Curso 6.828 Operating System Engineering

<https://pdos.csail.mit.edu/6.828/2017/schedule.html>

Debugging C with GDB

<https://www.youtube.com/watch?v=5yZIFmplXsw&list=PLMbKVy-fuVKF0iopxoNRvhE86xYWZz9-C>

info reg

registro $esp

x/24x $esp

x/8x $esp

You can use the command x (for “e**x**amine”) to examine memory in any of several formats, independently of your program’s data types.[1]

What is Kernel in Operating System Explained with its Types

<https://www.youtube.com/watch?v=rg6ww5U-XJ8>

* **Que es el Kernel?**

El kernel es el componente central de un Sistema Operativo, responsable de la comunicación entre software y hardware.

**Es responsable de:**

La administración de procesos.

La administración de dispositivos.

La administración de la memoria.

El manejo de interrupciones.

La comunicación de archivos de entrada y salida.

Los sistemas de archivos.

* **Qué tipos de kernel existen?**

Los kernels monolíticos.

Los microkernels.

Las responsabilidades del microkernel son mucho menores que las del kernel monolítico.

Windows usa un kernel híbrido monolítico y micro kernel, mientras que linux usa un kernel monolítico.

OS #01 Introduction to OS, its Roles & Types

<https://www.youtube.com/watch?v=k51934LHz3k&t=25s&index=1&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun>

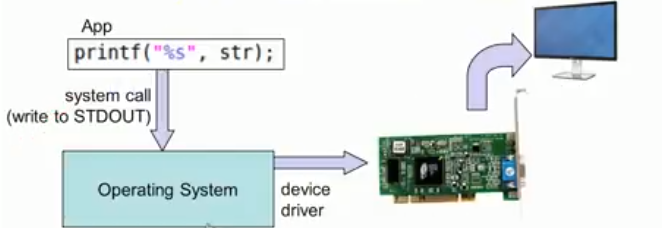
**Las capas en los sistemas**

* Aplicaciones
* Sistemas Operativos
* Organización
* VLSI
* Transistores

**Uso del Sistema Operativo**

* **Abstracción de Hardware:** convierte el hardware en algo que las aplicaciones pueden usar
* **Administración de Recursos:** Administrar los recursos del sistema

**Los SO Proveen abstracción (Ventajas)**



* **Fácil para programar aplicaciones:** No más detalles esenciales para los programadores.
* **Funcionalidad reutilizable:** Las aplicaciones pueden re-utilizar las funcionalidades del SO.
* **Portable:** Las interfaces de los sistemas operativos son consistentes. Las aplicaciones no cambian cuando el Hardware cambia.

**SO como administradores de recursos**

El SO debe administrar, cpu, red, memoria, almacenamiento secundario (disco duro), etc.

Administración de recursos:

* Permite que varias aplicaciones compartan recursos
* Protege aplicaciones entre sí
* Mejorar el rendimiento mediante la utilización eficiente de los recursos

**Tipos de SO**

**Aplicación Específica**

* **SO Embebidos:** Ej. Contiki OS para entornos extremadamente limitados de memoria
* **SO Móviles:** Android, iOS, Ubuntu Touch, Windows Touch
* **RTOS:** QNX, VxWorks, RTLinux
* **Entornos seguros:** SeLinux, SeL4
* **Para servidores:** Red Hat, Ubuntu, Windows Servers
* **Escritorio:** Mac OS, Windows, Ubuntu

OS #02 PC Organization/ Hardware, Address Types, Address Range

<https://www.youtube.com/watch?v=mio9OsckCZM&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun&index=2>

El corazón de cualquier sistema es el CPU

* **¿Todos los dispositivos pueden compartir una misma dirección?**

No, cada dispositivo tiene una única dirección de memoria

* **¿Qué tipos de memorias existen?**

Direcciones de memoria: donde se mapea la memoria principal (RAM)

Se accede mediante instrucciones load/store

Extended Memory

BIOS ROM? Basic I/O System

16-bit devices, expansión ROM’s

VGA Display

Low memory

* **Si tengo 16GB de RAM, que parte de la memoria cambiaria?**

La memoria extendida se extiende para abarcar parte de la memoria sin uso (la dirección inicial es 0x00100000 y la final depende del monto de la RAM).

Tipos de direcciones:

Direcciones de memoria

rango (2^32)-1

IO Ports

Usa un bus diferente comparado con el acceso a la memoria RAM

Cual es el rango?

los rangos son asignados por la cpu, (2^16)-1

Para qué sirven?

para recibir y pasar información a los periféricos de entrada y salida

Cómo se accede?

se accede por medio de las dirreccion ya previamente asignadas por la CPU, a estas direcciones asignadas no pueden ser utilizadas por ningún otro dispositivo

Memory mapped I/O

para que se lo utiliza?

métodos de implementar entradas/salidas entre los periféricos y la CPU en un computador

Dispositivos y RAM comparten el mismo espacio de memoria?

si, por esto este mecanismo mapea esto para poder acceder a los dipositivos.

Quien fija los rangos de direcciones?

la CPU

Standards

¿Las direcciones son fijas? ¿por qué?

tenian fijas todas las direcciones. ´porque eran compatibles con los estandares de IBM.

Dispositivos plug/play

¿Tiene direcciones fijas? Quien fija las direcciones? BIOS o OS?

no tiene direcciones fijas. quien fija esto es la BIOS

PC Organization

Processor’s

front side bus

North bridge

Memory bus

DRAM

PCI Bus

Interfaz entre PCI’s: PCI-PCI bridge

The x86 Evolution (8088)

uP 16 bits

Registros de 16 bits

Registros de propósito general: AX, BX, CD, DX

Registros de puntero:

Instruction pointer:

Segment registers:

Acceso a memoria

The x86 Evolution (80386)

uP 32 bits

32 bit external address bus

Registers are 32 bits

* características: protected operating mode, virtual addresses

The x86 Evolution (k8)

AMD k8 (2003)

RAX instead of EAX

X86-64, x64, amd64, intel64. all same things

OS #03 Programs & Processes, System Calls, OS Structure

<https://www.youtube.com/watch?v=n5lgcKch3Hk&index=3&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun>

OS #04 CPU Sharing, Race Conditions, Synchronization, CPU Scheduling

<https://www.youtube.com/watch?v=Cuo8GTklgoE&index=4&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun>

Compartiendo el CPU

Como el SO decide que app se ejecuta?

* Ejecuta App1, cuando termina, siguiente: es eficiente?
  + Qué sucede si en la aplicación se espera por un evento?
* Multiprogramacion: Cuando una app (App1) espera por un evento, se cambia a otra App (App2): cual es el pro y contra?
  + Pro:
  + Contra: App2: While(1)
* Multitasking: el tiempo del CPU esta dividido en slices (partes)
  + cada app se ejecuta dentro de un slice
  + da la impresion de que las apps se ejecutan concurrentemente
  + no starvation(inanicion). Performance mejorado
  + time slice/time quanta
  + Dentro de un slice puede ejecutarse más de una app?
* Otros recursos compartidos: impresoras, teclados, RAM, discos, archivos en disco, redes, etc.

Multiprocesadores

* Multiples chips procesadores en un sistema
* Multiples cores dentro de un chip
* multiples threads(hilos) en un core
* Cada procesador puede ejecutar un app independientemente de otras apps
* Race Conditions: cuando se da? que se requiere?
* Sincronizacion: tipo de candado asociado a un recurso. el candado asegura que solamente una app pueda acceder al recurso en un tiempo.
  + Secuencia: appx blinda el recurso, appx accede al recurso, mientras app y espera. appx desbloquea el recurso, app y accede al recurso.

Quíen se debe ejecutar luego?

* Scheduling: algoritmo que determina cual app se debe ejecutar luego.
* Debe ser justa
* Debe priorizar ciertas apps sobre otras

**OS y aislamiento (isolation)**

* Por que?
  + Múltiples apps se ejecutan concurrentemente, cada app puede ser de un usuario diferente. entonces se requiere aislamiento.
  + Prevenir que el malfuncionamiento de una app afecte otras apps.
* Estructura de anillo (ring):
  + primero se asegura que el OS se ejecute en modo protegido (parte interna del anillo, mayor privilegio)
  + El anillo externo -> aplicaciones de usuario, menor privilegio.
  + Cuando ejecuto una app como web browser, en qué anillo se ejecuta?
  + Puedo aislar las apps para diferentes usuarios?
* Como se consigue?
  + Control de acceso: administrar los recursos a los cuales se tiene acceso. Sólo usuarios autorizados pueden acceder a los archivos y otros recursos. Tipos de permisos (own, read, write, execute, etc)
  + Claves y criptografía
  + Biometrics (biometría)
  + Evaluación de seguridad: Cuan seguro es mi sistema? Análisis matemático, métodos de verificación manual/semi-automática

OS #05 Memory Management: Process, Fragmentation, Deallocation,

<https://www.youtube.com/watch?v=DdUeTN0qfuE&index=5&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun>

* **Como el sistema operativos maneja la RAM?**
* **Donde se almacena un archivo ejecutable?**

EN el disco duro

* **Donde se ejecute dicho archivo? como se llama cuando se ejecuta?**

En ram, se llama proceso

* **La RAM tiene recursos ilimitados de memoria?**

No

* **Solamente un proceso puede ser ejecutado en memoria?**

No

* **Como el SO controla los recursos de la RAM?**

Single Contiguous Model

* Se comparten procesos?
* El tamaño de memoria del proceso es restringido por el tamaño de la RAM
* Que pasa cuando el tamaño de un proceso supera al tamaño de la RAM?

Partition Model

* cuando hay suficiente espacio disponible, nuevos procesos son ubicados en memora.
* Tabla de particion: direccion de memoria, tamaño, proceso, uso
* ¿Que pasa cuando llega un nuevo proceso?
* ¿Que pasa cuando un proceso termina?
* Que problema puede acarrear este modelo? Fragmentation
* Se puede ejecutar un proceso en espacios de memoria no contiguos?
* Finding the right fit:
  + first fit: pros y cons
  + best fit: pros y cons (¿rendimiento se ve afectado, por que?)
* Deallocation
  + ¿Qué debe hacer el SO?

Limitaciones:

¿Todos los procesos deben estar en memoria?

OS #06 Virtual Memory & Demand Paging in Operating Systems

<https://www.youtube.com/watch?v=g9HTAK0WwkY&index=6&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun>

Virtual Memory

* **Como se particiona la RAM?**

La Memoria ram se divide en particiones del mismo tamaño( típicamente 4Kbs) llamadas Page Frames

* **como se llaman a las particiones?**

Page Frames

* **Procesos son particionados?**

Si, los procesos son particionados de manera similar, en bloques de igual tamaño

* **los bloques de procesos deben estar contiguos?**

Debido a que se utilizan las Process page tables, los bloques de un proceso particular no necesitan estan alocados en memoria ram de manera continua.

cada acceso a memoria está relacionado a una tabla lookup

* **donde estan las process page tables?**

Se ubican en la memoria RAM

kernel

overhead: se refiere a los pasos extra que deben tomarse cuando se desea acceder a un bloque en particular de un proceso. El overhead de un proceso al ejectarse utilizando memoria virtual es producido por la necesidad del CPU de buscar en la tabla para poder identificar el número de pageframe correspondiente.

context switch

* **¿en qué momento la process page table se vuelve activa?**

En el momento que su proceso se encuentre en estado de ejecución.

* **Se requiere cargar todos los bloques del proceso en memoria? Por que?**

No, porque no todas las partes de un programa van a ser accedidas simultáneamente. Hay partes que probablemente no sean ejecutadas

**Todo proceso que corre en memoria debe tener una tabla?**

Si

Tabla en memoria: block, page frame

Demanding paging:

**Swap space.-** espacio en el disco donde se encuentran los bloques de un programa, que son cargados a memoria solo cuando sea necesario

**present bit.-** bit en la tabla de paginas de procesos, que indica si un bloque se encuentra en RAM o no

page fault:Ocurre cuando se requiere acceder a un bloque del proceso que no esta presente aun en la memoria ram (Bit de presencia en 0)

**que hace el SO?**

El sistema operativo debe tratar de cargar la page frame desde el espacio de wap hacia la memoria ram y marcar el bit de presencia como 1. En el caso de que no haya mas espacio en Ram o en la process page table, debe remover bloques.

**cual de los bloques debe ser removido?**

algoritmo en el SO:

place replacement algoritm

First in first out

least recently used

least frequently used

**swap out?**

Es el proceso de mover un bloque desde RAM al Disco

**swap in?**

Es el proceso de mover un bloque desde Disco a RAM

**dirty bit: que indica si los contenidos del bloque en el block ha sido modificado**

Si el Dirty bit tiene el valor de uno, indica que el valor en Ram de ese bloque ha sido modificado y es distinto al valor del bloque que se encuentra en el espacio de swap.

**protection bits?**

Bits adicionales, que determinan atributos de un bloque particular.

**atributos de un bloque particular:**

* Ejecutable
* Read-Only
* Accesible por un user process

OS #07 MMU Mapping | How Virtual Memory Works?

<https://www.youtube.com/watch?v=AKGtJAi4wGo&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun&index=7>

**Secciones de memoria de un proceso:** Text(instrucciones), data (global y estático), Heap, Stack

**¿Como se mapea la memoria virtual en la física?**

Procesador -> dirección virtual

MMU (memory managment Unit): mapea la dirección virtual v a una dirección física p en RAM

**Proccess page table in RAM:**

Cuando un programa empieza a ejecutarse, el OS crea una tabla de página en RAM

PTPR (page table pointer register)

la dirección virtual del procesador tiene dos partes: table index y un offset

**Offset**

MMU mapping en 32 bit

dirección virtual es de 32 bits

table index 20 bits

offset: 12 bits

cada page frame es de 4KB, entonces se requieren 12 bits para direccionar una página. entonces el offset es de 12 bits y el índice 20 bits.

**Número de entradas en page table** es 2\*\*20 (alrededor de 4MB)

**Nótese que la page table requiere tener 4MB de memoria contigua!**

Entonces?

**Translation de Página de 2 niveles**

Dirección: 10bits, table: 10 bits, offset: 12 bits

Número de page tables: 2\*\*10=1024

Total page table size 4MB

**¡Pero no contiguas!**

Virtual Memory (sequence steps)

**3 entidades involucradas en la memoria virtual: ???**

OS, Procesador y MMU.

Cuando un programa inicia la ejecución, OS crea si page tables en RAM

**la page tables puede estar vacía**

…

para tomar la primera instrucción, CPU pone la memoria virtual

MMU looks up the page table

P: present bit

D: dirty bit

Processor

MMU

**Como se ejecutan varios procesos en memoria?**

Virtual address space, translation map and physical address space:

<https://www.youtube.com/watch?v=aNmo2bpW3dI>

OS #08 Memory Management: Segmentation & Fragmentation

<https://www.youtube.com/watch?v=SqYigYLFvcI&index=8&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun>

**Los programas son una colección de modulos logicos**

Segmentos usados comúnmente: Text, Data, Heap, Stack

Segment descriptos table (guardado en memoria)

registros en el procesador: segment selector, offset register, ptr to descriptor table

**logical address**: seg. selector + offset

**linear address?**

pointer to descriptor table: 0x3000

segment reg: 1

…

**segmentación y fragmentación**

**Fragmentación:**

x86 address translation

CPU

Segmentation unit

Paging Unit

Physical Memory

Pointer to Descriptor Table

GDTR

Segment Descriptor Access, Limit, base address

Segment and offset registers

offset registers: 32 bits

**segmentos asociados:** Code, Data, Stack

**Linear to Physical Address**

2-Level page translation

**Cuantas page tables están presentes?**

2\*\*10 page tables maximo

segmentation vs paging

OS #09 Memory Management in xv6 Systems

<https://www.youtube.com/watch?v=8LS8TCxJqCg&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun&index=9>

Traduccion de direccion en el x86: cuantos niveles de direccionamiento de memoria?

logical-linear-physical

Segmentation unit - Paging Unit

GDT: Global Descriptor Table

Segmentation unit en xv6?

Segment descriptor: 64bits

base add: 3 parts

segment limit: 2 parts

nivel de privilegio: 0: kernel, 3: USER

segment type: X, R, W

Segment descriptor: struct segdesc

macro: Normal Segment SEG

ej: SEG(STA\_W, 0, 0Xffffffff, DPL\_USER)

Segments in xv6

* Kernel Code
* K data
* U code
* User data

Loading GDT

Paging

Mapping kernel in xv6

KERNBASE

V2P: virtual to physical

P2V: physical to virtual

Physical mem

* Base
* IO
* Extended
* Devices

kmap: 4 regions (IO space, hern text+data, kern data+ memory, more devices)

Creating page talbe mapping for kernel:

* Enable paging
* Create/Fill page directory
* Create/Fill page tables (mappages): fill page table entries mapp...
* Load CR3 register: CR3: pointer to page directory

Page Table: 4KB 1023x32bit

Allocating Pages (kalloc):

* Free physical pages in a list (freelist)
* Page allocation removes from liste head (kalloc)
* Page free add to list

OS #10 PC Booting: How PC Boots | Explained in Detail

<https://www.youtube.com/watch?v=ZplB2v2eMas&index=10&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun>

Powering Up

* Power on Reset: reset signal

Regs = 0, except CS: 0xF000 IP(instruction pointer): 0xFFF0

* First instruction: 0xfff0

BIOS ROM:

* Power on self test
* Inicialize video card +
* Display Bios screen
* Memory test
* Set DRAM
* Configure plug and play
* Assign resources
* Identify the boot device: Read sector 0 from boot device into memory location 0x7c00, jump to 0x7c00

MBR Execution (Master boot record): 512 bytes

Bootloader: Loads the OS

OS:

* setup virtual mem
* init interrupt vectors
* Init: timers, monitors, hard disks, consoles, …
* init othre processor
* Startup user process

Multiprocessor Ortganization

* Mem Symmetry: all proc share same mem space
* IO simmetry: all proc share same IO system

Multiproc Booting

* One processor as Boot Processor (BSP)
* BIOS boots BSP
* BSP learn sys config
* BSP triggers boot of other App

OS #11 Process, Process Stack, PCB & Attributes

<https://www.youtube.com/watch?v=b4fsyrWegGo&t=41s>

**where does the kernel reside?**

Kernel: lower part of mem

virtual: user+kernel

Virtual add to Physical add

**kernel space in all process?**

Si

kernel data about a process

Process stacks: 2 stacks (user, kernel)

* User space stack: when exec user code
* kernel: use when kernel code in context of process

PCB : process control block

PID: process identifier

Process states

* NEW, READY, BLOCK, RUNNING

How one state to another?

EXIT state

Scheduling runnable Processes

* Queue of READY Processes

Entries in PCB

Storing procs in xv6

* NPROC: max number of proc in the siystem: 64

OS #12 Create, Execute & Exit from a Process | fork() System Call

<https://www.youtube.com/watch?v=V6PQClbz0zY&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun&index=11>

Creating process by cloning: fork

p -1, 0 , …

parent, child process

in parent: p=childs PID

in child: p=0

fork: from OS perspective

* Change state from new to ready

Copy on Write (COW)

Executing a new program

* fork and the exec
* exec (system call): find on hd the location of a.out executable, load on demand the pages required to executer a.out

Advantage of COW

OS #13 System Calls for Process Management | fork, wait, exec

<https://www.youtube.com/watch?v=aZKNAWpFVCg&index=12&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun>

fork clona un proceso identici al padre

fork retorna el id del hijo(child) si estoy en el proceso padre.

Si el proceso es del hijo, fork retorna cero.

Padre espera al hijo hasta que termine su proceso.

REGISTROS MODIFICADOS EN PROCESO HIJO

%eax = 0

%eip = forkret (excusivo de proceso hijo) codigo xv6

exit internals

1 Decrementa el contador, si contador es cero, se cierra el file(en linux todo proceso, device, etc es manejado como file)

2 Colocar referencia al inode en memoria

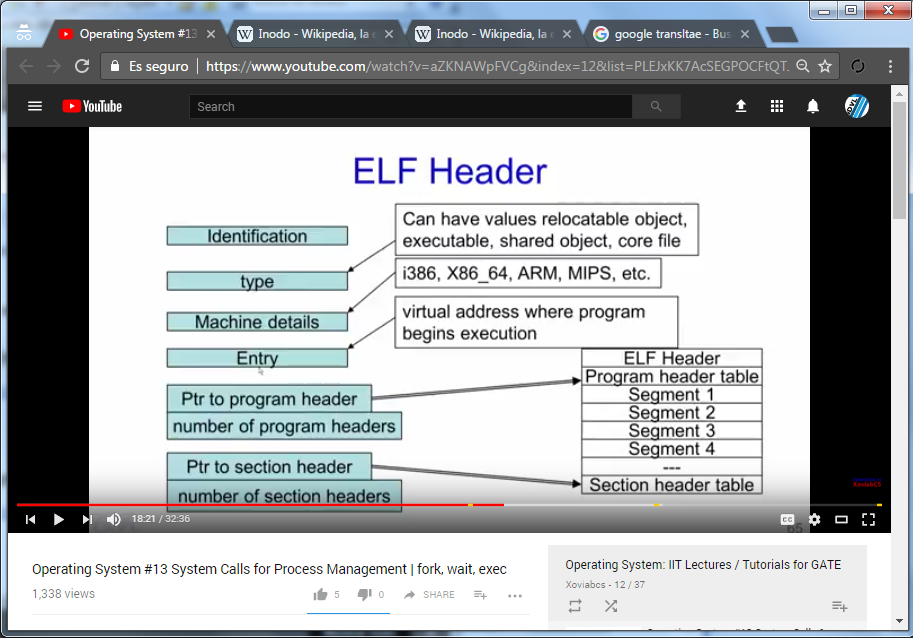
3 Despertar al padre si está en modo sleeping, hacerlo runnable.

4 hacer que init adopte hijos de proceso salido

Si el padre continua el proceso sin esperar al hijo, el hijo pasa al estado ZOMBIE.

**exec**

Carga en memoria otro programa y luego ejecuta



gcc hello.c -c crea el objeto a ejecutar

readelf -h hello.o muestra el ELF Header

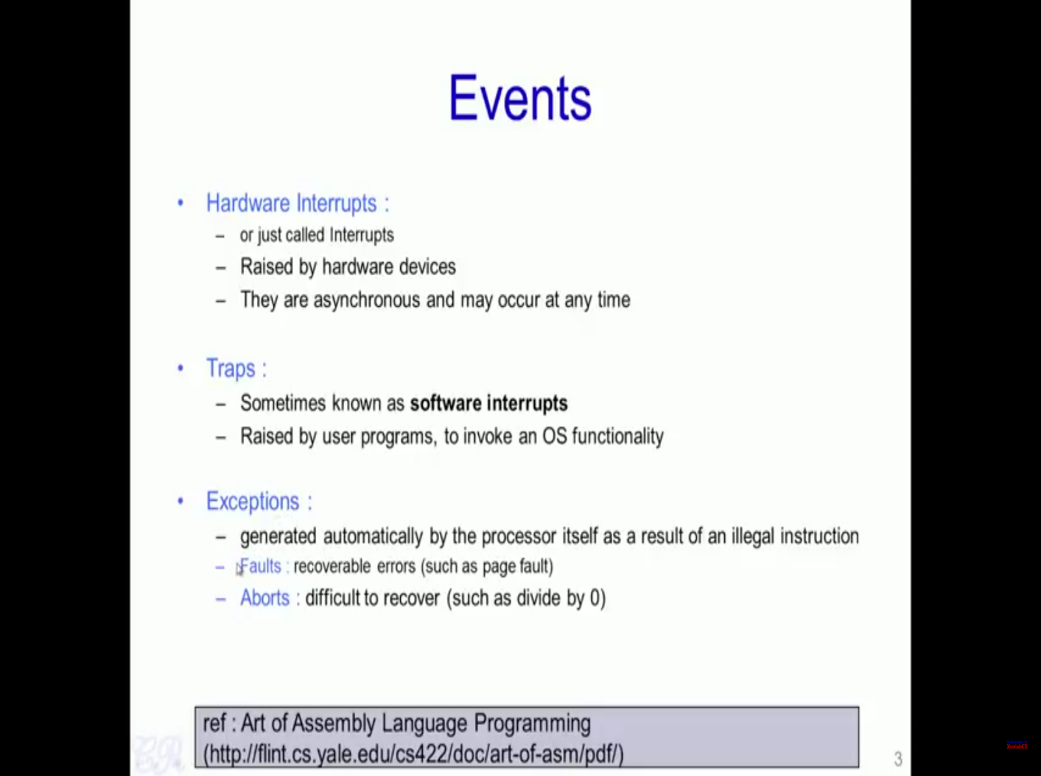
OS #14 What is an Interrupt? Types of Interrupts

<https://www.youtube.com/watch?v=rnGVincwk30&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun&index=13>

OS es un evento conducido, se ejecuta solo cuando existe una interrupcion.

del **user mode** se manda la interrupcion y esta es recibida por el **kernel mode** ejecutando lo correspondiente, para luego retornar al **user mode**.

**Eventos**

****

**-Hardware interrupts**

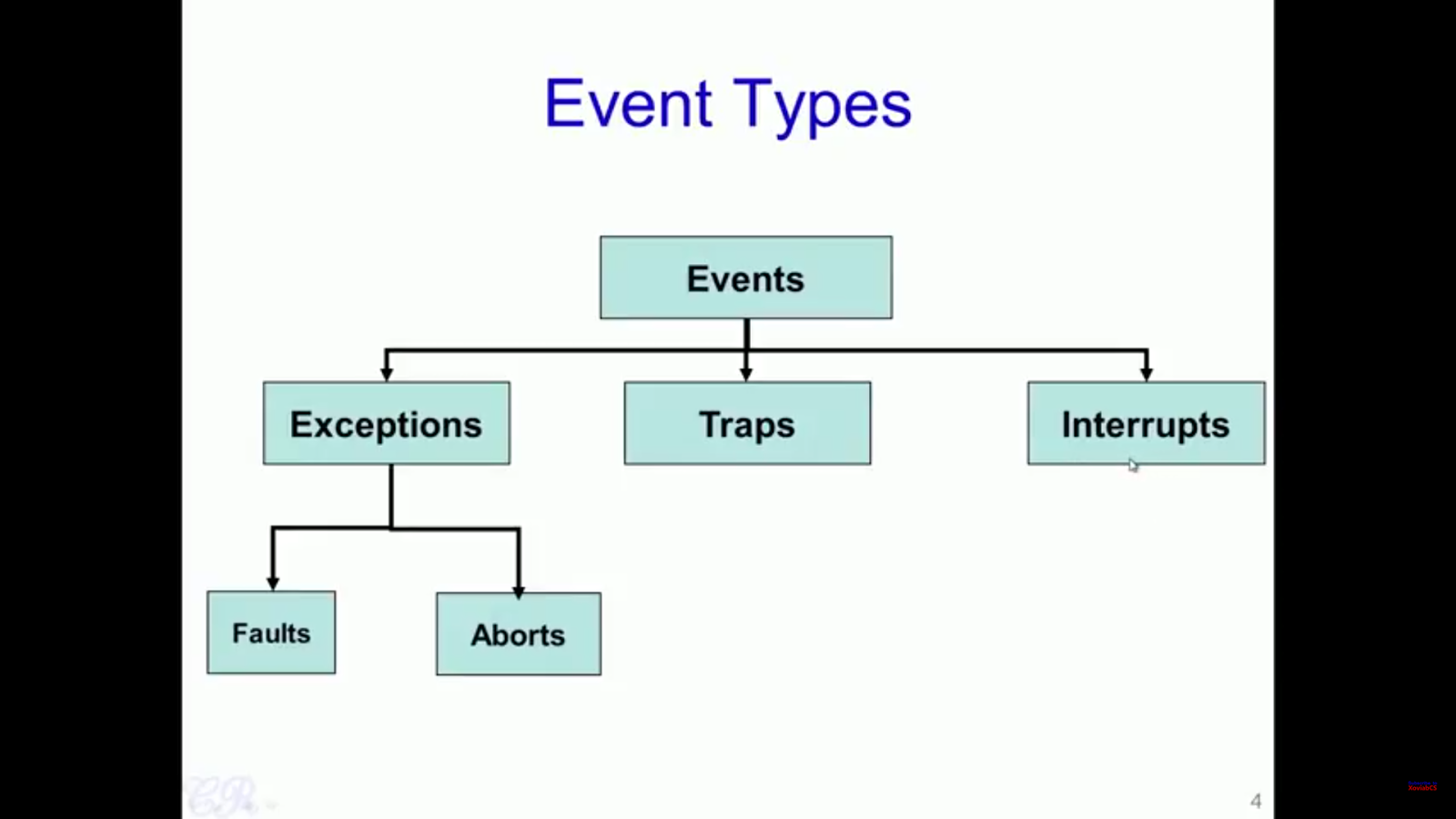
* Hardware devices
* Asincrono, ocurre en cualquier momento.
* Interrupt controller 8259(asegura que un pin de la interrupcion  es compartido entre los multiples devices, recibe la interrupcion de cada device y luego canaliza el pin del procesador, luego este e comunica con el interrupt controller para determinar cual device está generando actualmente la interrupción para finalmente ejecutar la rutina respectiva de la interrupción), más de 8 devices.

**-Traps**

* Software interrupts
* user programs, para invocar alguna funcionalidad del SO

**-Exceptions**

* Faults(page fault, cuando un proceso es ejecutado en el SO y carga la page requerida del swap space en la RAM)
* Aborts(DIVISIÓN para 0)



OS #15 Interrupt Handling Explained in Detail

<https://www.youtube.com/watch?v=V1TJ6b6_EjM&list=PLEJxKK7AcSEGPOCFtQTJhOElU44J_JAun&index=14>

**Que pasa cuando ocurre una Interrupción?(INT)**

1. El cpu detecta la línea de interrupción y obtiene el número IRQ desde PIC
2. Cambia hacia el kernel stack de ser necesario, y cambia el nivel de privilegio si se necesita.
3. El CPU guarda el estado básico del programa y suspende las actividades que se estaban ejecutando
4. El CPU realiza un Jump hacia el manejador de interrupciones (Interrupt Handler)
5. El SO ejecuta el manejador, y se encarga de las cosas más importantes como responder a la interrupción, guardar el estado del programa, organizar El Bottom Half, IRET
6. Luego de esto, el CPU realiza un retorno de la interrupción( una vez que el sistema operativo ha realizado todo lo que se necesitaba)
7. Finalmente, el SO realiza las acciones que organizó en el bottom half ( esto es, realiza todas las actividades de comunicación dadas en el interrupt)

**Todos los procesos tienen dos stacks:**

User Space Stack.- visible para los procesos de usuario

Kernel Space Stack.- Escondido para los procesos de usuario, usado para almacenar variables y tambien llamadas a funciones que se ejecutan en el kernel

**Por que se cambian stacks? (Stack Switching)**

El sistema operativo no puede confiar solamente en el stack de procesos de usuario (puede estar dañado)

Los procesos de usuario no son capaces de acceder al kernel Stack. Por ejemplo, si el proceso de usuario contiene codigo malicioso (un virus), no podrá acceder al kernel stack fácilmente.

**Como se efectúan estos cambios de stack(desde user stack a kernel sack)?**

Se realiza por el Task segment Descriptor, que cambia automáticamente  el Stack Segment y el Stack Pointer del procesador.

Así  mismo, cuando se ejecuta un cambio de stack, los privilegios cambian desde Bajo a Alto privilegio

TODO SE REALIZA AUTOMÁTICAMENTE POR EL CPU

**Por qué se debe Guardar estados luego de que ocurra una Interrupción?**

Para que el proceso pueda seguir ejecutándose una vez que la interrupción sea manejada

**Un típico interrupt Handler realiza tres operaciones:**

* Guarda información adicional acerca del proceso que está siendo invocado.
* Procesa la interrupción (donde normalmente se comunica con dispositivos de entradas y salidas)
* Restablece el contexto al Procesador para poder retornar al user process

**Latencia de Interrupción**

Tiempo desde que se dispara una interrupción hasta que el Manejador de interrupciones puede ser ejecutado.

Es importante dependiendo del sistema operativo

Puede variar entre un mínimo y un máximo de tiempo

El mínimo es definido por los controladores.

El máximo tiempo depende del sistema operativo

Para mejorar el tiempo de respuesta, se pueden activar interrupciones dentro de los manejadores, pero esto puede causar **interrupciones anidadas**, lo que ocasiona una dificultad al desarrollar y validar procesos en el sistema operativo.

Idealmente, se deberían diseñar los manejadores de interrupciones lo más pequeños posibles, para que no ocurran interrupciones anidadas.

**Linux realiza una técnica de manejo de interrupciones, dividiendo los manejadores en dos grupos**

**Top Half:** se ejecutan las interrupciones que requieren el mínimo trabajo, como por ejemplo guardar registros, realizar otras interrupciones, llamar al Bottom Half, restaurar registros y volver a contextos anteriores.

**Bottom Half:** Realiza todos los procesamientos que no se realizaron en el Top Half

OS #16 Software Interrupts | System calls in xv6

<https://www.youtube.com/watch?v=KqgMGZyiLnU>

Hardware Interrupts vs Software Interrupts:

**Hardware:**

Un dispositivo envia un pulso a un Pin específico del CPU para llamar a una interrupción.

**Software:**

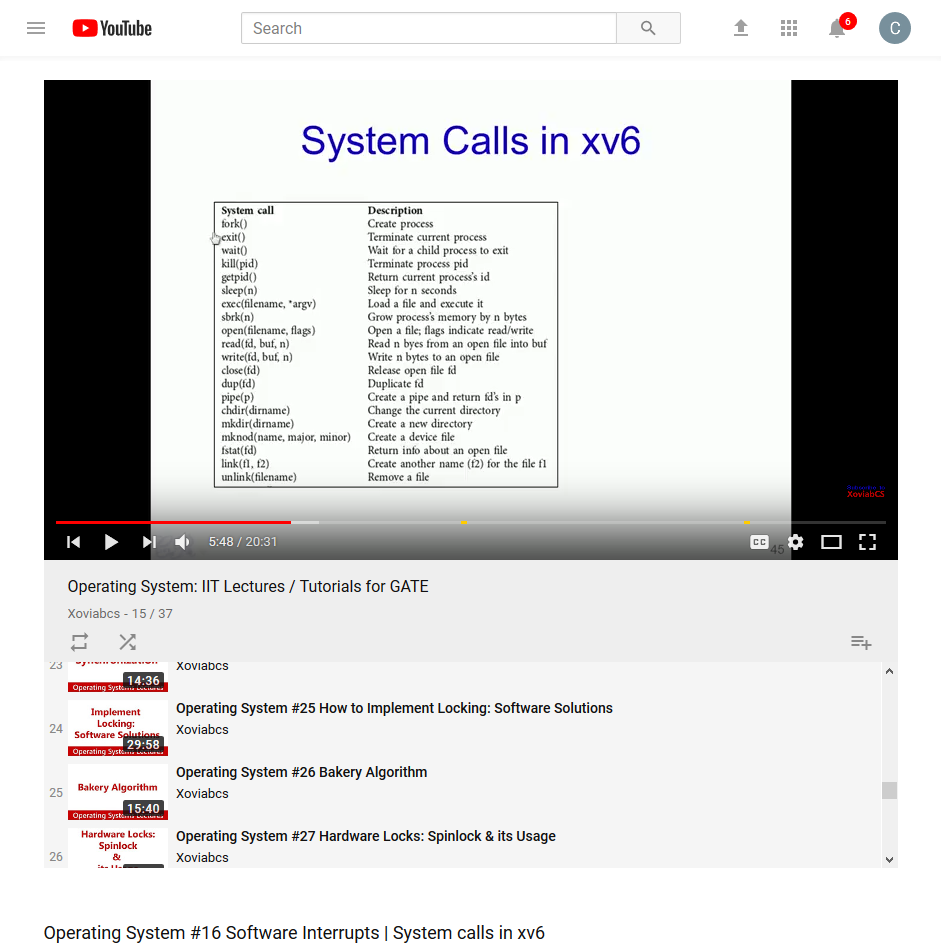
Una Instrucción que al ejecutarse causa una interrupción.

Las interrupciones de software son utilizadas para implementar llamadas de sistema (System Calls)

Linux: Utiliza  INT 128 para system calls

xv6: INT 64 para system calls

**System calls soportados en xv6**



**Como el SO puede distinguir entre system Calls?**

Este distintivo es definido por el User Process

int system\_call( resource\_descriptor, parameters) - int: return is generally ‘int’ (or equivalent) sometimes ‘void’. int used to denote completion status of system call sometimes also has additional information like number of bytes written to file.

- resource\_descriptor: What OS resource is the target here?

For example a file, device, etc. If not specified, generally means the current process.

- parameters: System call specific parameters passed.

• Passing parameters to system calls not similar to passing parameters in function calls. – Recall stack changes from user mode stack to kernel stack.

• Typical Methods

– Pass by Registers (eg. Linux)

– Pass via user mode stack (eg. xv6)

• Complex

– Pass via a designated memory region

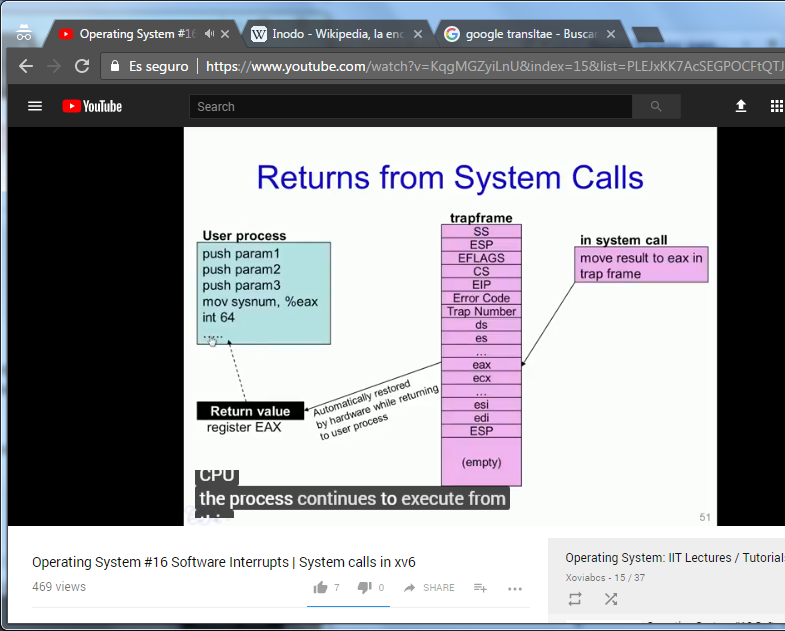
• Address passed through registers

• System calls with fewer than 6 parameters passed in registers

– %eax (sys call number), %ebx, %ecx,, %esi, %edi, %ebp

• If 6 or more arguments

– Pass pointer to block structure containing argument list



Instalación de xv6 en laboratorio 5 (Ubuntu)

**Dependencias:**

$ sudo apt-get install libsdl1.2-dev libtool-bin libglib2.0-dev libz-dev libpixman-1-dev bison flex

Luego de descargar el repositorio de qemu, y luego de instalar las dependencias necesarias pero antes de ejecutar la linea ./configure …..

$ git submodule update --init dtc

Ejecutar ./configure:

Al final ejecutar:

$ sudo make && sudo make install

Instalación qemu:

**Instalacion de QEMU por terminal:**

En primer lugar abre un terminal y ejecuta el comando:

***sudo apt-get install qemu qemu-kvm***

Descarga e instalación de XV6

Ahora que ya tenemos QEMU instalado podemos empezar a descargar xv6.

Para descargarlo es necesario tener ***git*** instalado. Para ello introduce esto en el terminal:

***sudo apt-get install git***Luego crear un directorio:  
$ mkdir 6.828  
$ cd 6.828

Ejecuntando el siguiente comando en terminal, descargamos los archivos necesarios para la instalación de xv6:  
$ git clone git://github.com/mit-pdos/xv6-public.git  
Build xv6 on Athena:

$ cd xv6-public  
$ make

Estado instalacion XV6 PC’s laboratorio 5

|  |  |
| --- | --- |
| PC | Observacion |
| 30 | W: chmod 0700 of directory /var/lib/apt/lists/partial failed - SetupAPTPartialDirectory (1: Operación no permitida)  E: No se pudo abrir el fichero de bloqueo «/var/lib/apt/lists/lock» - open (13: Permiso denegado)  E: No se pudo bloquear el directorio /var/lib/apt/lists/  W: Se produjo un problema al desligar el fichero /var/cache/apt/pkgcache.bin - RemoveCaches (13: Permiso denegado)  W: Se produjo un problema al desligar el fichero /var/cache/apt/srcpkgcache.bin - RemoveCaches (13: Permiso denegado)  E: No se pudo abrir el fichero de bloqueo «/var/lib/dpkg/lock» - open (13: Permiso denegado)  Y si me encontraba como superusuario |
| 20 | estudainte@GGIECLB5WRK020:~/qemu$ sudo apt-get install libsdl1.2-dev libtool-bin libglib2.0-dev libz-dev libpixman-1-dev bison flex [sudo] password for estudiante:  E: No se pudo abrir el fichero de bloqueo «/var/lib/dpkg/lock» - open (2: No existe el archivo o el directorio) E: No se pudo bloquear el directorio de administración (/var/lib/dpkg/), ¿está como superusuario?  En apt-get update:  Aborted (core dumped)  E: Problem executing scripts APT::Update::Post-Invoke-Success 'if /usr/bin/test -w /var/cache/app-info -a -e /usr/bin/appstreamcli; then appstreamcli refresh > /dev/null; fi'  E: Sub-process returned an error code  E: No se pudo abrir el fichero de bloqueo «/var/lib/dpkg/lock» - open (2: No existe el archivo o el directorio)  E: No se pudo bloquear el directorio de administración (/var/lib/dpkg/), ¿está como superusuario? |
| 22 | OK |
| 26 | OK |
| 37 | OK |
| 36 | OK |

<https://pdos.csail.mit.edu/6.828/2017/homework/xv6-boot.html>

$ mkdir 6.828  
$ cd 6.828  
$ git clone git://github.com/mit-pdos/xv6-public.git

Tools Used in 6.828

<https://pdos.csail.mit.edu/6.828/2017/tools.html>

**Problema:**

El archivo «/home/estudiante/Documentos/SO/xv6-public/.gdbinit» de carga automática ha sido rechazado por estar «auto-load safe-path» establecido a «%»

Para activar la ejecución de este archivo, añada

la línea     add-auto-load-safe-path /home/estudiante/Documentos/SO/xv6-public/.gdbinit

a su archivo de configuración «/home/estudiante/.gdbinit».

Para desactivar completamente esta protección de seguridad, añada

la línea     set auto-load safe-path /

a su archivo de configuración «/home/estudiante/.gdbinit».

Para más información sobre esta protección de seguridad, vea

la sección de «Auto-loading safe path» en el manual GDB. E.g., ejecuta desde ---Type <return> to continue, or q <return> to quit---

el shell:

   info «(gdb)Auto-loading safe path»

**Solución1:**

gdb -iex "set auto-load safe-path /"

Disable auto-loading safety for a single gdb session. This assumes all the files you debug during this gdb session will come from trusted sources [1]

**Solución2:**

La solución es crear el archivo de configuración de gdb “.gdbinit” en el home de usuario.

y dentro del archivo copiar las siguientes instrucciones:

* add-auto-load-safe-path “[ruta de carpeta]/xv6-public/.gdbinit”, como por ejemplo “/home/estudiante/6.828/xv6-public/.gdbinit”
* set auto-load safe-path /

Shell

sh.c file

<https://pdos.csail.mit.edu/6.828/2014/homework/sh.c>

[1] <http://student.itee.uq.edu.au/courses/csse2310/gdb-info/Auto_002dloading-safe-path.html>

Bibliografía

[1] http://sourceware.org/gdb/onlinedocs/gdb/Memory.html