

EXCEPCIONES

CÓDIGO MAIN



Ejecutamos Código de SO, acabando con while()

HANDLER: Prepara el sistema para la rutina de excepción

page-fault-handler.

call page-fault-routine

En excepciones NUNCA vuelve al Código original, entonces NO se retorna + NO se guarda contexto SW

RUTINA: Código ejecutado debido al salto de la excepción

void page-fault-routine (int error, int eip)

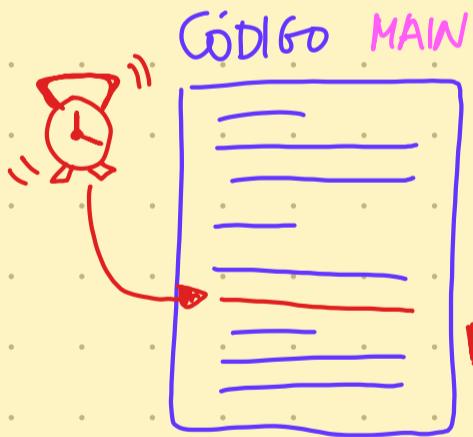
1

... //Codi executat
while(1)

}

- 1 Mientras se ejecuta el Código alguna instrucción salta la excepción, y vamos al handler para preparar el sistema para la rutina de la excepción
- 2 Cuando en el handler se llama la rutina ésta ejecuta su rutina y **NUNCA VUELVE**. Como probablemente queramos acceder a EIP (@ donde ha saltado la excepción), incluimos ERROR & EIP (en exc a un param. nón)

INTERRUPCIONES



1 Ejecutamos Código y en algún punto una interrupción HW salta (clock-int)

* Només aban en clk-hdl

HANDLER: Prepara el sistema para la rutina de excepción (que en este caso VOLVERÁ al main)

RUTINA: Acción provocada por la interrupción (vuelve al main)

4

clock_handler:

SAVE_ALL

call clock-routine

EOI

RESTORE_ALL

IRET

void clock-routine () {

... //Codi executat

return;

3

2 Después de ejecutar el handler paramos a la rutina de interrupción (por ejemplo, imprimir una tecla por terminal)

3 Al acabar la rutina volvemos al handler para ejecutar el EOI, y así permitir la ejecución de otras interrupciones (se ejecutan de manera SECUENCIAL)

4 Restauramos la pila de sistema igual que estaba antes de la interrupción y retornamos al código principal

SYSCALLS

CÓDIGO MAIN

```
-----  
|  
|  
|  
| write(...),  
|  
|  
|  
| ..
```

WRAPPER Realizamos el paso de parámetros entre modo sistema - usuario.

```
int write(int fd, char* buffer, int size) {  
    ...  
    movl $4, %eax // ID de la syscall  
    ...  
    ret
```

```
pushl %ebp  
movl %esp, %ebp  
pushl %edx  
pushl %ecx  
pushl %ebx  
...  
.
```

MAKE DYN. LINK

```
popl %ebx  
popl %ecx  
popl %edx  
popl %ebx  
...  
.
```

UNLINK

USER MODE

HANDLER UNIVERSAL

Punt central per on entrem en mode sistema (codi en EAX). Necesitem sys-call-table

ENTRY (system-call-handler)

```
SAVE_ALL // Guardem context SW&HW  
... // Comprobem paràmetres vàlids  
call *sys-call-table(,%eax,0x04)  
...  
movl %eax,0x18(%esp) // Guardem l'error en la  
RESTORE_ALL // pila per a qüe no es  
iret // perdi al restaurar ctxt
```

SERVICE ROUTINE Implementació

la rutina sys-write

```
int sys-write (int fd, char* buffer, int size) {  
    ... // 1) Comprovació de paràmetres  
    ... // 2) Copiem dades entre l'espai de sistema i usuari  
    ... // 3)ús de sys-write-control per l'I/O de dades  
    return;
```

SYSCALL_TABLE: Taula que conté totes les syscalls indexades per ordre (a més podem afegir noves)

ENTRY (sys-call-table)

```
... long sys-hi-syscall // 0  
... long sys-hi-syscall // 1  
... long sys-hi-syscall // 2  
... long sys-hi-syscall // 3  
long sys-write // 4  
...  
.globl MAX-SYSCALL
```

MAX-SYSCALL = (-sys-call-table)/4

Les hauríem de indicar amb sys-hi-syscall

- 1 El wrapper TIENE QUE estar escrito en assembly y crea un enlace entre el modo usuario y sistema para volver
- 2 Con las syscalls entramos a UN ÚNICO HANDLER en el que nosotros llamaremos a la sys-call-table indexando la rutina del write (4)
- 3 A través de esta tabla indexaremos las verdaderas rutinas de nuestras syscalls, que finalmente volverán a user mode