Téma 2 – Architektury OS a jejich služby

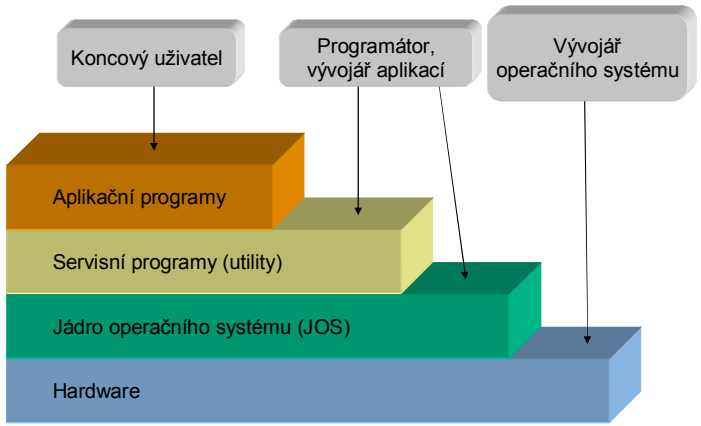
Obsah

1. Úkoly a skladba OS
2. Složky OS a jejich určení
3. Systémové programy
4. Standardy pro služby OS a typické služby JOS
5. Mechanismus volání služeb
6. Monolitické OS
7. OS s mikrojádrem
8. Virtuální stroje
9. Cíle návrhu OS, složitost OS

Operační systém

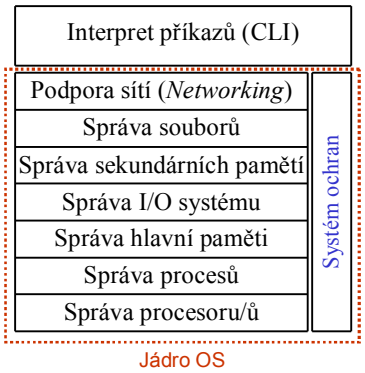
- Program, který řídí vykonávání aplikačních programů
- Styčná plocha (interface) mezi aplikačními programy a hardware
- Cíle OS:
  - Uživatelské „pohodlí“
  - Účinnost
    - Umožnit, aby systémové zdroje počítače byly využívány efektivně
  - Schopnost vývoje
    - Umožnit vývoj, testování a tvorbu nových systémových funkcí, aniž by se narušila činnost existujícího OS

Vrstvy ve výpočetním systému



Generické složky OS a jejich hierarchie

- Správa procesorů (CPU)
- Správa procesů
  - proces = činnost řízená programem
- Správa (hlavní, vnitřní) paměti
- Správa souborů
- Správa I/O systému
- Správa vnější (sekundární) paměti
- Podpora sítí (*Networking*)
- Systém ochran
- Interpret příkazů
  - CLI = Command Line Interpreter



Správa procesů a procesorů

- Provádění programu = proces (*process*, *task*)
  - Proces lze chápat jako rozpracovaný program
  - Proces má svůj **stav** (souhrn atributů a informací o rozpracovanosti)
- Proces potřebuje pro svůj běh jisté zdroje:
  - CPU (procesor), paměť, I/O zařízení, ...
- Správa procesů OS odpovídá za:
  - Vytváření a rušení procesů
  - Pozastavování (***blokování***) a obnovování procesů
  - Realizaci mechanismů pro
    - synchronizaci procesů
    - komunikaci mezi procesy
- Správa procesorů OS odpovídá za:
  - výběr procesoru pro běh procesu
  - výběr procesu, který poběží na dostupném procesoru

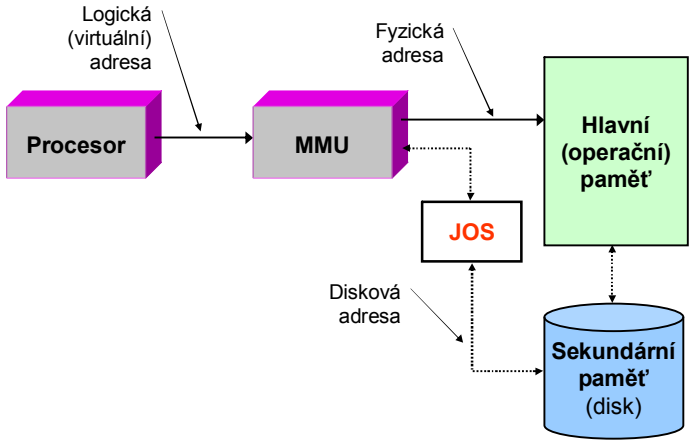
Správa (hlavní) paměti

- Hlavní (operační, primární) paměť
  - Pole samostatně adresovatelných slov nebo bytů
  - Repositář bezprostředně dostupných dat sdílený CPU (popř. několika CPU) a I/O zařízeními (resp. jejich řadiči)
  - Adresovaná fyzickými adresami (**FAP** = fyzický adresní prostor)
  - (Zpravidla) energeticky závislé zařízení
    - volatilní = pamatovaná data se ztrácí po výpadku energie
- OS je při správě (hlavní) paměti odpovědný za:
  - Přidělování a uvolňování paměti podle potřeb jednotlivých procesů
  - Vedení přehledu, který proces kterou část paměti v daném okamžiku využívá
  - Rozhodování, kterému procesu uspokojit jeho požadavek na prostor paměti po uvolnění prostoru v paměti

Virtualizace paměti

- Aplikační programátor i CPU vidí logické adresy (**LAP**)
  - Programy a data v LAP jsou zaváděny podle potřeby do FAP
- Struktury LAP
  - lineární (jednorozměrné pole)
  - dvojdimenzionální – kolekce samostatných lineárních **segmentů** (obecně proměnné délky)
- Zobrazování LAP do dostupného FAP pomocí hardware
  - Mechanismus **DAT**, *Dynamic Address Translation*
    - realizováno obvykle **jednotkou správy paměti MMU**, (= *Memory Management Unit*)
  - Při referenci logickou adresou místa, které není přítomno ve FAP
    - vznikne kritická výjimka (přerušení) a JOS převezme řízení a ve FAP nalezne vhodný „volný“ úsek paměti
    - na toto místo zavede se úsek s požadovanou informací z obrazu LAP na disku
- Nutná úzká spolupráce se specializovanou správou sekundární paměti
  - na vnější paměti JOS udržuje kopie (obraz) LAP procesu

Virtuální adresování a MMU



Správa vstupu a výstupu (I/O systém)

- OS spravuje soustavu vyrovnávacích pamětí
  - Paměť bloku přenášených dat je alokována v paměťovém prostoru jádra OS
  - To dovoluje uvolnit fyzickou paměť obsazovanou procesem během jím požadované I/O operace
    - řádově pomalejší I/O

- Drivery (ovladače) jednotlivých hardwarových I/O zařízení
  - Jsou specializované (pod)programy pro spolupráci a řízení konkrétní třídy vzájemně podobných periferních zařízení
- Jednotné rozhraní driverů (ovladačů) I/O zařízení
  - Všechny ovladače se jeví aplikačnímu programátorovi a nadřazeným vrstvám OS jako podprogramy s unifikovanou volací posloupností

Správa vnější paměti

- Hlavní (primární, operační) paměť
  - je volatilní, neschopná udržet informaci trvale
  - má relativně malou kapacitu a nelze v ní uchovávat všechna data a programy
- Počítačový systém musí mít energeticky nezávislou (persistentní) sekundární paměť s dostatečnou kapacitou
  - i za cenu nemožnosti přímé dostupnosti jejího obsahu procesorem
- Sekundární paměť obvykle realizují disky
- Jako správce vnější (sekundární) paměti je OS odpovědný za
  - Správu volného prostoru na sekundární paměti
  - Přidělování paměti souborům
  - Plánování činnosti relativně pomalých disků (např. minimalizace pohybů hlaviček disku)

Správa souborů

- Soubor
  - Identifikovatelná kolekce souvisejících informací vnitřně strukturovaná dle definice vytvořené tvůrcem souboru
  - Obvykle specializovaná reprezentace jak programů i dat
- Z hlediska správy souborů je OS odpovědný za:
  - Vytváření a rušení souborů
  - Vytváření a rušení adresářů (katalogů, „složek“)
  - Podporu elementárních operací pro manipulaci se soubory a s adresáři (čtení a zápis dat z/do souboru či adresáře)
  - Zobrazování souborů do sekundární paměti
  - Archivování souborů na energeticky nezávislá velkokapacitní média (např. magnetické pásky či DVD)

Podpora sítí, distribuované systémy

- Distribuovaný systém
  - Soustava počítačů, které nesdílejí ani fyzickou paměť ani hodiny („nesynchronizované kusy hardware“)
  - Každý počítač má svoji lokální paměť a pracuje samostatně
  - Počítače mohou mít i různé architektury
- Dílčí počítače distribuovaného systému jsou propojeny komunikační sítí
- Přenosy dat po síti jsou řízeny svými (zpravidla značně univerzálními) komunikačními protokoly
- Distribuovaný systém uživateli zprostředkovává přístup k různým zdrojům systému
- Přístup ke sdíleným zdrojům umožňuje
  - zrychlit výpočty (rozložení výpočetní zátěže)
  - zvýšit dostupnost dat (rozsáhlá data se nepřenášejí celá a nemusí být replikována)
  - zlepšit spolehlivost (havárie jedné části nemusí způsobit nefunkčnost celého systému)

Ochrany a chyby při běhu programů

- Ochrana
  - mechanismus pro kontrolu a řízení přístupu k systémovým a uživatelským zdrojům
- Systém ochran „prorůstá“ všechny vrstvy OS
- Systém ochran musí
  - rozlišovat mezi autorizovaným a neautorizovaným použitím
  - poskytnout prostředky pro prosazení legální práce
- Detekce chyb
  - Chyby interního a externího hardware
    - Chyby paměti, výpadek napájení
    - Chyby na ZVV či mediích („díra“ na disku)
  - Softwarové chyby
    - Aritmetické přetečení, dělení nulou
    - Pokus o přístup k „zakázaným“ paměťovým lokacím (ochrana paměti)
  - OS nemůže obsloužit žádost aplikačního programu o službu
    - Např. „k požadovanému souboru nemáš právo přistupovat“

Interpret příkazů

- Většina pokynů uživatele (operátora) je předávána operačnímu systému řídícími příkazy, které zadávají požadavky na
  - správu a vytváření procesů
  - ovládání I/O
  - správu sekundárních pamětí
  - správu hlavní paměti
  - zpřístupňování souborů
  - komunikaci mezi procesy
  - práci v síti, ...
- Program, který čte a interpretuje řídicí příkazy se označuje v různých OS různými názvy
  - Command-line interpreter (CLI), shell, cmd.exe, sh, bash, …
  - Většinou rozumí i jazyku pro programování dávek (tzv. skriptů)
  - CLI lze chápat jako nadstavbu JOS
    - systémový program (pracující v uživatelském režimu)
    - mnohdy těsně svázán se schopnostmi JOS

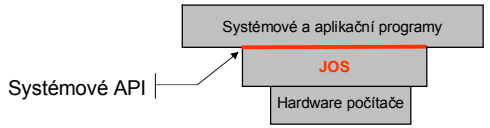
Systémové programy

- Poskytují prostředí pro vývoj a provádění programů
- Typická skladba
  - Práce se soubory, editace, kopírování, katalogizace, ...
  - Získávání, definování a údržba systémových informací
  - Modifikace souborů
  - Podpora prostředí pro různé programovací jazyky
  - Sestavování programů
  - Komunikace
  - Aplikační programy z různých oblastí
- Většina uživatelských rozhraní OS je vytvářená a podporovaná interprety příkazů – systémovými programy – a ne voláním systému (*system calls*)
- Systémové programy jsou v rámci OS řešeny formou výpočetních procesů, ne jako služby OS

Další funkce operačního systému

- Monitorování a „účtování“ systému
  - Sběr statistiky o využití systému
    - Pro dlouhodobé plánování v systému
    - Pro účtování „placených služeb“ (historická záležitost nabývající v současnosti znovu svůj význam při „cloud computing“)
  - Monitorování výkonnosti a chování systému
    - Slouží k podpoře dalšího vývoje systému

- Je třeba rozlišovat
  - služby OS jako celku
  - služby jádra OS **(JOS)** (*system calls*)



Služby JOS

- Standardy pro soustavy služeb OS (*system calls*)
  - Rozhraní systémových služeb – API (*Application Programming Interface*)
  - POSIX (IEEE 1003.1, ISO/IEC 9945)
    - Specifikuje nejen *system calls* ale i rozhraní standardních knihovních podprogramů a dokonce i povinné systémové programy a jejich funkcionalitu (např. **`ls`** vypíše obsah adresáře)
    - http://www.opengroup.org/onlinepubs/9699919799/nframe.html
  - Win32
    - Specifikace volání základních služeb systému v M$ Windows
    - objektová orientace
- Několik (zdánlivě nezávislých) skupin služeb JOS:
  - správa výpočetních procesů
  - přidělování a uvolňování paměti na žádost
  - přístup k datům v souborech a na periferiích
  - správa souborů a souborových systémů
  - služby pro podporu sítí
  - různé další služby
    - např. měření doby běhu úseku programu (profiling)

Základní služby jádra OS – POSIX (1)

Správa procesů

Služba	Popis
pid = fork()	Vytvoří potomka identického s rodičem
pid = waitpid(pid, &stat, options)	Čeká až zadaný potomek skončí
s = execve(name, argv, environp)	Nahradí „obraz“ procesu jiným „obrazem“
exit(status)	Ukončí běh procesu a vrátí status

Práce se soubory

Služba	Popis
fd = open(filename, how, ...)	Otevře soubor pro čtení, zápis, modif. apod.
s = close(fd)	Zavře otevřený soubor (uvolní paměť)
n = read(fd, buff, nbytes)	Přečte data ze souboru do pole *buff*
n = write(fd, buff, nbytes)	Zapíše data z pole *buff* do souboru
pos = lseek(fd, offset, whence)	Posouvá *ukazatel aktuální pozice* souboru
s = stat(filename, &statbuffer)	Dodá stavové informace o souboru

Základní služby jádra OS – POSIX (2)

Práce s adresáři souborů a správa souborů

Služba	Popis
s = mkdir(name, mode)	Vytvoří nový adresář s danými právy
s = rmdir(name)	Odstraní adresář
s = link(name1, name2)	Vytvoří položku *name2* odkazující na *name1*
s = unlink(name)	Zruší adresářovou položku
s = mount(spec, name, opt)	„Namontuje“ souborový systém
s = umount(spec)	„Odmontuje“ souborový systém

Další služby

Služba	Popis
s = chdir(dirname)	Změní „pracovní adresář“
s = chmod(fname, mode)	Změní „ochranné příznaky“ souboru
s = kill(pid, signal)	Zašle *signál* danému procesu
a mnoho dalších služeb	

Základní správa procesů

- Primitivní shell:

```
while(TRUE) {                                   /* nekonečná smyčka */
   type_prompt ();                              /* zobraz výzvu (prompt) */
   read_command (command, params)               /* přečti příkaz z terminálu */
   if (fork() > 0) {                            /* vytvoř nový synovský proces */
   /* Kód rodičovského procesu */
   waitpid(-1, &status, 0);                     /* čekej na ukončení potomka */
   } else {
   /* Kód "synovského" procesu */
      /* Zde lze připravit podmínky pro práci "synovského" procesu,        */
      /*   např. zařídit přesměrování vstupů a výstupů                     */
   execve(command, params, 0);                  /* vykonej příkaz command  */
   }
}
```

  - **tučně** jsou vyznačena přímá volání služeb JOS
  - ***tučnou kurzívou*** pak "funkce", které volání služeb JOS budou zcela jistě obsahovat

Porovnání služeb POSIX a Win32

POSIX	Win32	Popis
fork	CreateProcess	Vytvoř nový proces
waitpid	WaitForSingleObject	Může čekat na dokončení procesu
execve	--	CreateProcess = fork + execve
exit	ExitProcess	Ukončí proces
open	CreateFile	Vytvoří nový soubor nebo otevře existující
close	CloseHandle	Zavře soubor
read	ReadFile	Čte data ze souboru
write	WriteFile	Zapisuje data do souboru
lseek	SetFilePointer	Posouvá ukazatel v souboru
stat	GetFileAttributesExt	Vrací různé informace o souboru
mkdir	CreateDirectory	Vytvoří nový adresář souborů (složku)
rmdir	RemoveDirectory	Smaže adresář souborů
link	--	Win32 nepodporuje „spojky“ v soub. systému
unlink	DeleteFile	Zruší existující soubor
chdir	SetCurrentDirectory	Změní pracovní adresář

POSIX služby mount, umount, kill, chmod a další nemají ve Win32 přímou obdobu a analogická funkcionalita je řešena jiným způsobem

Hierarchické vrstvení JOS

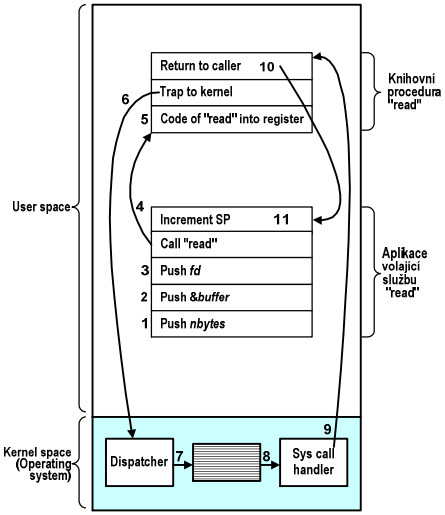
- JOS se dělí do jistého počtu vrstev (úrovní)
  - Každá vrstva je budována nad funkcionalitou nižších vrstev
  - Nejnižší vrstva (0) je hardware
  - Nejvyšší vrstva je uživatelské rozhraní
  - Pomocí principu modulů jsou vrstvy vybírány tak, aby každá používala funkcí (služeb) pouze vrstvy $n - 1$
- Řeší problém přílišné složitosti velkého systému
  - Dekomponuje se velký složitý problém na několik menších, snáze zvládnutelných (pod)problémů
  - Každá vrstva řeší konzistentní podmnožinu funkcí
  - Nižší vrstva nabízí vyšší vrstvě primitivní funkce (služby)
  - Nižší vrstva nemůže požadovat provedení služeb vyšší vrstvy
  - Používají se přesně definovaná rozhraní
    - jednu vrstvu lze uvnitř modifikovat, aniž to ovlivní ostatní vrstvy
    - rozhraní se volí tam, kde jsou nejméně složitá

Vykonávání služeb v klasickém OS

- Klasický monolitický OS
  - *Non-process Kernel OS*
  - Procesy – jen uživatelské a systémové programy
  - Jádro OS je prováděno jako monolitický (byť velmi složitý) program v privilegovaném režimu
- Služby OS lze plně vykonávat jako součást jádra nebo lze služby OS provádět v rámci běhu procesu
  - Obecně lze realizovat služby i v kontextu uživ. procesu
    - tj. jako jeho podprogram běžící při zamaskovaném přerušení a ležící v adresním prostoru uživatelského procesu – (kvůli bezpečnosti) užíváno relativně zřídkakdy
- Přerušení, volání služby
  - Vyvolá implicitně přepnutí režimu procesoru do systémového režimu, nepřepíná se však kontext volajícího procesu
  - K přepnutí kontextu (přechodu od jednoho procesu k jinému) $proces_1$ – OS – $proces_2$ dochází jen, je-li to nutné z hlediska plánování procesů po dokončení služby

Volání služeb jádra OS

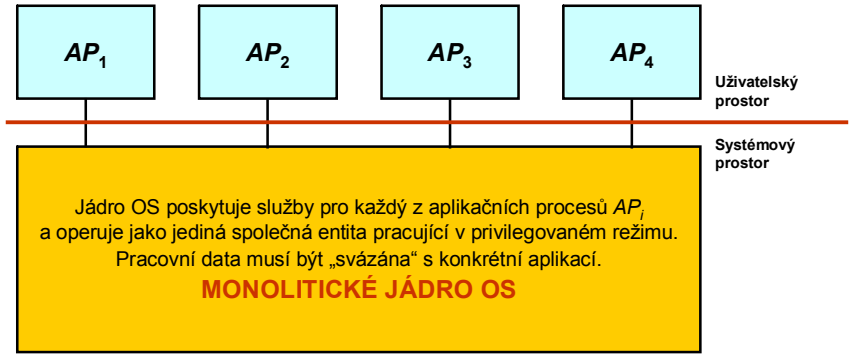
- Aplikační program (proces) volá službu OS:
  - Zavolá podprogram ze standardní systémové knihovny
  - Ten transformuje volání na systémový standard (*native API*) a vyvolá synchronní přerušení
  - JOS převezme řízení v privilegovaném režimu práce CPU
  - Podle kódu požadované služby dispečer služeb zavolá komponentu JOS odpovědnou za tuto službu
  - Po provedení služby se řízení vrací aplikačnímu programu s případnou indikací úspěšnosti



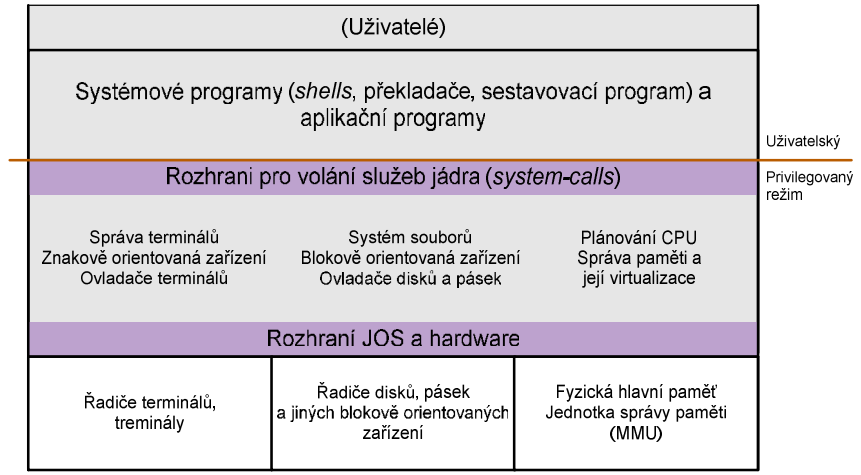
11 kroků k provedení služby read (fd, buffer, nbytes)

Služba OS plně jako součást JOS

- Tradiční řešení

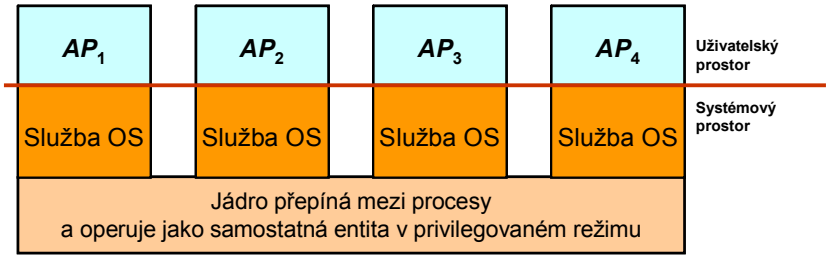


Příklad monolitické architektury: UNIX



Služba OS jako součást procesu

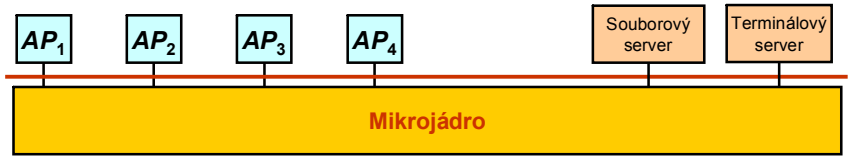
- Alternativní řešení



- Synchronní přerušení se obsluhuje v režii procesu → minimalizace přepínání mezi procesy.
- Používáno v UNIX SVR4
- Uvnitř JOS používá každý proces samostatný zásobník
- Kód a data JOS jsou ve sdíleném adresovém prostoru a jsou sdílena všemi procesy

Procesově orientované JOS, mikrojádro

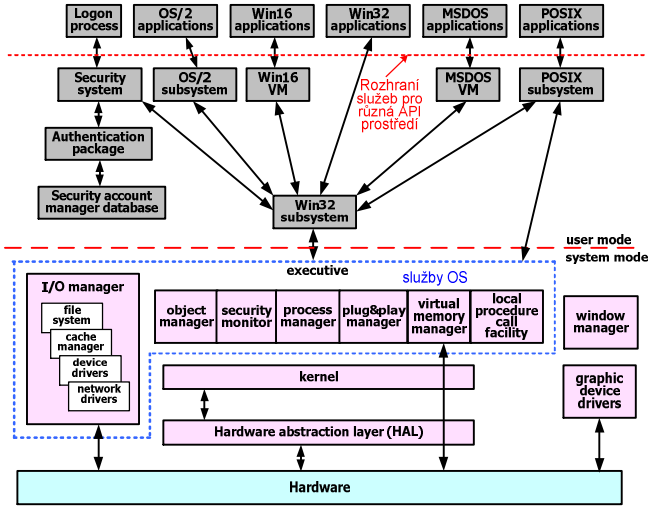
- OS je soustavou systémových procesů
- Funkcí jádra je tyto procesy separovat a přitom umožnit jejich kooperaci
  - Minimum funkcí je potřeba dělat v privilegovaném režimu
  - Jádro je pouze „ústředna“ pro přepojování zpráv
  - Řešení snadno implementovatelné i na multiprocesorech
- Malé jádro => mikrojádro (µ-jádro) – (microkernel W)



OS s µ-jádrem – výhody a nevýhoda

- OS se snáze přenáší na nové hardwarové architektury,
  - µ-jádro je malé
- Vyšší spolehlivost – modulární řešení
  - moduly jsou snáze testovatelné
- Vyšší bezpečnost
  - méně kódu se běží v privilegovaném režimu
- Pružnější, snáze rozšiřitelné řešení
  - snadné doplňování nových služeb a rušení nepotřebných
- Služby jsou poskytovány unifikovaně
  - výměnou zpráv
- Přenositelné řešení
  - při implementaci na novou hardwarovou platformu stačí změnit µ-jádro
- Podpora distribuovanosti
  - výměna zpráv je implementována uvnitř systému ale i v síti
- Podpora objektově-orientovaného přístupu
  - snáze definovatelná rozhraní mezi aplikacemi a µ-jádrem
- To vše za cenu
  - zvýšené režie, volání služeb je nahrazeno výměnou zpráv mezi aplikačními a systémovými procesy

Příklad OS s µ-jádrem – Windows XP



Ve Windows 7/8 by to vypadalo ještě mnohem složitěji.

Virtuální stroje (1)

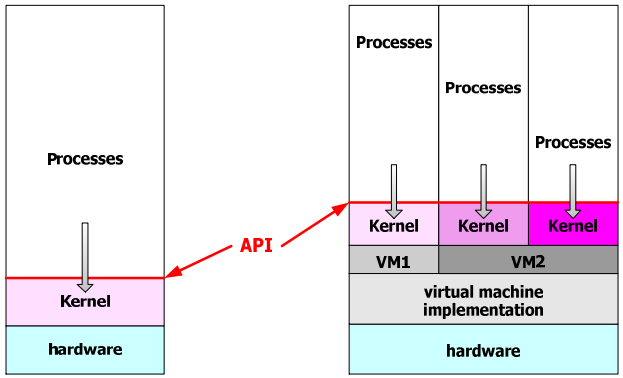
- Logická struktura OS s principem vrstvení dotaženým do extrému
  - Virtuální stroj je softwarový produkt, který chápe hardware a jádro operačního systému jako jednu společnou (hardwarovou) vrstvu
  - Virtuální stroj vyváží rozhraní identické s emulovaným holým (podloženým) hardwarem
- OS běžně vytváří ilusi prostředí, ve kterém běží více procesů
  - Každý proces běží na svém vlastním (virtuálním) procesoru vybaveném svou vlastní (virtuální) pamětí
  - Lze-li vytvořit iluzi souběžnosti více procesů, lze vytvořit i iluzi současnosti běhu více systémů s vlastnostmi původního fyzického počítače nebo i úplně jiného stroje resp. jiného OS, jiné verze téhož OS, ...
  - Každý uživatel na sdíleném stroji může tak užívat jiný OS

Virtuální stroje (2)

- Virtuální stroj zajišťuje úplnou ochranu systémových zdrojů
  - Každý virtuální stroj je izolován od všech ostatních
  - Taková izolace však neumožňuje přímé sdílení zdrojů
- Na virtuálním stroji může běžet jiný (další) virtuální stroj
  - usnadňuje to ladění částí OS v době běžného provozu
  - umožňuje to provést změnu parametrů bez restartování systému
- Zdroje fyzického počítače jsou sdíleny s cílem vytvořit ilusi existence virtuálních strojů
  - Plánování CPU dává uživatelům iluzi, že mají svůj vlastní procesor
  - Systém souborů může podporovat i virtuální tiskárnu, atd.
- Virtuální stroj je obtížné implementovat, protože musí modelovat přesný duplikát příslušného hardware
  - může být tragicky pomalé a neefektivní

Virtuální stroje (3)

- Standardní nevirtualizovaný stroj
- Jedno hardware
- Dvě virtuální hardwarové platformy
- Tři virtuální operační systémy



- Reálný příklad: Virtuální Windows XP ve Windows 7
  - Detaily http://www.microsoft.com/windows/virtual-pc/default.aspx

Cíle návrhu OS

- Uživatelský pohled
  - OS musí být snadno použitelný, snadno naučitelný, bezpečný, rychlý, ...
- Systémové hledisko
  - OS se musí dát snadno implementovat, udržovat a musí být přizpůsobivý, spolehlivý, bezchybný (?), ...

- Skutečné výsledky
  - Operační systémy jsou (a asi vždy budou)
    - obrovské – až desítky miliónů řádků zdrojového kódu
    - asynchronní (interaktivní)
    - (téměř vždy) plné chyb a (často) nespolehlivé a
    - silně závislé na konkrétním hardware, a tedy obtížně přenositelné

- Tradičně býval OS psaný v asembleru (*assembly language*). Nyní se OS píší v běžných programovacích jazycích vyšší úrovně (C, C++)
  - OS lze napsat rychleji
  - je kompaktnější
  - je srozumitelnější a lze ho snáze ladit
  - je (aspoň teoreticky) snáze přenositelný na jinou hardwarovou architekturu

Vytváření provozní verze OS (SYSGEN)

- Operační systém je obvykle připraven tak, aby běžel na jisté třídě hardwarových platforem / sestav počítače
- OS musí být konfigurovatelný na konkrétní sestavu
- Program SYSGEN
  - Na základě informace týkající se konkrétní požadované konfigurace a konkrétního hardwarového systému vytváří provozní verzi OS odpovídající skutečné skladbě HW prostředků
- Zavaděč systému (*Bootstrap program*)
  - Program uchovávaný v ROM, který umí nalézt jádro (zpravidla na disku), zavést ho do paměti a spustit jeho inicializaci a další provádění
- Zavádění systému (*Booting*)
  - Zavedením jádra a předáním řízení na jeho vstupní bod se spustí činnost celého systému
    - Jádro poté spustí počáteční aplikační proces, který čte různé konfigurační soubory a spouští inicializační dávky a startuje tím další komponenty systému

OS jsou funkčně složité

OS	Rok	Počet služeb jádra (*system calls*)
Unix	1971	33
Unix	1979	47
SunOS4.1	1989	171
4.3 BSD	1991	136
SunOS4.5	1992	219
SunOS5.6 (Solaris)	1997	190
Linux 2.0	1998	229
WinNT 4.0	1999	**3 443**

- Obrovská složitost vnitřních algoritmů (jádra) OS
  - Počty cyklů CPU spotřebovaných ve WinXP při
    - Zaslání zprávy mezi procesy: 6K – 120 K (dle použité metody)
    - Vytvoření procesu: ~3M
    - Vytvoření vlákna: ~100K
    - Vytvoření souboru: ~60K
    - Vytvoření semaforu: 10K – 30K
    - Nahrání DLL knihovny" ~3M
    - Obsluha přerušení/výjimky: 100K – 2M
    - Přístup do systémové databáze (*Registry*) : ~20K

OS jsou velmi rozsáhlé

- Historie Windows

OS	Rok	Počet řádků kódu [SLOC]
Windows 3.1	1992	3 mil.
Windows NT 3.5	1993	4 mil.
Windows 95	1995	15 mil.
Windows NT 4.0	1996	16 mil.
Windows 98 SR-2	1999	18 mil.
Windows 2000 SP5	2002	60 mil.
Windows XP SP2	2005	**78 mil.**

Dotazy