esquina superior izquierda:



El modo `0x03` usa `80x25` celdas de texto, para cada celda se puede indicar el carácter a mostrar y algunos atributos como color de fondo y color de texto. Toda esta información se encuentra a partir de la dirección `0x000B0000` o `0x000B8000` según sea un monitor monocromático o a color. La manera en que se almacena la información de un carácter es diferente a como se hace en el modo gráfico, en este caso, cada carácter que vemos en pantalla es representado por 16 bits (o 2 bytes), el byte más bajo provee el código ASCII del carácter y el byte más alto se compone de bits que indican los atributos que tendrá:



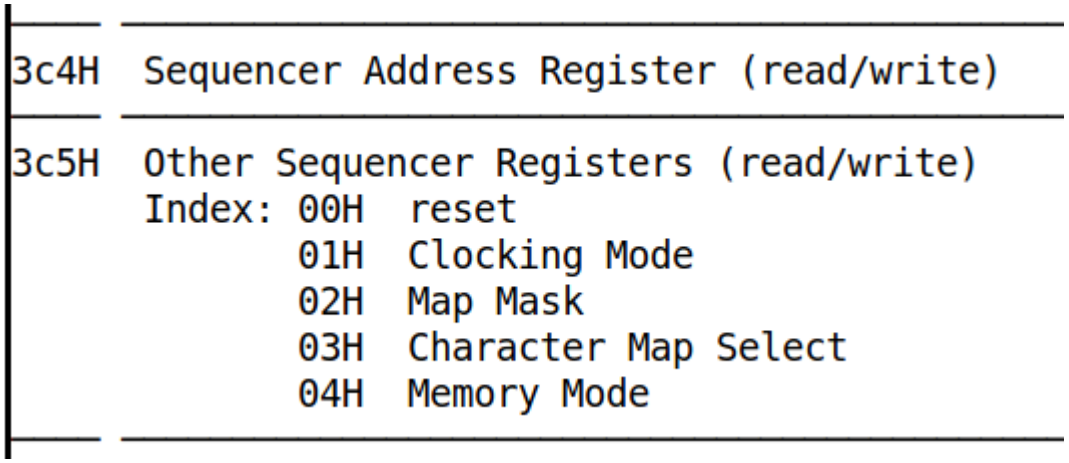
Así, por ejemplo, `0x0753`, representa la letra `S` en color gris y con fondo negro.

Notar que en ambos modos se solapan las direcciones: `0xA0000` - **`0xB0000 - 0xBFFFF`**. Esto es muy importante de analizar, ya que si estamos en modo gráfico y cambiamos a texto, quedaría *basura* en la memoria y no nos dejaría ver las fuentes originales (¡si quisiéramos volver a verlas, tenemos que recuperarlas!).

¿Cómo podemos entrar en un modo u otro? Para poder pasar de modo texto a gráfico y viceversa, necesitamos setear determinados valores en registros específicos del dispositivo. Los registros de la VGA son muchos, sin embargo ocupan un rango bastante pequeño de direcciones I/O. Esto que parece un dato ingenuo nos dice algo importante: muchos registros están indexados, esto básicamente quiere decir que no necesariamente leemos/escribimos directamente en un registro sino que lo hacemos indicando en un campo qué registro queremos leer/escribir y finalmente leemos/escribimos a través de otro campo.

Específicamente, el acceso a registros es a través de puertos. Los puertos, son la interfaz que nos permite escribir y/o leer los registros del dispositivo. Tienen direcciones específicas que indican el registro directo al que queremos acceder o bien la dirección base para los registros indexados (luego podemos manifestar a qué índice accederemos partiendo de esta base).

Por ejemplo, el registro de secuencia (Sequencer Register), es un tipo de registro indexado y tiene dos puertos asociados, uno para el registro de datos y otro para el registro de direcciones. Si en este último escribimos un byte igual al índice del (sub)registro sobre el que queremos operar, podemos leer/escribir en él a través del primero. Sabiendo esto podemos mirar el manual de referencia de la VGA para, por ejemplo, leer el valor del subregistro clocking mode:



El registro de direcciones se encuentra en el puerto `0x3C4` y el de datos en `0x3C5`. Así, podemos escribir en `0x3C4` el valor `01` (índice) y luego leer en `0x3C5`, en código sería algo así:

```
outb(0x3C4, 0x01);
value = inb(0x3C5);
```

Tareas

Parte I

Implementar en `console.c` una función `vgainit()` y llamarla al inicio del sistema. Esta función (por ahora) deberá mostrar un pie de pantalla con el mensaje `SO2020`. Por ejemplo, las siguientes lineas escriben `Holi!` (con algún color de caracter y de fondo, como describimos antes):

```
  *(int *)P2V(0xB8000) = 0x4348;
  *(int *)P2V(0xB8002) = 0x436F;
  *(int *)P2V(0xB8004) = 0x436C;
  *(int *)P2V(0xB8006) = 0x4369;
  *(int *)P2V(0xB8008) = 0x4321;
```

Notar que si ejecutamos estas lineas en espacio de usuario, *no funcionan* ya que como usuarios no tenemos privilegio para acceder a la memoria del dispositivo.

Parte II

Extender `console.c` con funciones que nos permitan cambiar de modo texto/gráfico, usando las configuraciones descriptas antes. Luego, usarlas para hacer el cambio de modo desde `vgainit()`. **¡Cuidado!** Al volver al modo texto luego de haber estado en gráfico las fuentes no serán legibles.

Parte III

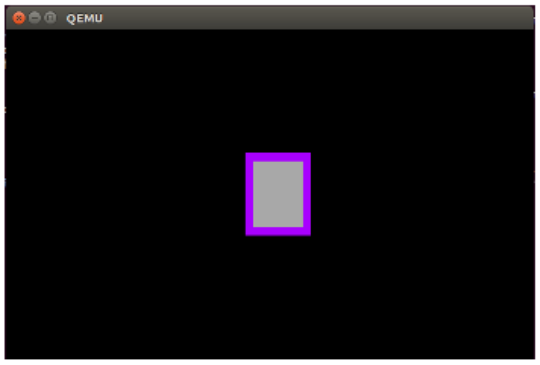
Ya funcionando la **Parte II** deberán modularizar correctamente, esto es:

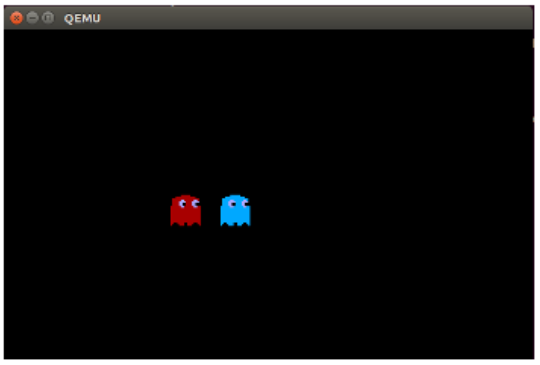
- Implementar una syscall `modeswitch` que hace el cambio de modo (texto/gráfico) según el argumento que recibe (`0` o `1`), para esta parte deberán usar la información en la ayuda abajo.

- Implementar una syscall `plotpixel(int x, int y, int color)` que pinte un pixel en pantalla. (los tres argumentos enteros se corresponden con las coordenadas `(x, y)` y el color a pintar).

Parte IV (la parte divertida)

- Les damos ~~un programita~~ dos programitas de ejemplo. Si todo va bien deberían poder verlos correctamente en modo gráfico.





- Ahora, hagan su propio programa de usuario que genere algún dibujo por pantalla. Exploten su creatividad!

Realmente hay margen de creatividad para este lab? Algunos ejemplos de años anteriores...

Extra

- Todo lo que implementaron puede ser modularizado de una manera más delicada. Teniendo en cuenta que son funciones para un mismo dispositivo pueden estar en un mismo archivo `vga.{c,h}`
- Agregar una nueva syscall `plotrectangle(int x1, int y1, int x2, int y2, int color)` para dibujar rectángulos en la pantalla (la idea es no tener que usar `for` para pintar un pixel por vez)
- Si bien el modo gráfico es de 256 colores, la paleta está programada para 64 colores: Programar la paleta para poder usar todos los colores.
- Recuperar las fuentes que se pierden cuando pasamos de modo gráfico a texto.

Ayudas

- Para incluir un programa de usuario recomendamos mirar el archivo `Makefile` y buscar alguno de los programas conocidos que vienen incluidos y se pueden ver en la consola de `xv6` (`ls` , `grep` , etc).

- Para agregar una nueva *syscall* deberán modificar **varios** archivos. Tomar alguna llamada a sistema y ver en todos los puntos que aparece. (Ya vimos en clases que

como usar `grep`).

- Cambiar de modo implica setear valores particulares en los registros del dispositivo. Para esto es importante tener una referencia de los puertos (ver *Puertos VGA* en las referencias abajo) Y la **super** ayuda se encuentra acá: http://files.osdev.org/mirrors/geezer/osd/graphics/modes.c
- Al escribir los Attribute Registers, el registro de direcciones y el de datos se encuentran en la misma direccion de memoria, por lo que hay que indicarle al sistema si lo que estamos escribiendo es la dirección de un subregistro o un dato siguiendo la documentacion en Accessing the Attribute Registers.
- Para el mapeo de memoria (física-virtual) pueden usar la función `P2V`.

Manejo básico de `qemu`

- Para listar los procesos dentro de `xv6` hacer `<CRTL-p>`.
- Salir de QEMU: `<CTRL-a> x`.
- Para que iniciar qemu CON pantalla VGA: `make qemu`.

Referencias

Free VGA

Free VGA / vga

Puertos VGA

Wikipedia text-mode

Algunos ejemplos (regalito) http://wiki.osdev.org/Printing_To_Screen