

Základy operačních systémů

I.

KIV/ZOS 2014

Kontaktní informace

- Ing. Ladislav Pešička
- UN 358
- *pesicka@kiv.zcu.cz*
 - Předmět zprávy začít: ZOS
- Úřední hodiny
 - St 13:30 až 14:30
 - Pá 10:00 až 11:00
- Všechny informace najdete v coursewaru

Požadavky na zápočet

- 2 zápočtové testy
- Semestrální práce
 - program + dokumentace

1. zápočtový test

- časově koncem října / začátek listopadu
- napsat složitější script v /bin/bash
- a teoretická otázka
- Např:

```
skript -p1 s1.txt
```

```
skript -p2
```

```
skript -p3 > vys.txt
```

0x00000000, 0x8016a950, 0x00000001, 0x0000000005)	
HANDLEP*** Address 8016a950 has base at 80106000	
.6.2 irql:if	SYSVER 0xf0000565
Name	Dll Base DateStamp - Name
ntoskrnl.exe	80010000 33247f80 ntkrnlmp.exe
atapi.sys	80007000 33248040 atapi.sys
Disk.sys	801db000 33601540 DISK2.SY
Ntfs.sys	80237000 344eeb40 ntfs.dll
NTice.sys	f1f48000 31ec6c6d floppy.SY
Cdrom.SYS	f228e000 31ec6c90 null.SYS
KSecDD.SYS	f2290000 33580000 ksecdd.SYS
win32k.sys	fe0c2000 34000000 win32k.dll
Cdfm.SYS	fdcda2000 34000000 cdfm.dll
nbf.sys	fdc35000 34000000 nbf.dll
netbt.sys	f1f68000 34000000 netbt.dll
afd.sys	f2008000 34000000 afd.dll
Parport.SYS	fdc14000 34000000 parport.dll
Wlan.sys	fidd0000 34000000 wlan.dll



2. zápočtový test

- teoretický
- časově začátek prosince
- otázky z přednášek
- řešení příkladů podobných těm na cvičení

ZOS cvičení

■ Základy Linuxu

- distribuce, jádro, struktura
- uživatelské ovládání
- příkazy, spojování příkazů
- příkazové skripty

■ Paralelní procesy, souběhy a ošetření

- ošetření kritické sekce, uvíznutí, ...
- reálná implementace Java, C, ..

■ Témata z přednášek

Zkouška

■ Písemný test

- Test na 60 min. bez pomůcek
- zvolit správnou odpověď, odpovědět na otázku, doplnit či nakreslit diagram atd..
- Ústní pohovor nad písemkou

ZOS

■ Obecné principy OS

- Není zaměřen na 1 systém, vychází z Unixu
- Není hodnocením, který systém je lepší

■ KIV/OS, KIV/PPR

- Pokračováním, Unix / Linux, paralelizace

■ Praxe

- Základy práce s Linuxem
- Práce se sdílenými zdroji, ošetření kritické sekce

ZOS přednášky = struktura OS (!)

■ modul pro správu procesů

- program, proces, vlákno, plánování procesů a vláken
- kritická sekce, synchronizace (semafory, ...)
- deadlock, vyhladovění

■ modul pro správu paměti

- virtuální paměť: stránkování, segmentace

■ modul pro správu I/O

■ modul pro správu souborů

■ síťování

■ bezpečnost

■ Kde všude můžete nalézt OS?

Ukázky zařízení
(s využitím materiálu
Introduction to embedded systems)



zubní kartáček

CPU: 8-bit

- řízení rychlosti
- časovač
- nabíjení

OS? NE

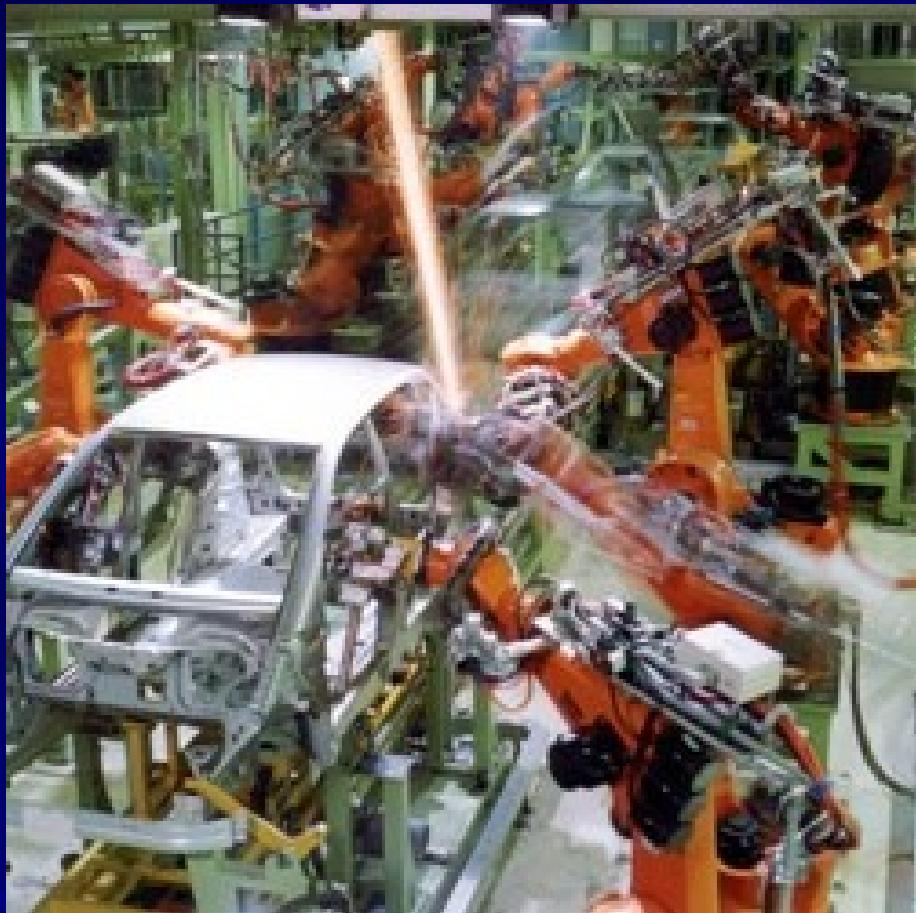


Prodejní terminál

Point-of-Sale (POS) Terminal

Microprocessor:
Intel X86 Celeron

OS: Windows XP Embedded



Kuka robot arms welding a Mercedes

Svařovací robot

Microprocessor: X86

OS: Windows CE OS &
Others

realtimový OS
správný výsledek v
požadovaném čase

OS v běžném životě - mobily

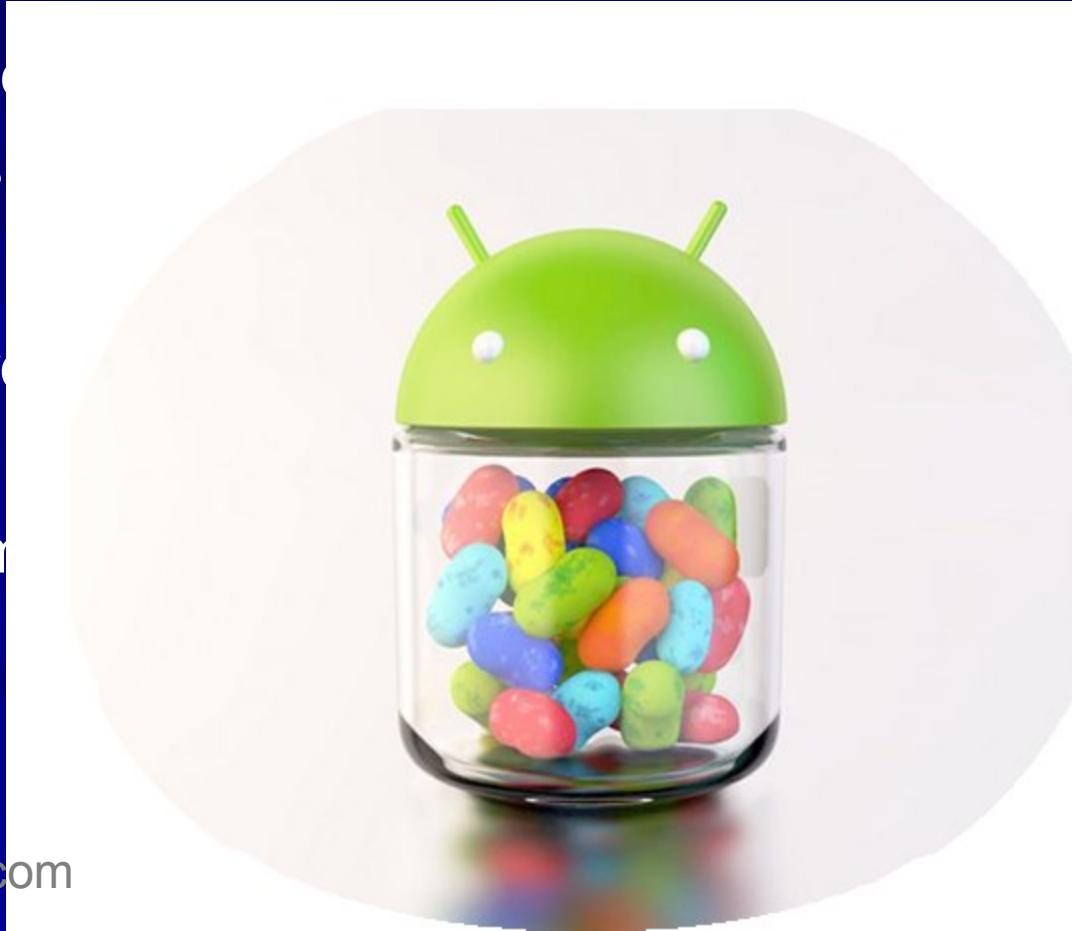
■ iOS 7.0, 8.0

- Apple iPhone, iPad
- Výrazná změna vzhledu
- Většina zařízení poskytuje

■ Android

- Poslední verze Androidu
- Na Linuxovém jádru
- Roztríštěnost verzí mimo

■ Windows Phone 8.1



OS ve vesmíru



NASA's Twin Mars Rovers.

Microprocessor:
Radiation Hardened
20Mhz PowerPC

Commercial Real-time OS

Software and OS was
developed during multi-year
flight to Mars
and **downloaded** using a
radio link

OS – příklady použití

- Servery, pracovní stanice, notebooky
 - MS Windows, GNU/Linux, Solaris
- Mobilní zařízení, tablety
 - Windows CE, Symbian, Linux, Android, ...
- Routery, AP, SOHO síťová zařízení
 - Cisco IOS, Linux, VxWorks
- Embedded zařízení
 - Bankomaty, stravovací systémy, lékařské přístroje
 - Windows CE, Windows XP embedded, Linux

Co všechno tvoří OS?

- Není všeobecná definice
- Vše co dodavatel poskytuje jako OS ?
 - Windows – kalkulačka, hra miny, malování, ...
- Program, běžící po celou dobu běhu systému ?
 - ale vLinuxu moduly zaváděné na žádost v případě potřeby
- SLOC (Source lines of code)

– Windows XP:	40 milionů řádků
– Linux kernel 3.10	16,9 mil. ř.
– Distribuce Debian 4.0	283 mil. ř.

Operační Systém - definice

OS je softwarová vrstva (základní programové vybavení), jejíž úlohou je spravovat hardware a poskytovat k němu programům jednotné rozhraní

- OS zprostředkovává aplikacím přístup k hardwaru
- OS koordinuje a poskytuje služby aplikacím
 - Analogie – dopravní systém, vláda, ..
- OS je program, který slouží jako prostředník mezi aplikacemi a hardwarem počítače.

Privilegovaný a uživatelský režim

■ Jádro OS běží v tzv. privilegovaném režimu

- Všechny instrukce CPU povoleny
- *Privileg. režim není např. v MS DOS*
- *Někdy část OS v uživatelském režimu (FUSE)*
- *Interpretované systémy (JVM)*

CPU ví v
jakém
režimu se
nachází

■ Aplikace – v uživatelském režimu

- Některé instrukce zakázány (tzv. privilegované instrukce)
např. přímý přístup k disku, jeho zformátování zákeřnou aplikací
- Aplikace musí požádat OS o přístup k souboru, ten rozhodne zda jej povolí

aplikace nemá
přímý přístup k
HW

■ OS může zasahovat do běhu aplikací (např. ukončit je)

■ Aplikace může požádat OS o službu

Jak CPU ví, v jakém je režimu?

- obecně
 - podle bitu ve stavové registru CPU
 - mode bit 0/1: privilegovaný/uživatelský
- konkrétněji
 - na kódový segment odkazuje CS registr, ten má v deskriptoru privilege level (2bity, ring 0..3)

<http://stackoverflow.com/questions/5223813/how-does-the-kernel-know-if-the-cpu-is-in-user-mode-or-kenel-mode>

OS

Dva základní pohledy na OS:

- **Rozšířený stroj** (shora dolů)
- **Správce zdrojů** (zdola nahoru)

OS jako rozšířený stroj

■ Holý počítač

- Primitivní a obtížně programovatelný (I/O)
- Např. disky ...
 - Práce s hlavičkou disku
 - Alokace dealokace bloků dat
 - Víc programů chce sdílet stejné médium

■ Jako programátor chceme

- Jednoduchý pohled – pojmenované soubory
- OS skrývá před aplikacemi podrobnosti o HW (přerušení, správu paměti..)

OS jako rozšířený stroj

- Strojové instrukce (holý stroj)
- Vysokoúrovňové služby (rozšířené instrukce)
 - Systémová volání
- Z pohledu programátora
 - Pojmenované soubory
 - Neomezená paměť
 - Transparentní I/O operace
- ZOS zkoumá, jaké služby a jak jsou v OS implementovány

OS jako správce zdrojů

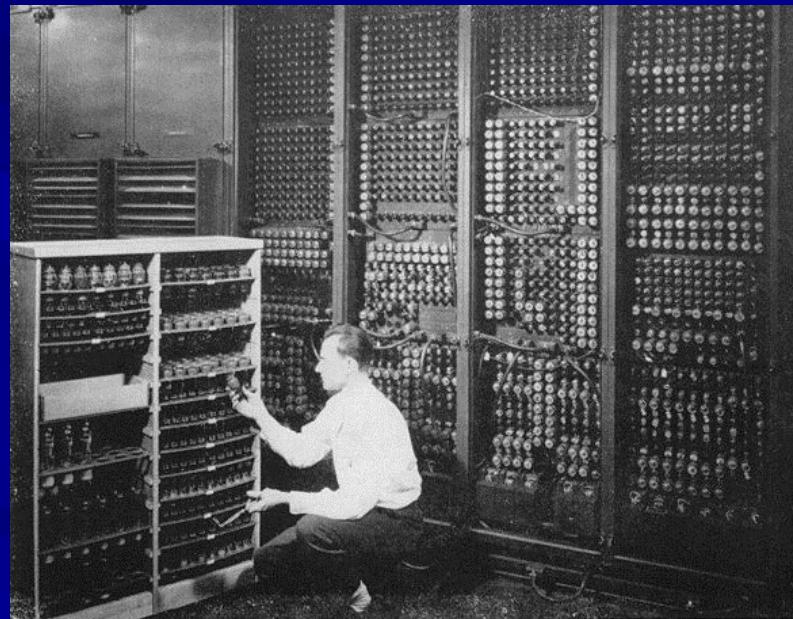
- OS jako poskytovatel / správce zdrojů (resource manager)
- Různé zdroje (čas CPU, paměť, I/O zařízení)
- OS – správná a řízená **alokace** zdrojů procesům, které je požadují (přístupová práva)
- **Konfliktní požadavky na zdroje**
 - V jakém pořadí vyřízeny
 - Efektivnost, spravedlivost

Historický vývoj

■ Vývoj hw -> vývoj OS

■ 1. počítač – ENIAC, 15.2.1946

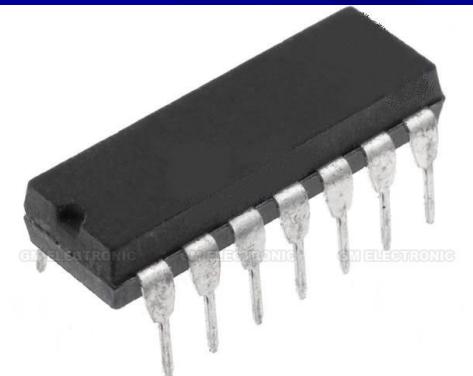
- Tělocvična
- 18 000 elektronek
- Regály, chlazení
- 5000 operací/s



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Generace počítačů

1. Elektronky
2. Tranzistory
3. Integrované obvody
4. LSI, VLSI (mikroprocesory,...)



1. Generace (1945-55)

- Elektronky, propojovací desky
- Programování
 - V absolutním jazyce
 - Propojování zdířek na desce
 - Později děrné štítky, assembly, knihovny, FORTRAN
 - Numerické kalkulace
- Způsob práce
 - **Stejní lidé** – stroj navrhli, postavili, programovali !
 - Zatrhnout blok času na rozvrhu, doufat, že to vyjde
- OS ještě neexistují

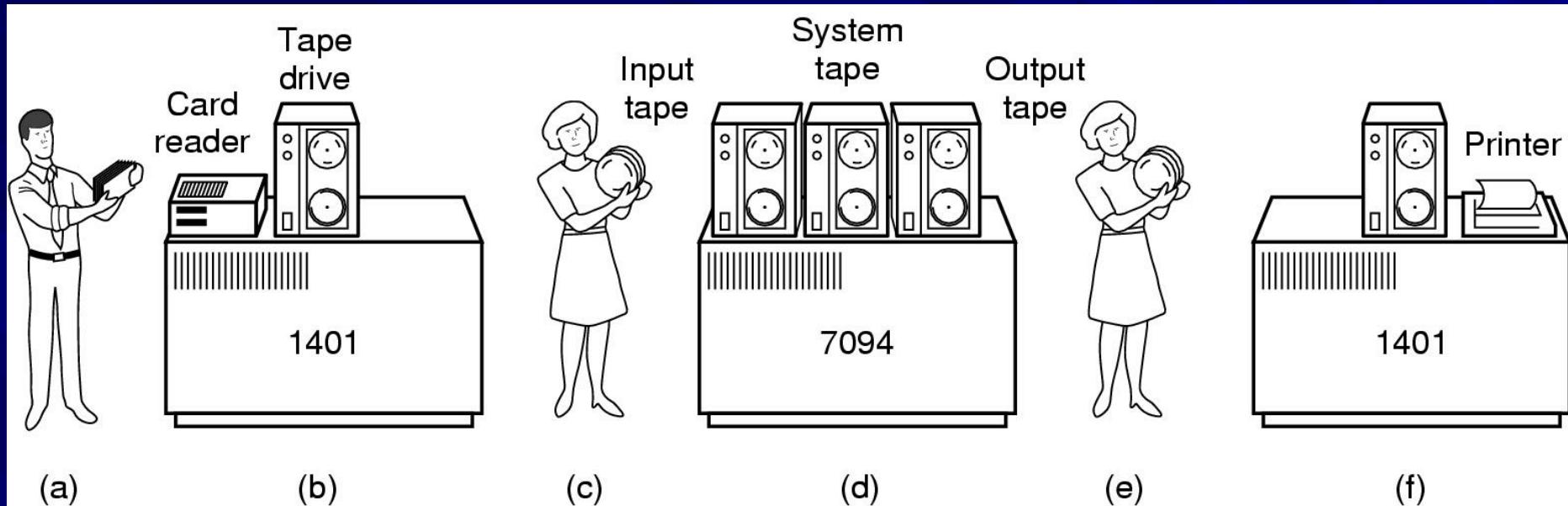
2. Generace (1955-65)

- Tranzistory, dávkové OS
- Vyšší spolehlivost; klimatizované sály
- Oddělení návrhářů, výroby, operátorů, programátorů, údržby
- Mil \$ - velké firmy, vlády, univerzity
- Způsob práce
 - Vyděrovat štítky s programem
 - Krabici dát operátorovi
 - Výsledek vytisknut na tiskárně
- Optimalizace
 - Na levném stroji štítky přenést na magnetickou pásku

2. generace

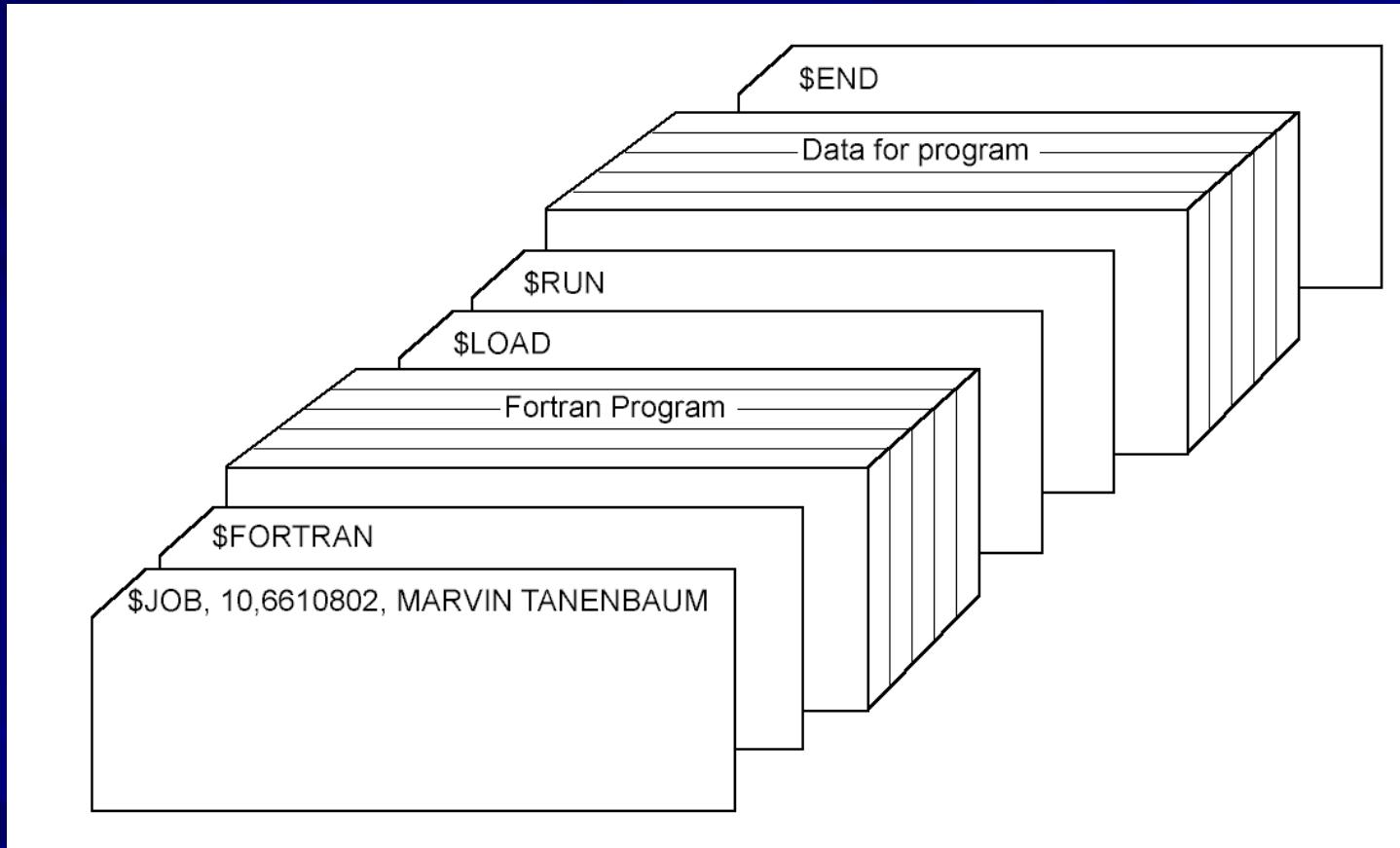
- Sekvenční vykonávání dávek
- Ochrana systému – kdokoliv dokázal shodit
- OS IBSYS = IBM SYSTÉM FOR 7094
- Pokud úloha prováděla I/O, CPU čekal..
 - Čas CPU je drahý
- Viz slidy Tanenbaum

Dávkové systémy



- vezmi štítky k 1401
- štítky se zkopírují na pásek (tape)
- pásek na 7094, provádí výpočet
- output tape na 1401 vytiskne výstup

History of Operating Systems



■ Struktura typické dávky – 2nd generation

3. Generace (1965-80)

- Integrované obvody, multiprogramování
- Do té doby 2 řady počítačů
 - Vědecké výpočty
 - Komerční stroje – banky, pojišťovny
- IBM 360 – sjednocení
 - Malé i velké stroje
 - Komplexnost – spousta chyb

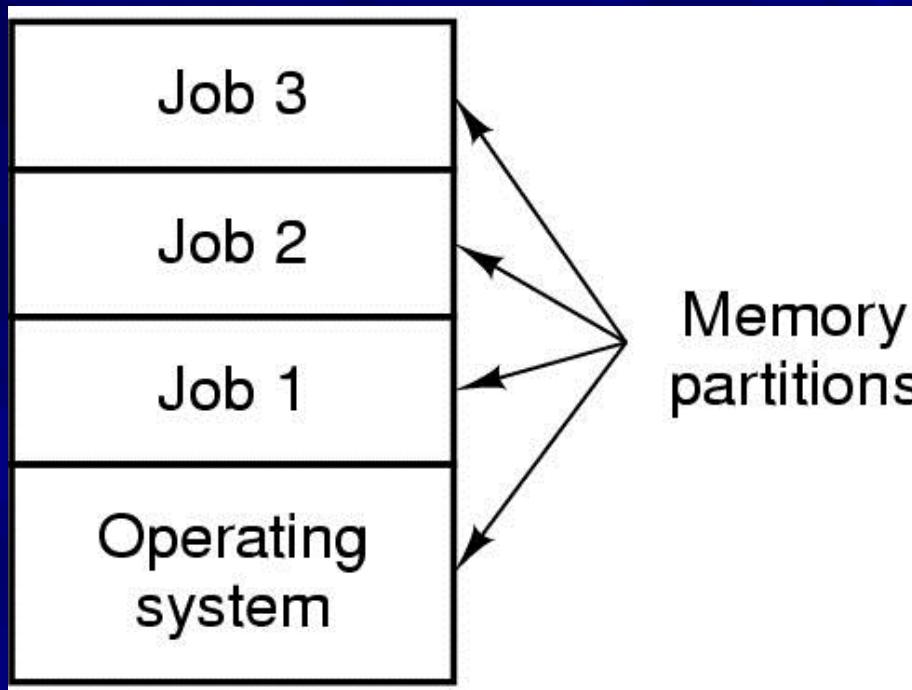
3. generace

■ Multiprogramování

- Doba čekání na I/O neefektivní
(věda OK, banky 80-90% čekání)
- Více úloh v paměti
 - Napřed konstantní počet
 - HW pro ochranu paměti

■ Každá úloha ve vlastní oblasti paměti; zatímco jedna provádí I/O, druhá počítá ...

History of Operating Systems



- Multiprogramming system
 - three jobs in memory – 3rd generation

3. generace

■ Spooling

- Na vstupu – ze štítků na disk, úloha se zavede z disku
- Na výstupu – výsledky na disk před výtiskem na tiskárně

spooling se dnes používá typicky pro sdílení tiskárny mezi uživateli
uživatel svůj požadavek vloží do tiskové fronty
až je tiskárna volná, speciální proces vezme požadavek z fronty a
vytiskne jej

■ Stále dávkové systémy

- Dodání úlohy, výsledek – několik hodin

3. generace

- Systémy se sdílením času
(time shared system)
 - Varianta multiprogramování
 - CPU střídavě vykonává úlohy
 - Každý uživatel má on-line terminál
- CTSS (MIT 1962) *Compatible Time Sharing Sys.*
- MULTICS

Minipočítače

- DEC PDP (1961)
 - Cca 3.5 mil Kč , „jako housky“
 - Až PDP11 – nekompatibilní navzájem
- Výzkumník Bell Labs pracující na MULTICSu
Ken Thompson – našel nepoužívanou PDP-7, napsal omezenou jednouživat. verzi MULTICSu vznik UNIXu a jazyka C (1969)

4. Generace (1980)

- Mikroprocesory, PC
- GUI x CLI
- Síťové a distribuované systémy
- MS DOS, Unix, Windows NT
- UNIX – dominantní na nonIntel;
 - Linux, BSD – rozšíření i na PC
 - Výzkum Xerox PARC – vznik GUI
 - Apple Macintosh
- film „Piráti ze Silicon Valley“

Dělení OS

Dle úrovně sdílení CPU:

- Jednoprocesový
 - MS DOS, v daném čase v paměti aktivní 1 program
- Multiprocesový
 - Efektivnost využití zdrojů
 - Práce více uživatelů

Dělení OS

Dle typu interakce:

■ Dávkový systém

- Sekvenční dávky, není interakce
- i dnes má smysl, viz. meta.cesnet.cz

■ Interaktivní

- Interakce uživatel – úloha
- Víceprocesové – interakce max. do několika sekund (Win, Linux, ..)

OS reálného času (!)

- Výsledek má smysl, pouze pokud je získán v nějakém omezeném čase
- Přísné požadavky aplikací na čas odpovědi
 - Řídící počítače, multimedia
- Časově ohraničené požadavky na odpověď
 - Řízení válcovny plechu, výtahu mrakodrapu ☺
- Nejlepší snaha systému
 - Multimedia, virtuální realita
- Př: RTLinux, RTX Windows, VxWorks

Hard realtime OS

- Zaručena odezva v **ohraničeném** čase
- Všechna zpoždění a režie systému ohraničeny

Omezení na OS:

- Často není systém souborů
 - Není virtuální paměť
 - Nelze zároveň sdílení času
-
- Řízení výroby, robotika, telekomunikace

Soft realtime OS

- Priorita RT úloh před ostatními
- Nezaručuje odezvu v daném čase
- Lze v systémech sdílení času
- RT Linux
- Multimédia, virtuální realita

Další dělení OS

■ Dle velikosti HW

- Superpočítáč, telefon, čipová karta

■ Míra distribuovanosti

- Klasické - centralizované 1 a více CPU

- Paralelní

- Síťové

- Distribuované

 - virtuální uniprocesor

 - Uživatel neví kde běží programy, kde jsou soubory

Další dělení OS

- Podle počtu uživatelů
 - Jedno a víceuživatelské

- Podle funkcí
 - Univerzální
 - Specializované (např. Cisco IOS)

Základní funkce operačního systému (!)

- správa procesů
- správa paměti
- správa souborů
- správa zařízení - I/O subsystém
- síťování (networking)
- ochrana a bezpečnost
- uživatelské rozhraní

Správa procesů

■ Program

- Spustitelný kód, v binární podobě
- Nejčastěji uložený na disku
- Např. C:\windows\system32\calc.exe

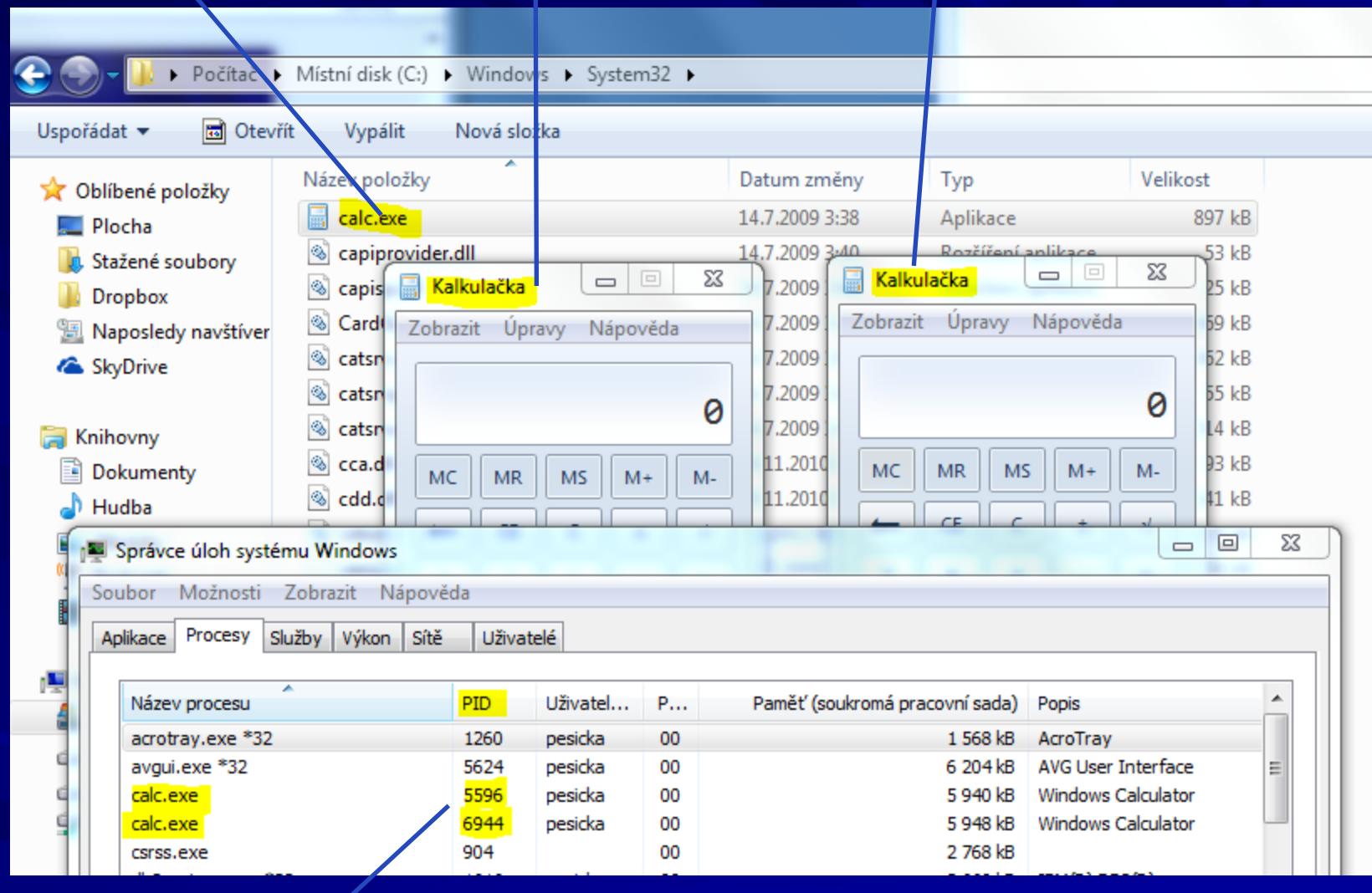
■ proces – instance běžícího programu

- Přidělen čas CPU
- Potřebuje paměťový prostor
- vstupy a výstupy
- *Dle jednoho programu můžeme spustit více procesů*

PROGRAM

PROCES

PROCES



PID (ID procesu) – základní identifikátor procesu !!

Správa paměti

■ správa hlavní paměti

- Přidělování paměti procesům
 - alokace / dealokace paměti dle potřeby
 - Virtuální adresování (stránkování, segmentace)
- Správa informace o volné a obsazené paměti
 - Která část paměti je volná, která obsazená a kým
- Odebírání paměti skončenému procesu
- Ochrana paměti
 - Přístup pouze pro oprávněné procesy

Soubory

■ soubory

- vytváření a rušení souborů
- vytváření a rušení adresářů
- primitiva pro manipulaci

■ se soubory

■ s adresáři

- správa volného prostoru vnější paměti
- mapování souborů na vnější paměť
- rozvrhování diskových operací

I/O subsystém

■ I/O subsystém

- správa paměti pro buffering, caching, spooling
- společné rozhraní ovladačů zařízení
- ovladače pro specifická zařízení

ovladače – kámen úrazu každého OS

Ochrana a bezpečnost

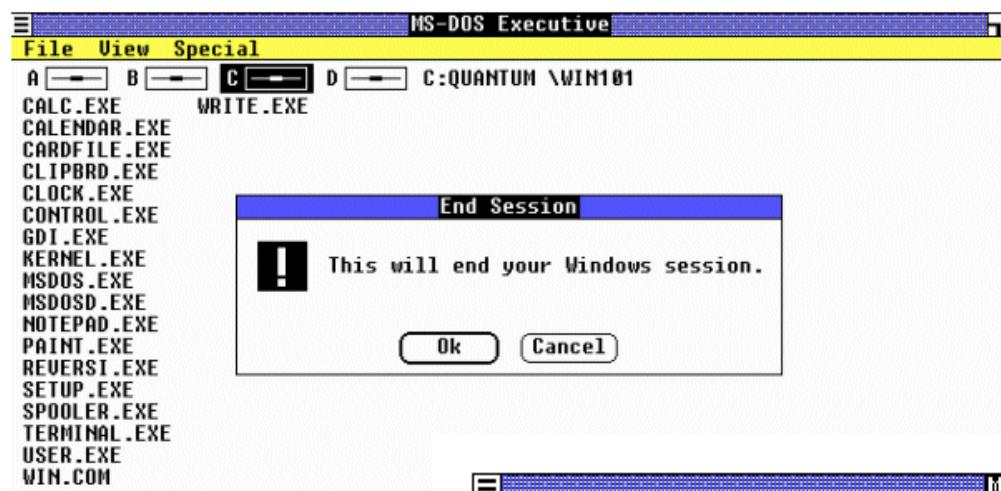
■ ochrana a bezpečnost

- ke zdrojům smí přistupovat pouze **autorizované procesy**
- specifikace přístupu
- mechanismus ochrany (souborů, paměti)

ACL, capabilities, ...

Uživatelské rozhraní

- uživatelské rozhraní
- CLI (command line interface)
- GUI (graphical user interface)
 - ukázky z www.zive.cz

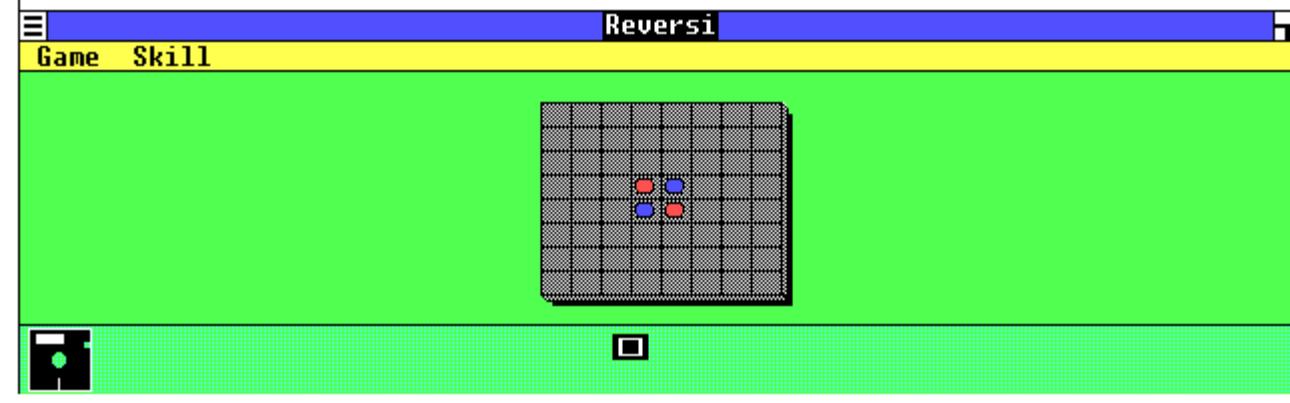


MS-DOS Executive

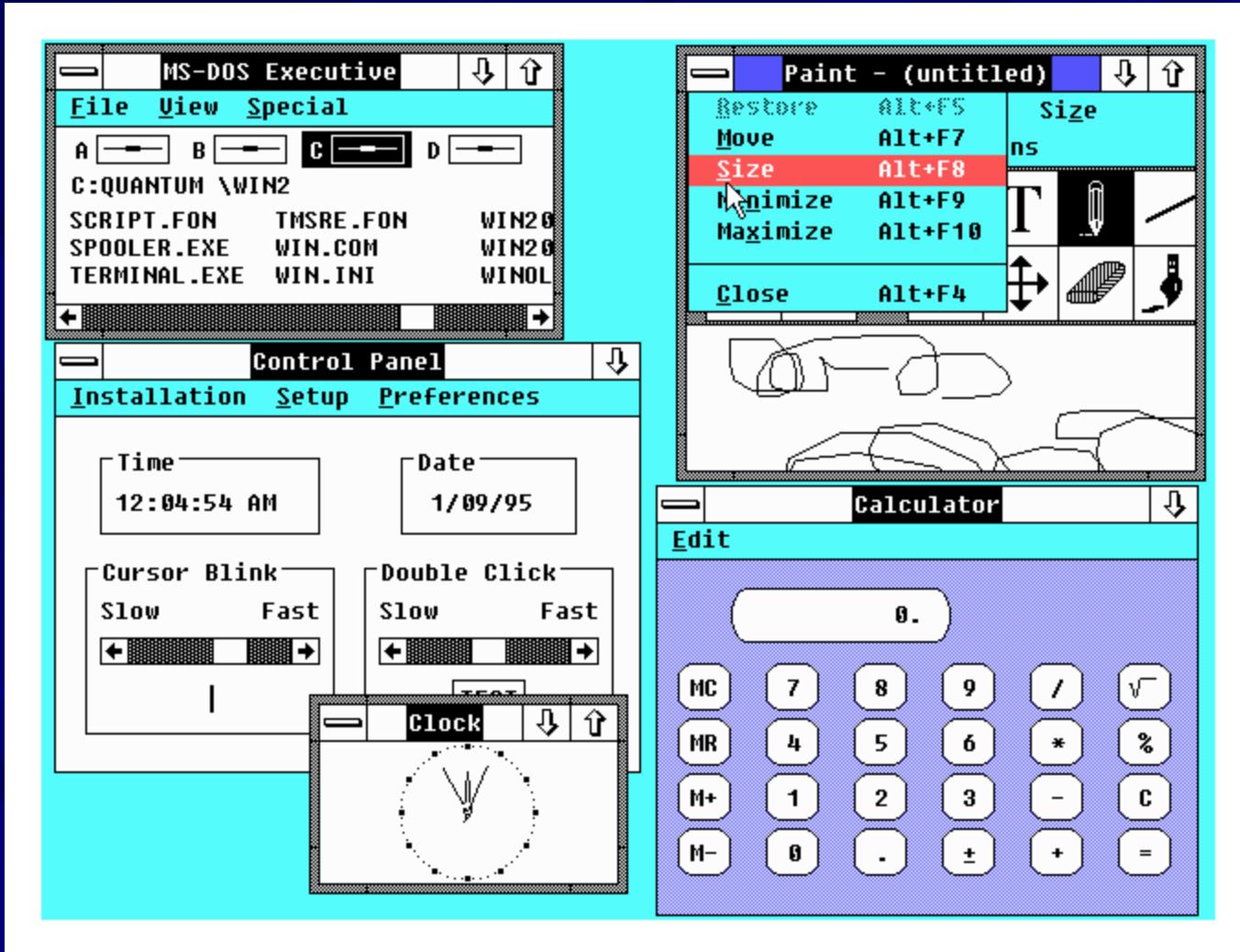
File View Special

A [] B [] C [] D [] E [] F [] G [] H [] G:AT \WIN_104

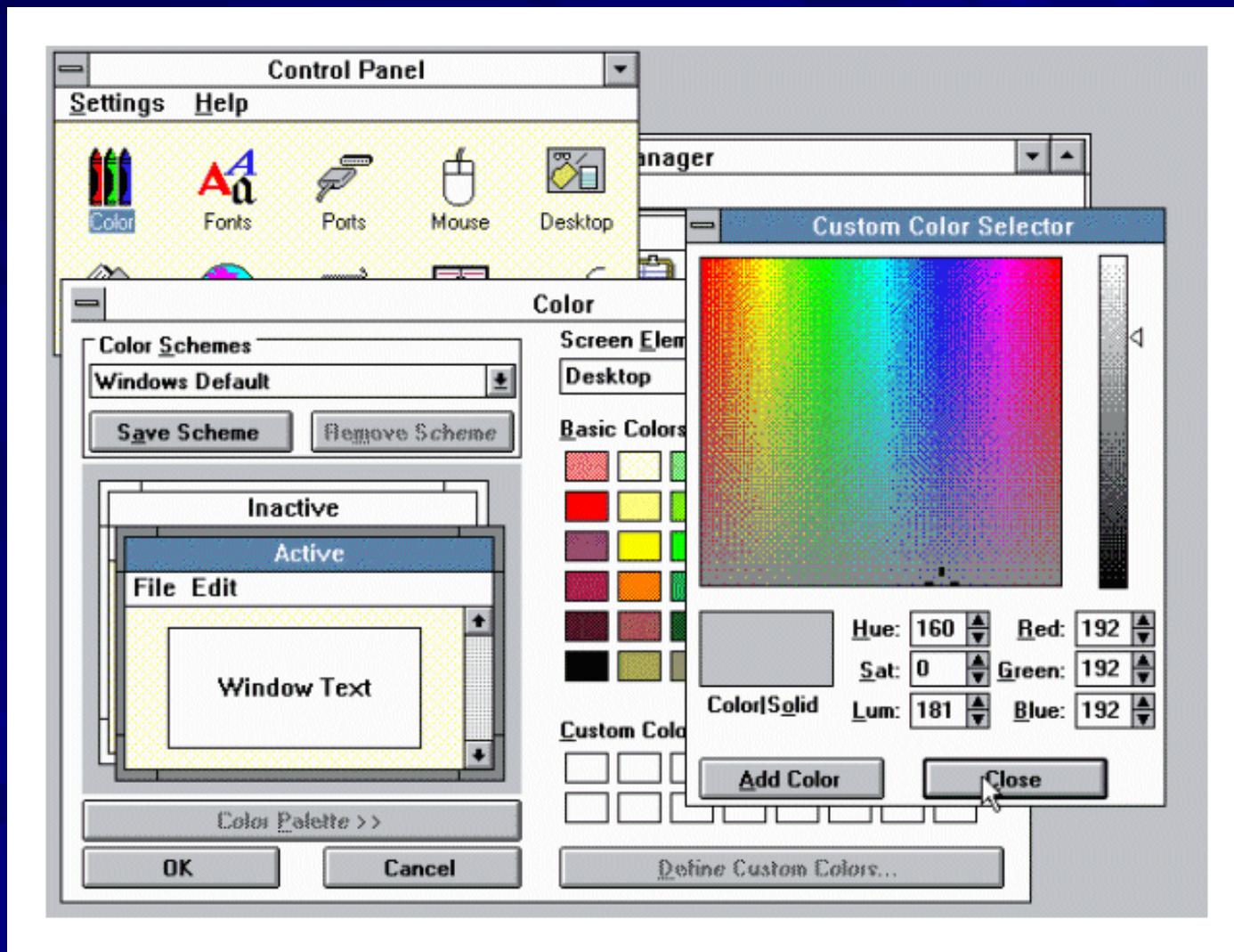
ABC.TXT	COURA.FON	NOTE PAD.EXE	SPOOLER.EXE	WIN100.OVL
CALC.EXE	COURB.FON	PAINT.EXE	TERMINAL.EXE	WINOLDAP.GRB
CALENDAR.EXE	DOTHIS.TXT	PRACTICE.WRI	TMSRA.FON	WINOLDAP.MOD
CARDFILE.EXE	HELVA.FON	README.TXT	TMSRB.FON	WRITE.EXE
CLIPBRD.EXE	HELUB.FON	REVERSI.EXE	WIN.COM	
CLOCK.EXE	MODERN.FON	ROMAN.FON	WIN.INI	
CONTROL.EXE	MSDOS.EXE	SCRIPT.FON	WIN100.BIN	



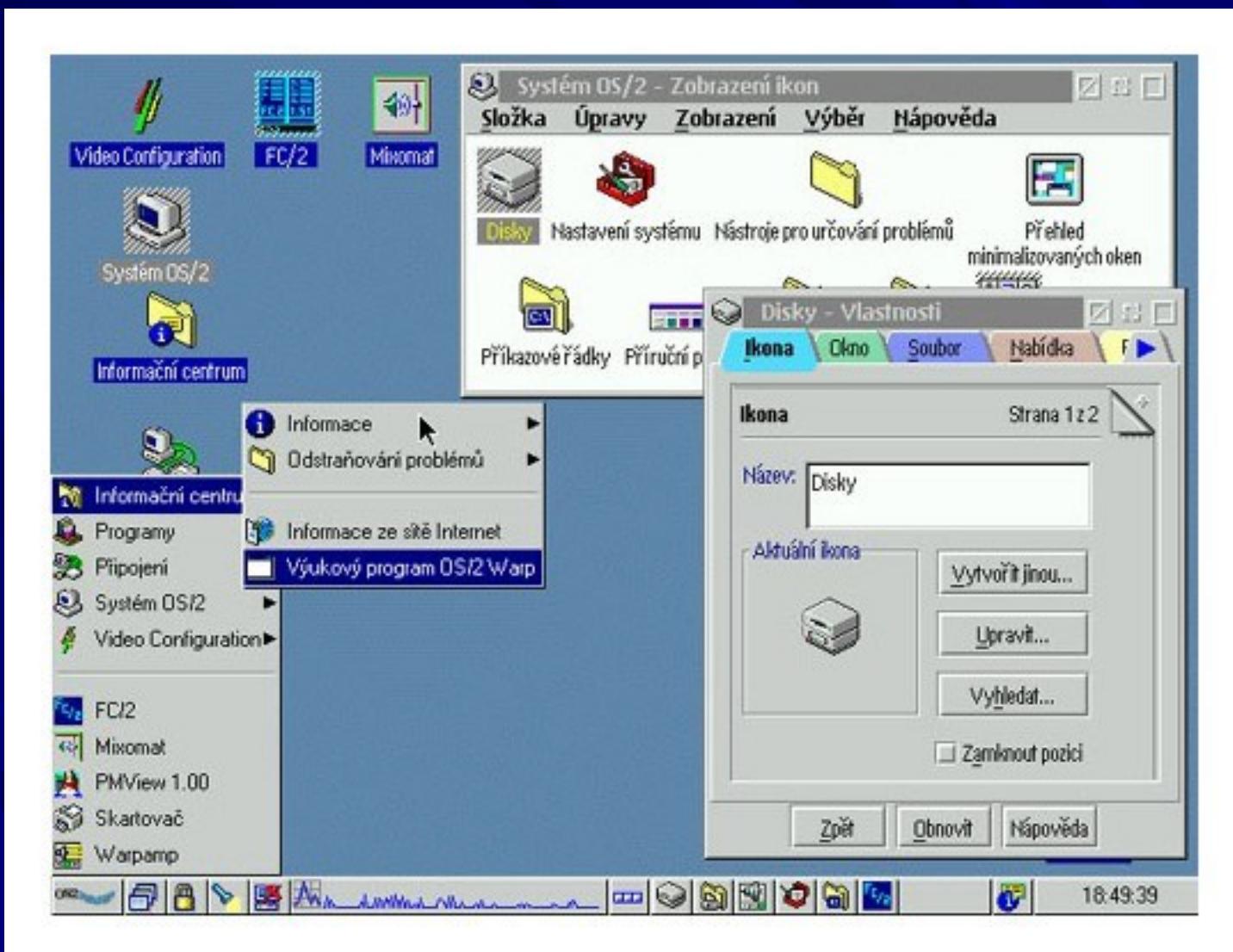
Win 2.0



Win 3.1



OS/2 Warp4



2 základní režimy OS

■ Uživatelský režim

- V tomto režimu běží aplikace (word, kalkulačka,...)
- Nemůžou vykonávat všechny instrukce, např. přímý přístup k zařízení (tj. zapiš datový blok x na disk y)
 - Proč? Jinak by škodlivá aplikace mohla např. smazat disk
 - Jak se tomu zabrání? Aplikace musí požádat jádro o službu, jádro ověří, zda aplikace má na podobnou činnost oprávnění a jádro činnost provede

■ Privilegovaný režim (režim jádra)

- Zde jsou povoleny všechny instrukce procesoru
- Běží v něm jádro OS, které mj. vykonává služby (systémová volání), o které je aplikace požádá

Jak se dostat z uživatelského režimu do režimu jádra?

Jde o přepnutí „mezi dvěma světy“, v každém z nich platí jiná pravidla

- Softwarové přerušení – instrukce INT 0x80
 - Stejně jako při hardwarovém přerušení (např. stisk klávesy): začne se vykonávat kód přerušení a vykoná se příslušné systémové volání
- Speciální instrukce (sysenter)
 - Speciální instrukce mikroprocesoru

Systémové volání

- Pojem **systémové volání** znamená vyvolání služby operačního systému poskytované jádrem
- Naše uživatelská aplikace nemohla sama vykonat, např. již zmíněný přístup k souboru na disku.
- Aplikace, která chce volat nějakou službu může volat systémové volání **přímo** (*open*, *creat*), nebo prostřednictvím **knihovní funkce** (v C např. *fopen*), která následně požádá o systémové volání sama.
- Výhodou knihovní funkce je, že je na různých platformách stejná, ať už se vyvolání systémové služby děje různým způsobem na různých platformách.

Systémové volání – příklad (!)

1. Do vybraného registru (EAX) uložím číslo služby, kterou chci vyvolat
 - Je to podobné klasickému číselníku
 - Např. služba 1- vytvoření procesu, 2- otevření souboru, 3- zápis do souboru, 4- čtení ze souboru, 5- výpis řetězce na obrazovku atd.
2. Do dalších registrů uložím další potřebné parametry
 - Např. kde je jméno souboru který chci otevřít
 - Nebo kde začíná řetězec, který chci vypsat
3. Provedu instrukci, která mě přepne do režimu jádra
 - tedy INT 0x80 nebo sysenter
4. V režimu jádra se zpracovává požadovaná služba
 - Může se stát, že se aplikace zablokuje, např. čekání na klávesu
5. Návrat, uživatelský proces pokračuje dále

Příklad

Jen ideový, v reálném systému se příslušné registry mohou jmenovat jinak

služba

LD AX, 5 .. Budeme volat **službu 5** (tisk řetězce)

LD BX, 100 .. Od **adresy 100** bude uložený řetězec

LD CX, 10 .. Délka řetězce, co se bude tisknout

INT 0x80 .. Vyvolání sw přerušení, přechod do světa jádra

... .. Vykonání systémového volání

... .. Kód naší aplikace pokračuje dále

parametry

Příklad reálný – Linux - getpid.c

```
int pid;
```

```
int main() {
```

```
    __asm__(
```

```
        "movl $20, %eax \n" /* getpid system call */
```

```
        "int $0x80 \n"      /* syscall */
```

```
        "movl %eax, pid \n" /* get result */
```

```
    );
```

```
    printf("Test volani systemove sluzby...\nPID: %d\n", pid);
```

```
    return 0;
```

```
}
```

- do registru EAX dáme číslo služby 20
- systémové volání přes int 0x80
- v registru EAX máme návratovou hodnotu pro naši službu (getpid)

Přehled systémových volání - Linux

<http://syscalls.kernelgrok.com/>

jde kliknut na jednotlivá volání – co znamenají
je vidět, co se dává do registrů (do EAX – číslo služby, ...)

Linux Syscall Reference



#	Name	eax	ebx	ecx	edx	esi	edi	Definition
0	sys_restart_syscall	0x00	-	-	-	-	-	kernel/signal.c:2058
1	sys_exit	0x01	int error_code	-	-	-	-	kernel/exit.c:1046
2	sys_fork	0x02	struct pt_regs *	-	-	-	-	arch/alpha/kernel/entry.S:716
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	-	-	fs/read_write.c:391
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	-	-	fs/read_write.c:408
5	sys_open	0x05	const char __user *filename	int flags	int mode	-	-	fs/open.c:900
6	sys_close	0x06	unsigned int fd	-	-	-	-	fs/open.c:969
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	-	-	kernel/exit.c:1771
8	sys_creat	0x08	const char __user *pathname	int mode	-	-	-	fs/open.c:933

Přehled systémových volání

další odkaz:

http://www.cli.di.unipi.it/~gadducci/SOL-11/Local/referenceCards/LINUX_System_Call_Quick_Reference.pdf

Z pohledu programátora

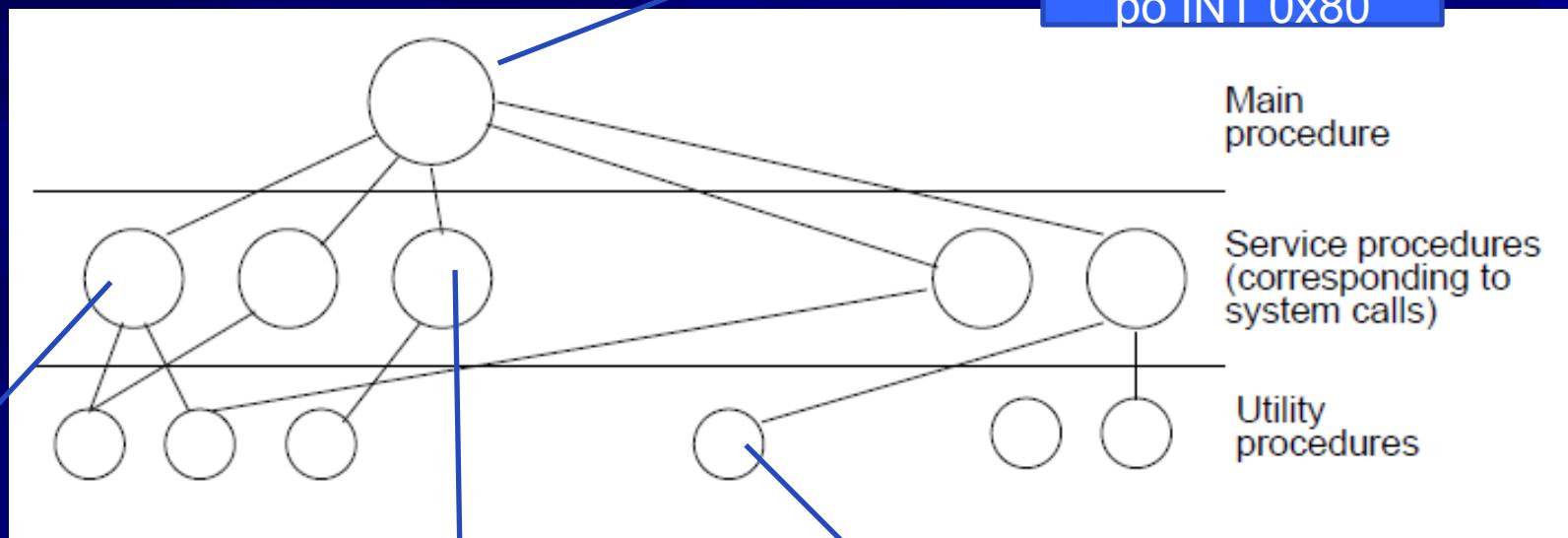
- inline assembler a INT 0x80
 - viz předchozí slide (reálný příklad Linux)
- použití instrukce **syscall()**
 - potřebujeme znát číslo funkce
 - interně použije INT 0x80
 - `id1 = syscall (SYS_getpid);`
- přímo je funkce, např. `getpid()`, `fopen()`
 - existuje „wrapper“ v libc knihovně
 - `id2 = getpid();`

Další možnosti uložení parametrů

- Musím nějak jádru říci, kterou službu chci a další parametry
- Informaci můžeme uložit
 - Do registrů (nejčastěji)
 - Na zásobník
 - Na předem danou adresu v paměti
 - Kombinací uvedených principů
- Příklad hypotetického volání:
chci po OS službu 2 (natočení piva) číslo služby uložím do EAX, do registru EBX uložím velikost piva (malé), do registru ECX stupeň (desítka)... vyvoláme INT 0x80
- Jádro pozná z EAX, že je zavolána služba 2 a ví, jak interpretovat hodnoty dalších registrů

Jak jádro implementuje jednotlivé služby?

Ukážeme si na monolitickém jádru:

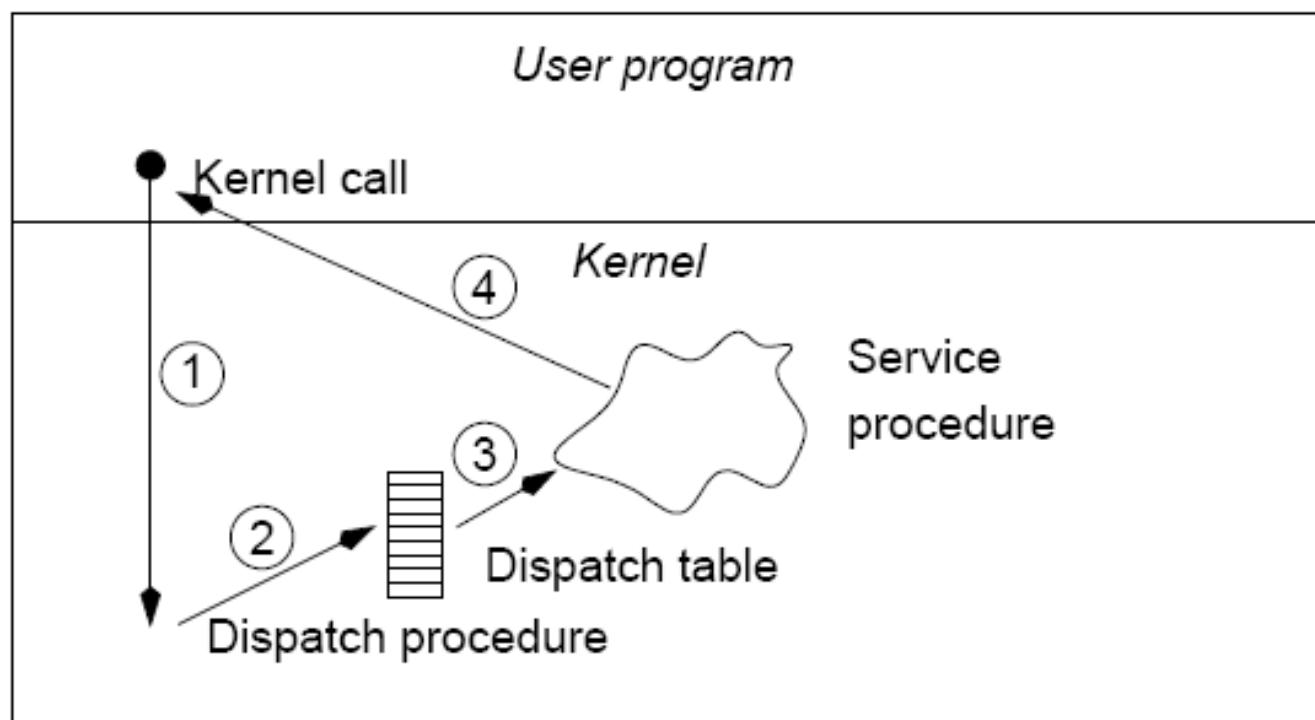


Služba 1
Např. otevři soubor

Služba 3
Např. vytvoř proces

Pomocné procedury,
Může je využívat i více služeb

Vyvolání služby systému (opakování)



Vyvolání služby systému (opakování)

- Parametry uložíme na určené místo
 - registry, zásobník
- Provedeme speciální instrukci (1)
 - vyvolá obsluhu v jádře
 - přepne do privilegovaného režimu
- OS převeze parametry, zjistí, která služba je vyvolána a provede službu
- návrat zpět
 - Přepnutí do uživatelského režimu

Poznámka

- Systémové volání nevyžaduje přepnutí kontextu na jiný proces
- Je zpracováno v kontextu procesu, který jej vyvolal

Co znamená INT x?

- instrukce v assembleru pro x86 procesory, která generuje SW přerušení
- x je v rozsahu 0 až 255
- paměť od 0 do je 256 4bytových ukazatelů (celkem 1KB), obsahují adresu pro obsluhu přerušení – **vektor přerušení**
- HW interrupty jsou mapovány na dané vektory prostřednictvím programovatelného řadiče přerušení

INT 0x80

- v 16kové soustavě 80, dekadicky 128
- pro vykonání systémového volání
- do registru EAX se dá číslo systémového volání, které chceme vyvolat

Jak se aplikace dostane do režimu jádra? (opakování)

■ Softwarové přerušení

- Volající proces způsobí softwarové přerušení
- Na platformě x86: instrukce `int 0x80`
- Přerušení se začne obsluhovat, procesor se přepne do režimu jádra a začne se provádět kód jádra

■ Speciální instrukce

- Novější, rychlejší
- Platforma x86: instrukce `sysenter`, `sysexit`

Může se lišit na různých platformách

Přerušení (interrupt)

- Přerušení patří k základním mechanismům používaným v OS
- **Asynchronní** obsluha události, procesor **přeruší** vykonávání sledu instrukcí (části kódu, které se právě věnuje), **vykoná** obsluhu přerušení (tj. instrukce v obslužné rutině přerušení) a **pokračuje** předchozí činností
- Analogie:
 - vařím oběd (vykonávám instrukce běžného procesu),
 - zazvoní telefon (přijde přerušení, je to asynchronní událost – kdykoliv)
 - Vyřídím telefon (obsluha přerušení)
 - Pokračuji ve vaření oběda (návrat k předchozí činnosti)
- Některé systémy mají **víceúrovňová** přerušení (vnoření)
 - (telefon přebije volání, že na někoho v sousedním pokoji spadla skříň)

Druhy přerušení (!!)

■ Hardwarové přerušení (vnější)

- Přichází z I/O zařízení, např. stisknutí klávesy na klávesnici
- **Asynchronní** událost – uživatel stiskne klávesu, kdy se mu zachce
- Vyžádá si pozornost procesoru bez ohledu na právě zpracovávanou úlohu
- Doručovány prostřednictvím **řadiče přerušení** (umí stanovit **prioritu** přerušením, aj.)

■ Vnitřní přerušení

- Vyvolá je sám procesor
- Např. pokus o dělení nulou, **výpadek stránky paměti** (!!)

■ Softwarové přerušení

- Speciální strojová instrukce (např. zmiňovaný příklad **INT 0x80**)
- Je **synchronní**, vyvolané záměrně programem (chce službu OS)
volání služeb operačního systému z běžícího procesu (!!)
uživatelská úloha nemůže sama skočit do prostoru jádra OS, ale má právě k tomu softwarové přerušení

■ Doporučuji přečíst:

<http://cs.wikipedia.org/wiki/P%C5%99eru%C5%A1en%C3%AD>

Kdy v OS použiji přerušení? (to samé z jiného úhlu pohledu)

- **Systémové volání** (volání služby OS)
 - Využiji softwarového přerušení a instrukce INT
- **Výpadek stránky paměti**
 - V logickém adresním prostoru procesu se odkazuje na stránku, která není namapovaná do paměti RAM (rámec), ale je odložená na disku
 - Dojde k přerušení – výpadek stránky
 - Běžící proces se pozastaví
 - Ošetří se přerušení – z disku se stránka natáhne do paměti (když je operační paměť plná, tak nějaký rámec vyhodíme dle nám známých algoritmů ☺)
 - Pokračuje původní proces přístupem nyní už do paměti RAM
- **Obsluha HW zařízení**
 - Zařízení si **žádá pozornost**
 - Klávesnice: stisknuta klávesa
 - Zvukovka : potřebuji poslat další data k přehrání
 - Sítová karta: došel paket

Vektor přerušení

- I/O zařízení signalizuje přerušení (*něco potřebuji*)
- Přerušení přijde na nějaké lince přerušení (IRQ, můžeme si představit jeden drát ke klávesnici, jiný drát k sériovému portu, další k časovači atd.)
- Víme číslo drátu (např. IRQ 1), ale potřebujeme vědět, **na jaké adrese** začíná obslužný program přerušení
- Kdo to ví? ... vektor přerušení
- **Vektor přerušení** je vlastně **index do pole**, obsahující **adresu obslužné rutiny**, vykonané při daném typu přerušení

Poznámka k přerušením

„signál“ operačnímu systému, že nastala nějaká událost, která vyžaduje ošetření (vykonání určitého kódu) - asynchronní

■ Hardwarové přerušení

- původ v HW
- Stisk klávesy, pohnutí myši
- Časovač (timer)
- Disk, síťová karta, ztráta napájení,..

■ Softwarová přerušení

- Pochází ze SW
- Obvykle z procesu v uživatelském režimu

Poznámka k přerušením

Příchod přerušení, z tabulky přerušení pozná, kde leží obslužný kód pro dané přerušení

Pozn. pro sw přerušení 0x80 ukazuje v tabulce přerušení (vektor přerušení) na vstupní bod OS

Maskování přerušení – v době obsluhy přerušení (musí být rychlá) lze zamaskovat méně důležitá přerušení

Sw přerušení jsou nemaskovatelná

Přijde-li přerušení... (!!)

- Přijde signalizace přerušení
- Dokončena rozpracovaná strojová instrukce
- Na zásobník uložena adresa následující instrukce,
tj. kde jsme skončili a kde budeme chtít pokračovat !!!!!!
- Z vektoru přerušení zjistí adresu podprogramu pro
obsluhu přerušení
- Obsluha - rychlá
 - Na konci stejný stav procesoru (hodnoty registrů) jako na začátku
- Instrukce návratu RET, IRET
 - Vyzvedne ze zásobníku návratovou adresu a na ní pokračuje !!!
- Přerušená úloha (mimo zpoždění) nepozná, že proběhla
obsluha přerušení

Knihovní funkce

- mechanismy volání jádra se v různých OS liší
- knihovní funkce
 - programovací jazyky zakrývají služby systému, aby se jevily jako běžné knihovní funkce
 - Jazyk C: `printf("Retezec");`
 - Knihovní funkce se sama postará, aby vyvolala vhodnou službu OS na dané platformě pro výpis řetězce
- Ne vždy musí volání knihovní funkce znamenat systémové volání, např. matematické funkce spočítá v uživatelském režimu

Architektury OS

OS = jádro + systémové nástroje

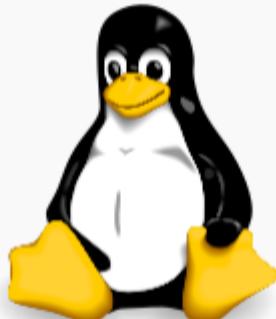
Jádro se zavádí do operační paměti při startu a zůstává v činnosti po celou dobu běhu systému

Základní rozdělení:

- Monolitické jádro – jádro je jeden funkční celek
- Mikrojádro – malé jádro, oddělitelné části pracují jako samostatné procesy v user space
- Hybridní jádro - kombinace

Architektura OS

Linux



Rodina OS:	Unix-like
Aktuální verze:	3.5 / 21. července 2012
Podporované platformy:	IA-32, x86-64, PowerPC, ARM, m68k, DEC Alpha, SPARC, hppa, IA-64, MIPS, s390 a další
Typ kernelu:	Monolitické jádro
Implicitní uživatelské rozhraní:	GNOME, KDE, Xfce a jiné
Licence:	GNU GPL a jiné
Stav:	Aktuální

Windows 7

Web:	Windows 7
Vyvíjí:	Microsoft
Rodina OS:	Windows NT
Druh:	Uzavřený vývoj
Aktuální verze:	Service pack 1 SP1 / 15.3.2011
Způsob aktualizace:	Windows Update
Správce balíčků:	Windows Installer
Podporované platformy:	x86, x86_64
Typ kernelu:	Hybridní jádro
Implicitní uživatelské rozhraní:	Grafické uživatelské rozhraní
Licence:	Microsoft EULA
Stav:	finální verze

Mac OS X



Web:	www.apple.com/macosx/
Vyvíjí:	Apple Inc.
Rodina OS:	BSD
Druh:	Uzavřený vývoj (s využitím open source komponent)
Aktuální verze:	10.8 / 19. července 2012
Podporované platformy:	x86, x86-64, PowerPC (32bitový i 64bitový)
Typ kernelu:	Hybridní jádro
Implicitní uživatelské rozhraní:	Aqua
Licence:	Apple SLA (část pod APSL)
Stav:	Aktivní

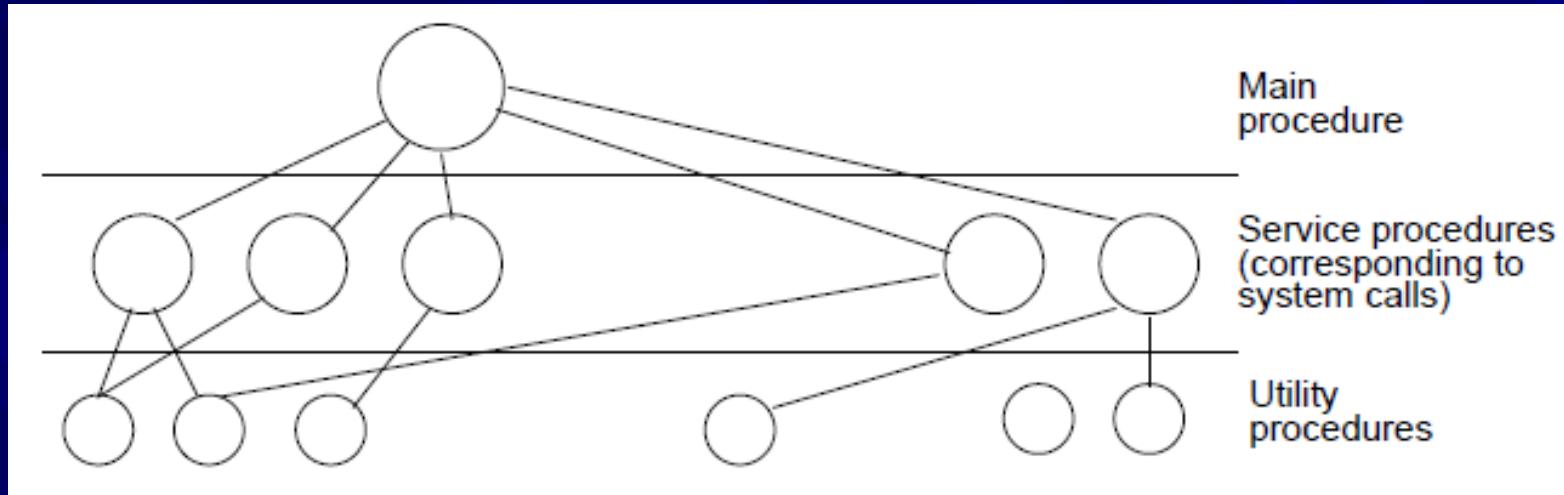
Monolitické jádro

- Jeden spustitelný soubor
- Uvnitř moduly pro jednotlivé funkce
- Jeden program, řízení se předává voláním podprogramů
- Příklady: UNIX, Linux, MS DOS

Typickou součástí jádra je např. souborový systém

Linux je monolitické jádro OS, s podporou zavádění modulů za běhu systému

Monolitické jádro (!)



Main procedure – vstupní bod jádra, na základě čísla služby (např. v EAX) zavolá servisní proceduru

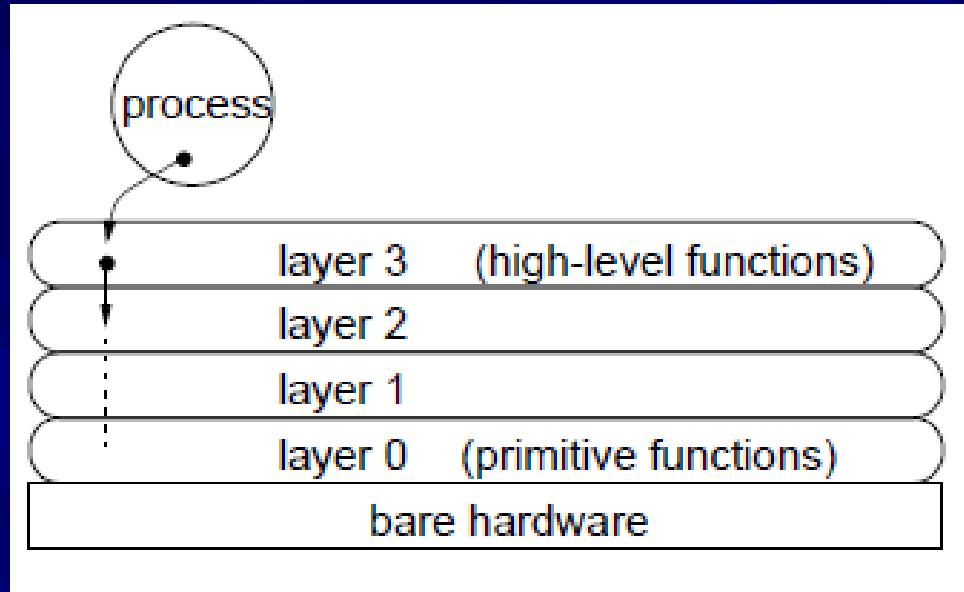
Service procedure – odpovídá jednotlivým systémovým voláním (zobrazení řetězce, čtení ze souboru, vytvoření procesu aj.)

Service procedure volá pro splnění svých cílů různé **pomocné utility procedures** (lze je opakovaně využít v různých voláních)

Vrstvené jádro

- Výstavba systému od nejnižších vrstev
- Vyšší vrstvy využívají primitiv poskytovaných nižšími vrstvami
- Hierarchie procesů
 - Nejníže vrstvy komunikující s HW
 - Každá vyšší úroveň poskytuje abstraktnější virtuální stroj
 - Může být s HW podporou – pak nelze vrstvy obcházet (obdoba systémového volání)
- Příklady: THE, MULTICS

Vrstvené jádro



OS THE:

Úroveň 0 .. Virtualizace CPU (přepínání mezi procesy)

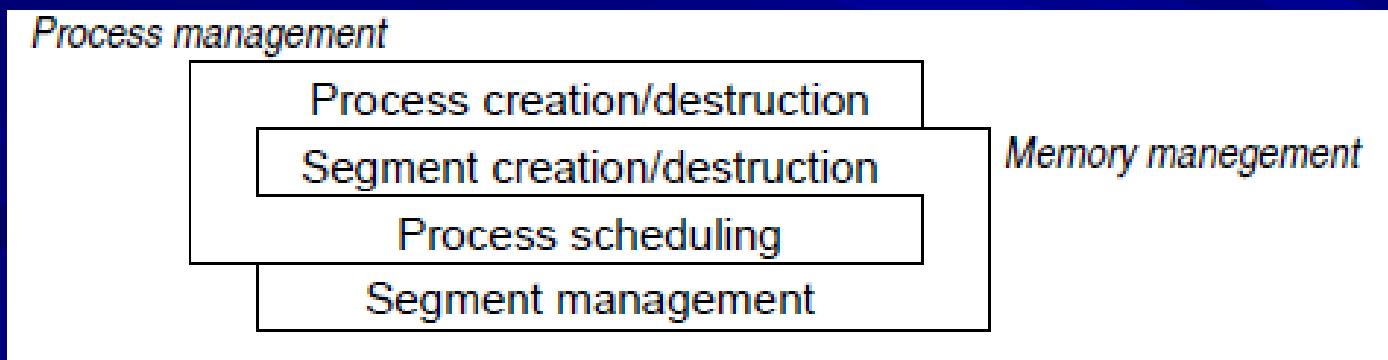
vyšší vrstva už předpokládá existenci procesů

Úroveň 1 .. Virtualizace paměti

vyšší vrstvy už nemusí řešit umístění části procesů v paměti

Funkční hierarchie

- Problém jak rozčlenit do vrstev
 - „dřív slepice, nebo vejce?“
 - správa procesů vs. správa paměti
- Některé moduly vykonávají více funkcí, mohou být na více úrovních v hierarchii
- Př: Pilot



Mikrojádro (!)

- Model klient – server
- Většinu činností OS vykonávají samostatné procesy mimo jádro (servery, např. systém souborů)
- Mikrojádro
 - Poskytuje pouze nejdůležitější nízkoúrovňové funkce
 - Nízkoúrovňová správa procesů
 - Adresový prostor, komunikace mezi adresovými prostory
 - Někdy obsluha přerušení, vstupy/výstupy
 - Pouze mikrojádro běží v privilegovaném režimu
 - Méně pádů systému

Mikrojádro

■ Výhody

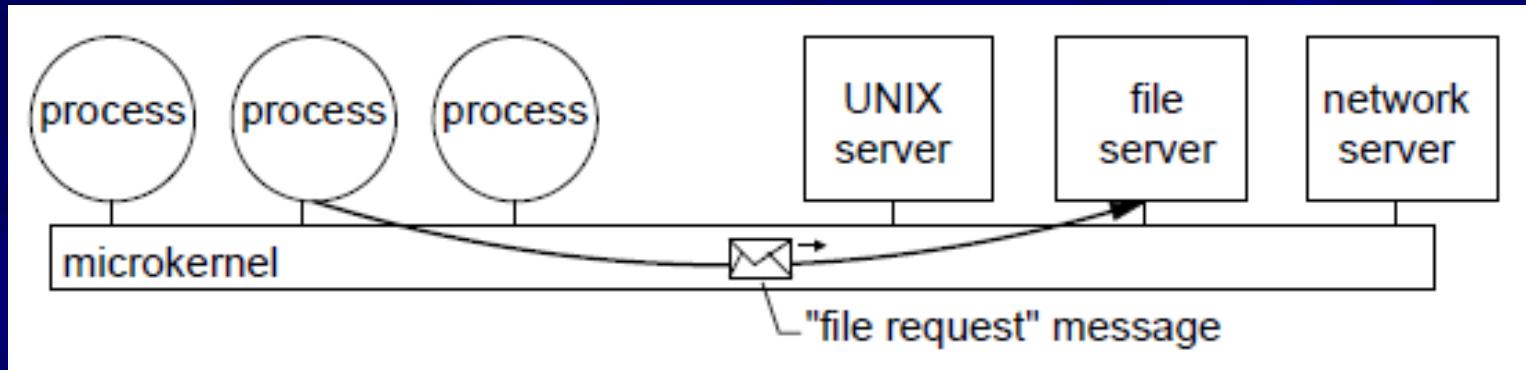
- vynucuje modulární strukturu
- Snadnější tvorba distribuovaných OS (komunikace přes síť)

■ Nevýhody

- Složitější návrh systému
- Režie
(4 x přepnutí uživatelský režim <-> jádro)

■ Příklady: QNX, Hurd, OSF/1, MINIX, Amoeba

Mikrojádro



Mikrojádro – základní služby, běží v privilegovaném režimu

1. proces vyžaduje službu
2. mikrojádro předá požadavek příslušnému serveru
3. server vykoná požadavek

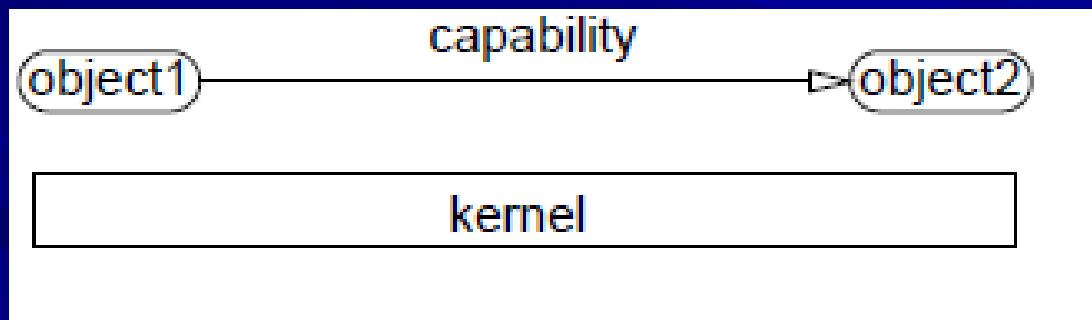
Snadná vyměnitelnost serveru za jiný

Chyba serveru nemusí být fatální pro celý operační systém
(není v jádře) – nespadne celý systém

Server může event. běžet na jiném uzlu sítě (distribuov. syst.)

Objektově orientovaná struktura

- Systém je množina objektů (soubory, HW zařízení)
- Capability = odkaz na objekt + množina práv definujících operace, spravuje jádro
- Jádro si vynucuje tento abstraktní pohled
- Jádro kontroluje přístupová práva
- Př: částečně Windows (NT,2000, XP, 7..) - hybridní



Hybridní jádro

- Kombinuje vlastnosti monolitického a mikrojádra
 - Část kódu součástí jádra (monolitické)
 - Jiná část jako samostatné procesy (mikrojádro)
-
- Příklady
 - Windows 7 (NT, Win 2000, Win XP, Windows Server 2003, Windows Vista,...)
 - Windows CE (Windows Mobile)
 - BeOS

GNU/Linux, GNU/Hurd

GNU/Linux

- GNU programy, např. gcc (R. Stallman)
- Monolitické jádro OS – Linux (Linus Torvald)

GNU/Hurd

- GNU programy, např. gcc
- Mikrojádro Hurd



Linux

- Pojem Linux jako takový označuje jádro operačního systému
- Pokud hovoříme o operačním systému, správně bychom měli říkat **GNU/Linux**, ale toto přesné označení používá jen málo distribucí, např. *Debian GNU/Linux*
- Flame war – debata mezi Linusem Torvaldsem a Andrewem Tanenbaumem, že Linux měl být raději mikrokernel

Linux - odkazy

Interaktivní mapa Linuxového jádra:

http://www.makelinux.net/kernel_map

Jaké jádro je nyní aktuální? (např. 3.16.3)

<http://kernel.org/>

Spuštění operačního systému bez instalace:

Live CD, Live DVD, Live USB

Např. *Knoppix* (<http://knoppix.org/>)

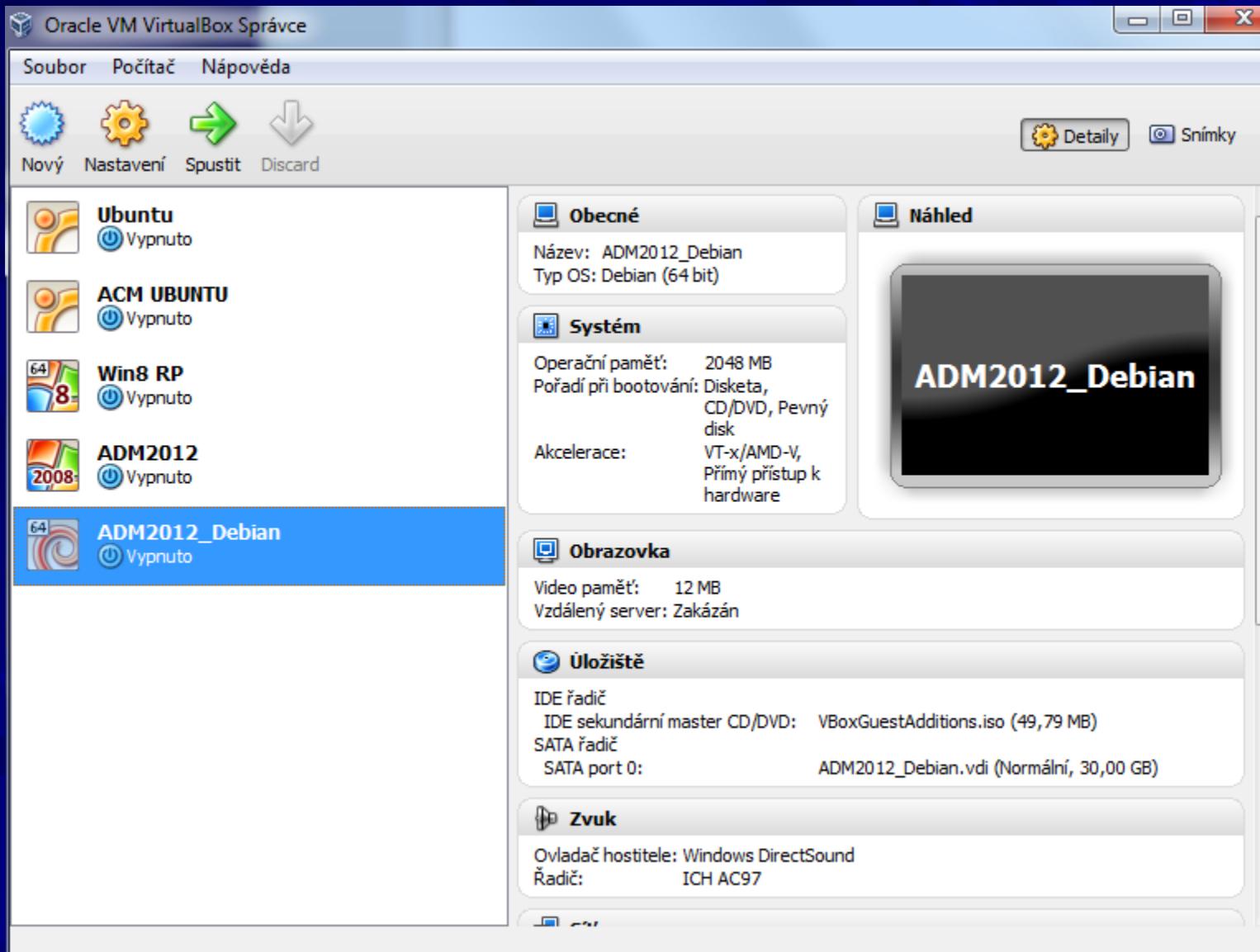
Kernel Version	Files	Lines
3.0	36,788	14,651,135
3.1	37,095	14,776,002
3.2	37,626	15,004,006
3.3	38,091	15,171,607
3.4	38,573	15,389,393
3.5	39,101	15,601,911
3.6	39,738	15,873,569
3.7	40,912	16,197,233
3.8	41,532	16,422,416
3.9	42,435	16,692,421
3.10	43,029	16,961,031

viz <http://www.zive.cz/clanky/podivejte-se-kdo-opravdu-vyviji-linux/sc-3-a-170587/default.aspx>

Virtualizace

- Možnost nainstalovat si virtuální počítač a provozovat v něm jiný operační systém, včetně přístupu k síti aj.
- VirtualBox
- VmWare
- Xen (např. virtualizace serverů), KVM aj.

VirtualBox



Literatura, použité zdroje

Obrázky z některých slidů (20, 21, 24) pocházejí z knížky

Andrew S. Tanenbaum: Modern Operating Systems

vřele doporučuji tuto knihu, nebo se alespoň podívat na slidy ke knize
dostupné mj. na webu předmětu v *Přednášky -> Odkazy*