Objetivos del curso:

Entender:

Diseño de SO

Implementación de SO

Hands-on: Construir SO

Propósito de SO:

Dar soporte a las aplicaciones: El SO es un intermediario entre las aplicaciones y el hardware.

Abstraer el HW para temas de conveniencia y portabilidad, si hago una aplicación en una computadora debo poder ejecutarla en otra.

Gestionar o multiplexar las aplicaciones, estas aplicaciones pueden compartir un recurso de HW.

Aislar las aplicaciones para evitar que estas afecten a otras aplicaciones.

Permitir gestionar la memoria para que las aplicaciones puedan compartir recursos.

Alto rendimiento, esto es, permitir que las aplicaciones se ejecuten lo más pronto posible.

Enfoque de diseño SO:

Pequeño: Cuando se administra las librerías de HW

Grande: Cuando se permite una abstracción con mejores propiedades de las máquinas físicas.

Orientación:

Capas:

Aplicaciones de usuario (vi, gcc, c):

Servicios del kernel:

HW (CPU, memoria, disco)

vi: editor en unix.

gcc es el compilador.

c es el lenguaje.

Servicios del Kernel de un SO:

Temas de procesos.

Temas de localización de memoria.

Temas de contenido de archivos.

Temas de nombres de directorios.

Temas de seguridad.

Otros temas: usuarios, IPC (Inter-Process Comunication), red, tiempo, etc.

Proceso: Es un programa en ejecución.

Las aplicaciones se comunican con el Kernel solamente mediante llamadas al sistema.

¿Por qué el diseño o la implementación de un SO es difícil o interesante (complejo)?

Porque debe ser eficiente (alto rendimiento, que son cuestiones de bajo nivel), portable (debe ser abstracto, que son cuestiones de alto nivel).

El entorno es variable, es decir, hay muchos elementos de hardware, de diferentes marcas y modelos.

Debe ser potente (que cumpla con varias características) pero también debe ser simple (pocos bloques, es decir, debe ser modular, pocas herramientas para interactuar con esas características).

Problemas abiertos de seguridad.

¿Por qué es interesante aprender de sistemas operativos?

Por la necesidad de trabajar y resolver los problemas antes mencionados.

Para saber que pasa por dentro del SO.

Para saber cómo construir sistemas de alto rendimiento.

Por la necesidad de diagnosticar bugs o problemas de seguridad.

El CPU se comunica con dispositivos I/O y la memoria principal.

La memoria retiene (holds) instrucciones y datos.

El CPU interpreta las instrucciones.

El CPU tiene un puntero que accede a las instrucciones y datos en la memoria principal.

El puntero de instrucción (EIP) me permite saber en qué parte de la memoria estoy.

Registros para espacio de trabajo, que pueden ser de 8, 16 y 32 bits.

Algunos registros son para propósito especial.

EFLAG register: sirven para saber los estados de lo que está sucediendo en el computador a nivel de procesos del SO.

Instrucciones de memoria: MOV, PUSH, POP para mover, poner y sacar en el manejo de pilas, respectivamente.

Algunas instrucciones pueden tomar una dirección de memoria.

Operaciones de manejo de pila:

Lenguaje ensamblador

Pushl %eax subl $4, %esp

movl %eax, (%esp)

Popl %eax movl $4, %esp

pushl %eip(\*)

Call 0x12345 movl $0x12345, %eip(\*)

Cada push cambia la dirección de memoria en la pila, que se llena de arriba para abajo, por cada push la dirección se va decrementando, y por cada pop se va incrementando.

Los 8086 y 80386 son procesadores.

8086 tiene 16 registros y un bus de direcciones de 20 bits, los registros de segmento indican:

CS: código de segmento

SS: segmento de pila

DS: segmento de datos

ES: otros segmentos de datos

80386 los datos son de 36 registros, ahora hay transacción a direcciones de 64 bits, y añade direcciones de memoria virtual.

Lpt\_pute(int )

{

/\*Cuerpo del código\*/

}

8086 solo maneja 1024 direcciones de I/O

Mapeo de memoria:

Direcciones de usuario normal

Works like magic memory

Esto es un mapeador de memoria de las I/O.

En la memoria se guardan datos en el siguiente orden:

Low memory

VGA display

16-bits devices

Extended memory

?

?

Instruction classes:

Movimientos de datos: MOV, PUSH, POP

Aritmética: TEST, SHL, ADD

I/O: IN, OUT

?

?

?

Convenciones de llamadas en Gcc para JOS

Push

Movl

Investigar eax PREGUNTAR

Ejemplo

Int main(void) {return f(0)}

Int f(int x) {return g(x)}

Int g(int x) {x+3}

Proceso de ejecutar un programa en c

.c – gcd – asm – gas—io –

.c –gcc—asm—gas—io—id—a.out—loader—memory

Compilador, ensamblador, enlazador y cargador.

QEMU hace lo que haría una computadora normal, únicamente implementado en software.

For(;;) //un lazo infinito

{

…

}

Nip: puntero de instrucciones, una vez que se ejecuta.

En la parte de la memoria hay 2 operaciones básicas, de lectura y escritura.

EMU también emula dispositivos.

Disco duro: usando un archivo del host.

PantVGA

¿

¿

Resumen:

Para el laboratorio se requiere una pc y que la pc sea Intel (x86)

**El tema del computador con programa almacenado: La memoria recibe las instrucciones y datos, el CPU se vuelve flexible al manejar las instrucciones de la memoria.**

**La pila se va llenando de arriba hacia abajo**

**Tema del mapeo de memoria a los dispositivos IO**

**Software = hardware**

**Dentro del laboratorio hay unos retos o challenge, no es necesario resolverlos todos, pero si algunos.**

DEBER:

Correr unos retos (algunos).