



# Operační systémy

Úvod

Strategický projekt UTB ve Zlíně, reg. č. CZ.02.2.69/0.0/0.0/16\_015/0002204



# Struktura počítačového systému

#### Uživatel

Lidé, stroje, jiné počítače

#### □ Aplikace (SW)

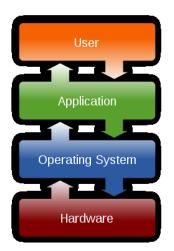
- Programy, které využívají systémové zdroje počítače k vyřešení zadání uživatelů
  - o Vývojové nástroje, kancelářské aplikace, webový prohlížeč, hry, ...

#### Operační systém (OS)

Řídí a koordinuje efektivní využití HW a poskytuje rozhraní pro SW.

#### □ Hardware (HW)

- Systémové zdroje
  - o CPU, operační paměť, úložiště, další vstupní a výstupní zařízení



(Silberschatz, Galvin & Gagne, 2013)

(Silberschatz, Galvin & Gagne, 2013; Staudek, 2013)

## Co znamená operační systém?

- □ V šedesátých letech byl definován jako SW, který řídí HW
  - Dnes je nutno použít obsáhlejší definici
  - Neexistuje žádná obecně platná

- Odděluje aplikace od hardwaru
- Jedná se o softwarovou vrstvu
  - Spravuje hardware
  - Poskytuje služby softwaru

(Deitel, Deitel & Choffness, 2004; Silberschatz, Galvin & Gagne, 2013)

## Definice operačního systému

- Operační systém je správce zdrojů
  - Spravuje všechny systémové zdroje (CPU, paměť, I/O zařízení, ...)
  - Řeší konflikty požadavků
    - o cílem je efektivní a spravedlivé využívání systémových zdrojů
- Poskytuje abstrakci systémových zdrojů
  - OS skrývá HW před SW (izolace) vytváří virtuální HW (Virtual Machine)
- Operační systém řídí provádění programů
  - Zabraňuje chybám a nesprávnému využití

(Deitel, Deitel & Choffness, 2004; Tanenbaum, 2015; Silberschatz, Galvin & Gagne, 2013)

## Proč studujeme operační systém?

- Jedná se o nejsložitější SW systémy
- Uplatňují se zde různé oblasti vědění
  - Algoritmizace, programování, správa zdrojů, souběžnost, krizové rozhodování
    - Vývoj lze následně uplatnit i v jiné oblasti.
- Příliš jednoduchý operační systém je nákladný
  - Nevyužívá systémové zdroje efektivně, neposkytuje vše dostupné

## Náš cíl v operačních systémech

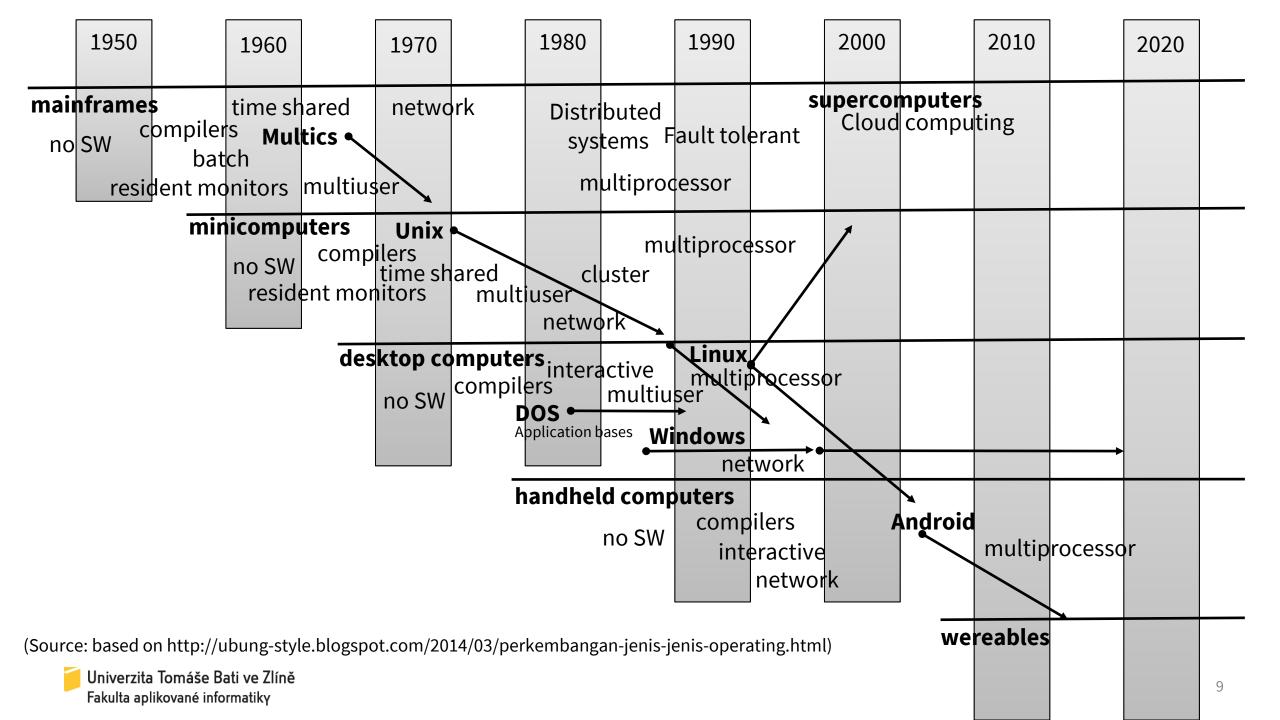
- Hlavním cílem je porozumět
  - Pochopit roli operačního systému
    - Co operační systém vlastně dělá
  - Pochopit jak operační systém pracuje
    - Design operačního systému
    - Principy funkce, služby, rozhraní
  - Naučit se používat operační systém
    - Microsoft Windows, Linux

#### Co všechno se naučíte

- Struktura operačního systému
  - Vnitřní návrh, procesy, vlákna, systémová volání
- Správa procesů
  - Plánování procesů a vláken
- Komunikace mezi procesy
  - Problémy synchronizace, mutexy, semafory, ...
- Správa paměti
  - Adresní prostor, virtuální paměť
- Úložiště
  - Souborové systémy

#### Historie

- □ 40. a 50. léta 20. století první generace
  - První počítače neměly operační systém, vykonávala se vždy jedna úloha.
     V 50. letech se začalo využívat dávkové zpracování.
- □ 60. léta druhá generace
  - Stále se používá dávkové zpracování, začíná multiprogramming a zpracování více úloh v čase, více uživatelů, interaktivita, virtuální paměť.
- □ 70. léta třetí generace
  - Rozvoj operačních systémů, multitasking, zpracování v reálném čase,
- 80. léta čtvrtá generace
  - Osobní počítače, GUI
- Pátá generace
  - Stále ještě čekáme



## Dělení operačních systémů

- Podle počtu procesorů
- Podle úrovně sdílení CPU
- Podle počtu uživatelů
- Podle způsobu nasazení/požadavků na odezvu
- Podle velikosti hardwaru
- □ Podle míry distribuovanosti
- Podle funkcí
- □ ...

## Cíle OS

- □ Efektivita
- Robustnost
- Škálovatelnost (nové zdroje)
- □ Rozšiřitelnost (nové technologie)
- □ Přenositelnost
- Bezpečnost
- □ Interaktivita
- Použitelnost



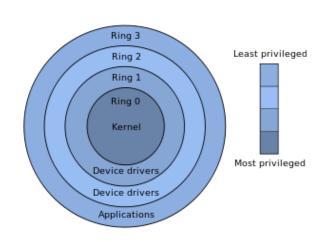
## **Části OS**

- Správa procesorů a procesů
- Správa hlavní paměti
  - Alokace a dealokace paměti podle potřeby
  - Udržuje informaci, která část paměti je používána a kým
- I/O podsystém, vstupy a výstupy
  - Společné rozhraní ovladačů zařízení, ovladače pro specifická zařízení
  - Rozvrh diskových operací
  - Souborový systém soubory a adresáře
- Uživatelské rozhranní
  - CLI (Command Line Interface)
  - GUI (Graphical User Interface)
- □ **Síť** (Network)

(Deitel, Deitel & Choffness, 2004; Silberschatz, Galvin & Gagne, 2013)

Linux kernel map functionalities human interface system processing memory storage networking layers HI char devices interfaces core processes memory access files & directories sockets access kernel/signal.c sys\_fork user sys\_brk access System Call Interface sys\_vfork cdev\_add sys\_mmap shm\_vm\_ops sys\_signal sys\_execve sys\_clone sys\_connect sys\_socket linux/syscalls.h space /proc /sysfs /dev do\_sigaction sys\_shmctl linux/uaccess.h sys\_shmat copy\_from\_user sysfs\_ops sys bind /proc/net/ interfaces linux\_binfmt sys\_listen sys\_futex sys\_mprotect tcp4\_seq\_show sys\_ioctl sys\_mincore cdev\_add
cdev\_map\_sys\_epoll\_create sg proc seg show dev system calls /dev/mem console\_fops sys\_times rt\_cache\_seq\_show and system files mem fons sys\_capset fb\_fops cdev sys\_reboot /proc/meminfo sys\_msync sys\_init\_module /proc/self/maps Virtual File System threads virtual memory protocol families **Device Model** security/ security vmalloc\_init INIT\_WORK | queue work | find vma prepare linux/kobject.h inet\_family\_ops security capset may open kobject work\_struct bus\_register kset linux/device.h virtual inet create inode security\_socket\_create unix\_family\_ops kthread create inode\_permission kernel thread security\_inode\_create device\_create security\_ops file system type do\_fork inet\_dgram\_ops inet\_stream\_ops virt\_to\_page super\_block generic file aid read do splice direct socket\_file\_ops driver\_register debugging synchronization device\_driver memory page cache socket networking mapping bridges log\_buf splice probe storage sock\_sendpage nfs\_file\_operations register\_kprobe do\_mmap\_pgoff cross-functional modules si\_swapinfo load module tcp\_sendpage kmem cache alloc smb fs type module kobject\_uevent\_init udp\_sendpage handle sysrg oprofile start vma\_link swap\_info kswapd wait\_event cifs file ops module\_param kobject uevent wake\_up spin\_lock\_irqsave spin\_unlock\_irqrestore sock\_splice\_read mm struct iscsi\_tcp\_transport tcp\_splice\_read kernel\_param vm\_area\_struct wakeup kswapd HI subsystems Scheduler logical memory logical protocols system run boot, shutdown file systems task struct \*tcp\_prot logical udp\_recvmsg udp\_sendmsg tcp\_v4\_rcv alsa ext4\_file\_operations schedule\_timeout\_ functions implementations kmalloc do initcalls video\_device kfree NE HOOK ip\_queue\_xmit ext4\_readdir run init process process timeout ip\_rcv sk buff **Page Allocator** abstract devices generic HW access interrupts core block devices network interface and request\irq request region **HID class drivers** dev\_queue\_xmit netif\_receive\_skb device block\_device\_operations pci register driver kmem cache drivers/input/ drivers/media/ \_\_free\_one\_page pci\_request\_regions kmem\_cache\_init kmem cache alloc do\_timer net\_device control scsi device console tick\_periodic get\_free\_pages scsi driver timer\_interrupt usb submit urb page \_\_alloc\_pages sd\_fops fb\_ops do\_IRQ\_irq\_desc usb hcd totalram\_pages try\_to\_free\_pages drm driver usb\_stor\_host\_template CPU specific HI peripherals device access physical memory disk controller network device drivers and bus drivers operations drivers device drivers x86\_init trap\_init hardware arch/x86/mm/ get\_page\_from\_freelist start thread native init\_IRQ switch\_to interfaces scsi\_host\_alloc ipw2100\_pci\_init\_one atkbd drv free\_area free\_list drivers, registers and interrupts e1000\_xmit\_frame i8042 driver Scsi\_Host out\_of\_memory die show\_regs ahci\_pci\_driver pci\_read pt\_regs atomic\_t num\_physpages do\_page\_fault pci write user peripherals I/O mem I/O **CPU** disk controllers network controllers memory electronics mouse graphics card audio

- Obecná architektura operačních systémů je budovaná s využitím dříve zmíněných základních komponent.
  - Základní rozdíly spočívají v tom, kde běží dané subsystémy a jak spolu navzájem komunikují, jaká mají oprávnění.
  - Pomáhá to zvládnout složitost OS.
- Systém běží ve dvou základních režimech
  - Uživatelský režim
  - Privilegovaný režim (kernel mode)



(Deitel, Deitel & Choffness, 2004; Silberschatz, Galvin & Gagne, 2013)

- Monolitická architektura
- Mikrokernel architektura
- Hybridní architektura
- Vrstvená architektura
- Virtuální stroj
- Model klient-server

(Deitel, Deitel & Choffness, 2004; Silberschatz, Galvin & Gagne, 2013, Lažanský, 2014)

## □ Monolitický operační systém (jádro)

- Všechny části OS jsou obsaženy v kernelu
  - o přímo komunikují s ostatními.
  - Mají stejná oprávnění.
  - Nízká režie, důraz na efektivitu.
  - o Možnost použití modulárního designu. Vše běží v paměťovém prostoru jádra.
- Kernel spuštěn s neomezeným přístupem k celému systému.
- Uživatelské programy mají minimální oprávnění.
- Většina starších operačních systémů (OS/360, VMS, Linux).

#### □ Monolitický operační systém

Memory management MM

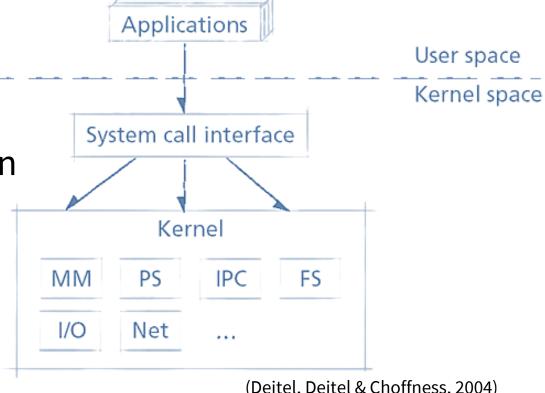
Process scheduler PS

IPC Interprocess communication

File systém FS

Input/output manager I/O

Network manager Net

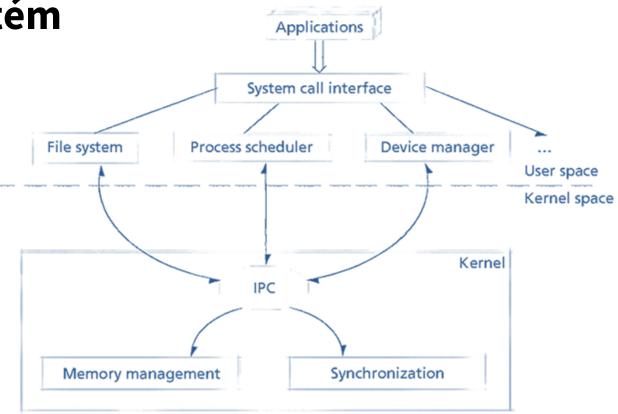


#### Mikrokernel operační systém

- Poskytuje základní služby v pokusu udržet jádro malé a škálovatelné.
  - o Základní synchronizace procesů potřebnou pro spolupráci mezi procesy
  - Nízko úrovňový paměťový management
  - Obsluha přerušení.
  - Komunikace mezi procesy
    - Při komunikaci je zajištěna autorizace.
- Další části OS spuštěny mimo jádro OS s menší úrovní práv
- OS Mach, GNU Hurd, MS Windows, ...

□ Mikrokernel operační systém

- Velká modularita
- Vyšší úroveň komunikace
  - Může degradovat výkon



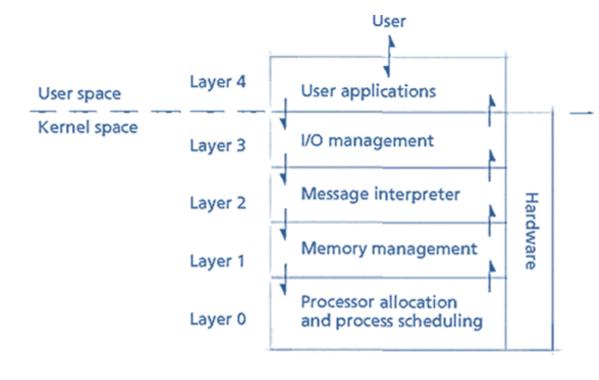
#### □ Hybridní operační systém

- Hybridní jádra jsou něco mezi monolitickým jádrem a mikrojádrem.
  - Hybridní jádro se snaží zkombinovat rychlost a jednoduchost designu monolitického jádra s bezpečnostními výhodami mikrojader.
  - Některé služby (jako souborový systém nebo implementace síťového protokolu) běží v jádře ke zredukování režie proti mikrojádrům, ale jiné části monolitického jádra (ovladače zařízení) běží jako server v uživatelském prostoru.
- MS WinNT

#### □ Vrstvený operační systém

- Vytvořena hierarchie procesů.
  - Nejníže položené vrstvy komunikují s hardwarem, každá další vyšší vrstva poskytuje abstraktnější virtuální stroj.
  - Jednotlivé vrstvy nelze obcházet, každá vrstva komunikuje výhradně se sousední vrstvou.
  - o Výhodou je možnost výstavby systému od nejnižších vrstev, modularita.
  - OS/2, OS "THE" (Technische Hogeschool Eindhoven)
  - Mnoho dnešních systémů obsahuje prvky vrstvené architektury

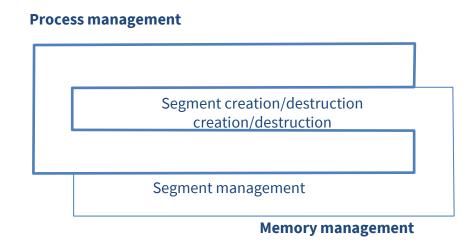
## □ Vrstvený operační systém



(Deitel, Deitel & Choffness, 2004)

#### □ Funkční hierarchie

- obtížné rozčlenit OS do striktní hierarchie
  - o vznikají závislosti





#### □ Virtuální stroj

- Logická struktura OS s principem vrstvení dotaženým do extrému
- Virtuální stroj je softwarový produkt, který chápe hardware a jádro operačního systému jako jednu společnou (hardwarovou) vrstvu
- Virtuální stroj vytváří duplikát původního hardware
- Současný běh více OS s vlastnostmi původního fyzického počítače
  - Každý uživatel na sdíleném stroji může tak užívat jiný OS
- Virtuální stroj zajišťuje úplnou ochranu systémových zdrojů
  - Každý virtuální stroj je izolován od ostatních

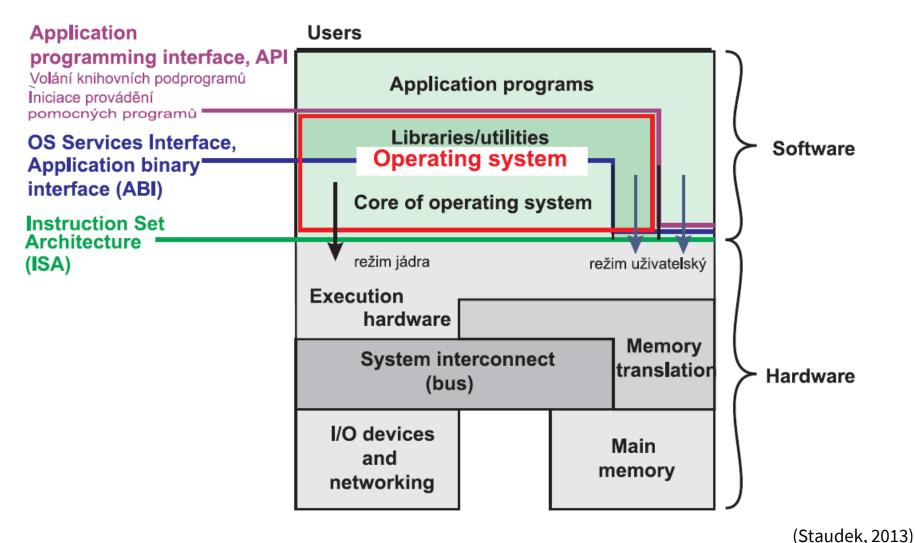
(Lažanský, 2014)

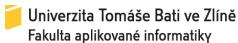
#### ■ Model klient-server

- Minimalizace rozsahu jádra (mikrojádro)
- Implementace funkcí OS jako uživatelský proces
- Požadavek na službu (např. čtení ze souboru) zasílá uživatelský proces (klient) na obsluhu souborů (file server).
  - Mikrojádro zajišťuje zejména komunikaci
- Vhodné pro distribuované OS

(Lažanský, 2014)

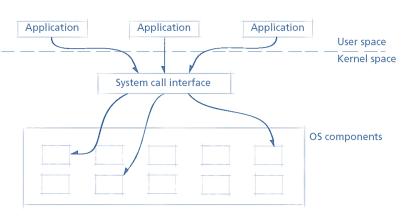
#### Struktura HW a SW





# Základ běhu aplikací

- Operační systém poskytuje API
  - Application Programming Interface
  - Specifikuje sadu funkcí, které jsou dostupné aplikačnímu programátorovi.
    - Programátor využívá knihovny pro přístup k API
  - Zjednodušuje přístup k HW pomocí abstrakce
    - o Zabraňuje přístupu k HW a poskytuje virtuální stroj
  - API poskytuje systémová volání
    - o Aplikace žádá operační systém o provedení akce.





## Příklad API

□ read() funkce popsaná v manuálových stránkách (man read)

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count)

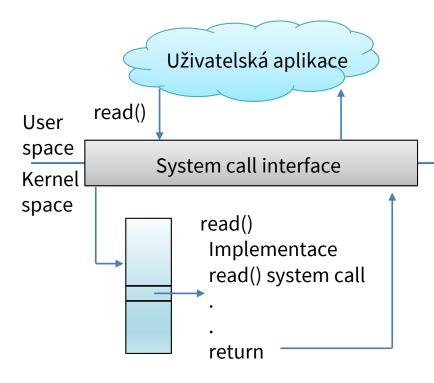
return function parameters
value name
```

- Funkce je deklarována v hlavičkovém souboru *unistd.h* 
  - Vstupní parametry
    - int fd deskriptor souboru; void \*buf buffer pro načtení dat; size\_t count max. velikost
  - Návratová hodnota
    - Nula znamená konec souboru, při chybě čtení vrací -1.

(Silberschatz, Galvin & Gagne, 2013)

# Systémové volání (System Call)

- Metody předání vstupních parametrů operačnímu systému
  - Nejjednodušší přístup je předat parametry pomocí registrů
    - o Problém s počtem a případně velikostí registrů
  - Parametry jsou uloženy v bloku a tento blok paměti je předán pomocí registru
  - Parametry jsou umístěny do zásobníku (stack), kde si je operační systém vyzvedne.



(Silberschatz, Galvin & Gagne, 2013)

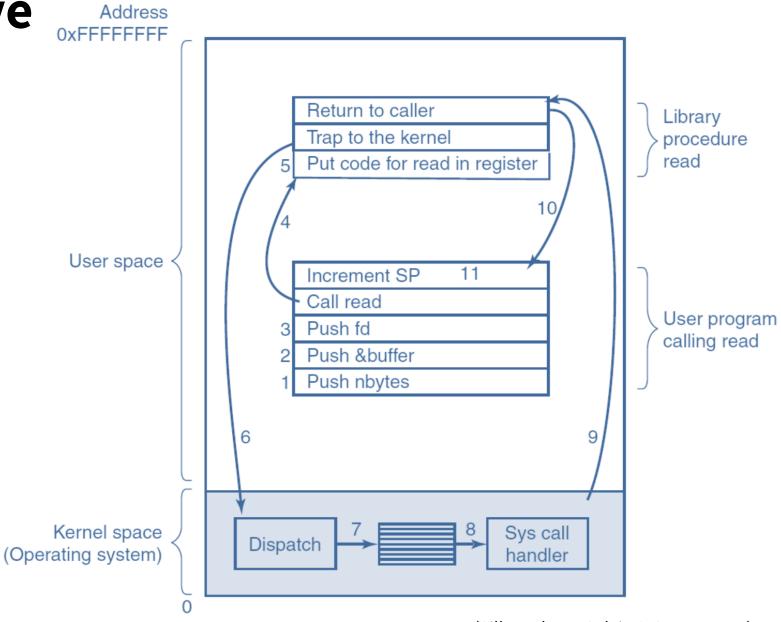
(Silberschatz, Galvin & Gagne, 2013)

## Zpracování systémového volání read()

- Vložení parametrů do zásobníku (1-3)
- Volání knihovní funkce (4)
- Vložení čísla systémového volání do registru (5)
- Provedení TRAP instrukce na přepnutí z uživatelského módu do kernel módu a začátek provádění kódu na fixní adrese v kernelu (6)
- Odeslání do odpovídající obsluhy systémového volání (7)
- Běh obsluhy systémového volání (8)
- Po dokončení je řízení předáno knihovně v uživatelském prostoru
   (9)
- Návrat do uživatelského programu, pomocí return (10)
- Vyčištění zásobníku po volání (11)

(Silberschatz, Galvin & Gagne, 2013)

# Systémové volání



## **Application Binary Interface – ABI**

- Definuje rozhraní na úrovni strojového kódu
  - V jakých registrech se předávají parametry
  - V jakém stavu je zásobník
  - Zarovnání vícebytových hodnot v paměti
- ABI se liší nejen mezi OS, ale i mezi procesorovými architekturami stejného OS.
  - Např: Linux i386, amd64, arm,...
  - Možnost podpory více ABI: int0x80, sysenter, 32/64bit



## Použitá a doporučená literatura

- □ DEITEL H. M., DEITEL P. J. & CHOFFNES D. R.: *Operating systems*. 3<sup>rd</sup> ed., Pearson/Prentice Hall, 2004. ISBN 0131246968.
- □ TANENBAUM A. S.: *Modern operating systems*. 4<sup>th</sup> ed. Boston: Pearson, 2015. ISBN 0-13-359162-x.
- SILBERSCHATZ A., GALVIN P. B. & GAGNE G.: Operating system concepts. 9<sup>th</sup> ed. Hoboken, NJ: Wiley, 2013. ISBN 978-1-118-06333-0.
- □ STALLINGS W.: Operating Systems: Internals and Design Principles. 8<sup>th</sup> ed., Pearson Education Limited, 2014.

## Použitá a doporučená literatura

- Štěpán Petr. Operační systémy. Přednášky FEL ČVUT v Praze, 2018.
- □ Lažanský Jiří. *Operační systémy a databáze*. Přednášky FELK ČVUT v Praze, 2014.
- Staudek Jan. Operační systémy. Přednášky FI MUNI, 2013.
- Vojnar Tomáš. Operační systémy. Přednášky FIT VUT v Brně, 2011.
- Klimeš Cyril. Principy výstavby počítačů a operačních systémů.
   On-line: https://publi.cz/books/11/Cover.html