

Operativni sistemi

dr.sc. Amer Hasanović

Distribucija hardver prekida

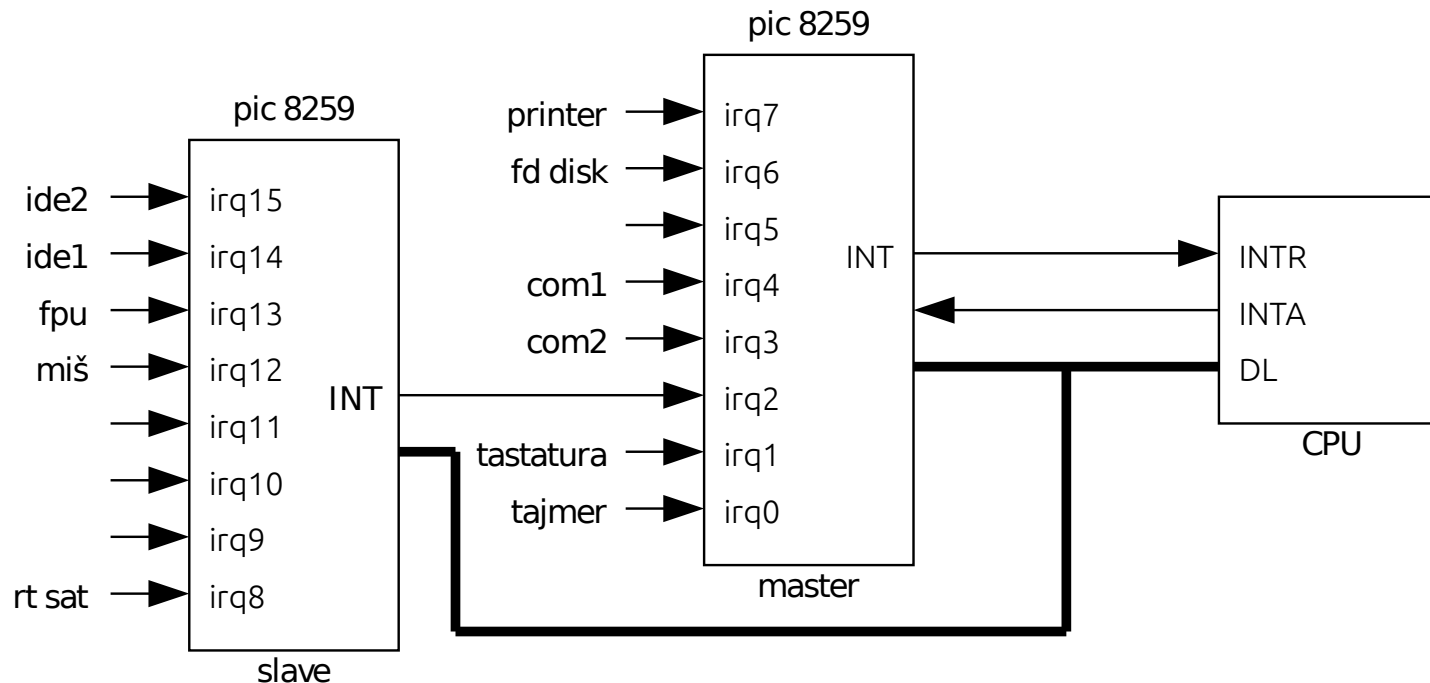
- Za generiranje konkretnih vektora prekida usljed djelovanja vanjskih hardverskih uređaja CPU zahtijeva poseban uređaj tzv kontroler prekida (programmable interrupt controller ili PIC)
- Postoje dva tipa kontrolera za Intel platformu:
 - 8259 kontroler za jednoprocesorske sisteme, stara Intel arhitektura.
 - Obično dva 8259 kontrolera u master-slave konfiguraciji
 - Svaki omogućava 8 linija prekida (IRQ) koje se konfiguriranjem mapiraju u konkretne vektore prekida
 - APIC (Advanced PIC) za multiprocesorske sisteme, moderni višezgredni CPU
 - Svako jezgro ima lokalni kontroler tzv LAPIC koji komunicira sa IOAPIC kontrolerom na koji su direktno vezani vanjski uređaji

- Kontroler prekida ima zadatak da identificira CPU-u uređaj koji zahtijeva trenutno servisiranje
 - zahtijev primljen sa jedne IRQ linije mapira se u vektor prekida radi aktiviranja adekvatne ISR;
 - nakon što tretira neki hardverski prekid, ISR kontroleru prekida treba da pošalje obavijest (tzv EOI end of interrupt) kako bi se mogli generirati novi prekidi.

Maskiranje prekida

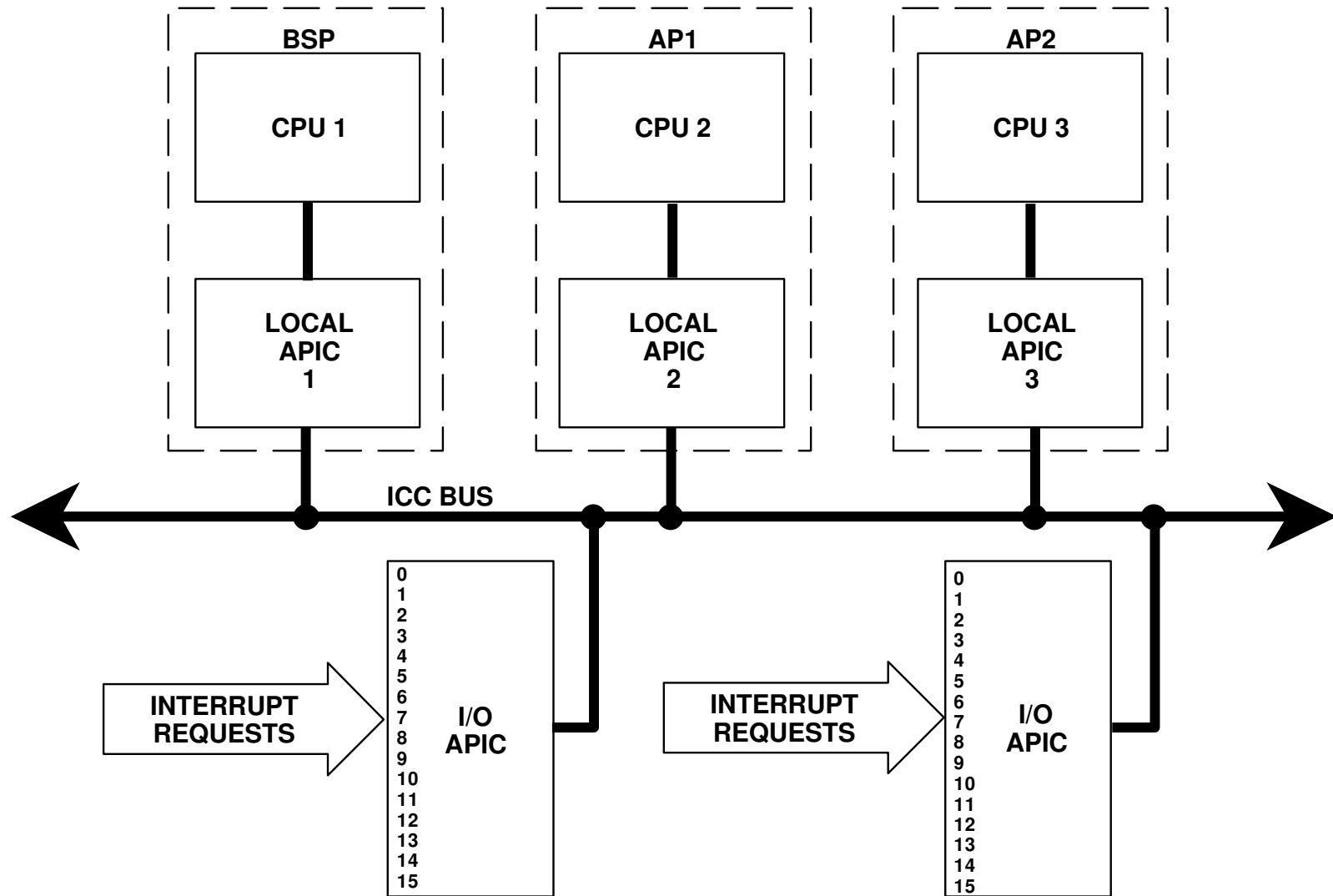
- Bez obzira na tip kontrolera, prekide je moguće maskirati
 - maskiranjem konkretnog bit-a u posebnom konfiguracionom registru PIC-a, prekidi od određenog uređaja ne proslijeđuju se ka CPU-u
- Svi hardverski prekidi mogu se maskirati direktno na procesorskom jezgri (tzv gašenje/paljenje prekida):
 - privilegovana instrukcija cli maskira, a sti demaskira sve prekide za to jezgro;
 - u slučaju maskiranja prekida, kontroler nastavlja generirati prekide, ali ih CPU ignorira;
 - IF (interrupt flag) bit u registru EFLAGS omogućava direktnu manipulaciju i/ili uvid u stanje maske (1 prekidi uključeni, 0 prekidi isključeni)

PIC 8259



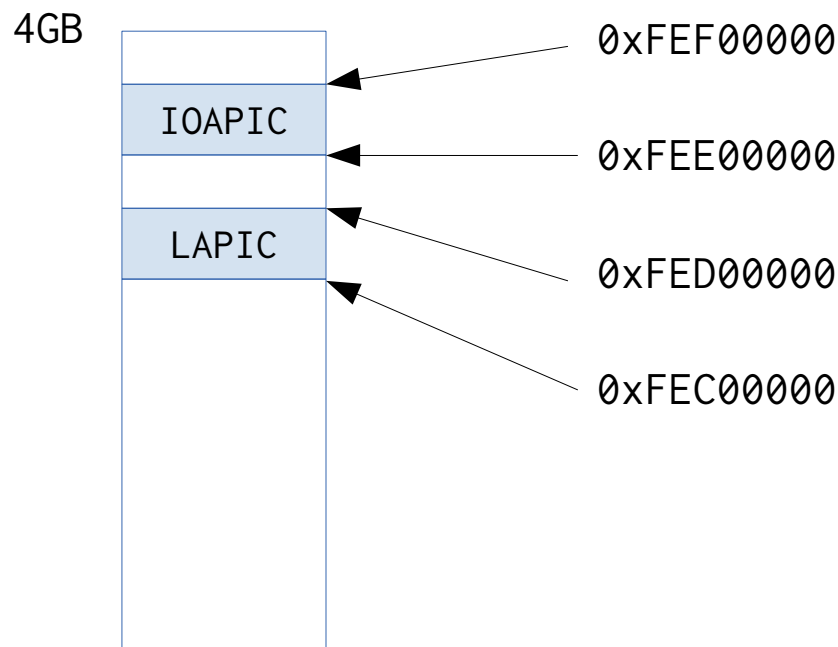
- Kontroleri se konfigurišu pomoću sekvenci in i out instrukcija, i to na adresama
 - 0x20 i 0x21 za master, 0xa0 i 0xa1 za slave.
- xv6 inicijalizira ovu konfiguraciju u funkciji `picinit`
 - Sve IRQ linije inicijalno su maskirane;
 - IRQ linije mapirane su u vektore prekida 32 pa na više

APIC infrastruktura



- Dvije funkcionalne jedinice implementiraju APIC
 - IOAPIC → vanjski kontroler obično sa 16 ili 24 linije za konektovanje na vanjske uređaje;
 - mapira linije u vektore prekida;
 - može se konfigurirati da određene linije usmjerava ka konkretnim CPU jezgrama ili da radi distribuciju spram trenutnog opterećenja jezgri.
 - LAPIC → kontroler integriran u konkretno procesorsko jezgro
 - konekovan na lokalne uređaje na jezgri npr: timer, senzor temperature itd

- Proizvljan broj LAPIC i IOAPIC uređaja komunicira preko ICC, magistrale za prekide:
 - međuprocorska komunikacija ostvaraju se slanjem prekida između dva LAPIC kontrolera
 - svaki kontroler ima jedinstveni identifikacioni broj tzv ID koji se koristi prilikom slanja prekida;
 - IOAPIC i LAPIC su memorijski mapirani
 - CPU pristupaju svojim LAPIC kontrolerima na istim adresama



- APIC arhitektura je u potpunosti simetrična:
 - svako jezgro ima jednak status, i svako jezgro može komunicirati sa bilo kojim drugim jezgrom.
 - sva jezgra dijele isti memorijski prostor i pristupaju tom memorijskom prostoru na istim adresama (memorijska simetrija)
 - sva jezgra imaju jednak pristup vanjskim uređajima (IO simetrija)
 - obično kroz memorijsko mapiranje;
- Memorijska simetrija izaziva potencijalne probleme, tzv **stanja utrke**, kada dva ili više jezgra pristupaju istom segmentu memorije
 - ovi problemi rješavaju se **sinhronizacijom**

BSP i AP

- Prilikom pokretanja sistema jedno procesorsko jezgro proglašava se kao *bootstrap processor* (BSP), ostala jezgra označavaju se kao *application processor* (AP)
 - BSP ima zadatak da inicijalizira komponente sistema i odradi boot sekvencu OS-a
 - AP-ovi su u zaustavljenom stanju dok ih BSP ne pokrene u posebnoj sekvenci međuprocorskih prekida

MP tabela

- Nakon inicijaliziranja haddrver komponenti BSP formira MP konfiguracione tabele u segmentu memorije koje pripadaju BIOS-u
 - MP tabele sadržavaju informacije o komponentama sistema, a koje koristi OS npr:
 - broj i tip CPU jezgri
 - broj i adrese IOAPIC kontrolera itd.
- xv6 na osnovu MP tabele inicijalizira interne strukture u funkciji mpinit:
 - funkcijama mpconfig i mpsearch BIOS prostor se skenira u potrazi za MP tabelom
 - inicijaliziraju se i slijedeće varijable:
 - lapic → adresa lapic kontrolera
 - ioapic, ioapicid → adresa i id ioapic kontrolera
 - ncpu, cpus → broj procesora, i niz struktura tipa cpu
 - ukoliko se tabele ne pronađu, xv6 označava ismp → 0, te se koristi jednoprocesorska konfiguracija sa 8259 PIC kontrolerom

```
// Bootstrap processor starts running C code here.  
// Allocate a real stack and switch to it, first  
// doing some setup required for memory allocator to work.
```

```
int main(void)  
{  
    kinit1(end, P2V(4*1024*1024)); // phys page allocator  
    kvmalloc(); // kernel page table  
    mpinit(); // collect info about this machine  
    lapicinit();  
    seginit(); // set up segments  
    cprintf("\ncpu%d: starting xv6\n\n", cpu->id);  
    picinit(); // interrupt controller  
    ioapicinit(); // another interrupt controller  
    consoleinit(); // I/O devices & their interrupts  
    uartinit(); // serial port  
    pinit(); // process table  
    tvinit(); // trap vectors  
    binit(); // buffer cache  
    fileinit(); // file table  
    iinit(); // inode cache  
    ideinit(); // disk  
    if(!ismp)  
        timerinit(); // uniprocessor timer  
    startothers(); // start other processors  
    kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must come after startothers()  
    userinit(); // first user process  
    // Finish setting up this processor in mpmain.  
    mpmain();  
}
```

Pokretanje AP jezgri

- AP jezgre pokreće BSP slanjem posebne sekvence međuprocorskih prekida (pogledati funkciju `lapicstartup`)
 - AP prilikom pokretanja traži adresu koda kojeg će početi izvršavati;
 - AP se starta u real modu i mora proći proces sličan boot procesu dok ne bude spreman da izvršava kod od kernela i/ili procesa;
 - BSP pokreće jezgra sekvencijalno i čeka u petlji dok AP kojeg trenutno pokreće eksplicitno ne postavi adekvatan status u polju `started cpu` strukture asocirane sa AP-om.

```

// Start the non-boot (AP) processors.
static void startothers(void)
{
    extern uchar _binary_entryother_start[], _binary_entryother_size[];
    uchar *code;
    struct cpu *c;
    char *stack;

    // Write entry code to unused memory at 0x7000.
    // The linker has placed the image of entryother.S in
    // _binary_entryother_start.
    code = p2v(0x7000);
    memmove(code, _binary_entryother_start, (uint)_binary_entryother_size);

    for(c = cpus; c < cpus+ncpu; c++){
        if(c == cpus+cpunum()) // We've started already.
            continue;

        // Tell entryother.S what stack to use, where to enter, and what
        // pgdir to use. We cannot use kpgdir yet, because the AP processor
        // is running in low memory, so we use entrypgdir for the APs too.

        stack = kalloc();
        *(void**)(code-4) = stack + KSTACKSIZE;
        *(void**)(code-8) = mpenter;
        *(int**)(code-12) = (void *) v2p(entrypgdir);

        lapicstartap(c->id, v2p(code));

        // wait for cpu to finish mpmain()
        while(c->started == 0)
            ;
    }
}

```

```

.code16
.globl start
start:
    cli
    xorw    %ax,%ax
    movw    %ax,%ds
    movw    %ax,%es
    movw    %ax,%ss
    lgdt    gdt_desc
    movl    %cr0, %eax
    orl     $CR0_PE, %eax
    movl    %eax, %cr0
    ljmpl   $(SEG_KCODE<<3), $(start32)

.code32
start32:
    movw    $(SEG_KDATA<<3), %ax
    movw    %ax, %ds
    movw    %ax, %es
    movw    %ax, %ss
    movw    $0, %ax
    movw    %ax, %fs
    movw    %ax, %gs

    # Turn on page size extension for 4Mbyte pages
    movl    %cr4, %eax
    orl     $(CR4_PSE), %eax
    movl    %eax, %cr4
    # Use enterpgdir as our initial page table
    movl    (start-12), %eax
    movl    %eax, %cr3
    # Turn on paging.
    movl    %cr0, %eax
    orl     $(CR0_PE|CR0_PG|CR0_WP), %eax
    movl    %eax, %cr0


    # Switch to the stack allocated by startothers()
    movl    (start-4), %esp
    # Call mpenter()
    call    *(start-8)
spin:
    jmp     spin

.p2align 2
gdt:
    SEG_NULLASM
    SEG_ASM(STA_X|STA_R, 0, 0xffffffff)
    SEG_ASM(STA_W, 0, 0xffffffff)

gdt_desc:
    .word   (gdt_desc - gdt - 1)
    .long   gdt

```

entryother.S



```
// Other CPUs jump here from entryother.S.
static void mpenter(void)
{
    switchkvm();
    seginit();
    lapicinit();
    mpmain();
}

// Common CPU setup code.
static void mpmain(void)
{
    cprintf("cpu%d: starting\n", cpu->id);
    idtinit();          // load idt register
    xchg(&cpu->started, 1); // tell startothers() we're up
    scheduler();        // start running processes
}
```