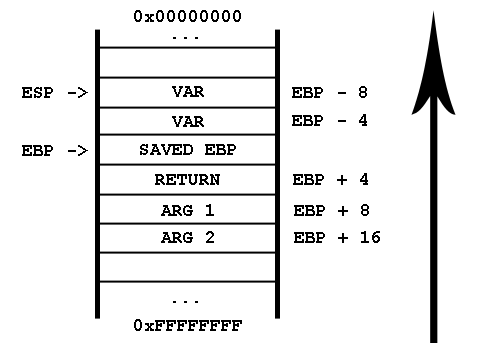
# Cosas útiles de ZeOS

## Para buildear el sistema operativo (cuando esté todo)

$ gcc -Wall -Wstrict-prototypes -o build build.c

$./build bootsect system.out user.out > zeos.bin

## Pila de usuario



## Donde se programa cada cosa?

* Handlers → entry.S
* Añadir interrupció a la IDT → interrupt.c
* Routine → interrupt.c
* Wrapper → libc.c

# Log de cosas hechas

## Implementar la macro RESTORE\_ALL i EOI

En entry.S, es simplemente hacer pop de todo lo que hemos puesto con el SAVE\_ALL:

#define RESTORE\_ALL \

popl *%ebx*; \

popl *%ecx*; \

popl *%edx*; \

popl *%esi*; \

popl *%edi*; \

popl *%ebp*; \

popl *%eax*; \

popl *%ds*; \

popl *%es*; \

popl *%fs*; \

popl *%gs*;

#define EOI \

movb $0x20, *%al*; \

outb *%al*, $0x20;

## Implementar el teclado

Primero hay que implementar el service\_routine (en interrupt.c). Leemos del puerto del teclado, el 0x60. El 7o bit inidca si es brak o make (pulsar o soltar). Del bit 0 al 6 inidcan el código de la tecla.

    // Rutina del teclat

*void* keyboard\_routine() {

*unsigned* *char* key;

   key = inb(0x60); // Llegeix el port del teclat

*unsigned* *char* translate = key & 0x7F; // Per saber el codi

*unsigned* *char* make = key & 0x80;        // Per saber si break o make

    if (make == 0x80) {

*char* c = char\_map[translate];

    if (c == '\0') printc\_xy(10, 20, 'C');

  else printc\_xy(10, 20, c);

  }

}

Luego implementar el handler (en entry.S)

// HANDLER DEL TECLADO

ENTRY(keyboard\_handler)

SAVE\_ALL;

EOI;

call keyboard\_routine;

RESTORE\_ALL;

iret;

En interrupt.c definimos la cabecera.

*void* keyboard\_handler();

Añadir la entrada a la IDT, en interrupt.c. Dentro de la función void setIdt()

setInterruptHandler(33, keyboard\_handler, 0);

Finalmente, en hardware.c habilitar la interrupción de teclado. El registro 0x21 contiene una máscara de interrupciones. 1 significa desabilitada, 0 habilitada.

Los bits se corresponden así:

\* bit 0 : Timer

\* bit 1 : Keyboard

\* bit 2 : PIC cascading

\* bit 3 : 2nd Serial Port

\* bit 4 : 1st Serial Port

\* bit 5 : Reserved

\* bit 6 : Floppy disk

\* bit 7 : Reserved

## Implementar la rutina system\_call\_handler

Para poder hacer llamadas a sistema, hay que hacer un handler que manipule estas llamadas. Esto se hace en entry.S

// HANDLER DE SYSTEMM CALLS

ENTRY(system\_call\_handler)

   SAVE\_ALL;                  // Guardar contexto

    cmpl $0, *%eax* // La llamada es negativa?

jl err // Si lo es, lanza error

cmpl $MAX\_SYSCALL, *%eax* // Es más grande que MAX\_SYSCALL? (4)

jg err // Si lo es, lanza error

call \*sys\_call\_table(, *%eax*, 0x04); // Llamar a la rutina // correspondiente

jmp fin // Acaba

err:

    movl $-38, *%eax*       // Mover el error al %eax

fin:

movl *%eax*, 0x18(*%esp*) // Cambiar el valor de %eax en la pila

RESTORE\_ALL; // Reestablecer contexto

iret;

## Inicializar la IDT con el handler

En interrupt.c, dentro de void setIdt():

setTrapHandler(0x80, system\_call\_handler, 3);

## Implementar la llamada write

Primero, implementar la rutina sys\_write.

*int* sys\_write(*int* fd, *char* \* buffer, *int* size) {

*int* res;

if (res = check\_fd(fd, ESCRIPTURA) < 0) return -1;

if (buffer == NULL) return -1;

    if (size < 0) return -1;

    return sys\_write\_console(buffer, size);

}

Luego modificar la sys\_call\_table para que contenga la entrada a write (será la 4). Esto lo hacemos en sys\_call\_table.S

ENTRY (sys\_call\_table)

    .long sys\_ni\_syscall     // 0

    .long sys\_ni\_syscall     // 1

    .long sys\_ni\_syscall     // 2

    .long sys\_ni\_syscall     // 3

.long sys\_write          // 4

  .globl MAX\_SYSCALL

*MAX\_SYSCALL =* (. - sys\_call\_table)/4

Finalmente, crear el wrapper para la system call (la función de la libc.c). La forma de ensamblador *inline* usada, define primero la instrucción en asm. La sigue los *output operands*, y después los *input operands*.

*int* write(*int* fd, *char* \*buffer, *int* size) {

*int* resultado;

*asm* ( "int $0x80"    // Instrucción asm

        : "=a" (resultado) // Output (eax = resultado)

        : "a" (4), "b" (fd), "c" (buffer), "d" (size) // Inputs

    );

    if (resultado >= 0) return resultado;

    else {

        errno = -resultado;

        return -1;

    }

}

## Implementar errno y perror

Creamos una variable en libc.c que se llama errno y es de tipo integer. Cuando hagamos una llamada a sistema, si nos devuelve un número 0 o positivo, significa que ha ido bien. Si devuelve un número negativo implica que ha habido un error. Por eso tenemos que guardar el valor absoluto de este número en el errno, y devolver -1.

El perror no es más que una función de libc.c que nos devuelve el errno.

*void* perror() {

*char* txt[100];

    itoa(errno, txt);

    write(1, txt, strlen(txt));

}

## Implementar el reloj

Queremos mostrar un reloj con el tiempo que ha pasado desde que hemos booteado ZeOS.

Primero inicializamos la entrada del clock en la IDT (entrada 32), en interrupt.c dentro de void setIdt().

setInterruptHandler(32, clock\_handler, 0);

También añadimos la cabecera del handler en este archivo.

*void* clock\_handler();

Luego escribimos el handler, en entry.S.

// HANDLER DEL RELOJ

ENTRY(clock\_handler)

SAVE\_ALL;

EOI;

call clock\_routine;

RESTORE\_ALL;

iret;

Escribimos la rutina de servicio en interrupt.c, que no hace más que sumar ticks y mostrar el reloj.

    // Rutina del clock

extern zeos\_ticks; // Extern la llama de fuera.

*void* clock\_routine() {

    zeos\_show\_clock();

    zeos\_ticks++;

}

Habilitamos la interrupción, poniendo el primer bit a 0 (máscara del registro 0x21 tendrá que ser 0xFC, para dejar habilitada la de teclado = 11111100).

Finalmente escribimos el wrapper en libc.c, que llama a la interrupcion y guarda el resultado en %eax como siempre. Luego trata el resultado con el errno.

*int* gettime() {

*int* resultado;

*asm* ( "int $0x80"

        : "=a" (resultado)

        : "a" (10)

    );

    if (resultado >= 0) return resultado;

    else {

        errno = -resultado;

        return -1;

    }

}

La llamada a sistema, que la implementamos en sys.c es así:

*int* sys\_gettime() {

    return zeos\_ticks;

}

Además, darse cuenta que tenemos una variable zeos\_ticks. Esta variable la hemos declarado en sys.c así:

*int* zeos\_ticks;

Luego, como la necesitamos en interrupt.c, tenemos que llamarla con la palabra clave exern, como se ve arriba. También la necesitamos en system.c. En el main, tenemos que poner esta variable a 0 para que empiece a contar desde que bootea. Lo añadimos en la línea 74 y 75 del main de system.c, después que cargue el kernel.

extern zeos\_ticks; // Extern la llama de fuera.

zeos\_ticks = 0;