Arquitectura de Computadoras

ITBA – 2° CUATRIMESTRE 2015

CORTÉS – GOFFAN – HORVAT

Informe

Objetivo

Realizar un sistema de 64 bits em arquitectura INTEL, capaz de administrar los recursos de hardware.

Introducción

Se obtuvieron y configuraron varios programas para correr el trabajo práctico en nuestras PC. La mayor dificultad se dio a la hora de hacer el sonido (**play_sound** –ver más adelante-) ya que al principio, en muchas computadoras del laboratorio no sonaba la melodía y, en computadoras particulares, habilitar el QEMU para que reproduzca un beep.

A su vez, virtualizamos en Windows y Mac una distribución de Linux (Ubuntu 15.04), e instalamos **nasm** y **qemu**.

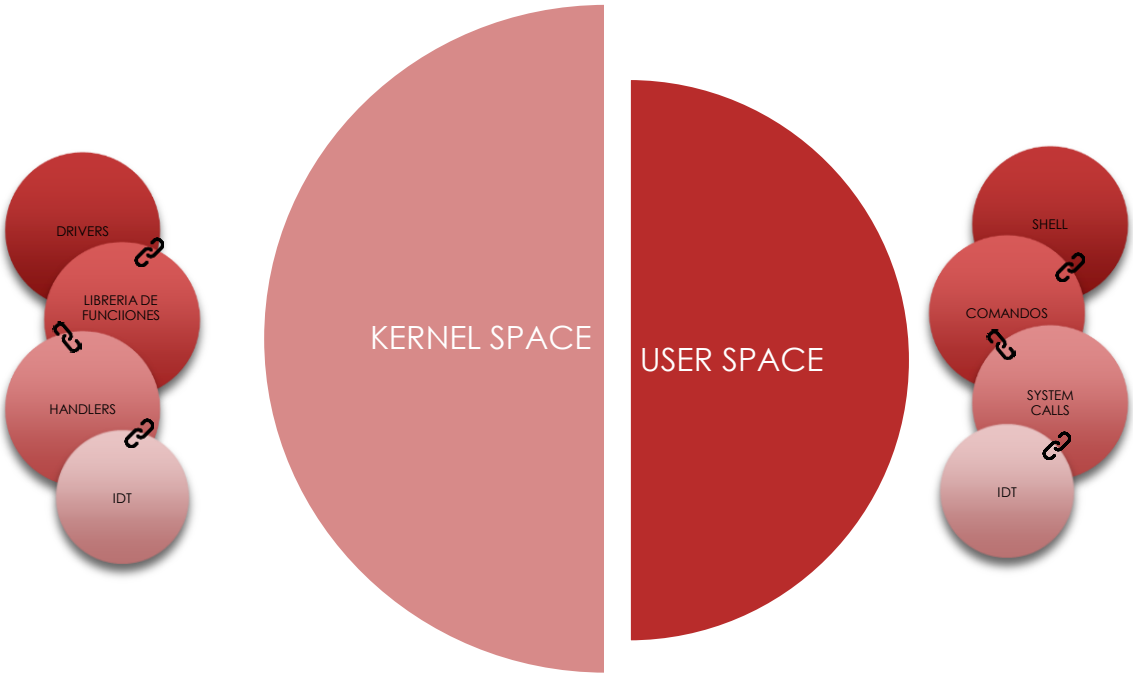
Una vez que finalizamos de instalar todas las herramientas necesarias, procedemos a clonar el repositorio otorgado por la cátedra (x64BareBones) y a partir de allí comenzamos a desarrollar un sistema operativo básico para la Arquitectura INTEL 64 bits.

Comenzamos el desarrollo aplicando el diseño bottom-up. Primero realizamos todas las funcionalidades del Kernel y luego procedimos a realizar las del User Land

CONTRAS: Realizábamos el código "a ciegas". La mayoría del tiempo no sabíamos si algo funcionaba o no hasta el momento de probarlo.

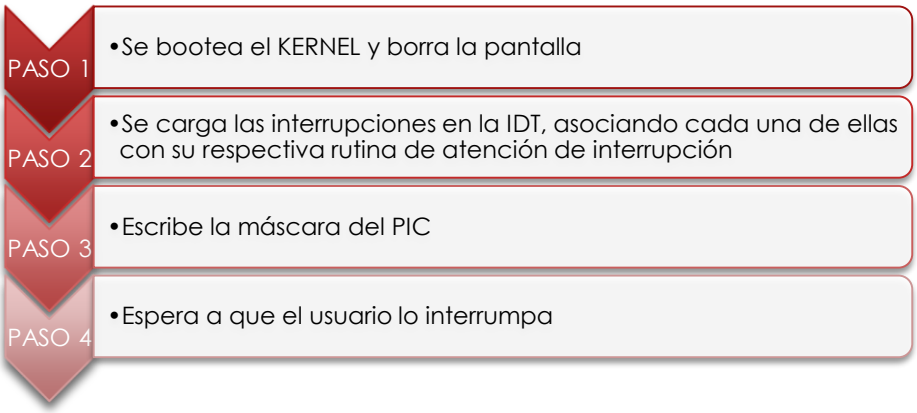
PROS: Pareciera ser que, como el Kernel es el único que tiene acceso al hardware, la implementación y realización del código es un poco más complejo que las del User Land. Al realizarlo primero que todo, pudimos tener más tiempo para interiorizarnos con los problemas que podían surgir.

Como vimos en clase, debe haber una clara separación entre el USER y el KERNEL, ya que no queremos que el usuario tenga acceso a funciones propias del Kernel.



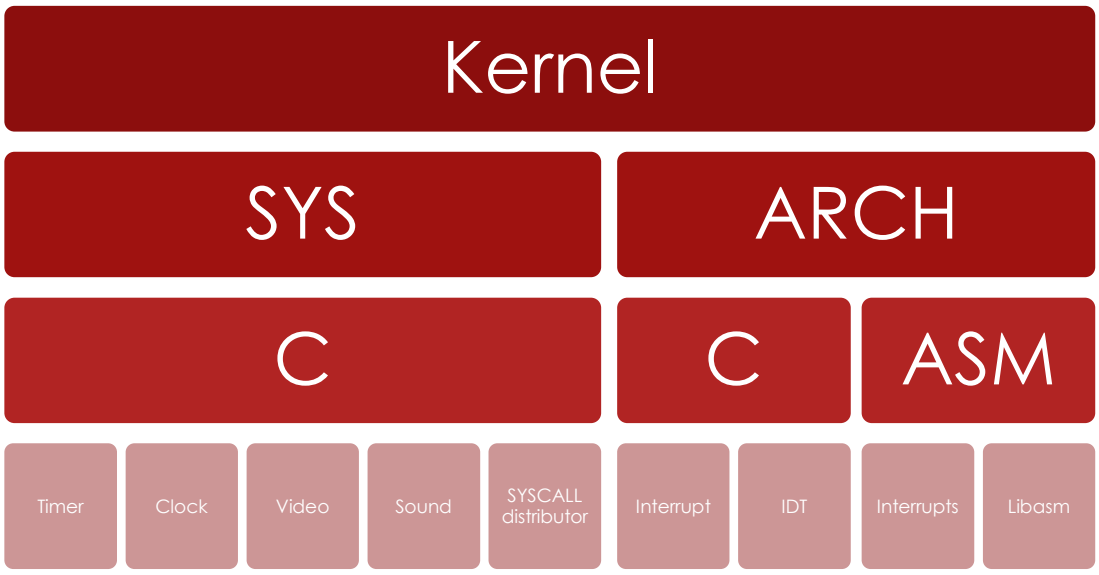
Kernel

Es el que está en contacto directo con el hardware y atiende los pedidos del usuario a través de la interrupción INT 80h.




```
int main() {

	init_idt();
	init_interrupts();
	init_timer();
	init_syscalls();
	init_keyboard();

	clc();

	((EntryPoint)sampleCodeModuleAddress)();

	return 0;
}
```

¿Cómo "dividimos" el Kernel?



SYS

1- CLOCK: Interacción directa con el hardware (_inport y _outport).
En la dirección 0x70 se inserta el tipo de dato que va a leer o escribir y en la entrada 0x71 se lee o escribe el dato.

   - ➔ 0 : Segundos
   - ➔ 2 : Minutos
   - ➔ 4 : Horas

2- VIDEO: Es el encargado de administrar no solo la memoria asignada para escribir cuando el usuario interrumpe cuando presiona una tecla, sino que implementa funciones donde imprime strings, números en binario, etc…
Al espacio asignado al video, se lo piensa como una matriz de 80x25 y se realizan cálculos para saber dónde está parado o si tiene que ir a una nueva fila.

3- SOUND: Interacción directa con el hardware, por medio de interrupciones. Es capaz de hacer un simple beep o hacer un beep con una cierta longitud y una frecuencia. Las notas, en este caso, se trabajan con frecuencias.

DO - C - 9121
RE - D - 8126
MI - E - 7239
FA - F - 6833
SOL - G - 6087
LA - A - 5423
SI - B - 4831

4- TIMER: Es el encargado de administrar el tiempo, obviamente. Posee dos funciones, un init_timer() y un wait, donde se pasa por parámetros la cantidad de milisegundos a esperar.

Userland

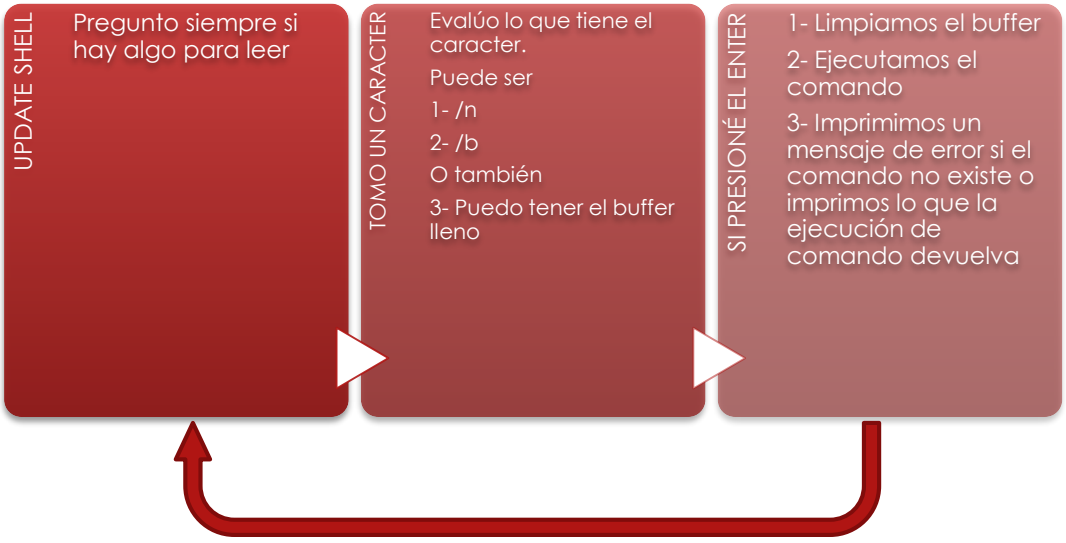
Shell / Consola

Una vez que ya inició el sistema, ya se cargaron las tablas IDT y todo lo que hemos explicado previamente en Kernel, se "habilita" la consola, donde el usuario podrá realizar diferentes tareas, dependiendo el comando que ellos ingresen.

Cada vez que el usuario presiona una tecla, internamente hay una interrupción por teclado y dicha tecla presionada se agrega al buffer.

Una vez que el usuario terminó de escribir y apreta enter, se procede a parsear y a validar la información. Si el comando no existe, se mostrará un mensaje de warning en rojo, sino se procederá a ejecutar la función del comando.

¿Qué sucede internamente en el Userland?



Conclusiones y Observaciones

El desarrollo de este trabajo práctico nos sirvió no solo para darnos cuenta de los principios básicos del manejo de un sistema operativo por completo, manejo con el hardware e interacción con el usuario; sino que también a investigar y leer mucha información de distintas fuentes. (ver REFERENCIAS debajo)

Aprendimos no solo a programar, sino que a investigar y tratar de llegar a todas las posibles soluciones ante un problema que surgía (y así, seguían apareciendo cada vez más errores).

Gracias a este trabajo práctico, aprendimos porque las cosas que vemos hoy en día funcionan de tal manera. Todo se debe a que pioneros en lo que hoy estamos estudiando, trataron de dar la mejor solución posible para facilitar las cosas a futuro e ir progresando en la rama de la informática.

REFERENCIAS

1- Forums.osdev.org
2- Wiki.osdev.org
3- The C programming language – Kernighan and Ritchie
4- University of Illinois (Online courses)
    a. https://courses.engr.illinois.edu/ece390/books/labmanual/io-devices-speaker.html
    b. https://courses.engr.illinois.edu/ece390/books/labmanual/nasm.html
    c. https://courses.engr.illinois.edu/ece390/books/labmanual/io-devices.html
    d. https://courses.engr.illinois.edu/ece390/books/labmanual/index.html