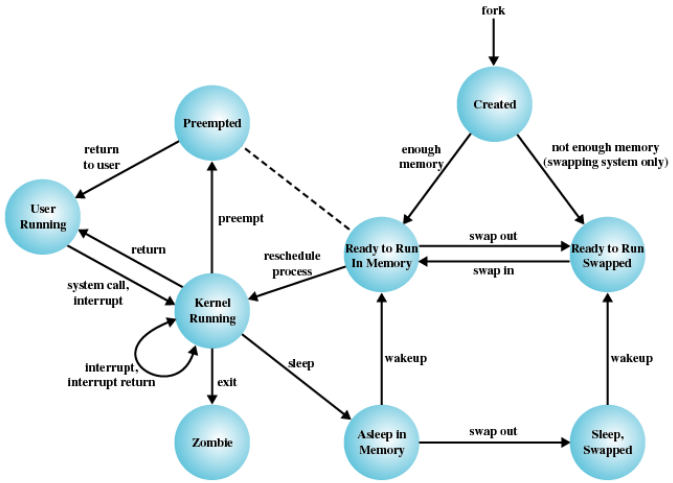
Operating Systems & Computer Networks

Subsystems, Interrupts, and System Calls



Content (1)

1. Introduction and Motivation
2. Subsystems, Interrupts and System Calls
3. Processes
4. Memory
5. Scheduling
6. I/O and File System
7. Booting, Services, and Security

Hierarchical System View

4. Interrupts
   - Objects: interrupt service routines (ISRs)
   - Operations: call, mask, unmask
3. Procedures
   - Objects: subroutines, call stack
   - Operations: stack pointer, call, return
2. Instruction Set
   - Objects: stack, microcode compiler, scalar and vector data
   - Operations: add, subtract, load, store, and branch
1. Electronic Circuits
   - Objects: registers, memory cells, logic gates
   - Operations: register or memory access

Hierarchical System View

7. Virtual Memory
   - Objects: segments, pages
   - Operations: read, write, load
6. Secondary Storage
   - Objects: blocks of data, device channels
   - Operations: read, write, lock, unlock
5. Primitive processes (program being executed)
   - Objects: simple processes, semaphore, ready lists
   - Operations: suspend, resume, wait, signal

Hierarchical System View

13. Shell
    - Objects: user programming interface
    - Operations: operations in shell command language
12. User Processes (incl. data on used resources)
    - Objects: user processes
    - Operations: create, terminate, suspend, resume
11. Directories
    - Objects: directories
    - Operations: create, delete, append, remove, search, list
10. Devices
    - Objects: external devices, e.g., printer, display, and keyboard
    - Operations: open, close, read, write
9. File System
    - Objects: named files
    - Operations: create, delete, open, close, read, write
8. Inter-process Communication (IPC)
    - Objects: channels, shared memory
    - Operations: create, delete, open, close, read, write

Content (1)

Interrupts - Motivation

- A lot of devices are connected to a computer, e.g.,
  - Keyboard, hard disk, network interface
- These devices occasionally need CPU service:
  - Keyboard: a key is pressed
  - Hard disk: a task is completed
  - Network interface: a packet has arrived
- BUT: it is **not predictable when** these devices need to be serviced
- How does the CPU find out that a device needs attention?
  - Two options: **Interrupts** and **Polling**

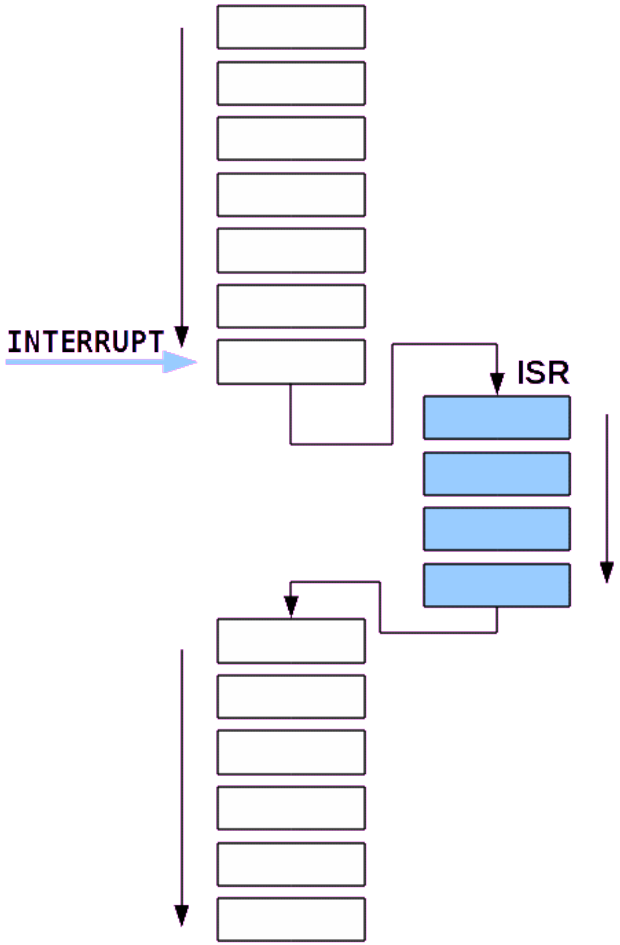
Interrupts versus Polling

Freie Universität Berlin

Interrupts	Polling
Give each device a wire that it can use to signal the CPU.	Ask the devices periodically if an event has occured.
Like a phone that rings when a call comes in.	Like a phone without a bell: You have to pick it up every few seconds to see if you have a call.
No overhead when no requests pending, efficient use of CPU time.	Takes CPU time even when no requests pending.
Devices are serviced as soon as possible - low latency.	Response time depends on polling rate.

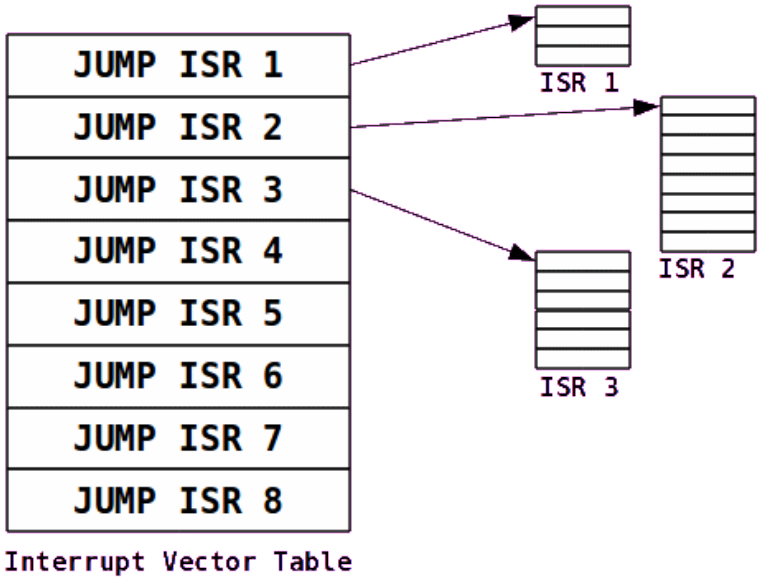
Interrupt Service Routines

- Interrupt handling is performed by the operating system (device drivers) in **interrupt service routines** (ISRs).
- Interrupts temporarily discontinue the currently executing application.



Interrupt Vector Table

- **Interrupt vector table** maps interrupts to service routines that handle them
- Table has one entry for each interrupt
- Each entry contains the address of the ISR (interrupt vector)
- Table resides in main memory at a constant address (interrupt base address)
- Interrupt number provides index into the table



Detecting Interrupts

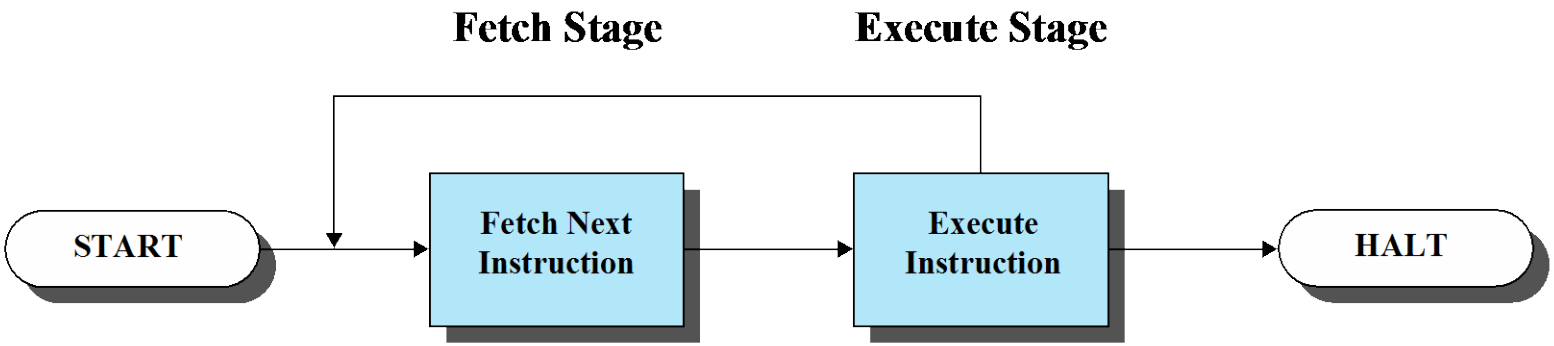


Figure 1.2 Basic Instruction Cycle

Detecting Interrupts

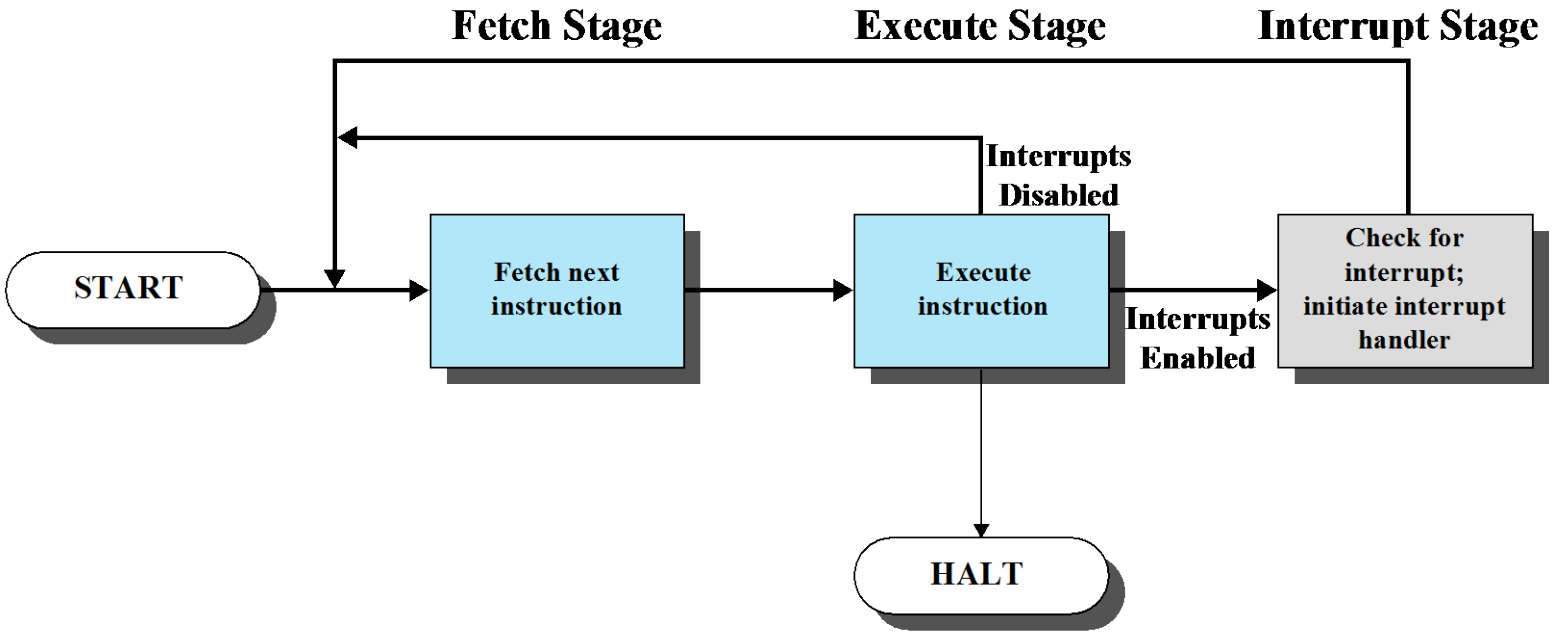
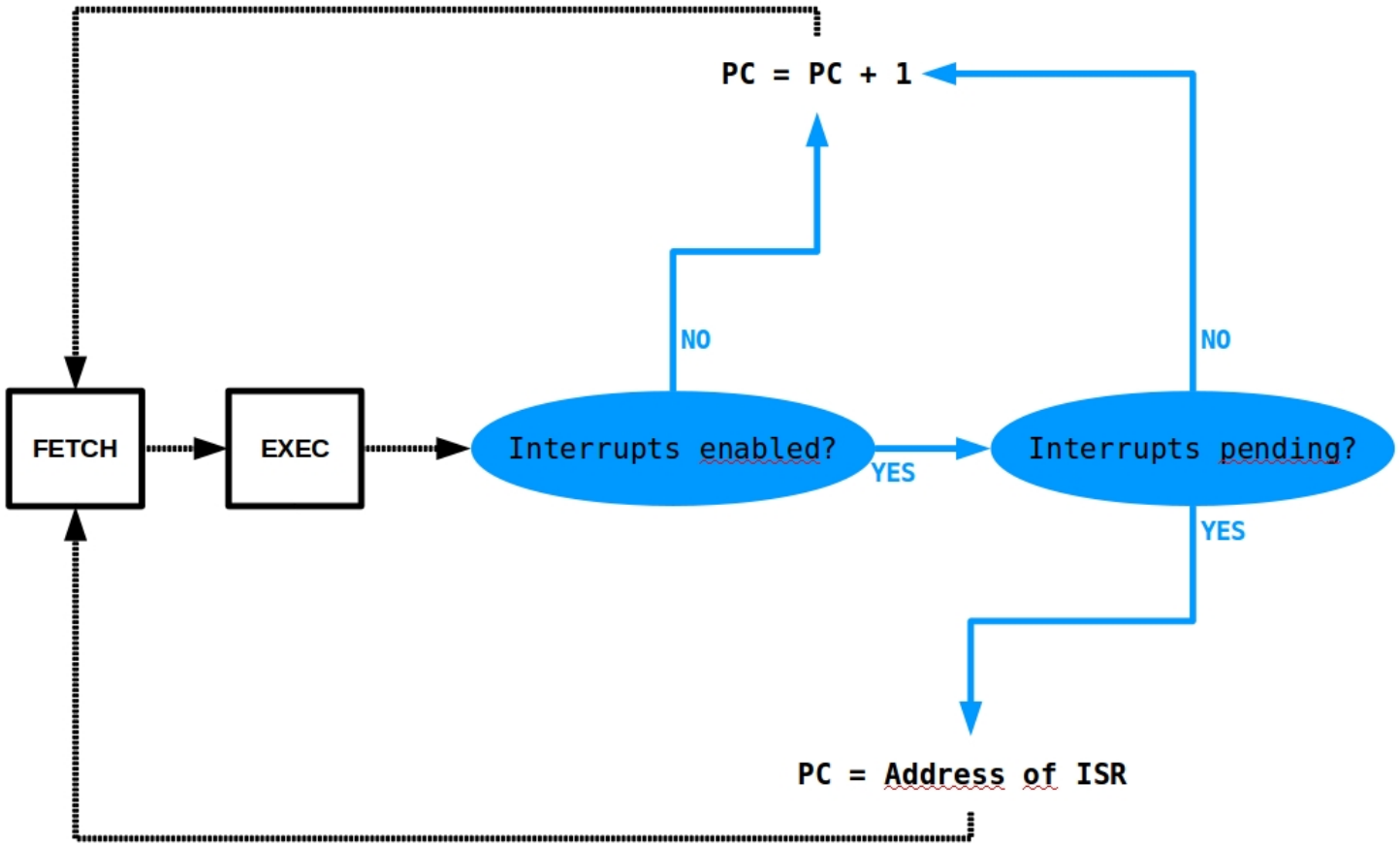


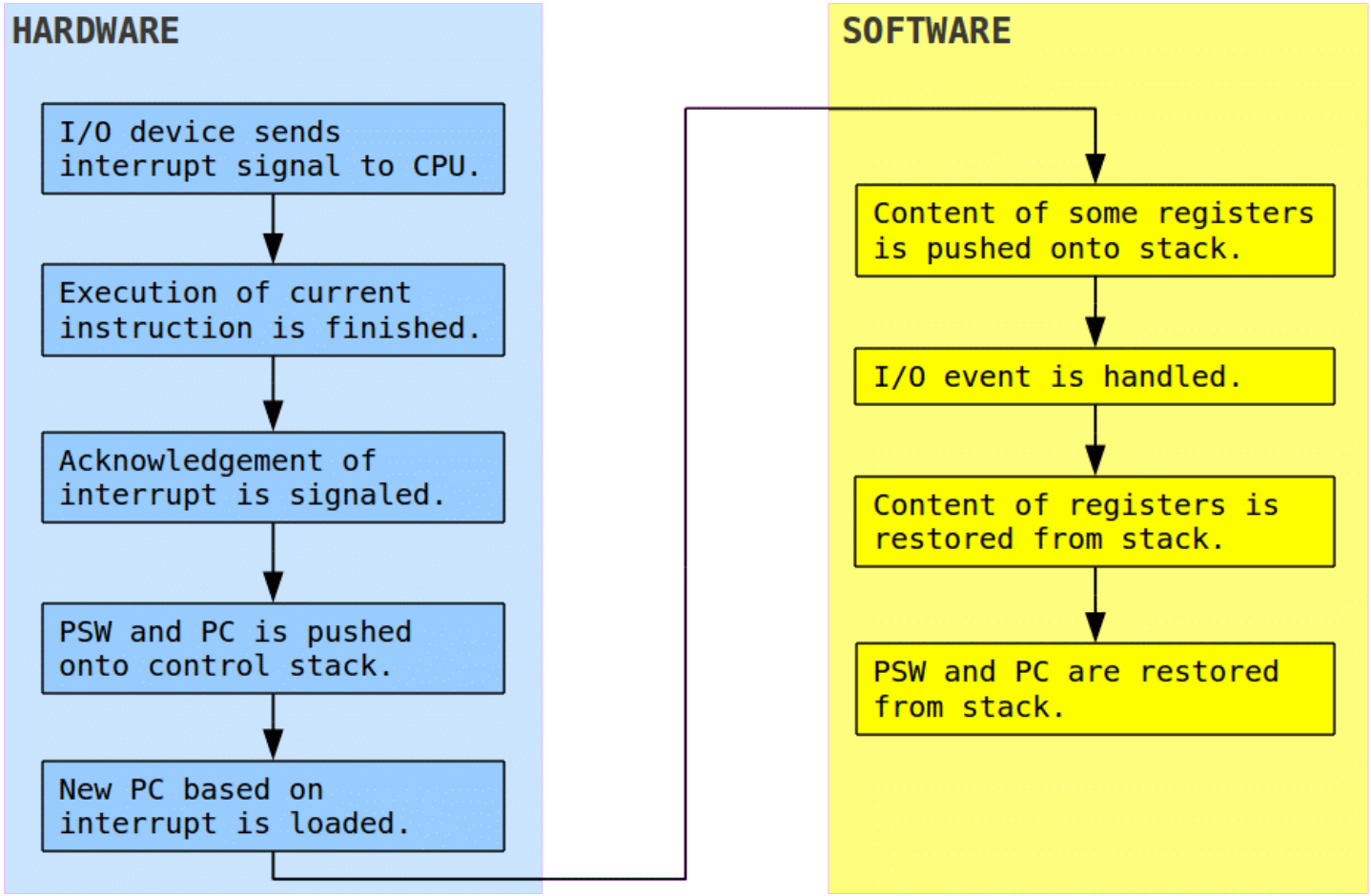
Figure 1.7 Instruction Cycle with Interrupts

Detecting Interrupts



Steps of Interrupt Handling

- Example: Handling of an I/O event

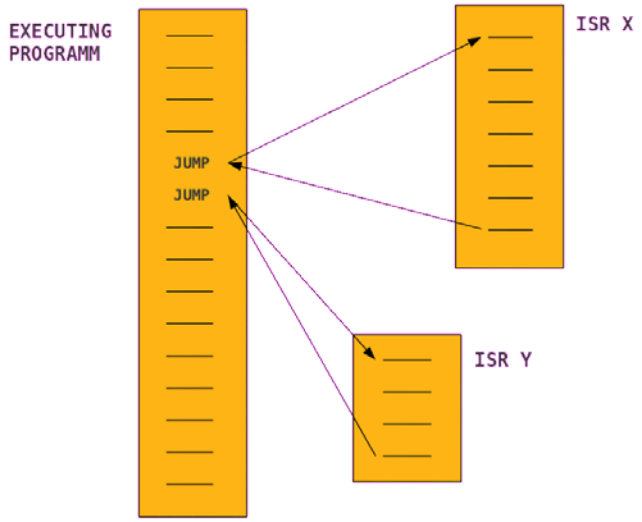


Types of Interrupts

- Hardware interrupts (asynchronous)
  - Triggered by hardware devices, e.g.,
    - Timer
    - I/O device
    - Printer
- Software interrupts (synchronous)
  - Triggered within a processor by executing an instruction
  - Often used to implement system calls
- Exceptions, e.g.,
  - Arithmetic overflow, division by zero
  - Illegal instruction
  - Illegal memory access

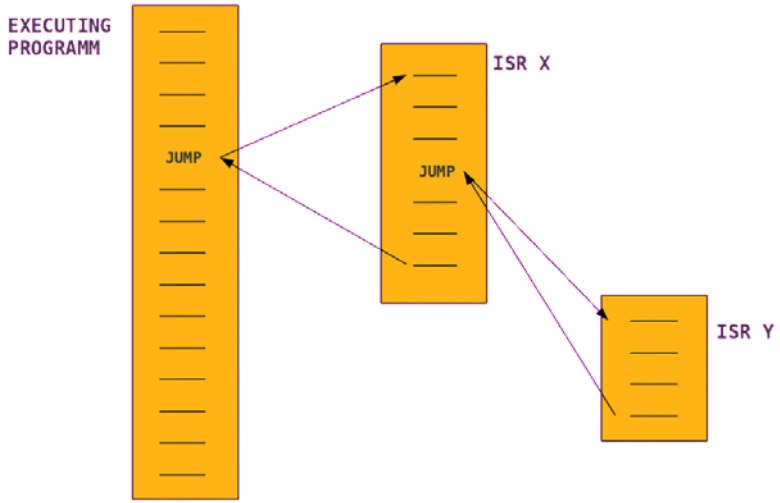
Multiple Interrupts

- **Sequential** interrupt processing:



- Delay of interrupt handling **unpredictable** under load

- **Nested** interrupt processing:

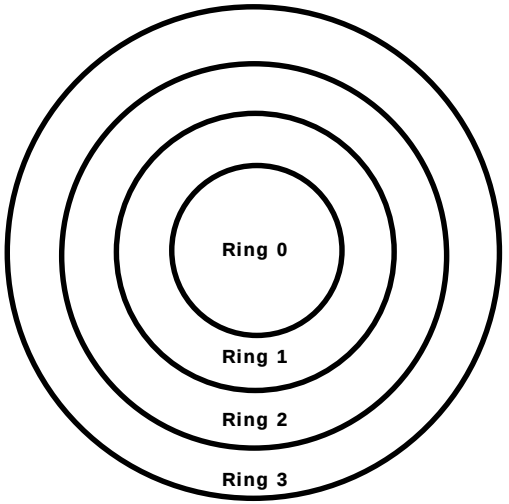


- Delay depends on interrupt priority level
- Highest priority guarantees **constant** delay
- Required for real-time applications

Content (1)

System Calls

- User applications access system services by calling **system call**s that are part of the **system interface**.



- Typical modes of execution:
  1. User mode (ring 3):
     - Typical mode for user processes
     - Limited access to hardware features
     - May request privileged services via system calls
  2. Kernel/privileged/system/control mode (ring 0):
     - Typical mode for kernel of operating system
     - Full access to hardware features
     - Memory access beyond own address space
     - Required for implementation of device drivers (low-level), scheduling, virtual memory

➢ Handling a user request within the kernel implies switching from user to kernel mode.

Context / Mode / Process Switch

"Context switch" may refer to:

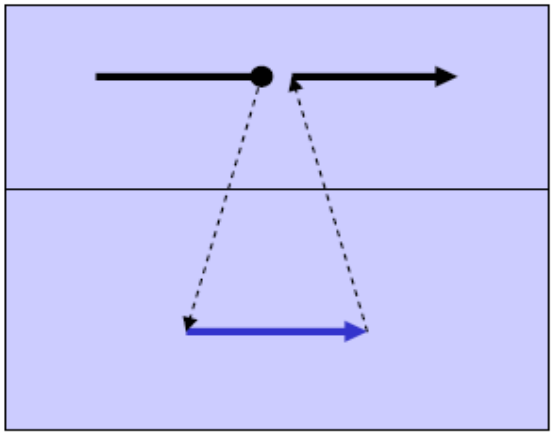
1. Mode switch between user and kernel mode
   - Short interruption of current process (e.g. while handling system call)
   - No modification of process state required

2. Process switch between different (user) processes
   - *May* occur (depending on scheduler) when flow of control moves from user process to operating system:
     - Interrupt: Response to external asynchronous event
       - Timer interrupt: periodic process switch
       - I/O interrupt: possibly event a process is waiting for
       - Memory fault: Loading of a swapped memory segment with interleaved execution of another process
     - Trap: Response to error caused by process
     - System call: Process requests OS service
   - More expensive than mode switch due to process state, processor caches, ...

Implementation of Syscalls (1)

- Subroutine call into operating system
  - Used in very simple operating systems without separate address spaces
    - No hardware-enforced security
    - ➢ User processes run with full access to hardware (ring 0)
  - Compiler / linker / loader insert call addresses into program
  - Interrupt handling terminates with a simple jump back into program (“RETI”)

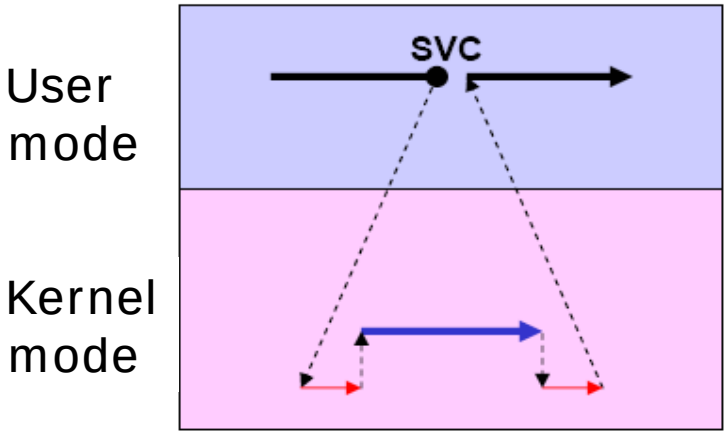
- Example:
  - MS-DOS & Embedded Systems



Implementation of Syscalls (2)

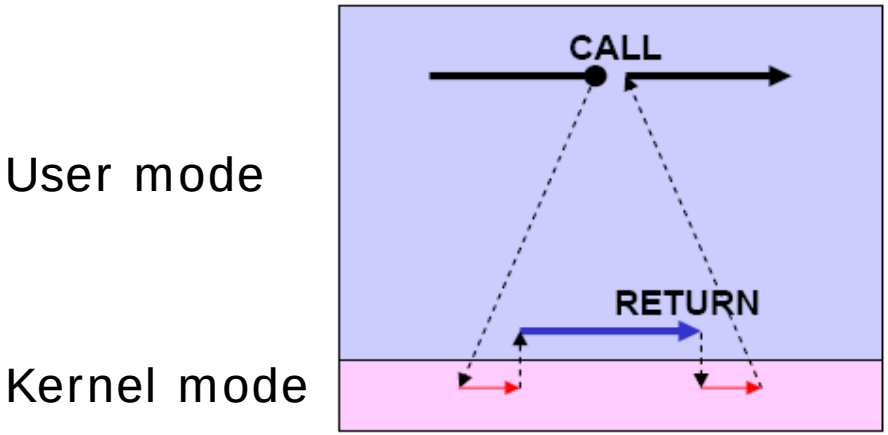
- Machine-level instruction “system call” (supervisor call, SVC)
  - Raises trap / exception / software interrupt
  - Interrupt service routine detects cause and branches into corresponding service routine
    - Possibly within same address space, so no process switch
  - Compiler inserts parameters for system calls
  - Terminates with return

- Example: most UNIX kernels



Implementation of Syscalls (3)

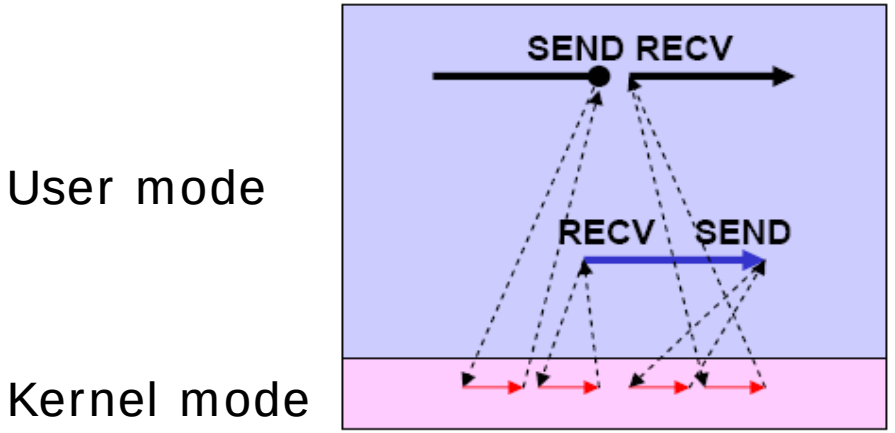
- Call of system module / object
  - Microkernel manages jump into address space of the corresponding system module in response to CALL alarm
    - Process switch required
  - Return into calling address space with RETURN



Implementation of Syscalls (4)

- Dispatching a task to a system process
  - Microkernel dispatches task to corresponding system process in response to SEND alarm
  - System process receives task with RECV
    - ➢ Process switch required
  - Same method used for delivering result

- Example: Mach, Minix



System Calls and System Library

- System library hides system calls from programmers
- Example **`write()`**:

```
        PUSH   EBX              ; EBX retten
        MOV    EBX,  8(ESP)     ; 1. Parameter
        MOV    ECX, 12(ESP)     ; 2. Parameter
        MOV    EDX, 16(ESP)     ; 3. Parameter
        MOV    EAX, 4           ; 4 steht für „write“
        INT    0x80             ; eigentlicher Systemaufruf
        JBE    DONE             ; kein Fehler
        NEG    EAX              ; Fehlercode
        MOV    errno, EAX
        MOV    EAX, -1
DONE:   POP    EBX              ; EBX wiederherstellen
        RET                     ; Rücksprung
```

- Value -1 in register EAX on error

System Calls and System Library

- Linux system calls

%ax	Name	Source	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int	-	-	-	-
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
3	sys_read	fs/read_write.c	unsigned int	char *	size_t	-	-
4	sys_write	fs/read_write.c	unsigned int	const char *	size_t	-	-
5	sys_open	fs/open.c	const char *	int	int	-	-
6	sys_close	fs/open.c	unsigned int	-	-	-	-
7	sys_waitpid	kernel/exit.c	pid_t	unsigned int *	int	-	-
8	sys_creat	fs/open.c	const char *	int	-	-	-
...							

System Calls and System Library

WRITE(2) Linux Programmer's Manual WRITE(2)

NAME

write - write to a file descriptor

SYNOPSIS

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

DESCRIPTION

write() writes up to count bytes from the buffer pointed buf to the file referred to by the file descriptor fd.

The number of bytes written may be less than count if, for example, there is insufficient space on the underlying physical medium, or the RLIMIT_FSIZE resource limit is encountered (see setrlimit(2)), or the call was interrupted by a signal handler after having written less than count bytes. (See also pipe(7).)

...

POSIX

- Portable Operating System Interface (POSIX)
- Standard for operating system API
  - IEEE 1003, ISO/IEC 9945
- Three main parts:
  - POSIX Kernel APIs (system call interface)
  - POSIX Commands and Utilities
  - POSIX Conformance Testing
- Operating system, that implement POSIX:
  - INTEGRITY, **Linux**, BSD/OS, A/UX, LynxOS, Mac OS X, MINIX, RTEMS, SkyOS, **Windows NT**

POSIX Versions

- POSIX.1, Core Services
  - (includes Standard ANSI C)
  - Process Creation and Control
  - Signals (IPC)
  - Floating Point Exceptions
  - Segmentation Violations (VMM)
  - Illegal Instructions
  - Bus Errors
  - Timers
  - File and Directory Operations
  - Pipes
  - C Library (Standard C)
  - I/O Port Interface Control
- POSIX.1b, Real-time extensions
  - Priority Scheduling
  - Real-Time Signals
  - Clocks and Timers
  - Semaphores
  - Message Passing
  - Shared Memory
  - Asynch and Synch I/O
  - Memory Locking
- POSIX.1c, Threads extensions
  - Thread Creation, Control, and Cleanup
  - Thread Scheduling
  - Thread Synchronization
  - Signal Handling

Content (1)