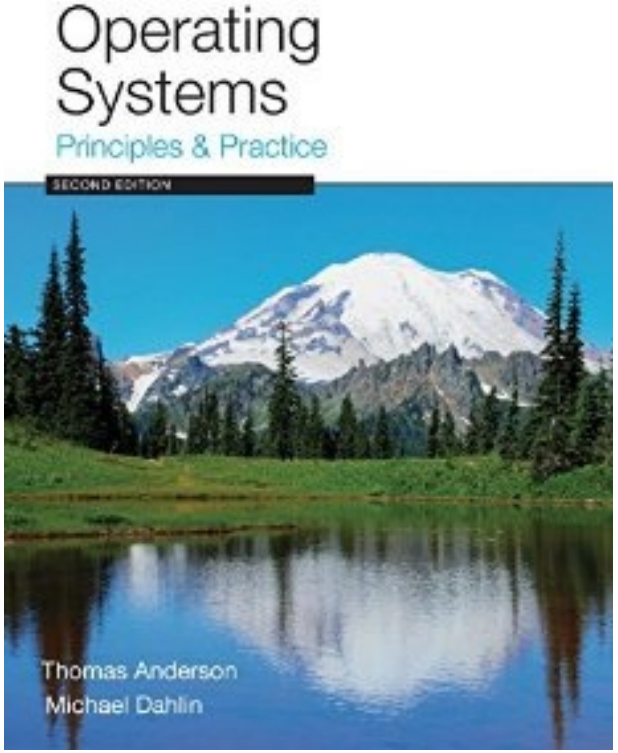
Sistemas Operativos

Capítulo 2 El Kernel

Prof. Javier Cañas R.

Estos apuntes están tomados en parte del texto: “Operating System: Principles and Practice” de T. Anderson y M. Dahin



Puntos Principales

- Concepto de Proceso
  - Un proceso es una abstracción del SO para ejecutar un programa con privilegios limitados.
  - Privacidad: Los datos sólo son accesibles a usuarios autorizados.
- Modo dual de operación: usuario vs. kernel
  - Modo kernel: ejecución con todos los privilegios.
  - Modo usuario: ejecución con privilegios restringidos.
- Cambio de modo seguro
  - ¿Cómo se pueden conmutar los modos?

Temario

1. El Concepto de Proceso

2.Modo Dual de Operación

3.Memoria y Direcciones Virtuales

4.Interrupciones

5.Llamadas al Sistema

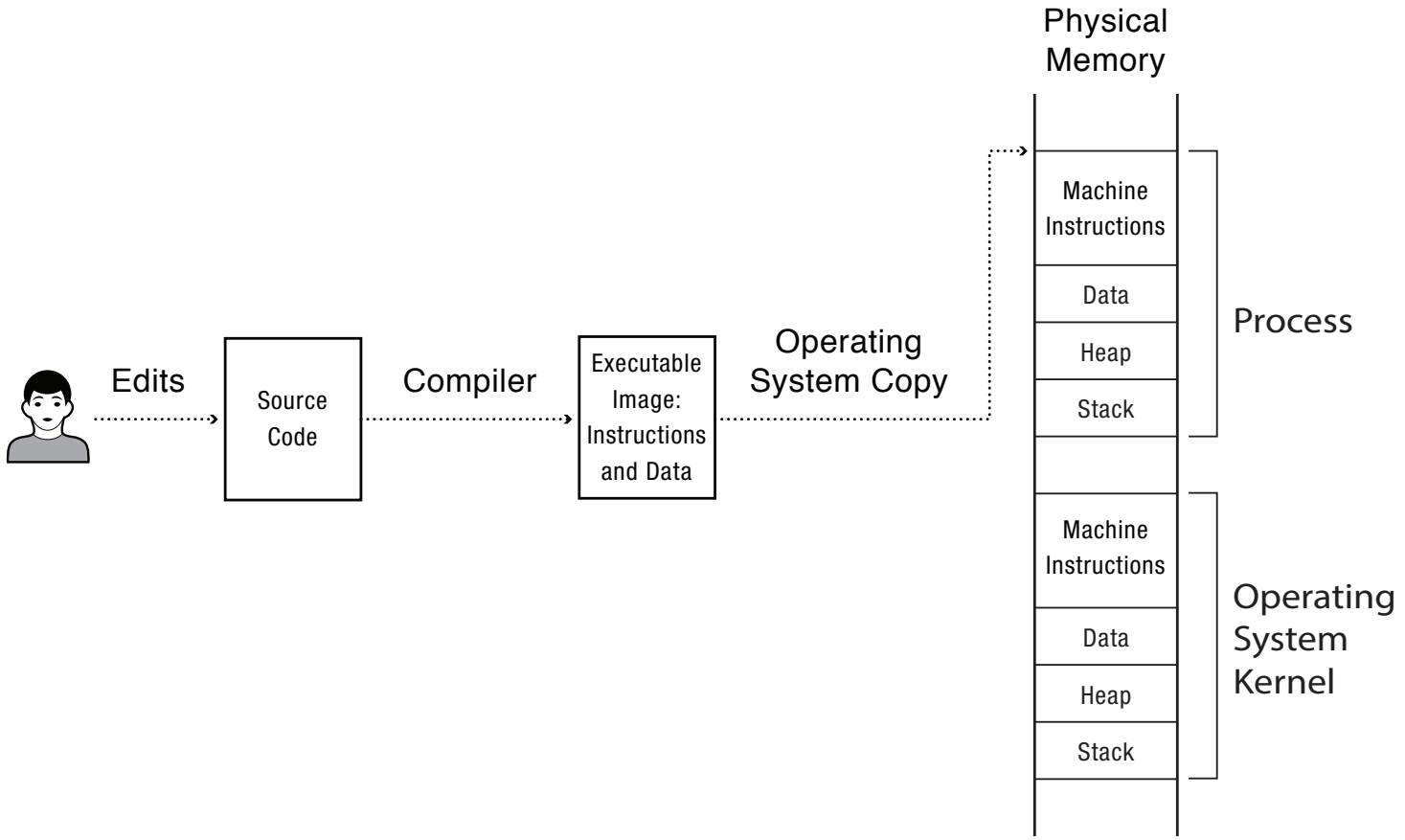
6.Booteo del SO

7.Máquinas Virtuales

Desafío: Protección

- ¿Cómo ejecutar código con privilegios restringidos?
  - Puede pasar que el código tiene errores o puede ser malicioso
- Ejemplos
  - Un script corriendo en un web browser.
  - Un programa recién descargado de Internet.
  - Un programa recién escrito pero no probado aún.

1 El Concepto de Proceso



Concepto de Proceso

- **Proceso**: instancia de un programa, corriendo con derechos limitados

- Process Control Block **PCB**: estructura de datos del SO que permite mantener el “estado” de un proceso

- Un proceso tiene dos partes
  - Thread: secuencia de instrucciones dentro de un proceso
    - Un proceso puede tener de 1 a n threads
    - Un thread se llama también “proceso liviano”
  - Espacio de direcciones: conjunto de derechos de un proceso
    - Memoria a la cual el proceso puede accesar
    - Otros permisos: archivos, procedimientos etc.

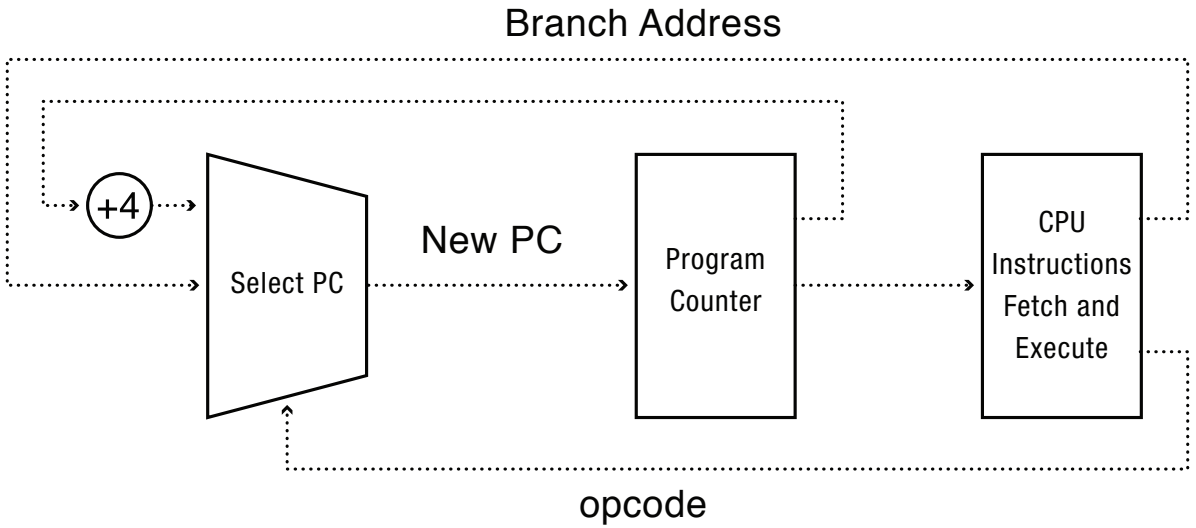
2 Modo Dual de Operación

- Experimento de pensamiento: ¿Cómo se puede implementar la ejecución con privilegios limitados?
  - Ejecutar cada instrucción en un simulador.
  - Si la instrucción se permite, ejecutarla, sino, parar el proceso.
  - Este es el modelo básico de Javascript
- ¿Cómo podemos hacer más rápido?
  - ¿ejecutar el código directamente en la CPU?

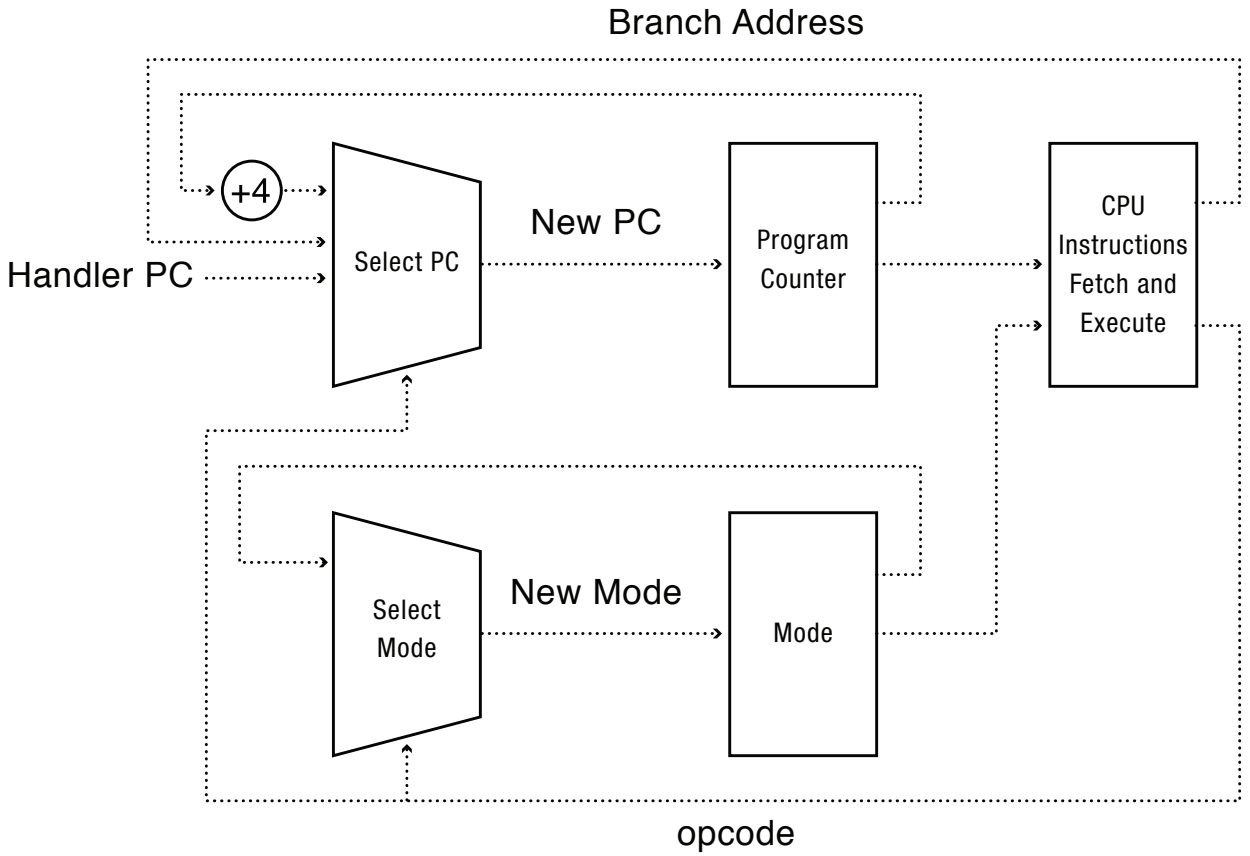
Soporte de HW: Modo dual

- Modo kernel
  - Ejecución con todos los privilegios del HW.
  - Read/write a cualquier dirección de memoria, acceso a cualquier dispositivo de E/S, read/write a cualquier sector de disco, enviar y leer cualquier paquete de datos.
- Modo usuario
  - Privilegios restringidos.
  - Sólo aquellos concedidos por el kernel del SO
- En la arquitectura x86, el modo está en el registro EFLAG

Modelo simple de CPU



Modelo de CPU con Modo Dual



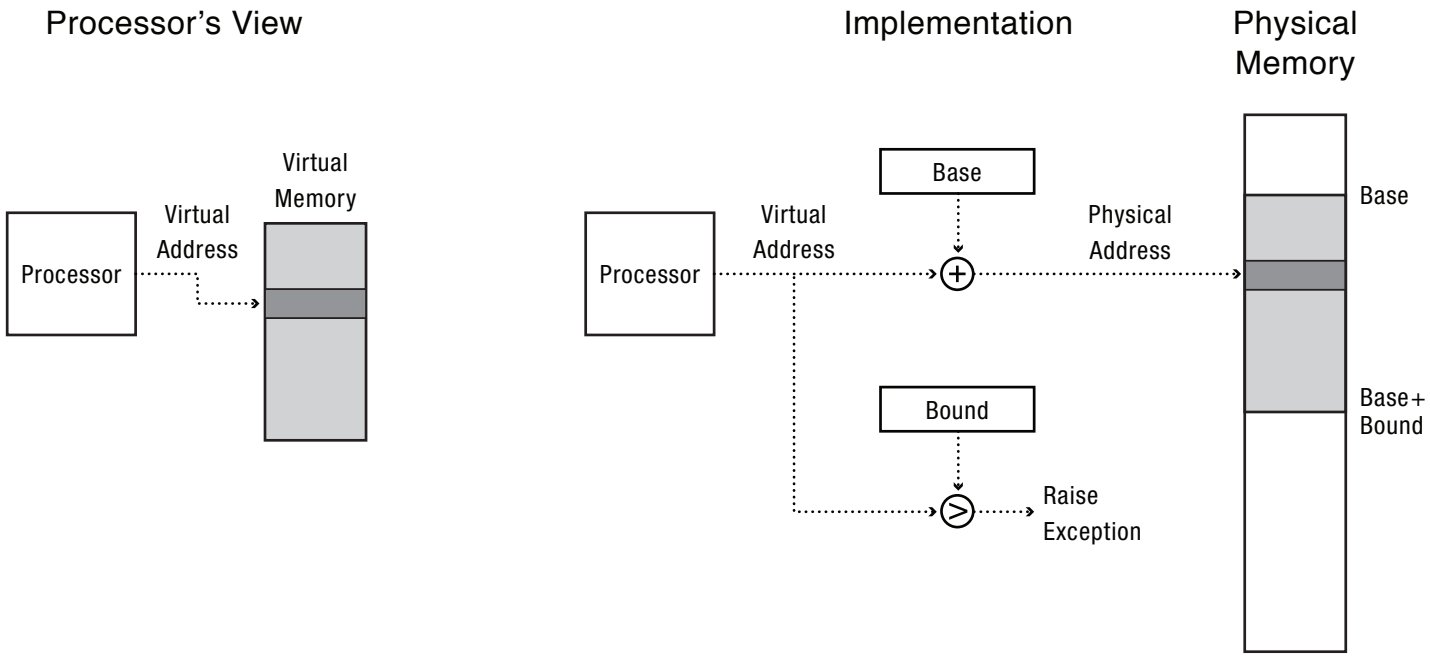
Soporte de HW: Modo dual

- Instrucciones privilegiadas
  - Disponibles para el kernel.
  - No disponibles en el código de usuario.
- Límites en acceso a memoria
  - Prevenir que el código usuario sobrescriba el kernel
- Timer
  - Prevenir
- Modo seguro de conmutar los dos modos

Quiz

- Ejemplos de instrucciones privilegiadas
- ¿Qué pasa si un programa usuario intenta ejecutar una instrucción privilegiada?

3 Memoria y Direcciones Virtuales



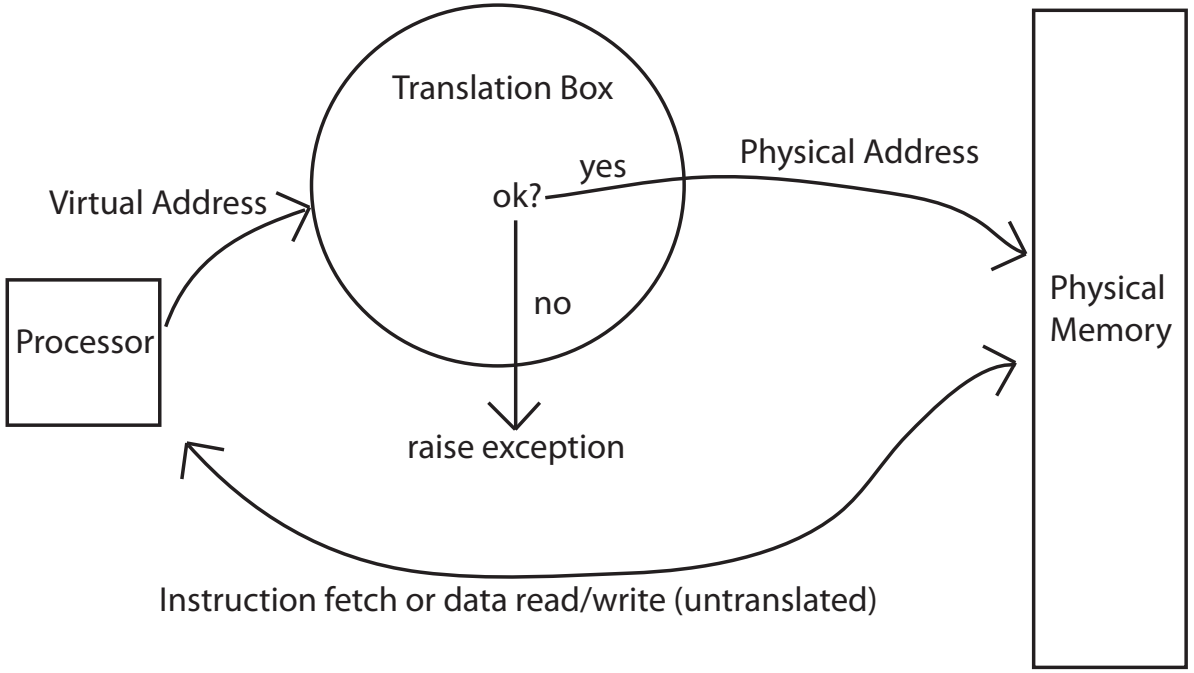
Protección de memoria

Hacia las Direcciones Virtuales

- ¿Qué problema existe con el par de registros Base y Límite (bound)?

Direcciones Virtuales

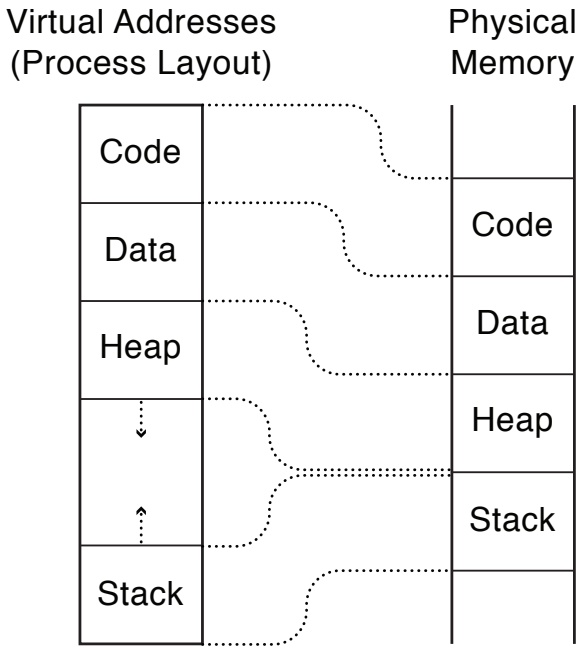
- Direcciones virtuales se convierten en direcciones físicas a través de una tabla.
- La tabla se actualiza sólo por el kernel



Layout de espacio virtual

- Al layout típico de un proceso, habría que agregar código compartido con otros procesos, bibliotecas dll, archivos mapeados en memoria….

- La conversión de direcciones se hace por hardware usando una tabla
- El kernel activa esta tabla



Ejemplo

- El siguiente código se utiliza para ver si un SO tiene memoria virtual. Se corren varias instancias del mismo programa. ¿Cómo se sabe si tiene memoria virtual?

```c
/*virtual_mem_test.c/
#include <stdio.h>
#include <unistd.h>
static unsigned int staticVar = 0;      // a static variable
int  main() {
    unsigned int localVar = 0;   // a procedure local variable

    staticVar += 1; localVar += 1;

    sleep(10);  // sleep causes the program to wait for x seconds
    printf ("static address: %p, value: %d\n", &staticVar, staticVar);
    printf ("procedure local address: %p, value: %d\n", &localVar, localVar);
}
```

En Linux:

```
$ gcc virtual_mem_test.c -o virtual_mem_test
$ ./virtual_mem_test &
[1] 6090
$ ./virtual_mem_test &
[2] 6091
$ ./virtual_mem_test &
[3] 6092
$ static address: 0x804a028, value: 1
procedure local address: 0xbfc2aaac, value: 1
static address: 0x804a028, value: 1
procedure local address: 0xbff8293c, value: 1
static address: 0x804a028, value: 1
procedure local address: 0xbfd6972c, value: 1

[1]   Salida 46               ./virtual_mem_test
[2]-  Salida 46               ./virtual_mem_test
[3]+  Salida 46               ./virtual_mem_test
```

4 Interrupciones

- Las interrupciones son eventos que requieren la atención del kernel.
- Si bien cualquier evento que requiera atención del kernel se denomina interrupción, éstas se clasifican en: interrupciones, excepciones y llamadas al sistema.
- Las interrupciones son asincrónicas respecto a la ejecución de instrucciones del programa. Las excepciones y llamadas al sistema son sincrónicas.

Interrupciones y Excepciones en la Arquitectura MIPS

Excepciones

- Una excepción es un llamado no planificado a una función.
- Las excepciones pueden ser causadas por hardware o software.
    - Ej. se oprime una tecla del computador.
- En el ejemplo de la tecla, es una excepción de hardware llama también *interrupción*.
- Si el programa encuentra un error en una instrucción

Traps

- Las excepciones de software se denominan comúnmente ***traps***.
- Causas de traps pueden ser por ejemplo:
  - división por cero
  - Acceso a dirección ilegal de memoria
  - Breakpoints en debugger
  - Overflow aritmético

Interrupciones

- Las excepciones de hardware se denominan *interrupciones*.
- Causas de interrupciones pueden ser por ejemplo:
  - Oprimir teclas (Ctrl -C por ejemplo)
  - Disco terminó de transferir bloques de datos
  - Timer
  - Llegó un paquete por la red
  - Batería baja

Acciones frente a una excepción

- Frente a una excepción, el procesador:
  - Registra la causa de la excepción
  - Jump a la rutina manejadora de excepciones (en la máquina MIPS en la dirección de instrucción 0x80000180)
  - Después de ejecutarse esta rutina, se vuelve al programa

Hardware para Excepciones

- El procesador MIPS tiene una parte que es utilizada sólo para funciones del sistema (no para correr programas). Esta parte se denomina Coprocesador 0. Maneja excepciones y diagnóstico del procesador.
- El Coprocesador 0 dispone de 32 registros de propósito especial. Dos de estos registros son **`Cause`** (registro 13) y **`EPC`** (registro 14).

Cause y EPC

- No son parte del Archivo de Registros de propósito general utilizados por las instrucciones de máquina.
  - **`Cause`**: Registra la causa de la excepción
  - **`EPC`** (Exception PC): Registra el PC donde ocurre la excepción
- Move from Coprocessor 0
  - `mfc0 $k0, EPC`
  - Moves contents of `EPC` into `$k0`. Es un registro de propósito general.

Exception Causes

Exception	Cause
Hardware Interrupt	0x00000000
System Call	0x00000020
Breakpoint / Divide by 0	0x00000024
Undefined Instruction	0x00000028
Arithmetic Overflow	0x00000030

Flujo ante una Excepción

- El procesador salva la causa y el exception PC en `Cause` y `EPC` respectivamente.
- El procesador salta a la rutina que maneja interrupciones (dirección **`0x80000180`**)

… Flujo ante una Excepción

- La rutina del manejador de excepción:
  - Salva registros en Stack
  - Lee el registro `Cause`:

```
mfc0 $k0, Cause
```

  - Maneja la excepción
  - Restaura registros
  - Retorna al programa

```
mfc0 $k0, EPC
jr $k0
```

Excepciones Vectorizadas

- Algunas arquitecturas utilizan excepciones vectorizadas.
- Estas arquitecturas no utilizan registro Cause.
- La dirección a la cual el control es transferido está determinado por la causa de la excepción.
- En esta caso, el SO conoce la razón de la excepción por la dirección a la cual el control es transferido

FIN

Interrupciones y Excepciones en la Arquitectura MIPS

El Timer

- Dispositivo de hardware que periódicamente interrumpe al procesador
  - Al interrumpir, el control lo toma la *rutina de interrupción de tiempo*.
  - La frecuencia de interrupción es fijada por el kernel. Nunca por el código usuario.
  - Las Interrupciones pueden ser temporalmente diferidas
    - Nunca por el código usuario!
    - Aspecto crucial para implementar mecanismos de exclusión mutua

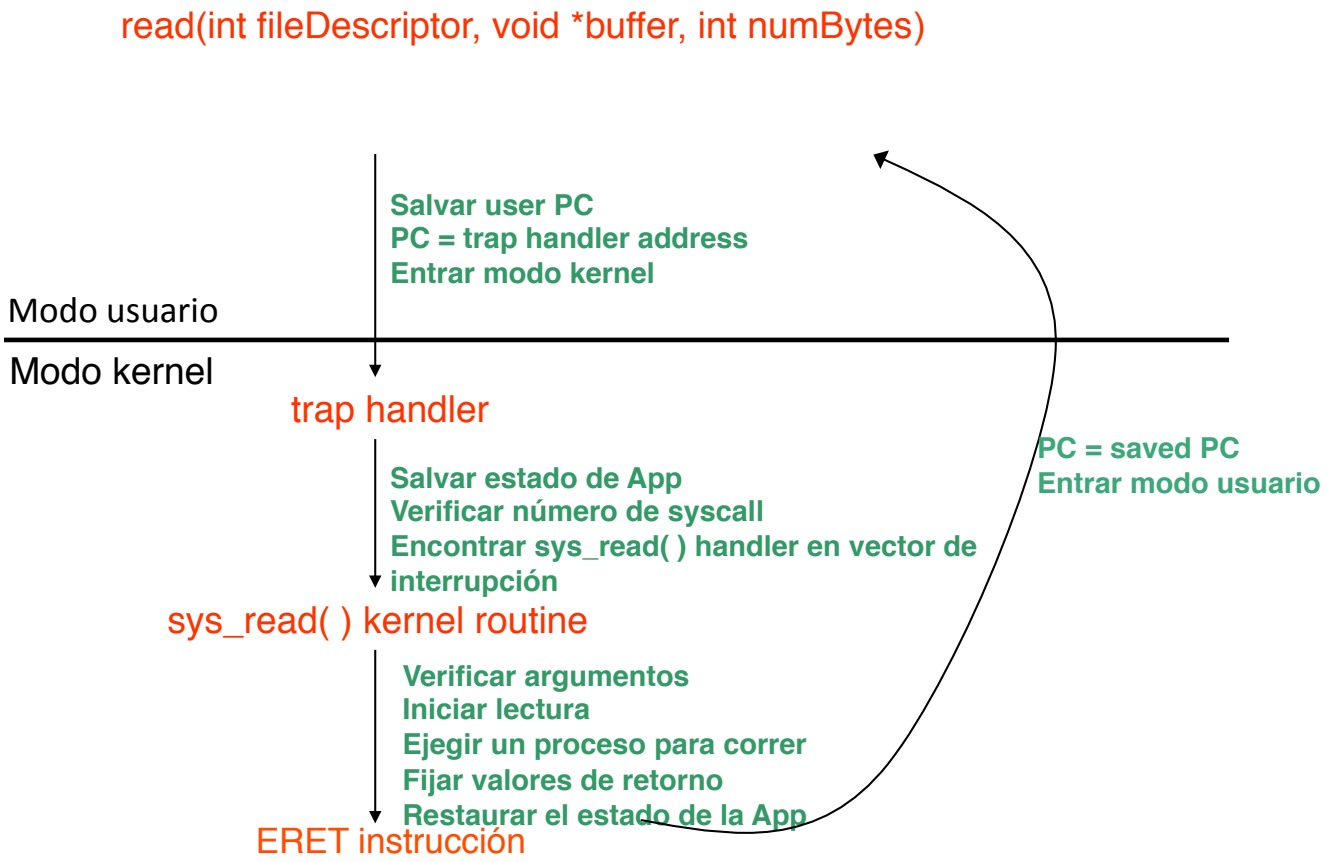
Conmutación de modos: Modo usuario a modo kernel

- Interrupciones
  - Son gatilladas por el timer y dispositivos de E/S.
- Excepciones
  - Son gatilladas por comportamientos no esperados de un programa o también por sw malicioso.
- Llamadas al Sistema (aka Llamada a Procedimiento Protegido)
  - Requerimiento de un programa al kernel, para realizar alguna operación por su intermedio.
  - Su número es limitado y están cuidadosamente escritas.

El SO es SW que sólo responde a interrupciones

Conmutación de modos: Modo kernel a modo usuario

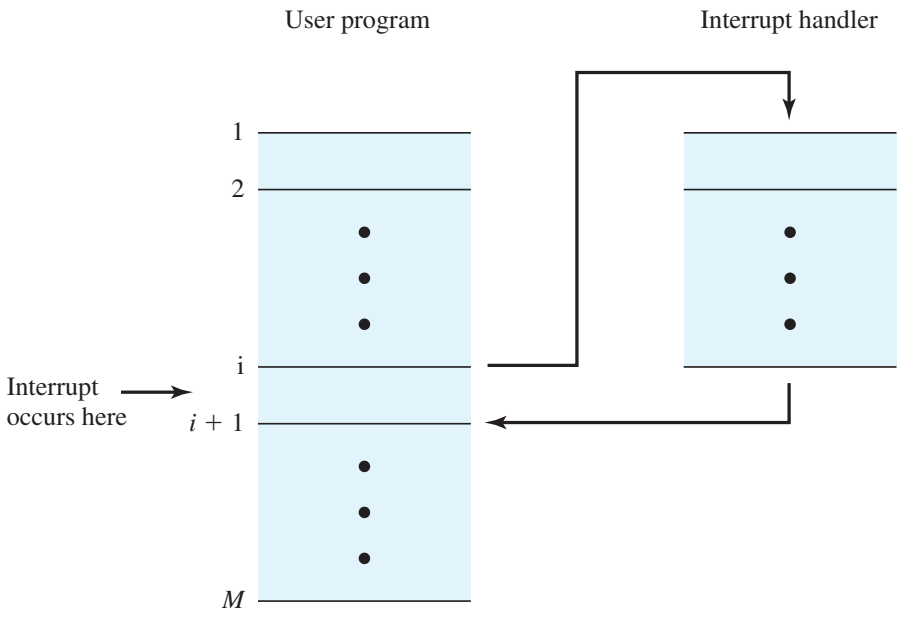
- Al comienzo de un nuevo proceso o un nuevo thread
  - Salto a la primera instrucción en un programa/thread.
- Retorno desde una interrupción, excepción o llamada al sistema
  - Reasumir una ejecución que estaba suspendida.
- Conmutación a otro proceso
- Notificación asincrónica al programa usuario. Programas de usuario pueden recibir notificaciones asincrónicas de eventos (Upcall)


read(int fileDescriptor, void *buffer, int numBytes)
Salvar user PC
PC = trap handler address
Entrar modo kernel
Modo usuario
Modo kernel
trap handler
Salvar estado de App
Verificar número de syscall
Encontrar sys_read( ) handler en vector de interrupción
sys_read( ) kernel routine
Verificar argumentos
Iniciar lectura
Ejegir un proceso para correr
Fijar valores de retorno
Restaurar el estado de la App
ERET instrucción
PC = saved PC
Entrar modo usuario

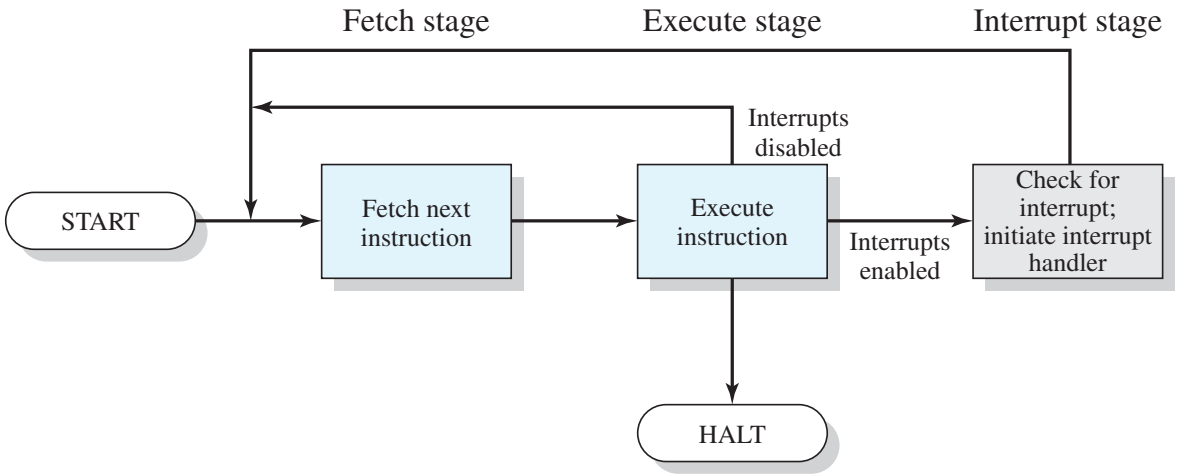
De las Interrupciones

- Elementos
  - El Vector de Interrupciones
  - El Stack de Interrupciones del Kernel
  - PSW: Processor Status Word. Registro de HW con interrupciones activas
- Procesos relacionados con interrupciones
  - Enmascaramiento.
  - Transferencia de control atómica (no interrumpible)
  - Reactivación transparente de un proceso

Transferencia de control por interrupción

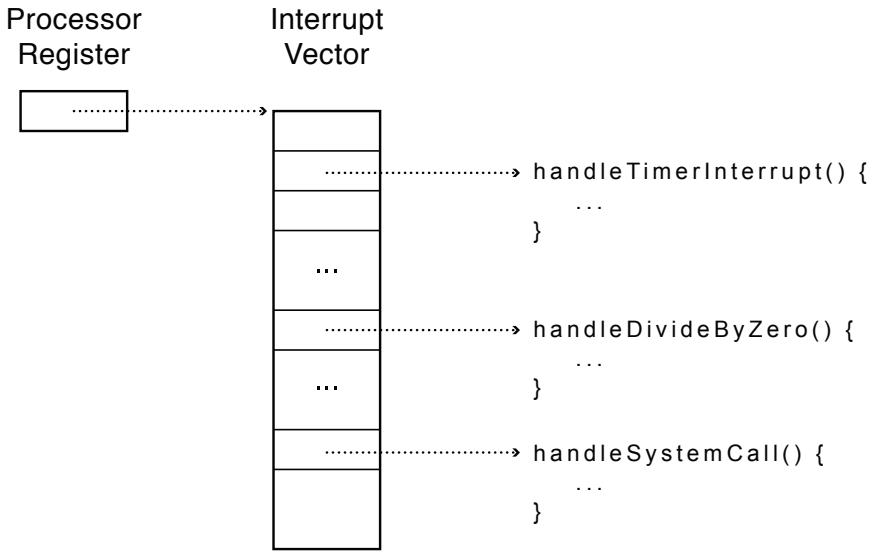


Ciclo de instrucción con interrupción



El Vector de Interrupciones

- Tabla manejada por el kernel. Normalmente en direcciones bajas de memoria. Contiene punteros a códigos que se ejecutan ante diferentes eventos.



Stacks

- ¿Para qué sirven?
  - Cuando se llama a una función, es necesario guardar la dirección de retorno, sus argumentos y variables locales. La estructura (stack frame) de almacenamiento es el Stack y accesible por el Stack Pointer.
  - Entre otras cosas, el Stack permite llamadas recursivas a funciones.
  - Los Stack manejados por el SO son distintos a los Stacks de usuarios (protección).

El Stacks de Interrupciones

- Este Stack es por procesador. Se ubica en la región de memoria del kernel.
- SO modernos asignan un Stack de interrupciones por cada proceso a nivel usuario. Entonces un proceso tiene ambos: stack de kernel y stack de usuario.
- ¿Por qué el manejador de interrupciones no puede correr en el stack del proceso interrumpido?

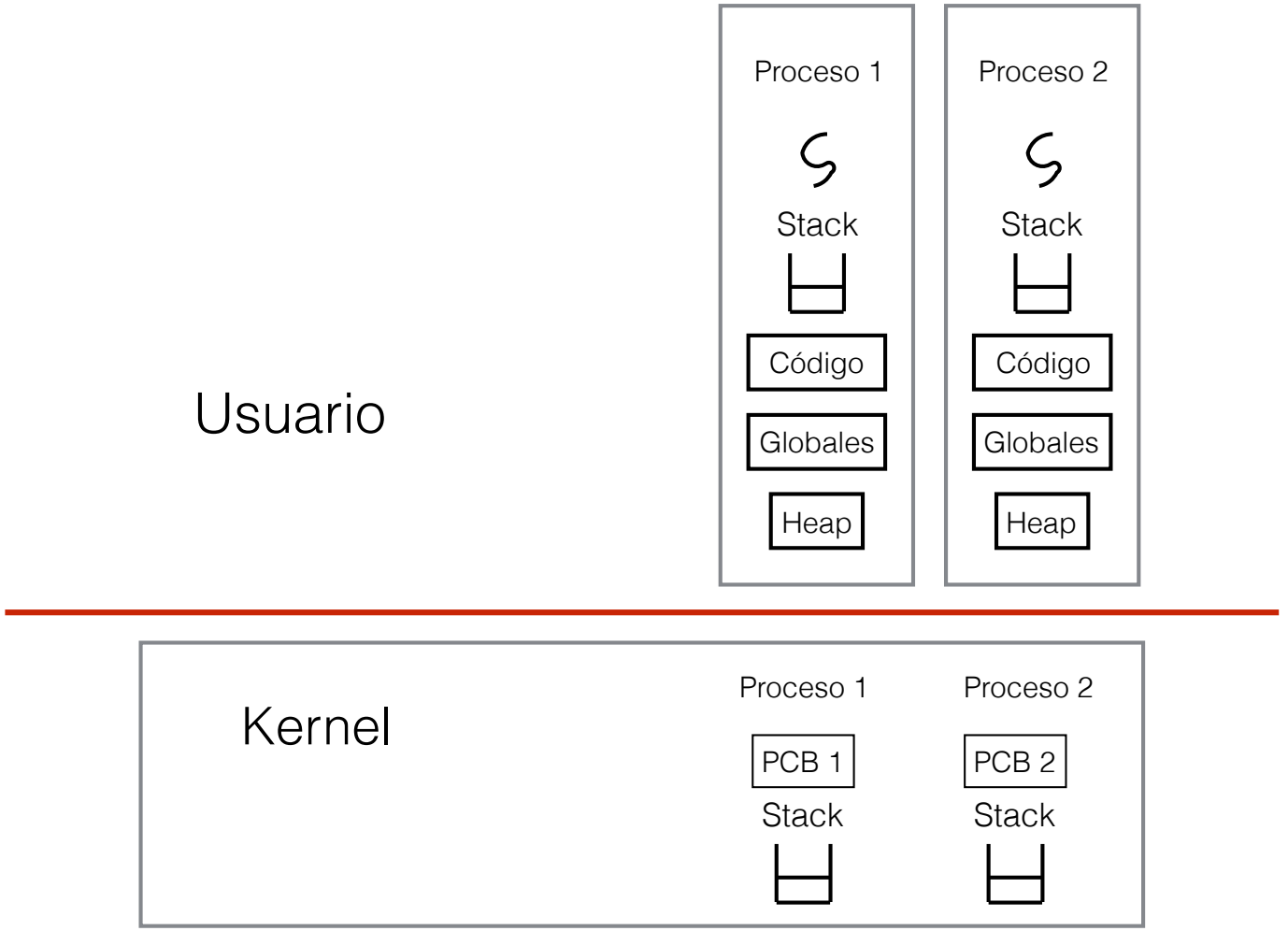
¿Por qué el manejador de interrupciones no puede correr en el stack del proceso interrumpido?

- Hay dos razones:

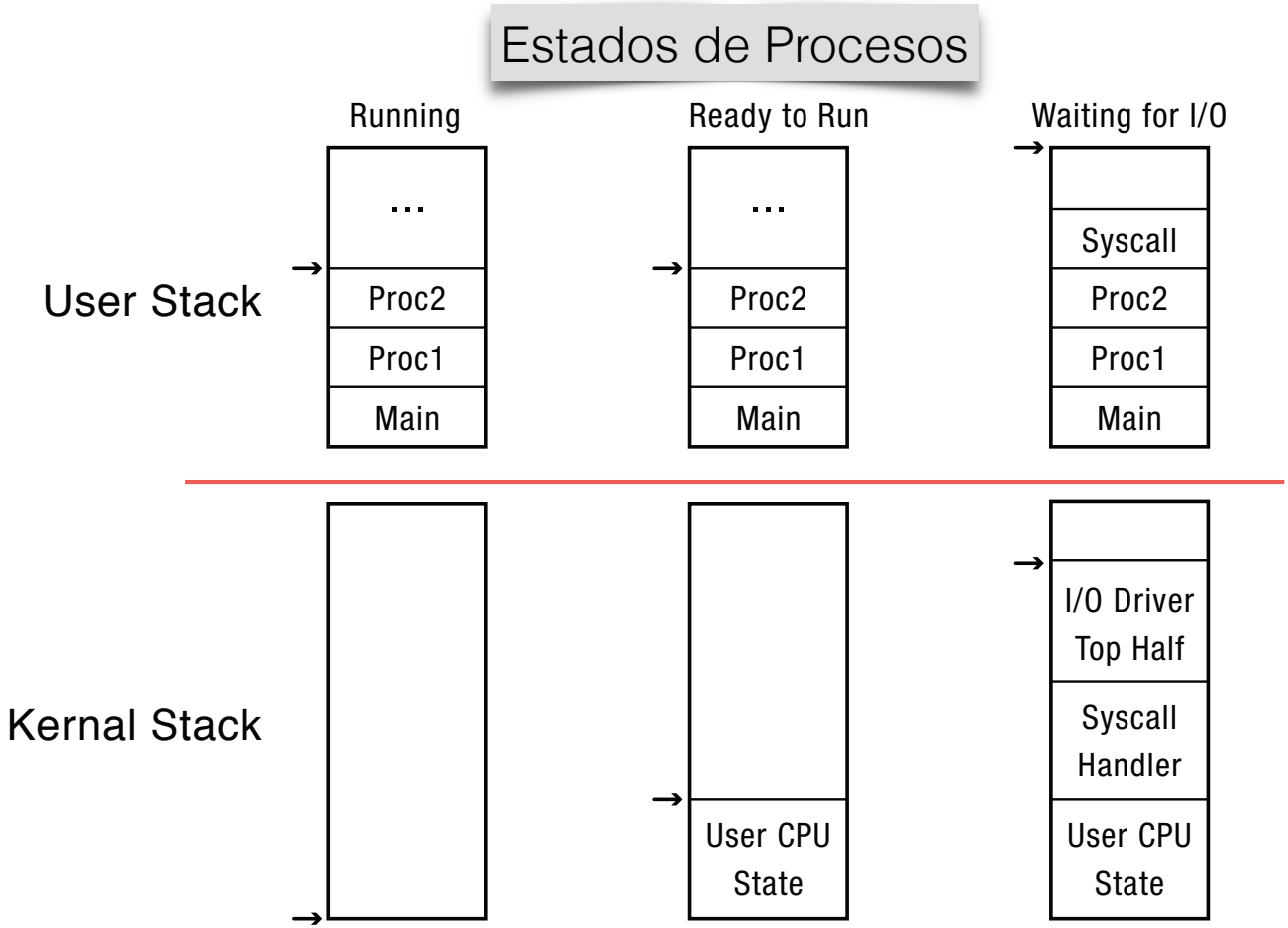
1. **Confiabilidad**: Se el SP (stack pointer) por un bug apunta a una dirección errónea, el manejador del kernel continuar trabajando en forma apropiada.

2. **Seguridad**: En un multiprocesador, corriendo en el mismo proceso pueden modificar la memoria usuario durante la llamada al sistema y puede modificar la dirección de retorno del kernel.

Dos Stacks por proceso



Stacks e Interrupciones



Context Switch

- Cuando la CPU conmuta a otro proceso, el sistema debe salvar el estado del antiguo proceso y cargar el estado salvado del nuevo proceso. Esta acción se denomina Context Switch (Conmutación de contexto).

- El contexto de un proceso está representado en la PCB

Enmascaramiento de Interrupciones

- El código que maneja interrupciones (interrupt handler), corre con interrupciones desabilitadas.
  - Cuando la interrupción se completa, se vuelve a habilitar.
  - Deshabilitar, sólo significa diferir en tiempo.

- El kernel también puede desabilitar interrupciones
  - Cuando elige el siguiente proceso/thread para correr.
  - Si el tiempo es muy largo, se perderán eventos de E/S.
  - En la arquitectura x86, los registros:
    - CLI: deshabilita interrupciones
    - STI: habilita interrupciones

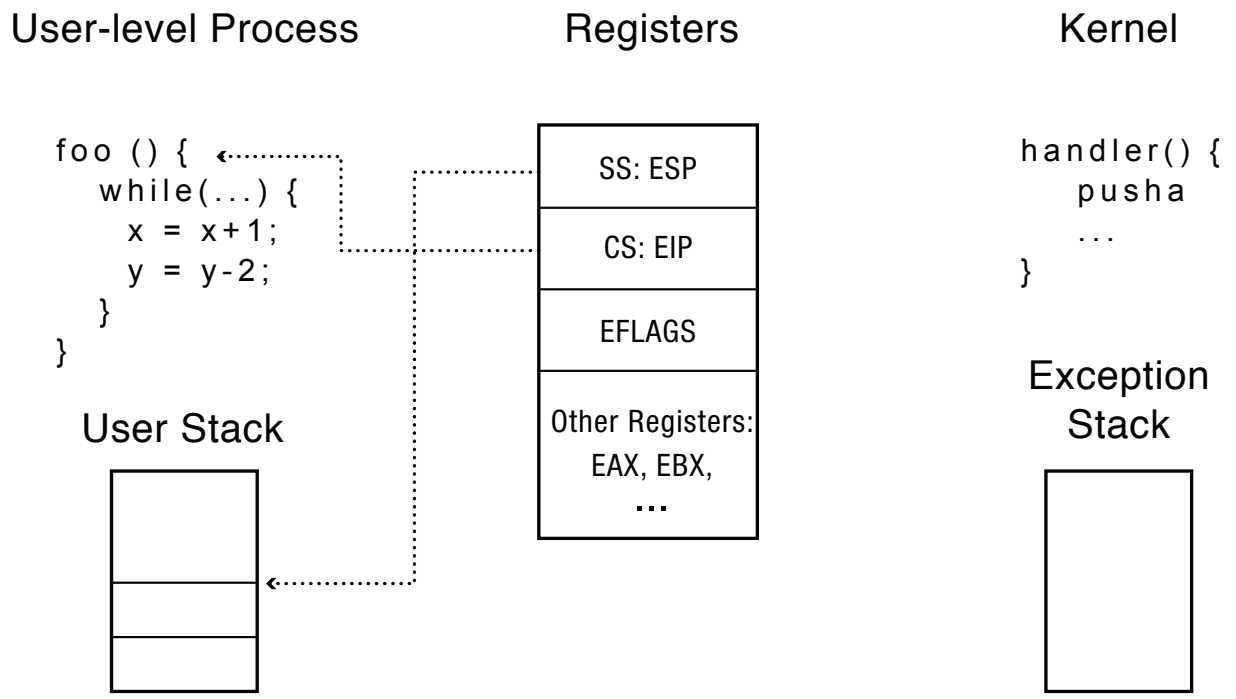
Manejadores de Interrupciones (*interrupt handler*)

- Un manejador de interrupciones no se puede bloquear (no bloqueante). Corre hasta terminar
  - Debe ser rápido para permitir capturar la siguiente interrupción
  - Toda espera debe ser acotada en tiempo
- Los drivers de dispositivos corren como threads de kernel
  - Colas de trabajo para manejadores de interrupciones.
  - También el manejador (a veces) espera por interrupciones.

Modo de transferencia atómico (no interrumpible)

- Por ejemplo, la arquitectura x86 ante una interrupción:
  - Salva el SP actual
  - Salva el Program Counter (PC)
  - Salva el Process Status Word (PSW) actual (registro de códigos de condición)
  - Conmuta al stack del kernel: Push SP, PC y PSW en Stack
  - Conmuta a modo kernel
  - Direcciona el vector de interrupción
  - El manejador de interrupción salva registros que puedan corromperse

x86 antes de la interrupción

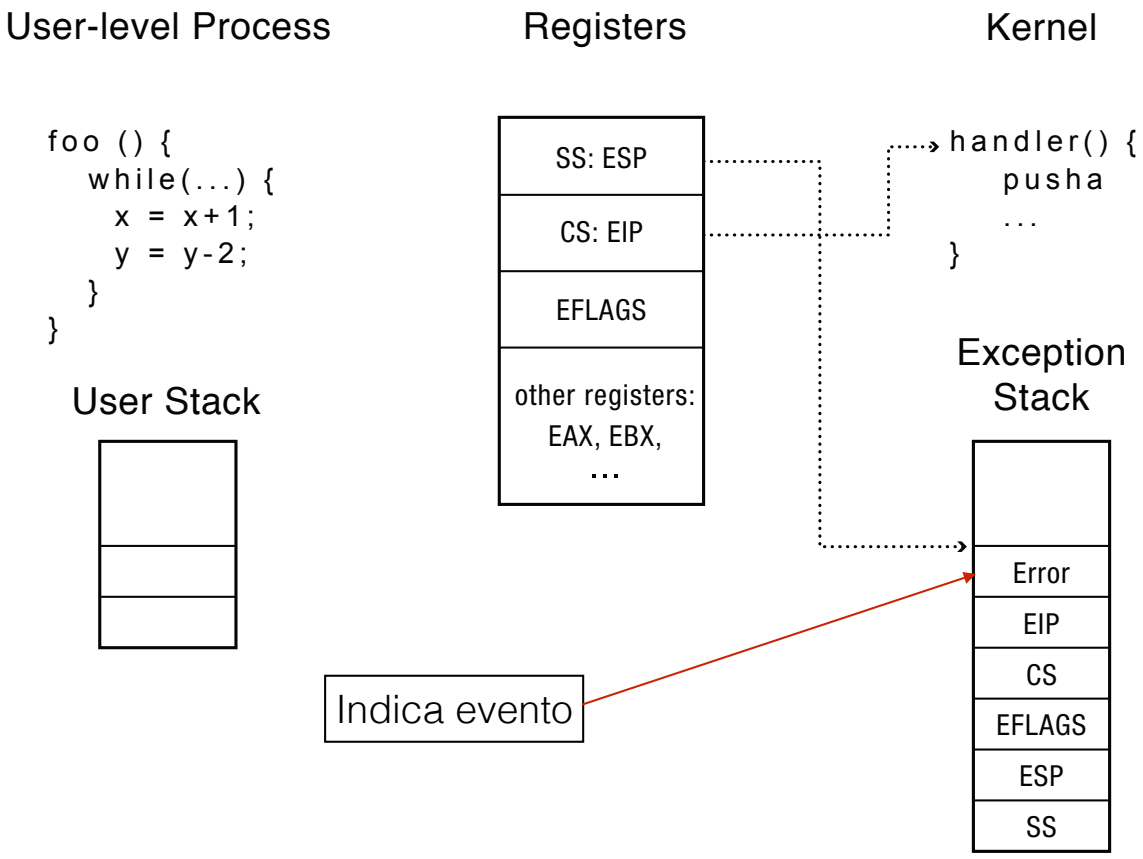


ESP: Stack pointer
EIP: Program counter

SS: Stack segment
CS: Code segment

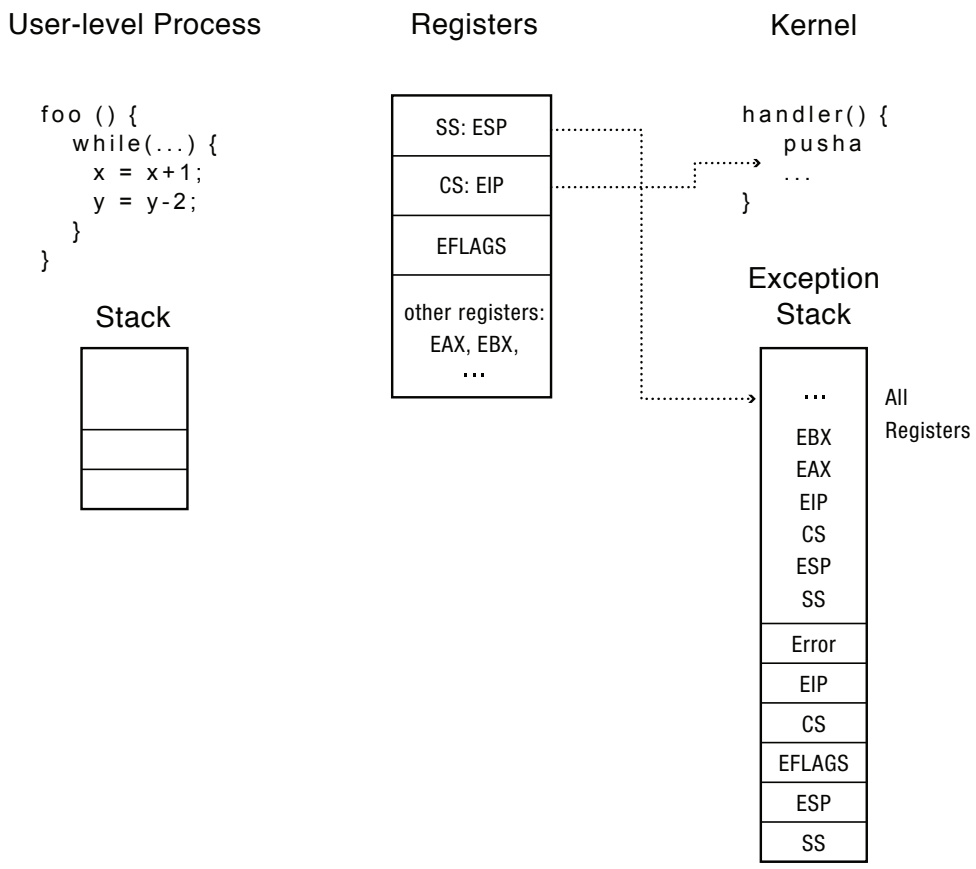
x86 durante la interrupción

Justo en el momento que salta al handler



x86 después de la interrupción

El handler a comenzado la ejecución



Al final de la la interrupción

- El Handler restaura los registros que estaban guardados en el stack.
- Atómicamente el proceso/thread que fue interrumpido:
  - Restaura el PC
  - Restaura el SP
  - Restaura el PSW
  - Conmuta a modo usuario

Para el proceso, todo esto es transparente

Interrupciones de SW

- Un Upcall es una interrupción por sw o también una interrupción a nivel usuario. También "*interrupciones virtualizadas*"
  - Se denominan también señales (signals en UNIX y eventos asincrónicos en Windows)
  - Es un mecanismo que notifica a un proceso usuario de un evento que necesita ser manejado en forma adecuada.

- Es un símil directo con las interrupciones del kernel
  - Signal handler: puntos de entrada fijos
  - Stack separado para el manejo de señales
  - En forma automática se salvan/restauran registros y se reasume en forma transparente.
  - Enmascaramiento: se inhiben señales mientras se ejecuta el signal handler

Signals: Upcall

- Las señales permiten que una aplicación se comporte de una forma similar a un kernel, es decir responda a interrupciones.
- Los sistemas UNIX utilizan señales para notificar a procesos que ha ocurrido un evento particular.
- Para manejar las señales recibidas se utiliza un “signal handler”. La secuencia es:
  - Una señal es generada por un evento particular
  - Una señal es entregada a un proceso
  - La señal es manejada por el proceso

… Signals

- La función estándar de UNIX para entregar una señal a un proceso es:

```
kill(pid_t pid, int signal)
```

- kill especifica el pid del proceso al cual una señal será despachada.

… Signals

- La llamada **`signal`** altera la acción por defecto. Por ejemplo:

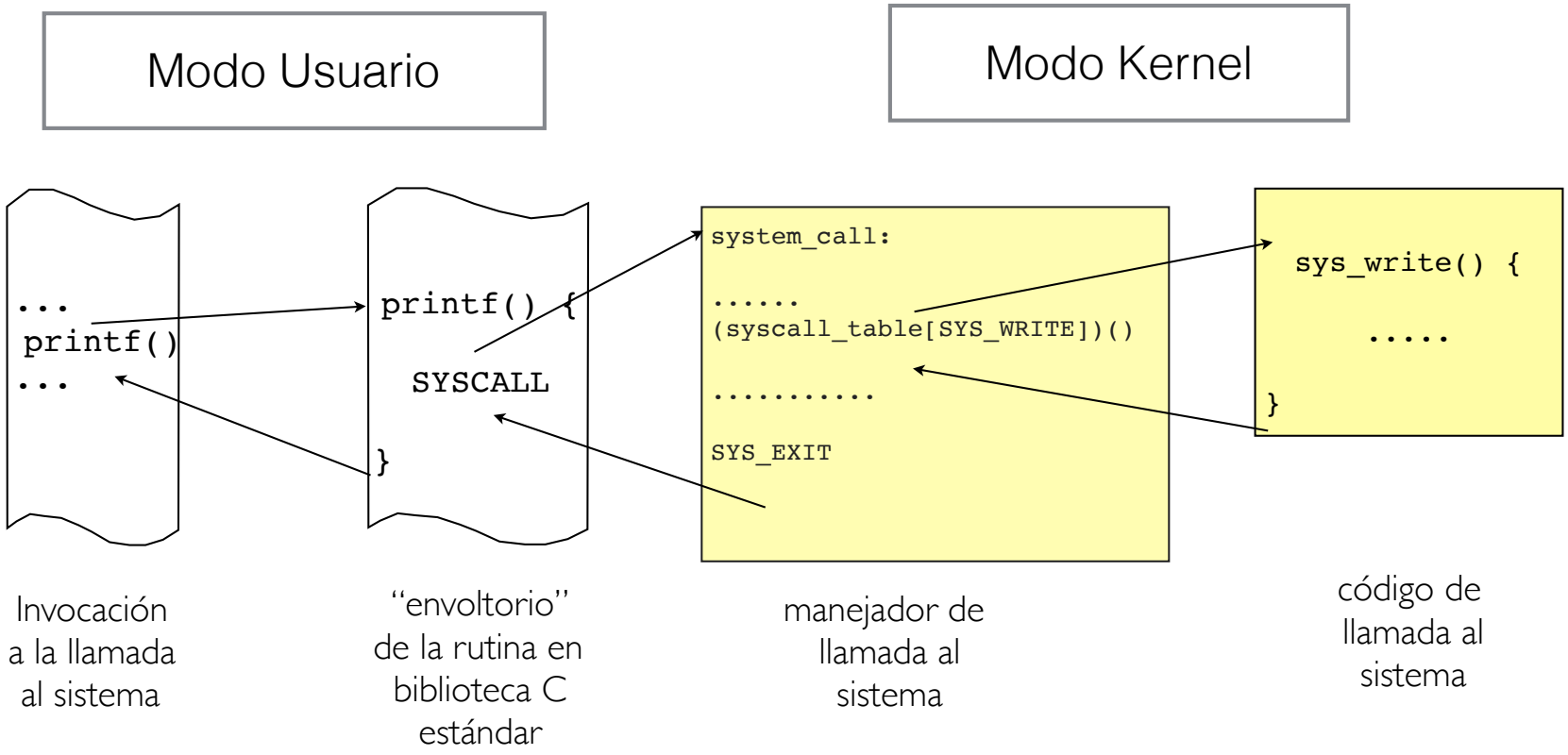
```
#include <signal.h> //contiene lista de posibles argumentos
signal(SIGINT, SIG_IGN) // cause the interrupt to be ignored

signal(SIGINT, Dir_func) //en este caso al llegar la señal se
                         //ejecuta la función cuya dirección
                         // es Dir_func
```

5 Llamadas al Sistema

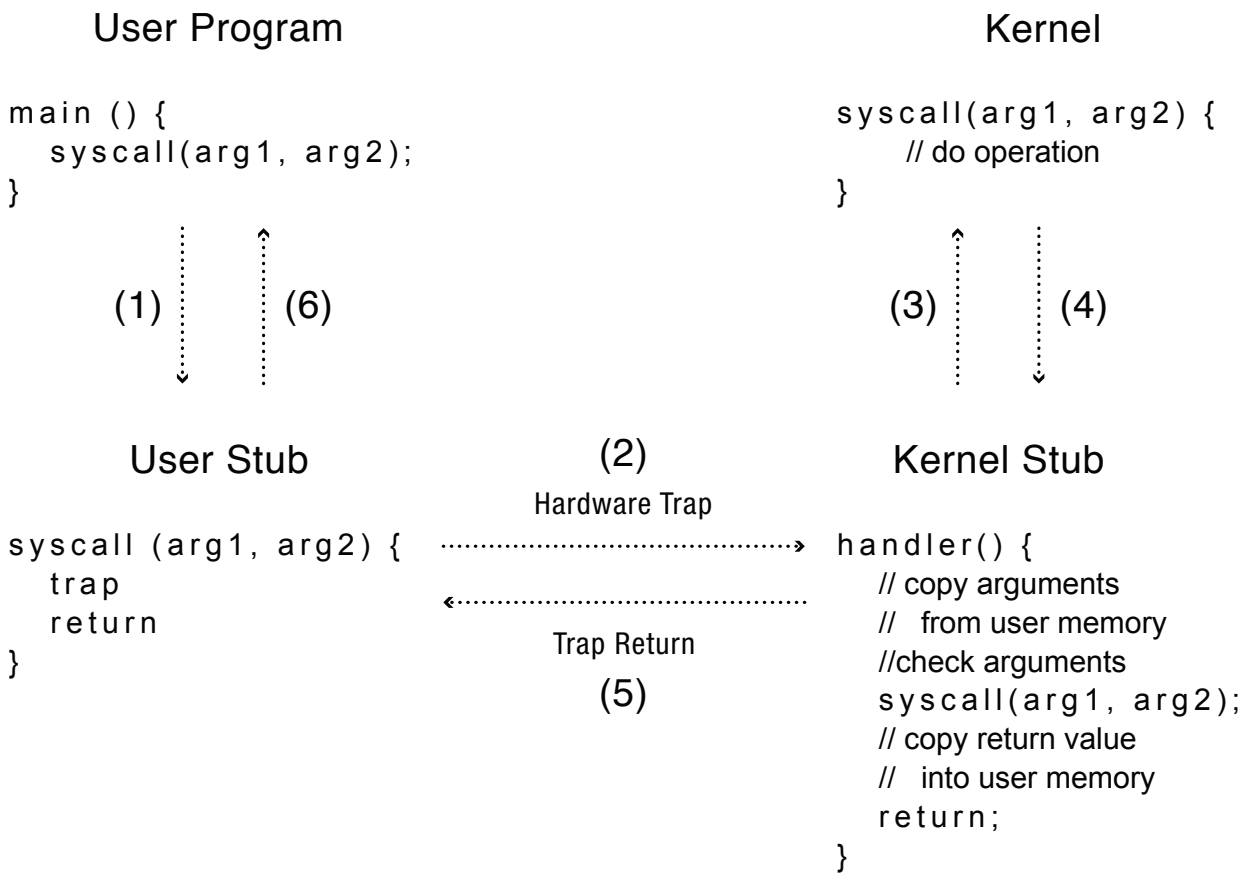
- Para imprimir un número entero en MIPS R3000:

```
li $v0, 1 #Codigo para imprimir entero
li $a0, 5 #entero a imprimir
syscall
```

Llamas al sistema y bibliotecas



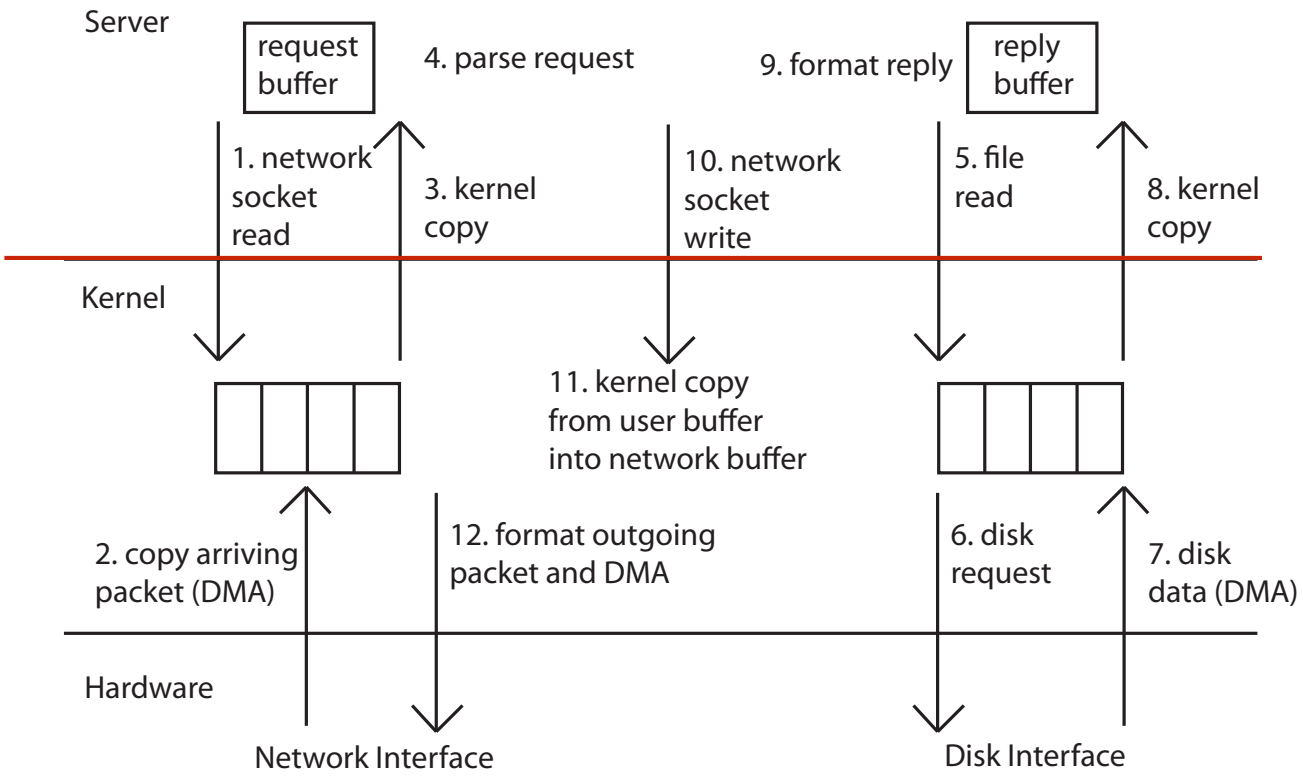
Generalización



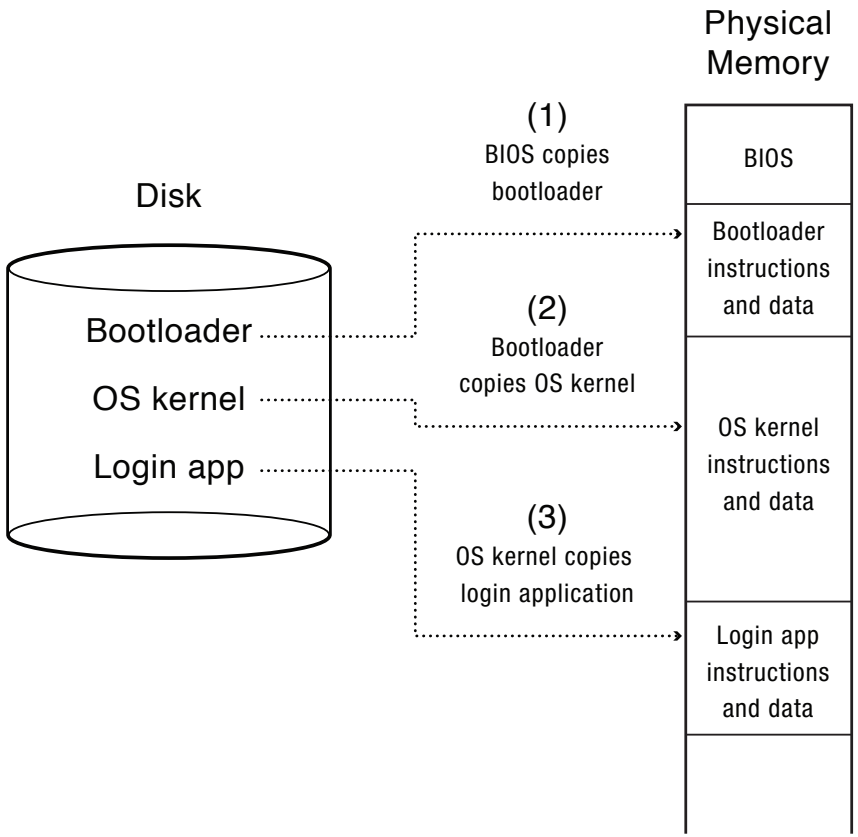
El Manejador de llamadas al sistema del Kernel (handler)

- Localiza los argumentos
  - En registros o en stack usuario
- Copia los argumentos
  - De la memoria usuario a memoria kernel
  - Pone cuidado con operaciones maliciosas
- Valida argumentos
  - Protege al kernel de errores en código usuario
- Copia los resultados en memoria usuario

Ejemplo: Web Server



6 Booteo del SO



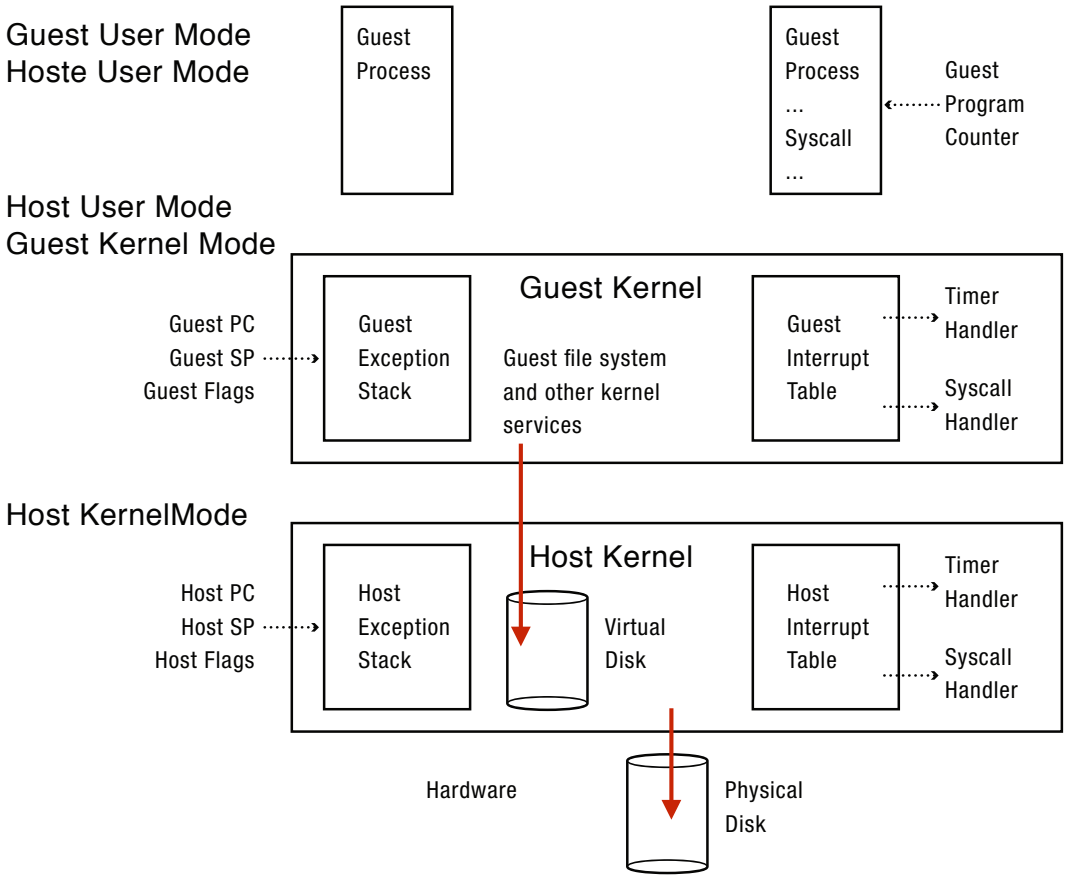
7 Máquinas Virtuales

- Algunos kernels de SO proporcionan la abstracción de una completa máquina virtual a nivel de usuario.
- ¿Cómo se puede lograr esto?. Por ejemplo interrupciones, excepciones y llamadas al sistema en este contexto.
- Definiciones: El SO que proporciona la abstracción de máquina virtual se denomina ***SO host***. El SO que corre dentro de la máquina virtual se denomina ***SO guest***.

… Máquinas Virtuales

- El SO host proporciona la ilusión que el kernel guest está corriendo en un HW real.
- Para simular el disco del guest, el SO host utiliza el sistema de archivos. De la misma forma la memoria, interfaces de red, etc… se simulan con memoria virtual e interfaces de red virtuales.
- En el proceso de booteo, el host inicializa sus vectores de interrupción, los handler en su memoria.
- Al activar la máquina virtual el host carga el *bootloader* del guest desde el disco virtual. Este carga el kernel guest desde el disco virtual y comienza su ejecución inicializando sus tablas de interrupción.

Estructura


Guest User Mode
Hoste User Mode
Guest
Process
Guest
Process
...
Syscall
...
Guest
Program
Counter
Host User Mode
Guest Kernel Mode
Guest Kernel
Guest PC
Guest SP
Guest Flags
Guest
Exception
Stack
Guest file system
and other kernel
services
Guest
Interrupt
Table
Timer
Handler
Syscall
Handler
Host KernelMode
Host Kernel
Host PC
Host SP
Host Flags
Host
Exception
Stack
Virtual
Disk
Host
Interrupt
Table
Timer
Handler
Syscall
Handler
Hardware
Physical
Disk

Máquina Virtual: Nivel Usuario

- ¿Cómo trabaja una VM (máquina virtual)?
  - Una VM corre como cualquier aplicación nivel usuario
  - ¿Cómo captura instrucciones privilegiadas, interrupciones, E/S, etc…?

- Se instala un driver de kernel, transparente al kernel del host
  - Se requieren permisos de administrador
  - Se modifica la tabla de interrupciones para redireccionarla al código del kernel VM
  - Si la interrupción es para la VM, **upcall,** si es para otro proceso, reinstalar tabla de interrupciones y reasumir kernel

Sistemas Operativos

Capítulo 2 El Kernel

Prof. Javier Cañas R.