Writing a Kernel from Scratch

Γιάννης Τσιομπίκας

nuclear@member.fsf.org

9 Μαΐου 2012

Γιατί;

- It's FUN!
- Εξοικείωση με το hardware.
- Εμβάθυνση στον θαυμαστό κόσμο των λειτουργικών συστημάτων.
- Μια καλή δικαιολογία να γράψουμε assembly (FUN!!!).
- Post-apocalyptic computing syndrom...

Booting

Booting

Η εκτέλεση ξεκινάει από την ROM (BIOS) σε 16 bit real mode.

BIOS

- Φορτώνει το πρώτο sector (512 bytes) στην διεύθυνση 7c00.
- jump στην διεύθυνση 7c00.

Boot Loader

- Φορτώνει τον kernel στην μνήμη.
- Ενεργοποιεί το A20 line.
- Βάζει τον επεξεργαστή σε protected mode.
- jump στο entry point του kernel.

Booting

Η εκτέλεση ξεκινάει από την ROM (BIOS) σε 16 bit real mode.

BIOS

- Φορτώνει το πρώτο sector (512 bytes) στην διεύθυνση 7c00.
- jump στην διεύθυνση 7c00.

Boot Loader

- Φορτώνει τον kernel στην μνήμη.
- Ενεργοποιεί το A20 line.
- Βάζει τον επεξεργαστή σε protected mode.
- jump στο entry point του kernel.

Booting

Η εκτέλεση ξεκινάει από την ROM (BIOS) σε 16 bit real mode.

BIOS

- Φορτώνει το πρώτο sector (512 bytes) στην διεύθυνση 7c00.
- jump στην διεύθυνση 7c00.

Boot Loader

- Φορτώνει τον kernel στην μνήμη.
- Ενεργοποιεί το A20 line.
- Βάζει τον επεξεργαστή σε protected mode.
- jump στο entry point του kernel.

Multiboot

Multiboot header:

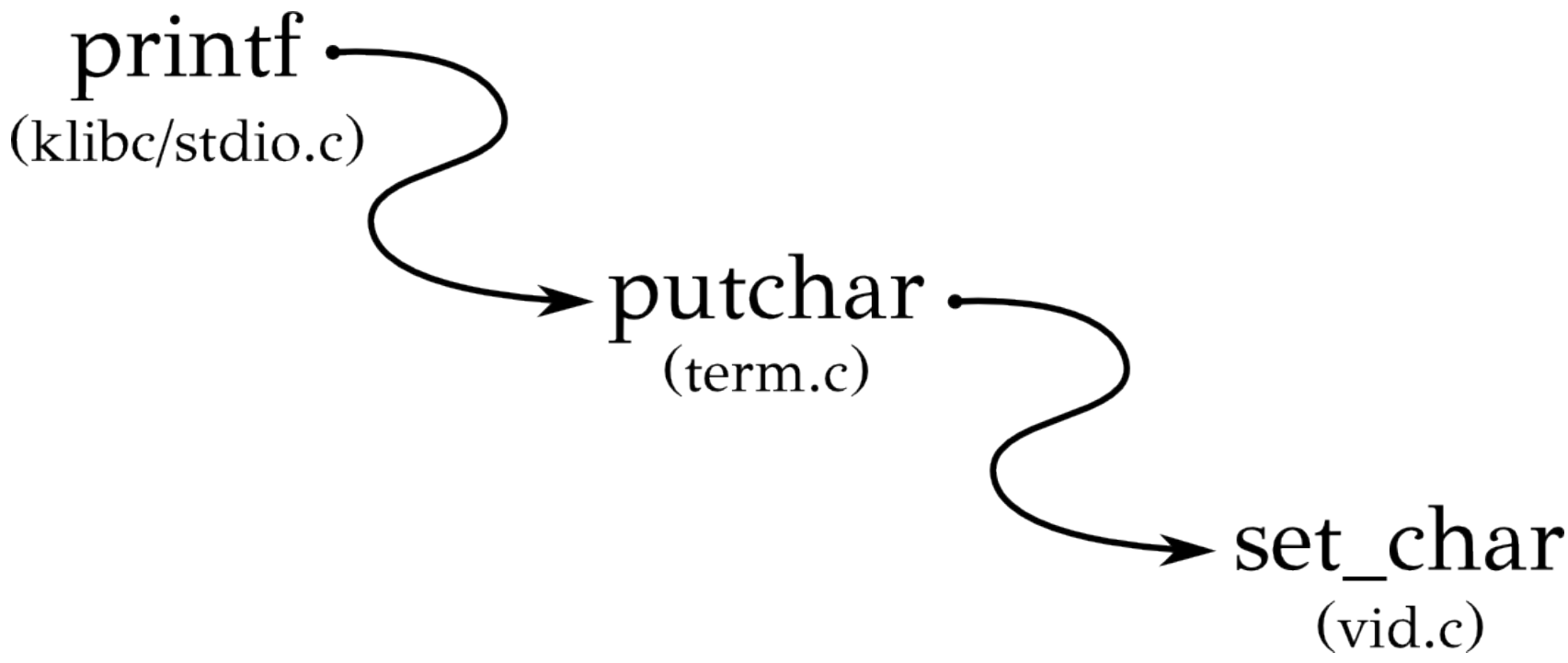
offset	μέγεθος	πεδίο
0	4	magic identifier (1badb002)
4	4	flags
8	4	checksum
12	4	header address
16	4	load address
20	4	load end address
24	4	bss end address
28	4	entry address
32	4	video mode type
36	4	video mode width
40	4	video mode height
44	4	video mode color depth

Entry code

```
#define MAGIC 0x1badb002
#define FLAGS 2
#define STACK_SIZE 0x4000
        .text
        /* multiboot header */
        .long MAGIC
        .long FLAGS
        .long -(MAGIC + FLAGS)   /* checksum */
        .globl kentry
kentry: /* setup a temporary kernel stack */
        movl $(stack + STACK_SIZE), %esp
        pushl $0  /* reset eflags register */
        popf
        /* call the kernel main function. ebx points to
         * the multiboot information structure */
        push %ebx
        call kmain
        /* dropped out of main, halt the CPU */
        cli
        hlt
        /* space for the temporary kernel stack */
        .comm stack, STACK_SIZE
```

VGA Text Mode Driver

Text output

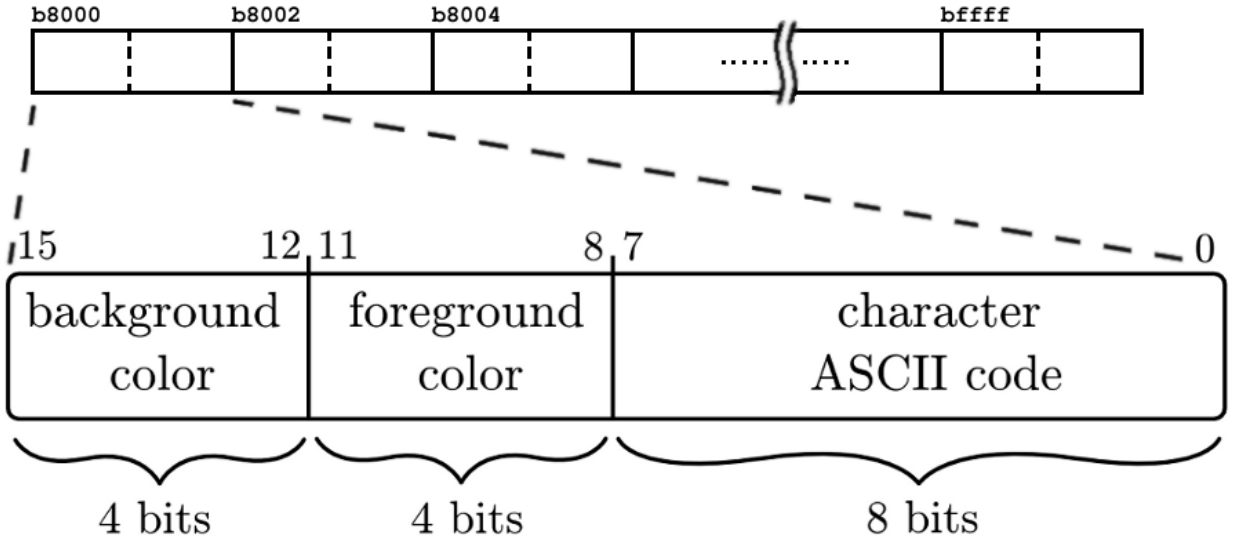


Text output - putchar

```
int putchar(int c)
{
    switch(c) {
    case '\n':
        cursor_y++;
    case '\r':
        cursor_x = 0;
        break;
    /* ... more ... */
    default:
        if(isprint(c)) {
            set_char(c, cursor_x, cursor_y, fg, bg);
            if(++cursor_x >= WIDTH) {
                cursor_x = 0;
                cursor_y++;
            }
        }
    }
    if(cursor_y >= HEIGHT) {
        scroll_scr();
        cursor_y--;
    }
    set_cursor(cursor_x, cursor_y);    /* set VGA cursor */
    return c;
}
```

Text output - set_char

VGA text mode video memory: b8000

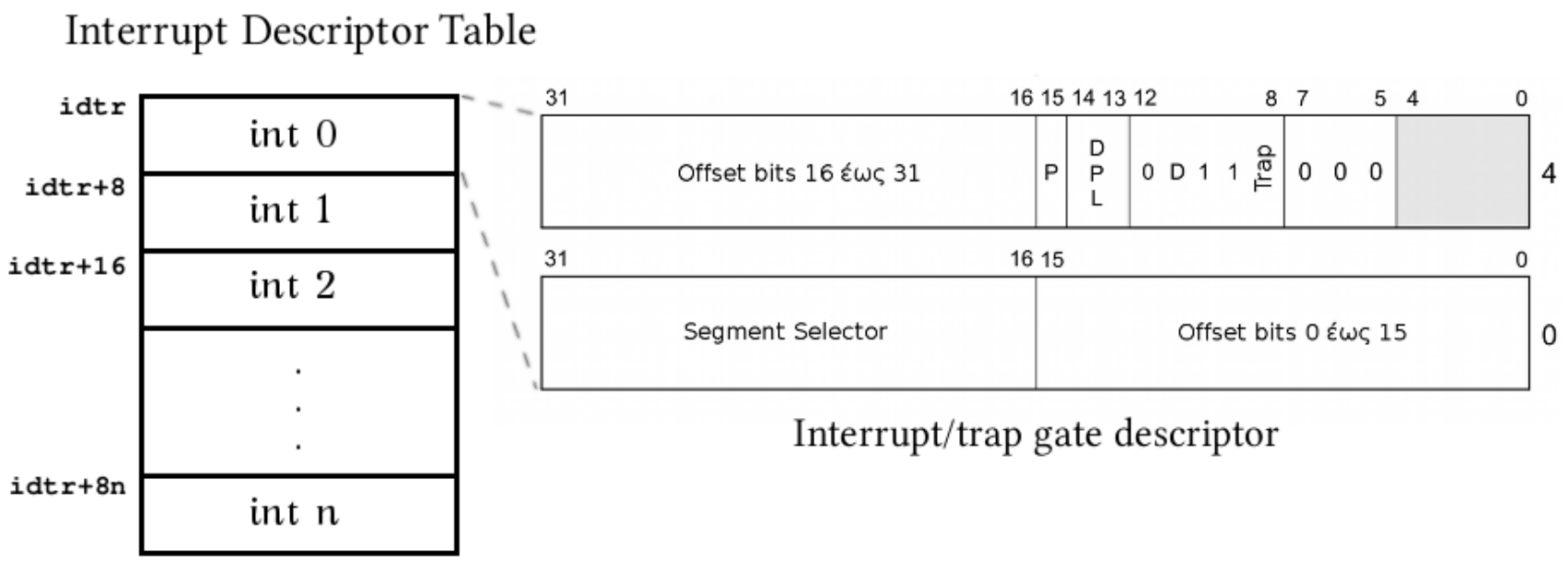



```
#define VMEM_CHAR(c, fg, bg) \
    ((uint16_t)(c) | (((uint16_t)(fg) & 0xf) << 8) | \
    (((uint16_t)(bg) & 0xf) << 12))

void set_char(char c, int x, int y, int fg, int bg)
{
    vmem[(y + start_line) * WIDTH + x] = VMEM_CHAR(c, fg, bg);
}
```

Interrupts

Τύποι interrupts

- Hardware interrupts
- Software interrupts
- Exceptions



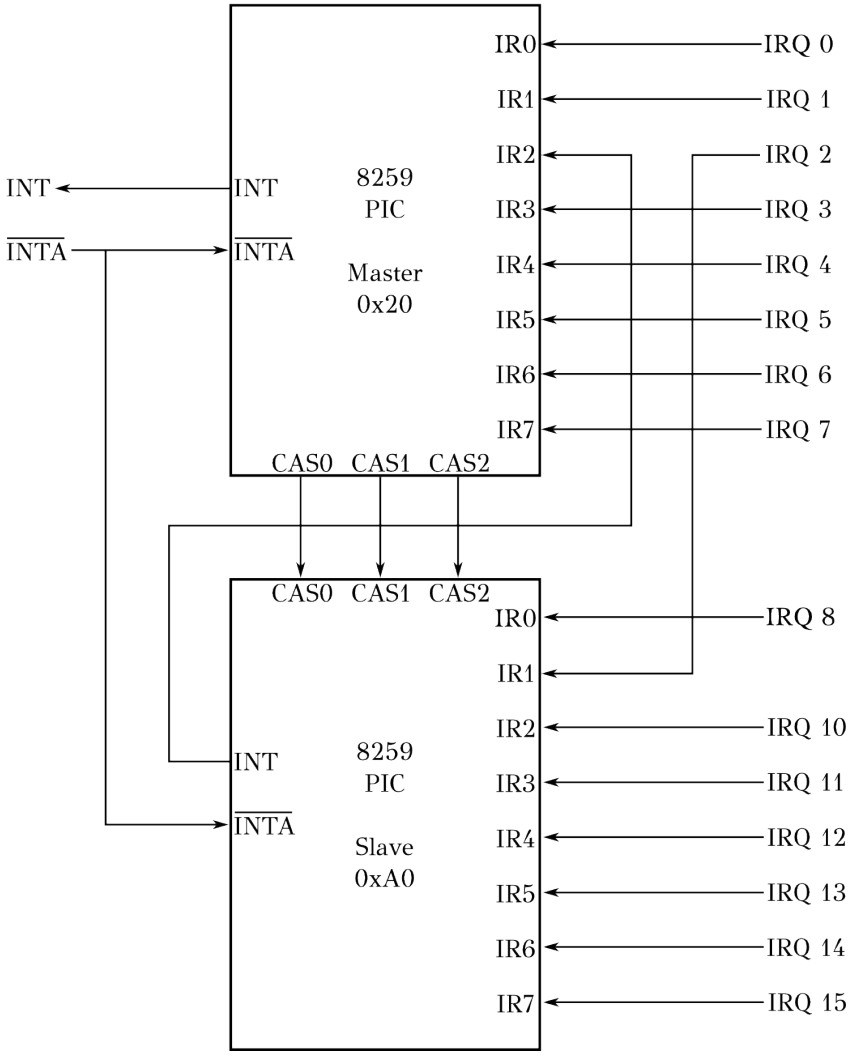
Exceptions

#	name	type	error code
0	divide error	fault	no
1	debug	trap/fault	no
2	NMI	N/A	N/A
3	breakpoint	trap	no
4	overflow	trap	no
5	bound range exceeded	fault	no
6	invalid opcode	fault	no
7	device not available	fault	no
8	double fault	abort	0
9	co-proc segment overrun	abort	no
10	invalid TSS	fault	selector
11	segment not present	fault	selector
12	stack fault	fault	selector or 0
13	general protection	fault	selector or 0
14	page fault	fault	special flags
15	reserved	N/A	N/A
16	floating point	fault	no
17	alignment check	fault	EXT bit
18	machine check	abort	no
19	SIMD floating point	fault	no

Τα interrupts 0 έως 31 είναι reserved για CPU exceptions.

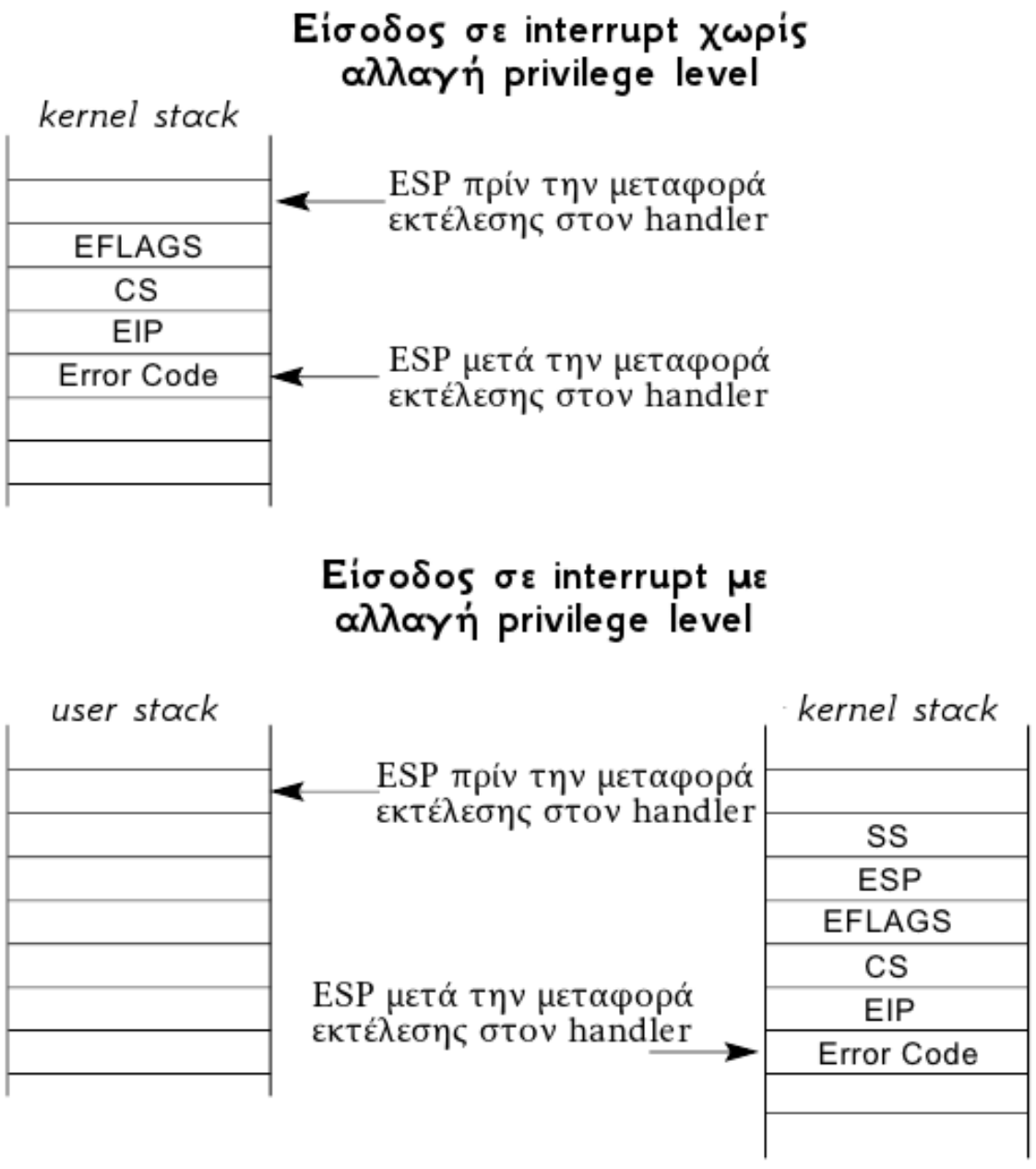
Hardware interrupts

2 cascaded intel 8259A PIC chips




```
static void init_pic(int offset)
{
    /* send ICW1 saying we'll follow with ICW4 later
           on */
    outb(ICW1_INIT | ICW1_ICW4_NEEDED, PIC1_CMD);
    outb(ICW1_INIT | ICW1_ICW4_NEEDED, PIC2_CMD);

    /* send ICW2 with IRQ remapping */
    outb(offset, PIC1_DATA);
    outb(offset + 8, PIC2_DATA);

    /* send ICW3 to setup the master/slave rel */
    /* ... set bit3 = 3rd int pin cascaded */
    outb(4, PIC1_DATA);
    /* ... set slave ID to 2 */
    outb(2, PIC2_DATA);

    /* send ICW4 to set 8086 mode (no calls) */
    outb(ICW4_8086, PIC1_DATA);
    outb(ICW4_8086, PIC2_DATA);

    /* done, reset the data port to 0 */
    outb(0, PIC1_DATA);
    outb(0, PIC2_DATA);
}
```

Interrupt stack



Interrupt entry

```
/* interrupt entry with error code */
    .macro ientry_err n name
    .globl intr_entry_\name
intr_entry_\name:
    pushl $\n
    jmp intr_entry_common
    .endm
```

```
/* interrupt entry without error code
 * pushes a dummy error code (0) */
    .macro ientry_noerr n name
    .globl intr_entry_\name
intr_entry_\name:
    pushl $0
    pushl $\n
    jmp intr_entry_common
    .endm
```

```
/* common code used by all entry points */
    .extern dispatch_intr
intr_entry_common:
    pushl %gs  /* save the current */
    pushl %fs  /* data segment     */
    pushl %es  /* selectors        */
    pushl %ds
    pusha  /* save general purpose regs */
    mov %ss, %eax  /* set the kernel   */
    mov %eax, %ds  /* data segment to  */
    mov %eax, %es  /* all data segment */
    mov %eax, %fs  /* selectors        */
    mov %eax, %gs
    call dispatch_intr
intr_ret_local:
    popa /* restore general purpose regs */
    popl %ds  /* restore data      */
    popl %es  /* segment selectors */
    popl %fs
    popl %gs
    /* pop error code and intr num */
    add $8, %esp
    iret
```

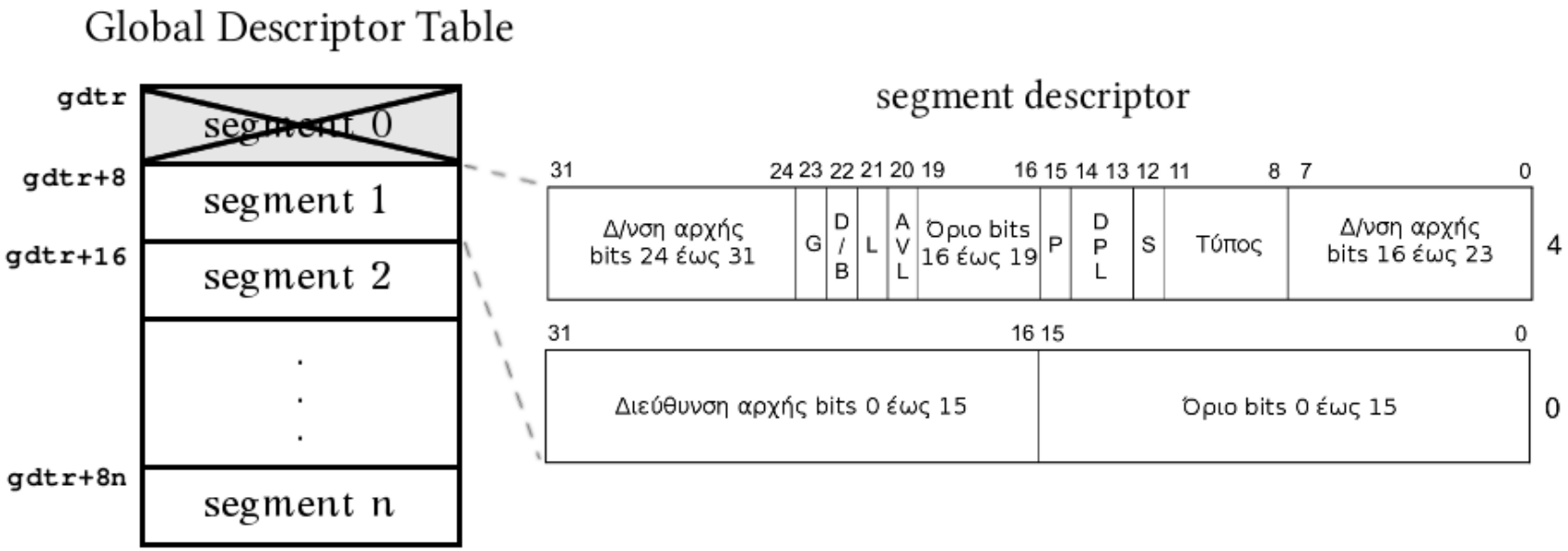
Interrupt entry macros

```
#ifdef ASM
/* included from intr-asm.S */
#define INTR_ENTRY_EC(n, name)      ientry_err n, name
#define INTR_ENTRY_NOEC(n, name)    ientry_noerr n, name
#else
/* included from intr.c inside init_intr() */
#define INTR_ENTRY_EC(n, name) \
    void intr_entry_##name(void); \
    set_intr_entry(n, intr_entry_##name);
#define INTR_ENTRY_NOEC(n, name)    INTR_ENTRY_EC(n, name)
#endif /* ASM */

INTR_ENTRY_NOEC(0, div)
INTR_ENTRY_NOEC(1, debug)
INTR_ENTRY_NOEC(2, nmi)
...
INTR_ENTRY_EC(12, stack)
INTR_ENTRY_EC(13, prot)
INTR_ENTRY_EC(14, page)
...
```

Memory Management

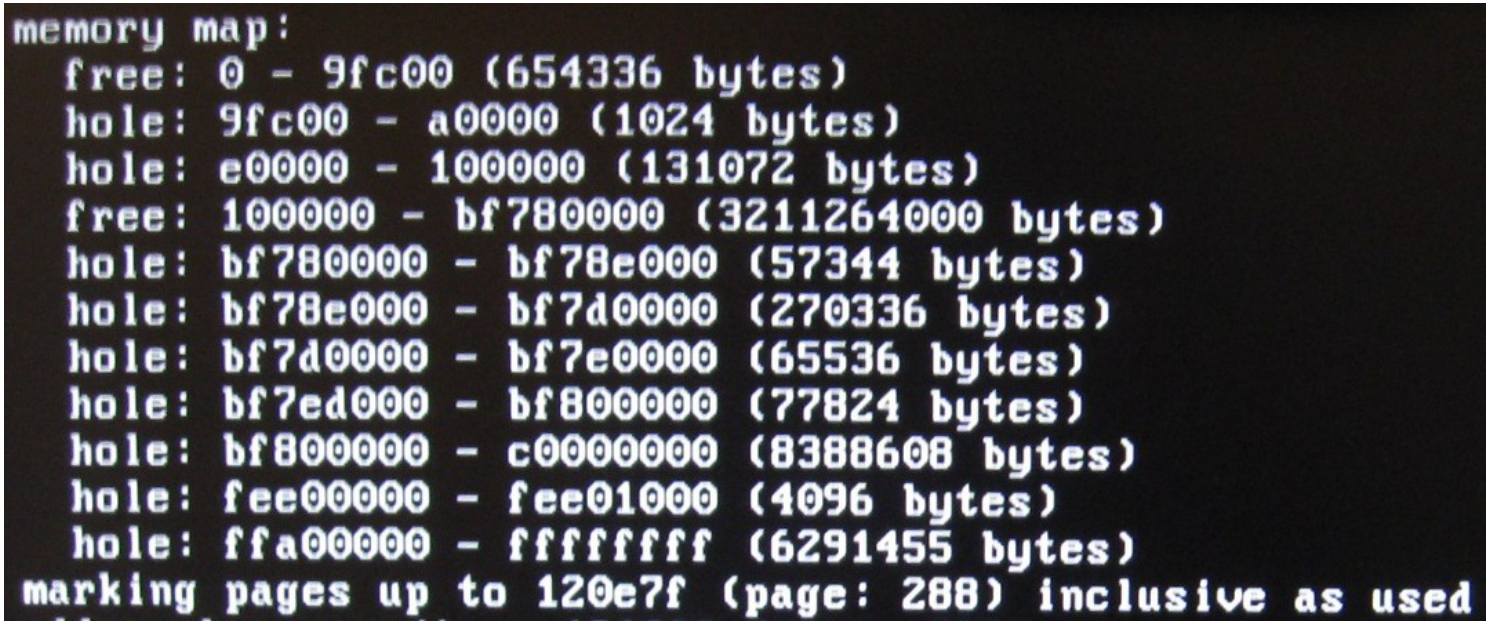
x86 memory model

- Segments ορίζονται από 8-byte descriptors στον GDT (ή LDT).
- Λογικές διευθύνσεις αποτελούμενες από segment και offset. Segment selector registers επιλέγουν ποιο segment χρησιμοποιείται σε κάθε περίπτωση.
- Προαιρετικό paging με page tables 2 επιπέδων.



Physical Memory Management

Multiboot memory map (απο το GRUB)

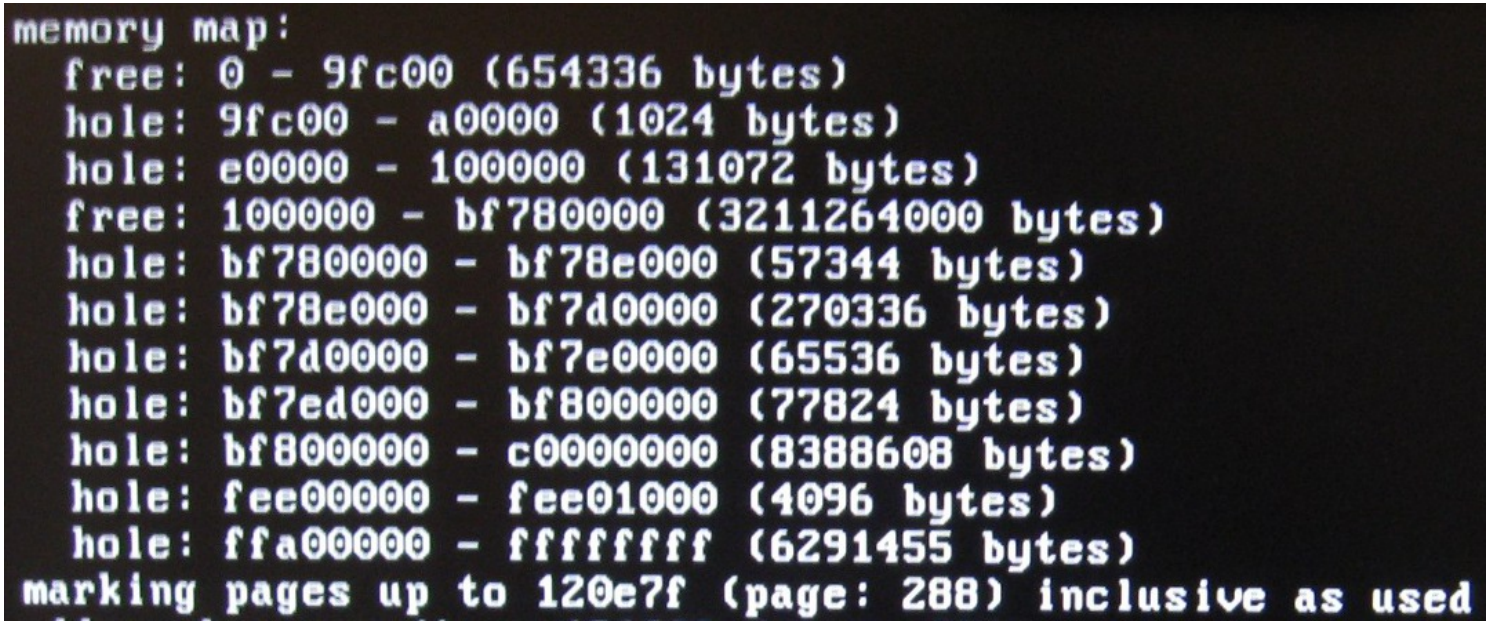


Bitmap allocator (alloc_phys_page / free_phys_page)

```
#define BM_IDX(pg) ((pg) / 32)
#define BM_BIT(pg) ((pg) & 0x1f)
#define ISFREE(pg) ((bmap[BM_IDX(pg)] & (1 << BM_BIT(pg))) == 0)
```

Physical Memory Management

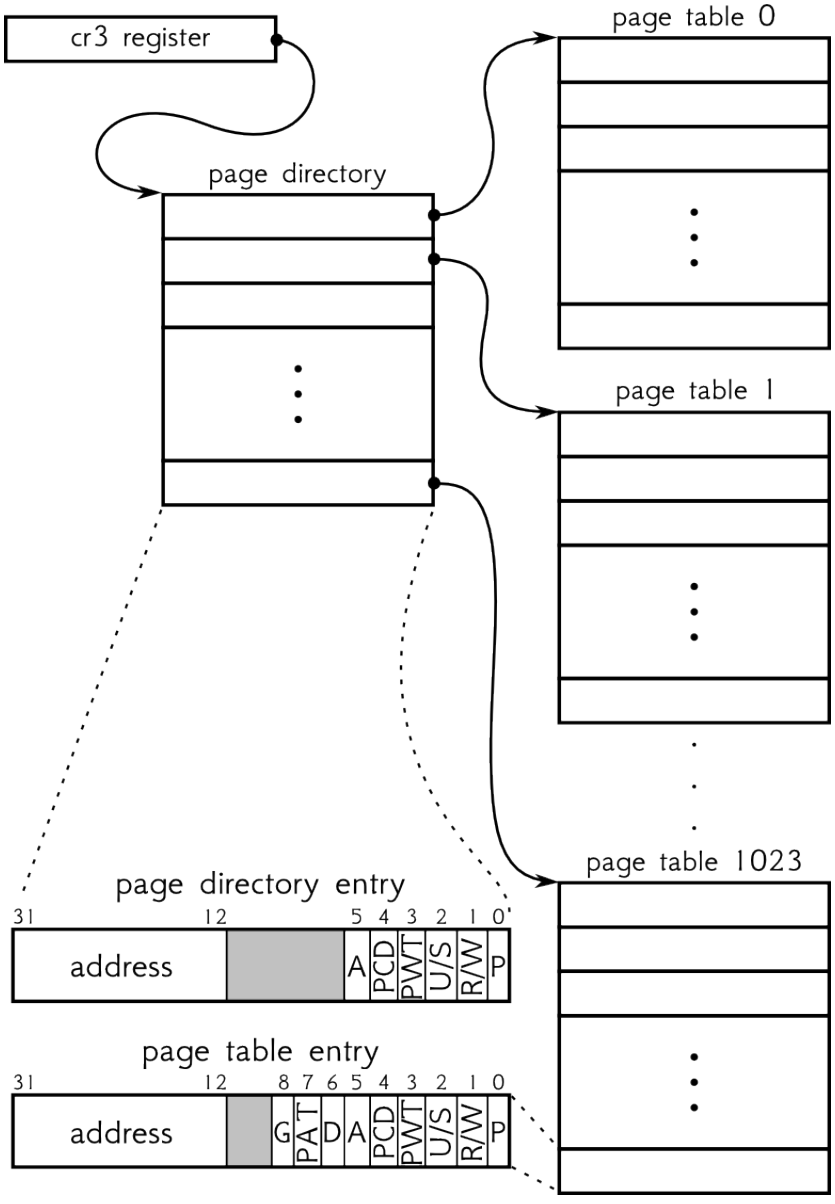
Multiboot memory map (απο το GRUB)



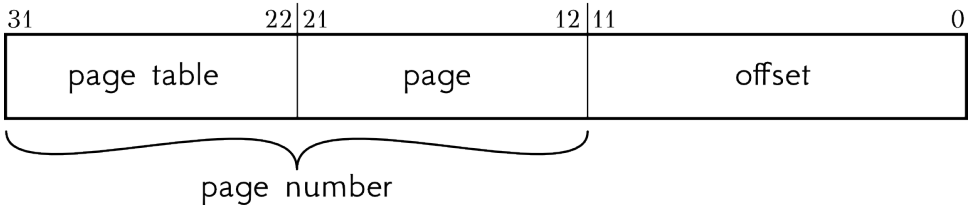
Bitmap allocator (alloc_phys_page / free_phys_page)

```
#define BM_IDX(pg) ((pg) / 32)
#define BM_BIT(pg) ((pg) & 0x1f)
#define ISFREE(pg) ((bmap[BM_IDX(pg)] & (1 << BM_BIT(pg))) == 0)
```

Page Translation



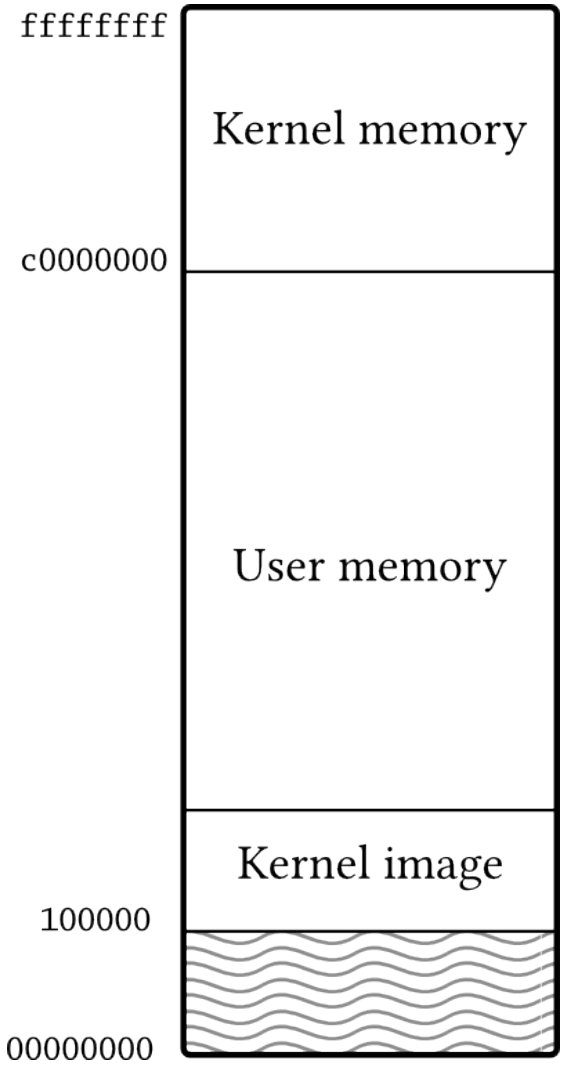
Virtual address translation



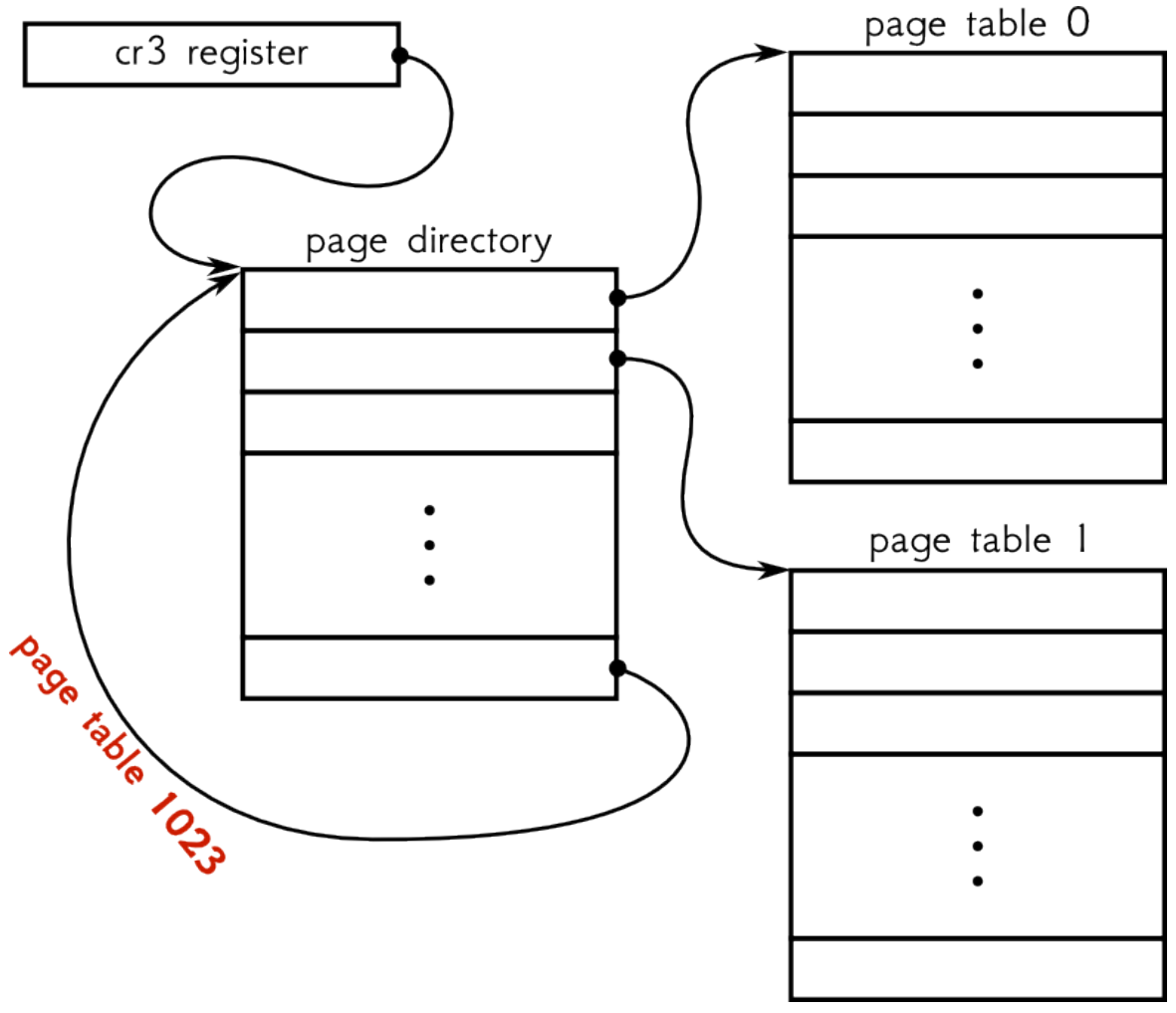
- Τα 10 ανώτερα bits $[22, 31]$ είναι index στο page directory (διαλέγουν page table).
- Τα επόμενα 10 bits $[12, 21]$ είναι index στο επιλεγμένο page table (διαλέγουν page).
- Τα κατώτερα 12 bits είναι το offset μέσα στο page.

Virtual Memory Management

- map_page / unmap_page
- TLB management
  (flush_tlb / flush_tlb_addr)
- Page range list allocator
  (pgalloc / pgfree)
- Higher level allocator
  (malloc / free)
- Page fault handling
  (more on that later)
- Recursive page tables
  (cont'd next slide...)



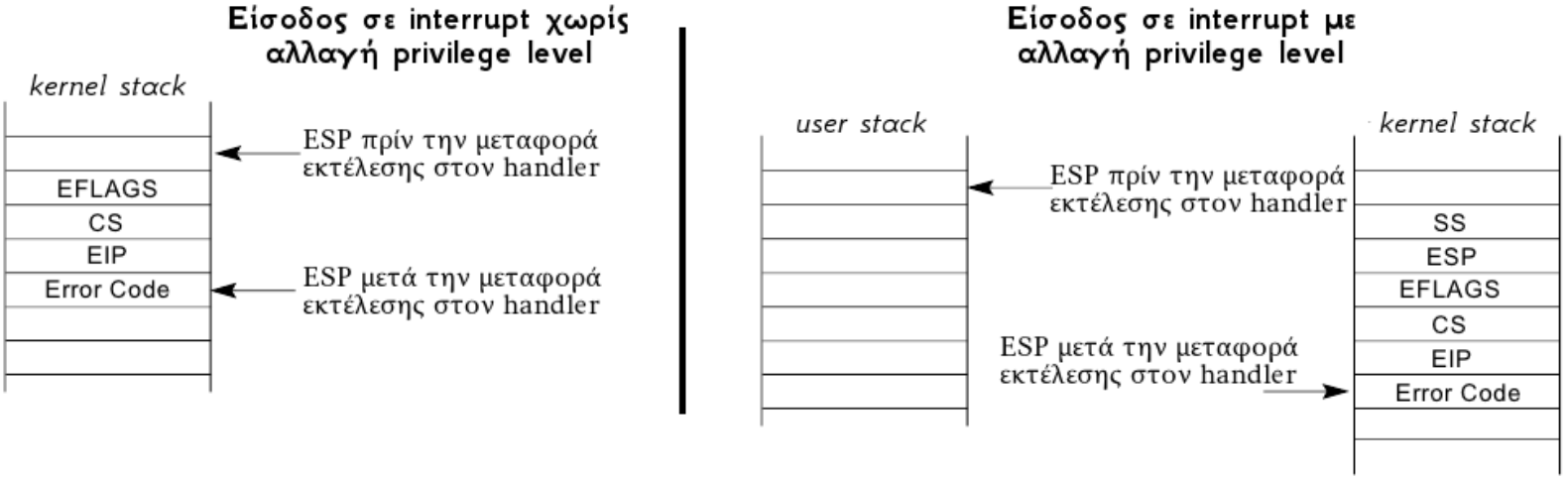
Recursive page tables



Processes

Kernel entry / exit

- 4 priviledge levels $[0, 3]$
- Αλλαγή του priviledge level όταν φορτώνεται segment descriptor με διαφορετικό `dpl`.
  - interrupts (int/iret)
  - call gates (intersegment call/ret)
  - sysenter/sysexit and friends
- Κάθε priviledge level < 3 έχει δικό του stack (ss/esp) στο TSS.



System calls

- interrupt 128
- system call number → eax
- arguments → ebx, ecx, edx, esi, edi
- result → eax

```
sys_func[SYS_HELLO] = sys_hello;
sys_func[SYS_SLEEP] = sys_sleep;      /* timer.c */
sys_func[SYS_FORK] = sys_fork;        /* proc.c */
sys_func[SYS_EXIT] = sys_exit;        /* proc.c */
sys_func[SYS_WAITPID] = sys_waitpid; /* proc.c */
sys_func[SYS_GETPID] = sys_getpid;    /* proc.c */
sys_func[SYS_GETPPID] = sys_getppid; /* proc.c */
```

System call interrupt handler

```
static void syscall(int inum)
{
    struct intr_frame *frm = get_intr_frame();
    int idx = frm->regs.eax;

    if(idx < 0 || idx >= NUM_SYSCALLS) {
        printf("invalid syscall: %d\n", idx);
        return;
    }
    /* the return value goes into the interrupt frame copy of
     * the user's eax so that it'll be restored into eax before
     * returning to userland.
     */
    frm->regs.eax = sys_func[idx](frm->regs.ebx, frm->regs.ecx,
        frm->regs.edx, frm->regs.esi, frm->regs.edi);

    /* we don't necessarily want to return to the same process
     * might have blocked or exited or whatever, so call
     * schedule to decide what's going to run next.
     */
    schedule();
}
```

Scheduler

Συνάρτηση `schedule`:

- Διαλέγει πάντα το πρώτο process απο την λίστα.
- Άν εξαντλήσει το timeslice του, αφαιρείται απο μπροστά και μπαίνει πίσω.
- Καλεί την `context_switch`.
- Άν το runqueue είναι άδειο καλεί την `idle_proc` που κάνει halt την CPU.

Context Switching

Τα context switches ξεκινούν με κλήση στην `context_switch`, η οποία:

- Αλλάζει page tables (cr3).
- Θέτει το kernel stack του καινούριου user process στο TSS.
- Κάνει push το state του current process στο stack του.
- Καλεί την `switch_stack` η οποία *επιστρέφει στο καινούριο context*!
- Επαναφέρει το state απο το καινούριο stack.

Σημείωση

Μετά απο fork η `swich_stack` ΔΕΝ επιστρέφει στο ίδιο σημείο και δέν εκτελείται το υπόλοιπο της `context_switch`.

Context Switching

Τα context switches ξεκινούν με κλήση στην `context_switch`, η οποία:

- Αλλάζει page tables (cr3).
- Θέτει το kernel stack του καινούριου user process στο TSS.
- Κάνει push το state του current process στο stack του.
- Καλεί την `switch_stack` η οποία *επιστρέφει στο καινούριο context*!
- Επαναφέρει το state απο το καινούριο stack.

Σημείωση

Μετά απο fork η `swich_stack` ΔΕΝ επιστρέφει στο ίδιο σημείο και δέν εκτελείται το υπόλοιπο της `context_switch`.

Context switch code

```
void context_switch(int pid)
{
    static struct process *prev, *new;
    prev = proc + last_pid;
    new = proc + pid;

    set_current_pid(new->id);
    /* switch to the new process' address space */
    set_pgdir_addr(new->ctx.pgtbl_paddr);
    /* make sure we'll return to the correct kernel
     * stack next time we enter from userspace */
    tss->esp0 = PAGE_TO_ADDR(new->kern_stack_pg) +
        KERN_STACK_SIZE;
    /* push all registers onto the stack before
     * switching stacks */

    push_regs();
    /* XXX: when switching to newly forked processes this
     * switch_stack call WILL NOT RETURN HERE. It will
     * return to just_forked instead. */
    switch_stack(new->ctx.stack_ptr, &prev->ctx.stack_ptr);
    /* restore registers from the *new* stack */
    pop_regs();
}
```

Process creation - fork

Processes δημιουργούνται με το system call `fork`.

- Allocate entry στο process table.
- Allocate kernel stack για το process.
- Αντιγράφει το current interrupt frame στο καινούριο stack, ώστε επιστρέφωντας σε user space να συνεχιστεί η εκτέλεση απο το ίδιο σημείο.
- Αλλάζει την τιμή του eax στο interrupt frame σε 0.
- Τοποθετεί την διεύθυνση της `just_forked` στο καινούριο stack για να επιστρέψει εκεί η `switch_stacks`.
- Καλεί την `clone_vm` για να πάρει το καινούριο process, αντίγραφο της μνήμης του current process.
- Προσθέτει το καινούριο process στο runqueue.

Process creation - clone_vm

Κάθε process έχει ενα `vmmap` που περιέχει ένα `vm_page` sturcture για κάθε mapped virtual page στο memory space του.
Το vmmap είναι ένα balanced binary search tree, για γρήγορη αναζήτηση με βάση το page number.

Η `clone_vm`:

- Αντιγράφει τα page tables (για το user memory space) του current process αφού καθαρήσει τα write bits.
- Αντιγράφει το kernel κομμάτι του page directory ώστε να χρησιμοποιηθούν τα ίδια page tables για την μνήμη του πυρήνα σε όλα τα processes.
- Δημιουργεί καινούριο `vmmap` με pointers στα ίδια `vm_page` αφού αυξήσει το reference count τους (`nref`).

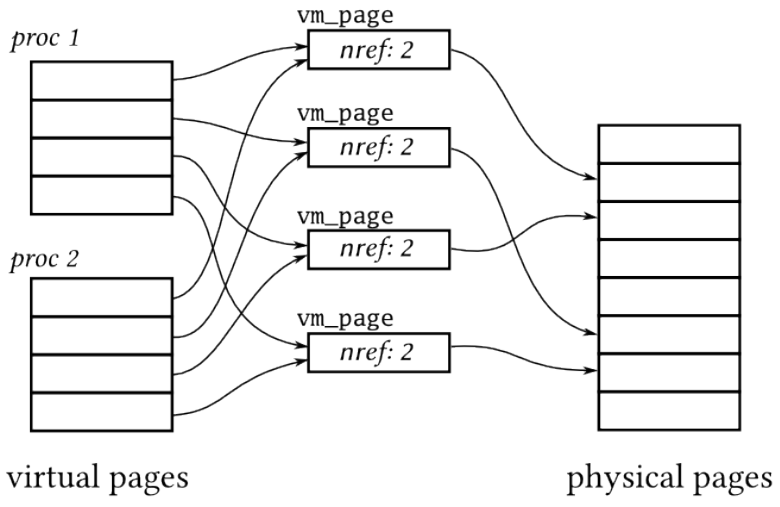
Copy on Write

Εγγραφή σε κάποιο απο τα κοινά pages σηκώνει pagefault. Ο page fault handler κάνει τα εξής:

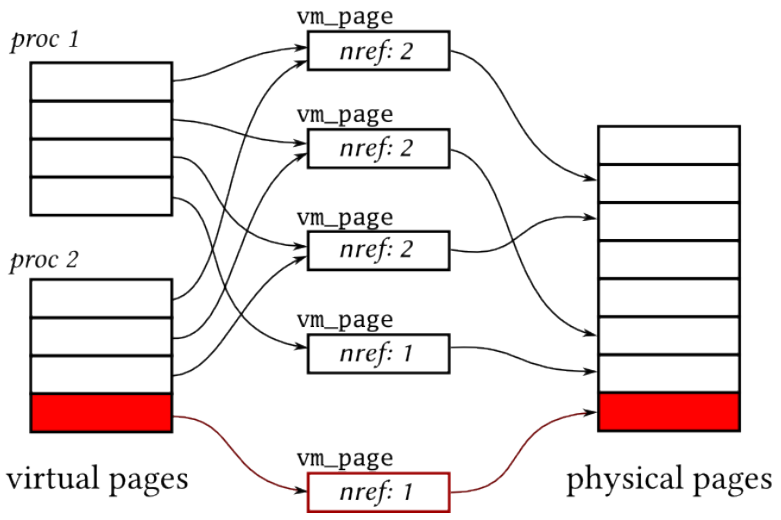
- Αναζητά το `vm_page` που αντιστοιχεί στο συγκεκριμένο page.
- Αν το `vm_page` είναι writable άλλα σηκώθηκε write fault πρόκειται για CoW fault.
- Στην οποία περίπτωση κάνει allocate καινούριο physical page και το κάνει map.
- Αντιγράφει τα περιεχόμενα του page που faultαρε στο καινούριο.
- Δημιουργεί καινούριο `vm_page` (με nref=0) και το αντικαθηστά το παλιό στο vmmap με το καινουριο.
- μειώνει το reference count του original `vm_page`.

Copy on Write - diagram

Μετά το fork όλα τα pages είναι κοινά...



Αντιγραφή μόνο σε απόπειρα εγγραφής.



Copy on Write - page fault handling

```
int pgnum = ADDR_TO_PAGE(fault_addr);

if((frm->err & PG_WRITABLE) && (get_page_bit(pgnum, PG_WRITABLE,
     0) == 0)) {
    /* write permission fault might be a CoW fault or just an
     * error. fetch the vm_page permissions to check if this is
     * suppoosed to be a writable page (meaning we should CoW)
     */
    struct vm_page *page = get_vm_page_proc(proc, pgnum);

    if(page->flags & PG_WRITABLE) {
        /* ok this is a CoW fault */
        if(copy_on_write(page) == -1) {
            panic("copy on write failed!");
        }
        return;  /* done, allow the process to restart the
            instruction and continue */
    } else {
        /* TODO: SIGSEGV the process, for now just panic */
        goto unhandled;
    }
}
```

First process

Το πρώτο process το κατασκευάζει η `start_first_proc`

- Γράφει το process image στη μνήμη.
- Κάνει allocate user stack και kernel stack.
- Τοποθετεί το kernel stack στο TSS.
- Του δίνει το υπάρχων page table.
- Δημιουργεί vmmap με βάση το υπάρχων page table.
- Το κάνει current και το προσθέτει στο runqueue.
- Κατασκευάζει ψεύτικο interrupt frame.
- καλεί την `intr_ret` για να κάνει fake επιστροφή απο interrupt στο καινούριο process, ώστε να γίνει η πρώτη αλλαγή priviledge level $0 \rightarrow 3$.

Blocking

- Τα processes blockάρουν σε kernel space καλώντας `wait` με την διεύθυνση αυτού που περιμένουν (wait channel).
- Η `wakeup` ξυπνάει όλα τα processes που περιμένουν σε ένα συγκεκριμένο wait channel.
- Hash table για αντιστοιχία wait channel $\rightarrow$ process.

Wait and wakeup

```
void wait(void *wait_addr)
{
    struct process *p;
    int hash_idx;

    disable_intr();

    p = get_current_proc();
    assert(p);
    /* remove it from the runqueue ... */
    remove(&runq, p);

    /* and place it in the wait hash table
     * based on sleep_addr
     */
    hash_idx = hash_addr(wait_addr);
    ins_back(wait_htable + hash_idx, p);

    p->state = STATE_BLOCKED;
    p->wait_addr = wait_addr;

    /* call the scheduler to give time to
          another process */
    schedule();
}
```

```
void wakeup(void *wait_addr)
{
    int hash_idx;
    struct process *iter;
    struct proc_list *list;

    hash_idx = hash_addr(wait_addr);
    list = wait_htable + hash_idx;

    iter = list->head;
    while(iter) {
        if(iter->wait_addr == wait_addr) {
            /* found one, remove it, and
             * make it runnable
             */
            struct process *p = iter;
            iter = iter->next;

            remove(list, p);
            p->state = STATE_RUNNABLE;
            ins_back(&runq, p);
        } else {
            iter = iter->next;
        }
    }
}
```

Test process run

```
#include <syscall.h>
    .text
    .globl test_proc
test_proc:
    /* fork another process */
    movl $SYS_FORK, %eax
    int $SYSCALL_INT
    /* getpid */
    movl $SYS_GETPID, %eax
    int $SYSCALL_INT
    push %eax
    xor %ecx, %ecx /* zero iter counter */
infloop:
    movl $SYS_HELLO, %eax
    int $SYSCALL_INT
    /* sleep for (pid * 2) seconds */
    movl (%esp), %ebx
    shl $1, %ebx
    movl $SYS_SLEEP, %eax
    int $SYSCALL_INT
    inc %ecx /* inc loop counter */
    /* let proc 2 exit after 2 iter */
    cmpl $2, (%esp)
    jne 1f
    cmpl $2, %ecx
    je exit_proc
1:  jmp infloop
exit_proc:
    movl $SYS_EXIT, %eax
    movl $0, %ebx
    int $SYSCALL_INT
    .globl test_proc_end
test_proc_end:
```

```
copied init process at: 112000
DBG:page fault in user space (pid:1)
process 1 says hello!
process 1 will sleep for 2 seconds
DBG:page fault in user space (pid:2)
process 2 says hello!
process 2 will sleep for 4 seconds
idle loop is running
timer going off!!!
process 1 says hello!
process 1 will sleep for 2 seconds
idle loop is running
timer going off!!!
timer going off!!!
process 1 says hello!
process 1 will sleep for 2 seconds
process 2 says hello!
process 2 will sleep for 4 seconds
idle loop is running
timer going off!!!
process 1 says hello!
process 1 will sleep for 2 seconds
idle loop is running
timer going off!!!
timer going off!!!
process 1 says hello!
process 1 will sleep for 2 seconds
process 2 exit(0)
idle loop is running
```

Timekeeping

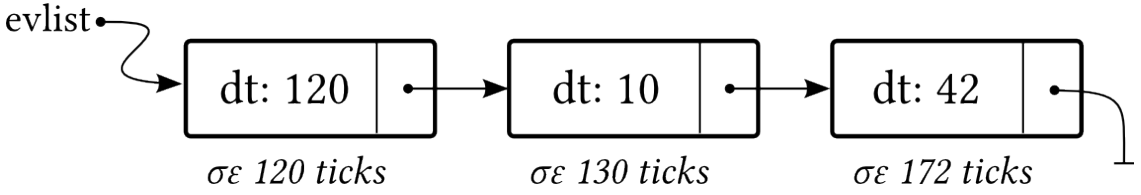
8253 Programmable Interval Timer

- Driven απο κρύσταλλο 1.193182 MHz
- 3 16bit binary counters
  - Channel 0: Γενικής χρήσεως, IRQ 0 στο underflow
  - Channel 1: DRAM refresh (unusable)
  - Channel 2: PC speaker
- Πολλαπλά modes λειτουργίας (one shot, rate, square wave, strobe...).

```
#define OSC_FREQ_HZ      1193182
#define TICK_FREQ_HZ     250

reload = DIV_ROUND(OSC_FREQ_HZ, TICK_FREQ_HZ);
```

Timer handling

- Global ticks counter (nticks).
- Λίστα με events, χρόνος σχετικός με προηγούμενο στη λίστα.

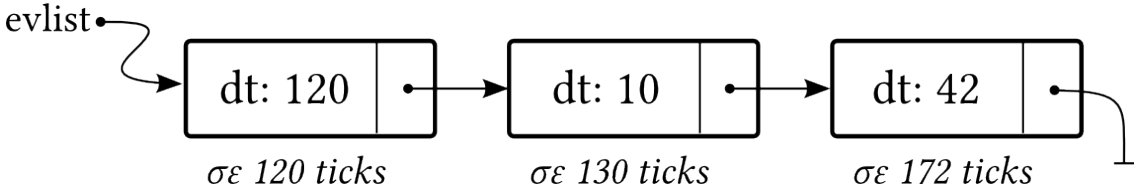


Interrupt handling

- increment `nticks`
- decrement `dt` του πρώτου event και wakeup αν είναι 0.
- call `schedule`

Timer handling

- Global ticks counter (nticks).
- Λίστα με events, χρόνος σχετικός με προηγούμενο στη λίστα.



Interrupt handling

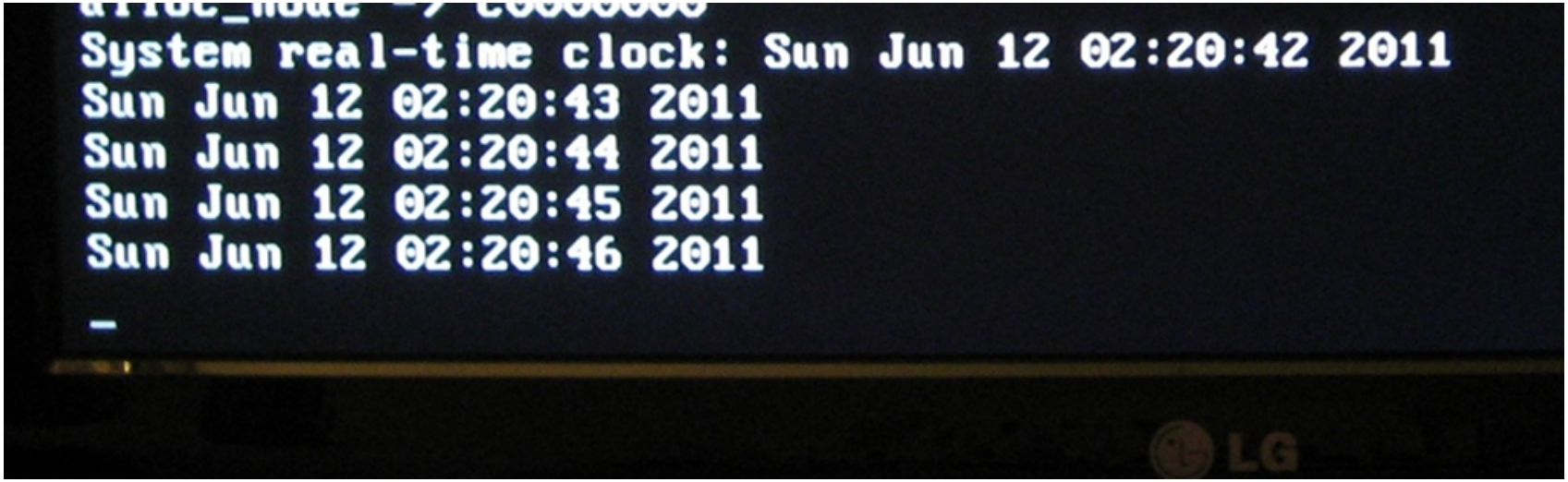
- increment `nticks`
- decrement `dt` του πρώτου event και wakeup αν είναι 0.
- call `schedule`

Timer handling code

```
static void timer_handler(int inum)
{
    struct process *p;

    nticks++;  /* increment global tick counter */

    /* find out if there are any timers that have to go off */
    if(evlist) {
        evlist->dt--;

        while(evlist && evlist->dt <= 0) {
            struct timer_event *ev = evlist;
            evlist = evlist->next;
            /* wake up all processes waiting on this address */
            wakeup(ev);
            free(ev);
        }
    }
    /* decrement the process' ticks_left and call the scheduler
     * to decide if it's time to switch processes */
    if((p = get_current_proc())) {
        p->ticks_left--;
    }
    schedule();
}
```

Real Time Clock

- RTC registers σε battery-backed static RAM (CMOS)
- Πιθανές μορφές δεδομένων στούς RTC registers, ορίζεται στον status register:
  - BCD ή decimal (συνήθως BCD !?@?!)
  - 24h ή 12h με high-order bit AM/PM
  - Έτος '2011' ή '11' (συνήθως '11' ?@?#!)
- Στον kernel: seconds από UNIX epoch & standard C mktime/asctime κ.τ.λ στην klibc.



Ερωτήσεις;

Links

- https://nuclear.mutantstargoat.com/hg/kern
- http://nuclear.mutantstargoat.com/articles/kerneldev
- http://codelab.wordpress.com
- http://www.linuxinside.gr