**Tutorial 12**

Una vez que ya tenemos nuestro código listo para poder cargar nuestro kernel final a memoria, es tiempo de que podamos entender en detalle lo que un kernel significa. Básicamente, un kernel es el núcleo de todo sistema operativo. No todo sistema computacional ocupa un kernel para ejecutar aplicaciones como por ejemplo, un sistema de video juegos de primeras generaciones corre un solo programa sobre el hardware y si necesita cargar otro juego pues es necesario apagar la consola, insertar un nuevo programar y iniciarla nuevamente. Entonces, porque utilizamos un kernel? Es por su capa de abstracción y facilidad que le da a los programas para que se puedan ejecutar sobre cualquier plataforma, indistintamente del hardware que contenga la computadora.

Para poder ejecutar programas en un sistema operativo es necesario establecer la seguridad y entornos de ejecución adecuados para los programas de usuarios. El kernel se ejecuta en el Ring 0, mientras que el Ring 1 y 2 se utiliza para drivers y finalmente el Ring 3 para programas de usuario. Esto le da bastante aislamiento a el código de usuario del código de sistemas porque si un programa de usuario se ejecutase en Ring 0 y crea una excepción, todo el sistema se viene abajo mientras que si lo mantenemos a niveles separados, esa capa de abstracción mantiene el sistema arriba mientras se causa un error a nivel de usuario.

Básicamente entonces, el kernel provee a los otros niveles de ejecución, las cosas básicas para que se puedan ejecutar sin problemas y no preocuparse de la interacción con el hardware. El kernel ofrece manejo de la memoria, manejo de programas, multitasking, pagina de memoria, un sistema de archivos, una terminal, una interface grafica y un espacio multiusuario. Con esto se cubre todo requerimiento básico de cualquier sistema operativo de hoy en día. Sin embargo, todos lo practican de maneras ligeramente diferentes logrando siempre el mismo resultado o mejor que otros.

El primer problema con el que se tiene que lidiar es con el manejo de memoria para los programas. Como es posible ejecutar dos programas a la vez y no sobrelapar la información en memoria de un programa con otro? Para esto, el kernel implementa algo que se llama memoria virtual. Esto es, que cada programa tiene su propio espacio virtual de memoria, que comienza en 0x0 y no tiene un limite establecido. Aunque la memoria RAM física del equipo no supla la demanda del programa, este siempre creerá que tiene mucha mas memoria RAM, en el caso de sistemas operativos de 32 bits, 3-4GB de RAM.

Entonces, como separamos la memoria del kernel y la del usuario? El kernel deberá implementar el manejo de memoria virtual, pero es también en la tabla GDT, en la cual se colocan nos niveles de acceso de la memoria de usuario y tienen diferentes descriptores:offset a los que tiene el espacio de memoria del kernel. Luego esta el problema de como interactuar con el hardware desde un programa de usuario si no se tiene control directo sobre el hardware. Para esto existen las llamadas al sistema o System Calls que deben ser implementadas por todo kernel para que los programas puedan hablar con le hardware. Por ultimo, que pasa si una instrucción que generamos en nivel de usuario ocasiona un error a nivel de procesador (división por 0?). En este caso, el procesador dispara un ISR o Interrupt Service Routine, que se debe encargar de manejar la excepción y mostrarnos el error. Esto es totalmente programable a nivel de kernel.