**Tutorial 0**

Todos estos tutoriales tienen que ver con la explicación y entendimiento de como funcionan los sistemas operativos desde que iniciamos un computador, corre el sistema operativo y utilizamos las aplicaciones que esta contiene. Para poder entender muy bien como es que funciona un sistema operativo primero debemos entender cada una de las partes que lo componen y para comenzar debemos aprender que tipos de lenguajes de programación se usan y porque son estos los que se escogieron para hacer este tipo de tareas.

Uno de los lenguajes mas utilizados para implementar sistemas operativos es el lenguaje C. Sin embargo hay ciertas restricciones para poder utilizarlo, como por ejemplo: al inicio, la computadora arranca en “real mode” donde solo pueden correr programas de 16 bits. C soporta programar en 16 bits pero si quieres utilizar la máxima potencia de un computadora en estos días tendríamos que utilizar un compilador de 64 bits o 32 bits y el código de 16 bits y 32 o 64 bits no se puede mezclar entre si. Otra razón por la que no se puede utilizar C al inicio de un sistema operativo es que la BIOS se encarga de cargar el **Boto Loader** en memoria y ejecutar la primera instrucción encontrada en ella, sin embargo el formato en que se compilan los programas de C utiliza nombres simbólicos y rutinas que no están ordenadas de forma “plana”, es decir, la primera instrucción o el “main” no esta en el primer byte del programa. Solo hay un solo lenguaje que permite hacer eso y se llama ensamblador.

Los punteros funcionan de un forma distinta cuando estamos creando nuestro propio sistema operativo. Por ejemplo char\* puntero = 0 esta apuntando a una dirección 0 en memoria real, no virtual como se maneja en una aplicación corriendo sobre un sistema operativo. La alocución de memoria también se maneja de forma directa en un sistema operativo y es que funciones como mallot(), free(), new o delete no existen en nuestro sistema operativo o a nivel de sistema porque no existen tales librerías. Tenemos todo el control del sistema como tal a nivel de núcleo o kernel. Como no existen librerías, tampoco es probable usar funciones de STD Library como stunt(), printf() o scanf() y es necesario poder crear nuestra propia librería de funciones de sistema o RTL.

El debugging no existe como tal a nivel de sistema operativo, sin embargo, estaremos utilizando BOCHS como emulador de un sistema x86 lo cual nos puede dar cierta información de los registros y estado de memoria en cuanto ocurra un error. Escribir un sistema operativo desde 0 no es tan fácil, sin embargo, suena como un reto muy interesante en el cual se pueda aprender mucho.

URL de Proyecto : https://github.com/CEUTEC-Courses/2016-ii-1272-so1-osdevseries-jorgefer00