Informe de Trabajo Práctico
Arquitectura de las Computadoras

JOAQUIN BATTILANA

LAUTARO PINILLA

2° Cuatrimestre 2018

El objetivo del trabajo práctico era realizar un Kernel sobre la base “BareBones”. Este informe es complementado por la “Guía Ayuda” agregada al junto al TP hecha por nosotros para ayudarnos entre nosotros para realizar el trabajo comentando cuestiones tanto prácticas como teóricas, nos referiremos a esta como “la guía complementaria” en este documento.

La primer cuestión grande de diseño con la que nos encontramos fue sobre realizar el TP enteramente en modo protegido, esto se debe a que no tenemos la herramienta [1] para poder cambiar de modo texto a modo video en tiempo real, ya que la interrupción que se utiliza para esto es la interrupción número 10 (que luego la propia IDT lo pisa). Por lo tanto, decidimos realizar todo el trabajo en modo video, eso nos llevó a tener que crear nuestro propio driver de video para poder mostrar caracteres principalmente.

Para el modo video utilizamos el modo número 13h, la razón está justificada la guía complementaria aunque se podría simplificar en que como el modo era muy utilizado por desarrolladores de videojuegos, asumimos que existe mucha documentación sobre programación de gráficos en bajo nivel.

Este modo nos presentó un par de Pros y Contras:

Como un punto a favor, este modo tenía acceso directo a memoria, lo conocido como “chunky graphics” en la jerga de programación de gráficos. Esto significa que la pantalla está “mapeada” directamente en la memoria de forma contigua, a cambio de otros modos en donde la pantalla está dividida en “planos”, lo que no permite un mapeo 1:1 con la memoria y por ende, presenta una complicación. Además, podemos agregar que con este modo contábamos con 256 colores personalizables.

Como punto negativo, este modo solo permite acceder aproximádamente al ¼ del “VRAM” (Video RAM, el espacio en RAM asignado a la placa de video). Esto no nos permite utilizar técnicas conocidas como el “Page Flipping”, técnica explicada a más detalle en la guía complementaria. Aparte, podríamos agregar que la cantidad de memoria no demasiada aunque para lo requerido, era suficiente.

Para hacer el PONG nos aprovechamos de la poca memoria disponible del modo 13h para hacer el método de “double buffering” (explicado con más detalle en la guía complementaria) creando un arreglo estático (debido a que no tenemos forma de alocar memoria dinámicamente) de 64k. Esta técnica nos permitió no tener problemas de “flickering” o “tearing” al hacer el refresco de la pantalla al renderizar el juego.

En el driver de teclado nos basamos en un buffer circular en el cual tenemos un máximo tamaño y se va pisando cada vez que la cantidad de teclas supera este máximo. En nuestro caso solo metemos en este buffer las teclas necesarias, que no son todas, las de control nos las quedamos para activar o desactivar cierta combinación de teclas. Estaria bueno

[1] Revisar en la guía complementaria la sección “Modo Video” en la Página 6 donde se cuestiona esta hipótesis y se plantea una posible solución.

para el futuro meter todas las teclas en este buffer y luego hacer una logica para saber la combinación de teclas, pero por el tiempo no llegamos a implementarlo de esta manera.

En el driver de sonido usamos el PIT (timer-tick) para imponer la frecuencia en la que queríamos que suene la bocina y luego mediante una espera de tiempo (llevada a cabo por una lógica dentro del mismo timer tick también, comparamos los ticks en un momento y en otro), hacíamos que deje de sonar para que solo se escuche el típico “beep”.

En el trabajo usamos de “STDOUT” y “STDERR” la pantalla, pero con diferentes colores. Esto estaría bueno en el futuro implementarlo dentro de “archivos” para que cada “proceso” que corre pueda tener su propio STDOUT y STDERR y no estén ligados a la pantalla como pasa en nuestro caso. Esto se ve propiamente en “scan_f” que la misma función aparte de tomar de STDIN tiene que imprimir, ya que al tomar las teclas directamente del buffer de teclado no hay una lógica antes que elija imprimir y le pase las teclas necesarias. Por eso estaría bueno poder implementar un STDOUT que sea un archivo manejado por el OS y que la lógica de dibujar esté separada de la lógica de leer del buffer. Esto no pudimos implementarlo en parte por tiempo y en parte porque todavía no tenemos las herramientas necesarias.

Con las excepciones no tuvimos muchos problemas, el unico es que por alguna razón nuestra lógica de imprimir números nos imprime basura en RCX pase lo que pase, no pudimos encontrar la razón. Decidimos que luego de que una excepción se lance se “reinicie” la computadora y vuelva a correr userland. Esto en el futuro estaría bueno cambiarlo para que no se pierda todo el trabajo anterior.