Interrupts and System Calls

Operating Systems I

Viktor Iakovlev (Victor Yacovlev)

x86 commands

Bytes	**0**	**1**	**2**	**3**	**4**	**5**
nop	0x00					
halt	0x10					
rrmovl *rA*, *rB*	0x20	*rA rB*				
irmovl *V*, *r*B	0x30	0x8 *rB*	*V*			
call *Des*t	0x80	*Dest*				
pushl *rA*	0xA0	rA 0x8				
popl *rA*	0xB0	rA 0x8				

C*omplex* **I***ntruction* **S***et* **C***omputing*

Commands are encoded using arbitary byte size

Generic Purpose Registers

- Intel 8080/8085 - 8 bits (A,B,C,D)
- Intel 8086 - 16 bits (AX, BX, CX, DX)
- Intel 80386 - 32 bits (EAX, EBX, ECX, EDX)
- AMD Opteron - 64 bits
  (RAX, RBX, RCX, RDX)

Intel 386 Architecture

- Builds for i386 are guaranteed to be working on any modern PC
- Multithreaded applications relies on next generation i486 due to CMPXCHG support
- i686 architecture features (Ubuntu minimum required):
  - CMOV - Conditional Move
  - SYSENTER/SYSCALL

gcc -march=ИМЯ_ПРОЦЕССОРА или native

https://gcc.gnu.org/onlinedocs/gcc/x86-Options.html#x86-Options

Additional Command Sets

- Vector Computing (MMX, SSE, AVX)
- Virtualization Extensions (VT-X or AMD-V)

cat /proc/cpuinfo

Floating Point Support

- Classic x86: distinct chip called co-processor x87
- Cons: the main CPU sleeps until FPU processes it’s own command
- x87 works with it’s own stack, changing data to CPU using main memory
- The x87 is a Standard for IA-32
- ... even it is not really used
- x86-64 completly dropped x87

Vector Arithmetics

- Special registers called XMM/YMM/ZMM
- Horizontal - maps vector to scalar
- Vertical - maps vector(s) to vector

Remember on low-precision **float** *type?*
It might be useful!

An Example: **dpps** command

```
dpps xmm0, xmm1, 0xF1
```

- Calculates a **dot product** for a vector of 16 floating-point values
- The constant only last operand describes:
  - Hight 4 bits (0xF) encodes what part of vectors to be in use
  - Low 4 bits (0x1) encodes where to place the result

Intel Command Sets

- MMX - integer only (128 bits)
- SSE - adds support for floating (128 бит)
- AVX и AVX-2 - floating point and integers (256 бит)
- AVX-512 - floating point and integers (512 бит)
  Notice: only Hi-End processors supports it

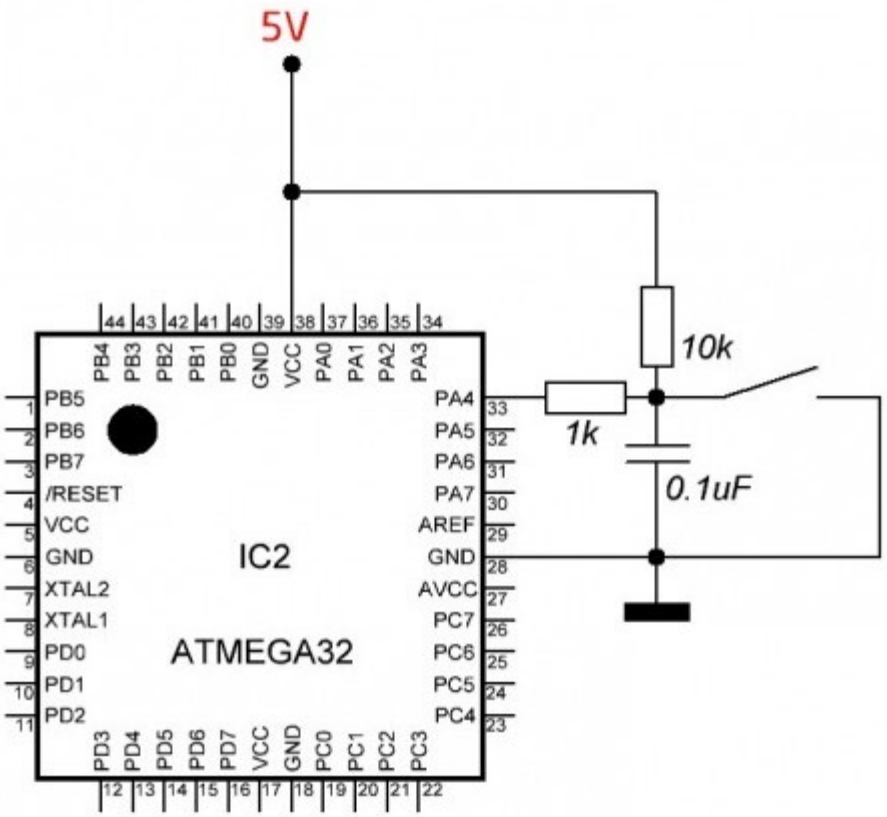
- Some compilers provides support for intrisicts to be used with C/C++ language, but not ASM:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/

Why to learn an assembly at «Operating Systems» course?

The Interrupts

Imaging a Hardware



Hardware Interrupts:

1. Button press
2. The next byte received by a transmission bus
3. Timer event
4. Aliens arrived

Interrupts Handling

- Save Instruction Pointer to stack
- Jump to instruction pointed by IDTR register + offset

What Happens on Interrupt

Before Interrupt Handle:

- Pipeline flush and disabling Out-of-order execution
- Virtual to physical mapping address table switch
- Switch stack pointer

To be Processed by Handler:

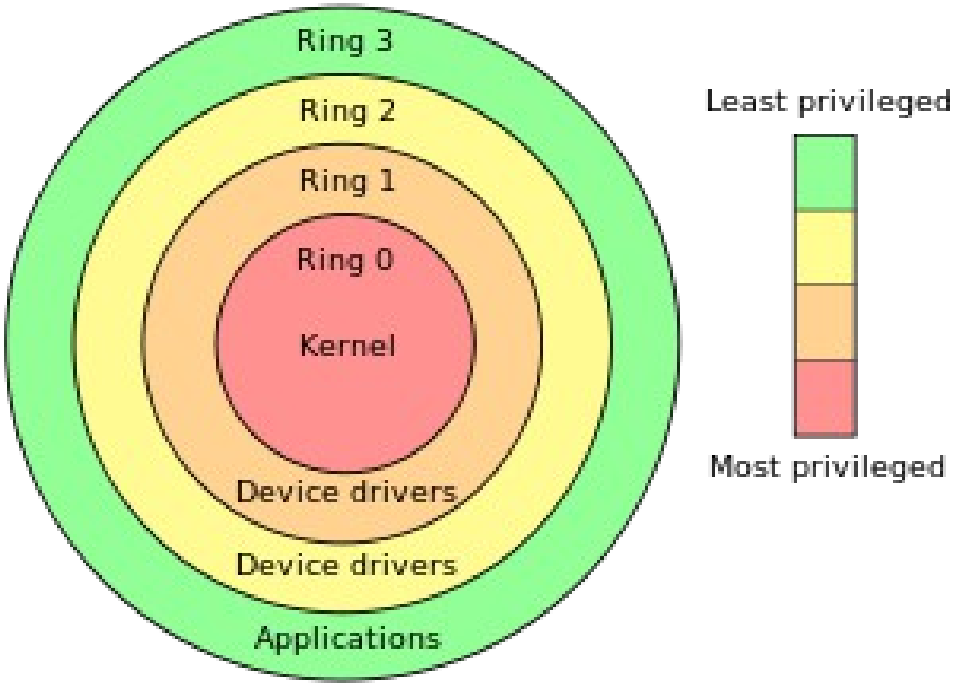
- Save processor state to stack
- .... [handle an event]
- Restore processor state back from stack

Event Handler

- Interact with hardware caused interrupt
- Requires direct access to hardware and memory
- Regular programs can’t do it
  (even started by **root** *superuser)*

x86 Isolation Rings

- Each process has it's own memory
- Regular programs has no access to I/O ports, but privileged has



Software Interrupts

- **int NN** for the most instruction sets (Intel, ARM etc.)
- Have the same behaviour like hardware interrupts

- Handled by BIOS before OS loaded
- Operating System might (bu not required) to modify table of interrupt handlers

BIOS Interrupts Example

AH used for command, AL - for it's operand

BIOS

- INT 0x10 - Text ouput
- INT 0x13 - Disk I/O
- INT 0x15 - UART (COM-port) handling
- INT 0x16 - Keyboard

DOS

- INT 0x21 - used by DOS API

The Kernel

- Just a program that operates at highest (in most cases) processor level
- Has access to everything

System Call

- Some function from Kernel API
- Operates in Kernel Mode: the most privileged level
- Switches back to unprivileged level after execution finished

INT 0x80

- Unified interrupt number in Linux
- The EAX register stores the number of system call
  (see /usr/include/asm/unistd_32.h)
- Parameters are stored at EBX, ECX, EDX, ESI, EDI, EBP
- The value returns within EAX register

Linux System Calls

- Documentation covered by section 2 of man pages
- The C standard library has C wrappers according to calling conventions

Example: string output

```
static const char S[] = "Hello";
write(1, S, sizeof(S));
/* man 2 write
    #include <unistd.h>
    ssize_t write(int fd, const void *buf, size_t count);
*/
```

```
#include <asm/unistd_32.h>

        mov eax, __NR_write  // system call number
        mov ebx, 1           // the first argument
        mov ecx, S_ptr       // the second argument
        mov edx, 6           // the third argument
        int 0x80             // do it!

S:      .string "Hello"
S_ptr: .dword S
```

linux-vdso.so (linux-gate.so)

```
$ ldd /usr/bin/cat
        linux-vdso.so.1 (0x00007ffe3ea87000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f9cea333000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f9cea6d6000)
```

- Virtual «library» to wrap some system calls as regular functions
- Maps functions that do not require high privileges:
  - __vdso_clock_gettime
  - __vdso_getcpu
  - __vdso_gettimeofday
  - __vdso_time

Modern Processors System Call Support

- sysenter/sysexit - for IA-32
- syscall и vsyscall - for AMD x86-64