ARQUITECTURA DE LAS COMPUTADORAS

TRABAJO PRÁCTICO ESPECIAL PC 2019

INTEGRANTES:

- *57610 - Ignacio Grasso*
- *58404 - Martín Galderisi*
- *58546 - Nicolás de la Torre*

Driver de Video

El driver de video está basado en:
-https://wiki.osdev.org/VESA_Video_Modes
-https://wiki.osdev.org/Drawing_In_Protected_Mode.

La matriz usada fue extraída de la guía del arduino, la estructura utilizada fue extraída de unas de las páginas anteriormente mencionadas, llenando dicha estructura con los datos proporcionados por el VBEInfoBlock.

Muchas funciones, para dibujar rectángulos, fueron implementadas en Userland, ya que solo eran necesarias para el funcionamiento del snake.

Para mostrar los pixeles en pantalla, utilizabamos la syscall que llamaba a putPixel.

Driver de Teclado

Para el driver de teclado buscamos los Scan-Codes de las teclas. Se programó basado en un teclado con formato QWERTY y lenguaje USA. Se tuvieron en cuenta los dos posibles valores que pueden tener las teclas, es decir, con o sin el SHIFT apretado.

Excepciones

Tuvimos en cuenta dos posibles excepciones, división por cero y Operación inválida. Como nuestro proyecto no tiene una biblioteca de operaciones matemáticas, simulamos la división por cero utilizando el comando “cero”. También se utilizó una simulación para la operación invalida y se invoca utilizando el comando “invalidOpCode”. Al invocar cualquiera de los dos comandos, se imprime la lista de registros y los valores que contenían a la hora de la excepción.

Syscalls

Agregamos la función _syscall, cargada en la posición 80h de la IDT, ésta se asemeja al comportamiento real de los system calls. Hay 7 system calls, sys_total_ticks, dada por el código 0x01, nos retorna la cantidad total de ticks transcurridos, sys_ticks_per_second, con el codigo 0x02, nos devuelve la cantidad de segundos transcurridos. Luego tenemos las syscall read y write, con los codigos 0x03 y 0x04 respectivamente, estas syscalls son las encargadas de leer y escribir lo que hay en el buffer de teclado.

También agregamos las syscall sys_time, con el código 0x05, guarda en un buffer la hora, sys_pixel con codigo 0x06, esta syscall fue elemental a la hora de imprimir en pantalla en el videoDriver, así como sys_clear, con el código 0x07, esta syscall nos limpia la pantalla completamente.

En Userland creamos funciones en asm para llamar al system call que deseamos, donde dentro de la función se carga en RDI el código correcto del system call. Luego de cargar el RDI y los demás registros con distintos parámetros, con la instrucción int 80h se va a la IDT a buscar la syscall cargada en la posición 80h. Dentro de ésta se compara RDI con los posibles códigos, y llama a la syscall correcta.

IDT Loader

El IDT_Loader no sufrió grandes modificaciones del provisto por la cátedra, utilizamos _cli() para deshabilitar las interrupciones, agregamos las interrupciones de teclado, timer tick, así como las excepciones. Luego volvimos a habilitar las interrupciones mediante el comando _sti().

Stdio/Stdlib

Simplemente implementamos las funciones necesarias de stdio y stdlib.

Para stdio :Incluimos las funciones printf, getchar y putchar en su formato estándar, también realizamos una implementación particular de scanf, una especie de readLine, simplemente la utilizamos para leer los comandos para luego enviarlos a un parser y ejecute la instrucción correspondiente.

Para stdlib : Incluimos las funciones strlen, strcmp, strncmp tradicionales, basándonos en su implementación estandar. Estas funciones fueron útiles para luego calcular la longitud del string que luego imprimiríamos pixel a pixel en el videoDriver. Como mencionamos anteriormente, strcmp y strncmp fueron utilizadas en el parser del intérprete de comandos.

Snake

Para el snake lo primero que hicimos fue implementarlo en c, utilizando la biblioteca estándar y imprimiendo en consola, para así diseñar la lógica del juego y luego, adaptarlo a modo video.

Empezamos con el diseño de la víbora. Cada parte de su cuerpo es una estructura de datos que contiene la posición en los ejes X e Y, por lo tanto, la víbora es un vector de estructuras, siendo la primer posición su cabeza.

Luego, implementamos el movimiento de la víbora, y para esto tuvimos que crear 2 variables adicionales dentro de la estructura(ModX,ModY). Estas variables sirven para saber la dirección en la que se debe actualizar la víbora(hacia donde avanza). La víbora se desplaza por el campo de juego, que en sí es una matriz.

En cada movimiento comparamos la posición de la cabeza de la víbora con los bordes del campo de juego o con alguna posición de su cuerpo. Esto sirve para saber si la víbora ha muerto o si el usuario puede seguir jugando .

El resto consistió en implementarlo en modo video y agregar otras funcionalidades como la puntuación. No nos alcanzó el tiempo para hacer que crezca cada 15 segundos, así que decidimos implementar un contador que aumenta luego de presionar 5 veces cualquier tecla.