Operační systémy a sítě

Petr Štěpán, K13133
KN-E-229
stepan@labe.felk.cvut.cz

Téma 2. Služby a architektury OS

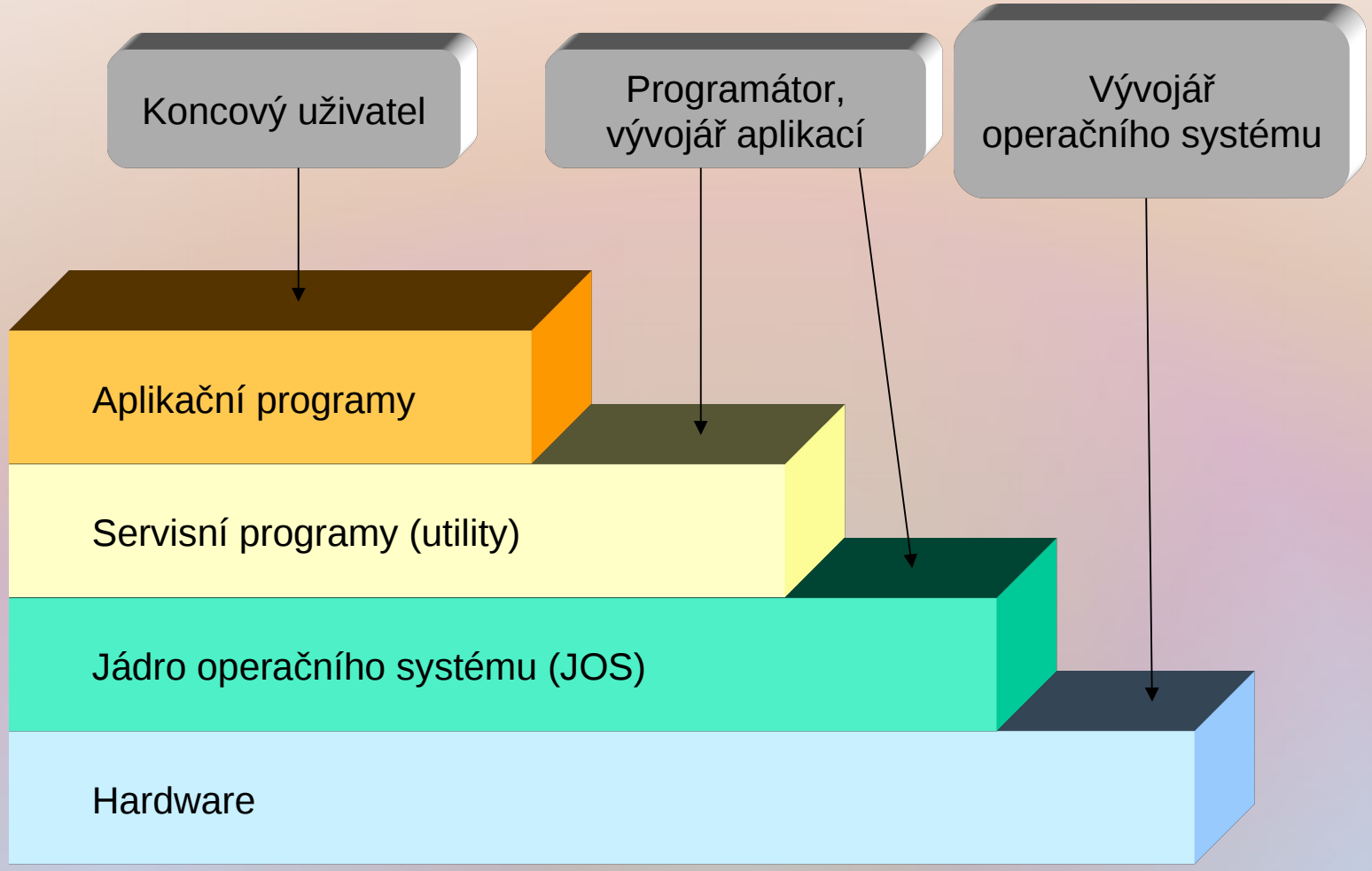
Obsah

- Funkce operačního systému
- Části operačního systému
- Systémové programy
- Služby operačního systému
- Mechanismus volání služeb
- Monolitické operační systémy
- Operační systémy s mikrojádrem
- Virtuální počítač
- Cíle návrhu OS a složitost OS

Operační systém

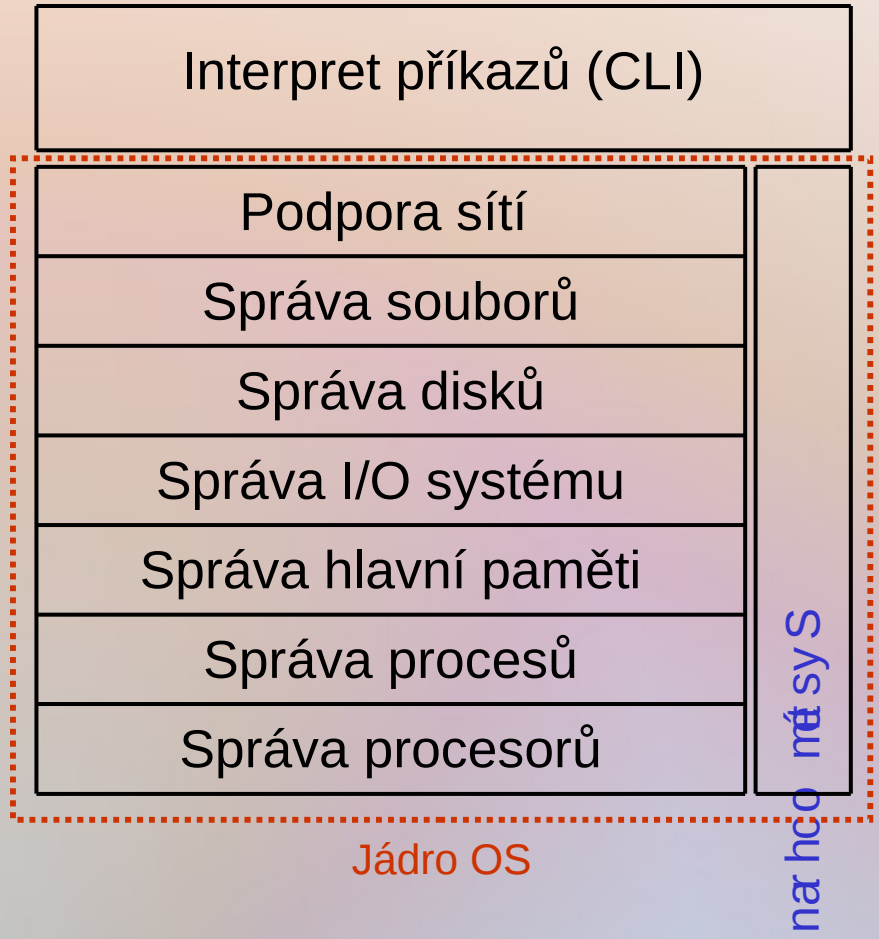
- Program, který řídí vykonávání aplikačních programů
- Styčná plocha (interface) mezi aplikačními programy a hardware
- Cíle OS:
  - Uživatelské „pohodlí“
  - Účinnost
    - Umožnit, aby systémové zdroje počítače byly využívány efektivně
  - Schopnost vývoje
    - Umožnit vývoj, testování a tvorbu nových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

Vrstvy počítače



Složky OS

- Správa procesorů
- Správa procesů
  - proces = činnost řízená programem
- Správa (hlavní, vnitřní) paměti
- Správa I/O systému
- Správa disků - vnější (sekundární) paměti
- Správa souborů
- Podpora sítí
- Systém ochran
- Interpret příkazů



Správa procesů a procesorů

Provádění programu = proces (process, task)

- Proces lze chápat jako rozpracovaný program
- Proces má svůj stav (souhrn atributů 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 paměti

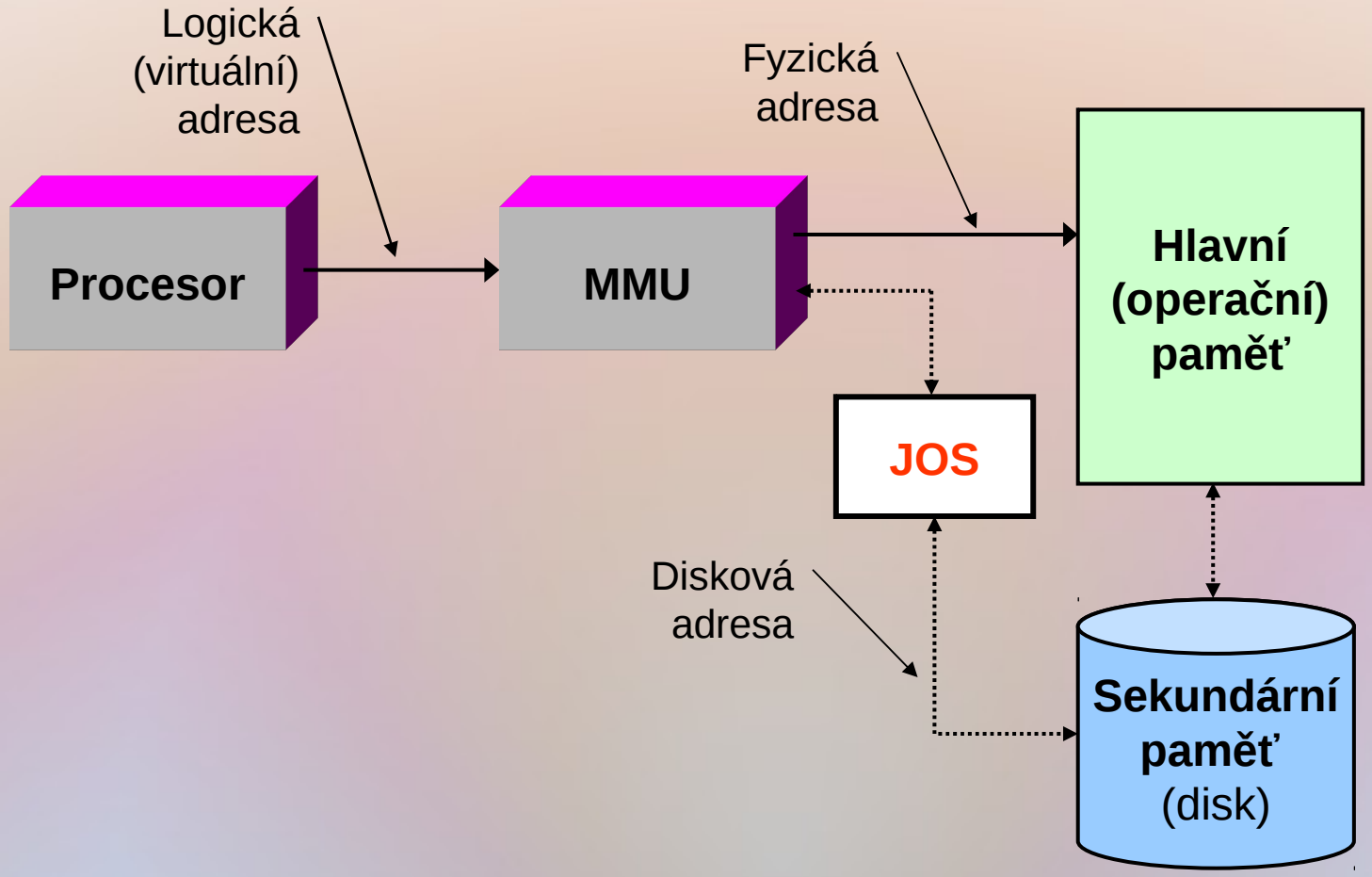
Hlavní (operační, primární) paměť

- 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
- pamatovaná data se ztrácí po výpadku energie

OS je při správě (hlavní) paměti odpovědný za:

- 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
- Přidělování a uvolňování paměti podle potřeb jednotlivých procesů

Virtualizace paměti



Správa vstupně/výstupních zařízení

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 a vedlejším efektem těchto podprogramů je pak práce s periferií

Správa disků – sekundární paměti

Hlavní paměť (RAM) je energeticky závislá, 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

- ať už klasické pevné disky nebo SSD bez mechanických částí

Jako správce vnější (sekundární) paměti je OS odpovědný za

- Správu volného prostoru na sekundární paměti
- Přidělování paměti souborům
- Plánování činnosti relativně pomalých disků
  - organizace vyrovnávacích pamětí
  - minimalizace pohybů hlaviček disku, minimalizace přepisu buněk u SSD, apod.

Správa souborů

Soubor

- Identifikovatelná kolekce souvisejících informací vnitřně strukturovaná dle definice navržené 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)
- Mapování souborů do sekundární paměti
- Archivování souborů na energeticky nezávislá velkokapacitní média (např. CD, DVD, magnetické pásky)

Podpora sítí

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)

Interpret příkazů

Většina zadání uživatele je předávána operačnímu systému řídícími příkazy, které zadávají požadavky na

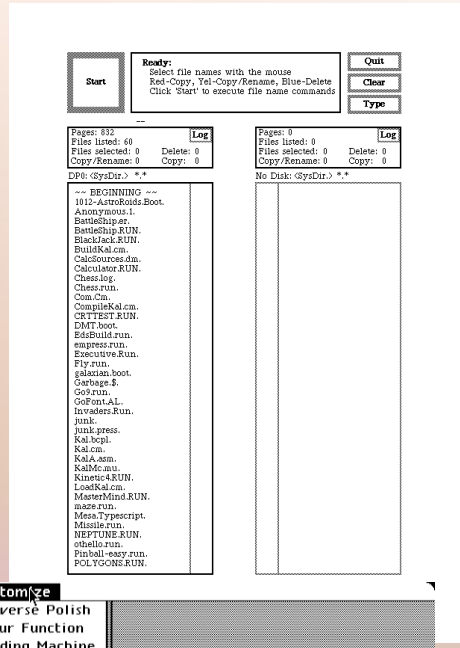
- správu a vytváření procesů
- ovládání I/O
- správu sekundárních pamětí
- správu hlavní paměti
- zpřístupňování souborů
- komunikaci mezi procesy
- práci v síti, ...

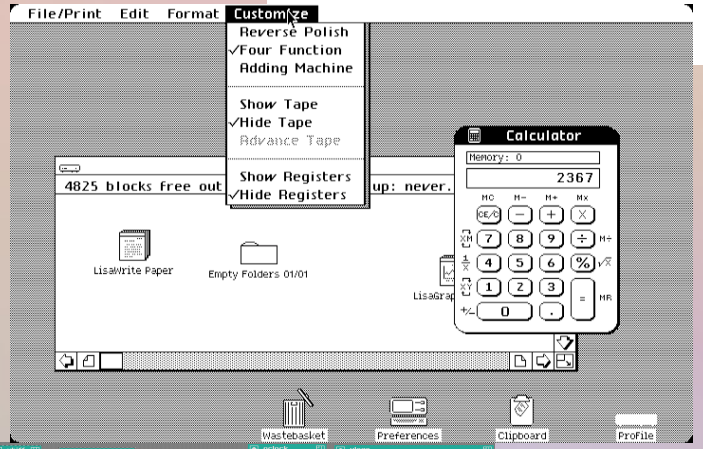
Program, který čte a interpretuje řídicí příkazy se označuje v různých OS různými názvy

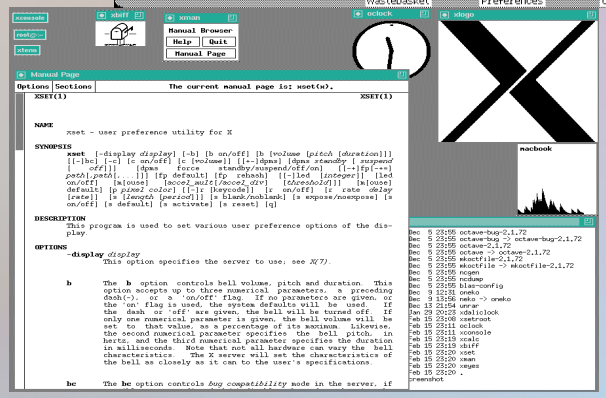
- Command-line interpreter (CLI), shell, cmd.exe, sh, bash, …
- Většinou rozumí jazyku pro programování dávek (tzv. skriptů)
- Interpret příkazů lze chápat jako nadstavbu vlastního OS
  - systémový program (pracující v uživatelském režimu)

GUI

- První Xerox Alto (1973)
- Apple Lisa (1983)
- X window (1984) – MIT, možnost vzdáleného terminálu přes síť
- Windows 1.0 pro DOS (1985)
- Windows 3.1 (1992) podpora 32-bitových procesorů s ochranou paměti, vylepšená grafika
- Windows NT (1993) – preemptivní multitasking, předchůdce Windows XP (2001)







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
- Anti-virové programy
- Šifrování a bezpečnost
- Aplikační programy z různých oblastí

Systémové programy jsou v rámci OS řešeny formou výpočetních procesů, ne jako služby OS

Ochrana jádra OS

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 vstupně/výstupních zařízeních 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“

Ochrana jádra OS

Základ ochrany OS přechod do systémového módu

- Intel PSW obsahuje 2bity privilegovaného módu 0 – jádro OS, 3 – uživatelský mód, bezpečný

Přechod ze systémového módu do uživatelského

- prostou změnou systémového módu
- návrat z přerušení

Přechod z uživatelského módu do systémového

- pouze programově vyvolaným přerušením
- speciální instrukce trap nebo int
- nejde spustit cokoliv, spustí se pouze program připravený operačním systémem
- Systémové volání – služby jádra - system calls

Služby jádra OS

X86 System Call Example Hello World on Linux

```
   .section .rodata
greeting:
    .string "Hello World\n"
    .text


_start:
   mov $12,%edx                 /* write(1, "Hello World\n", 12) */
   mov $greeting,%ecx
   mov $1,%ebx
   mov $4,%eax                  /* write is syscall no. 4 */
   int $0x80


   xorl %ebx, %ebx             /* Set exit status and exit */
   mov $0xfc,%eax
   int $0x80

   hlt                          /* Just in case... */
```

Služby jádra OS

Služby jádra jsou číslovány

- Registr eax obsahuje číslo požadované služby
- Ostatní registry obsahují parametry, nebo odkazy na parametry
- Problém je přenos dat ze systémové oblasti do uživatelského prostoru
  - malá data lze přenést v registrech – návratová hodnota funkce
  - velká data - uživatel musí připravit prostor, OS tam nakopíruje data

Volání služby jádra není jednoduché

- Je nutné použít assembler, volání musí být implementováno přesně
- Zapouzdření pro programovací jazyky – API

Linux system call table – http://docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html
Windows system call table - http://j00ru.vexillium.org/ntapi/

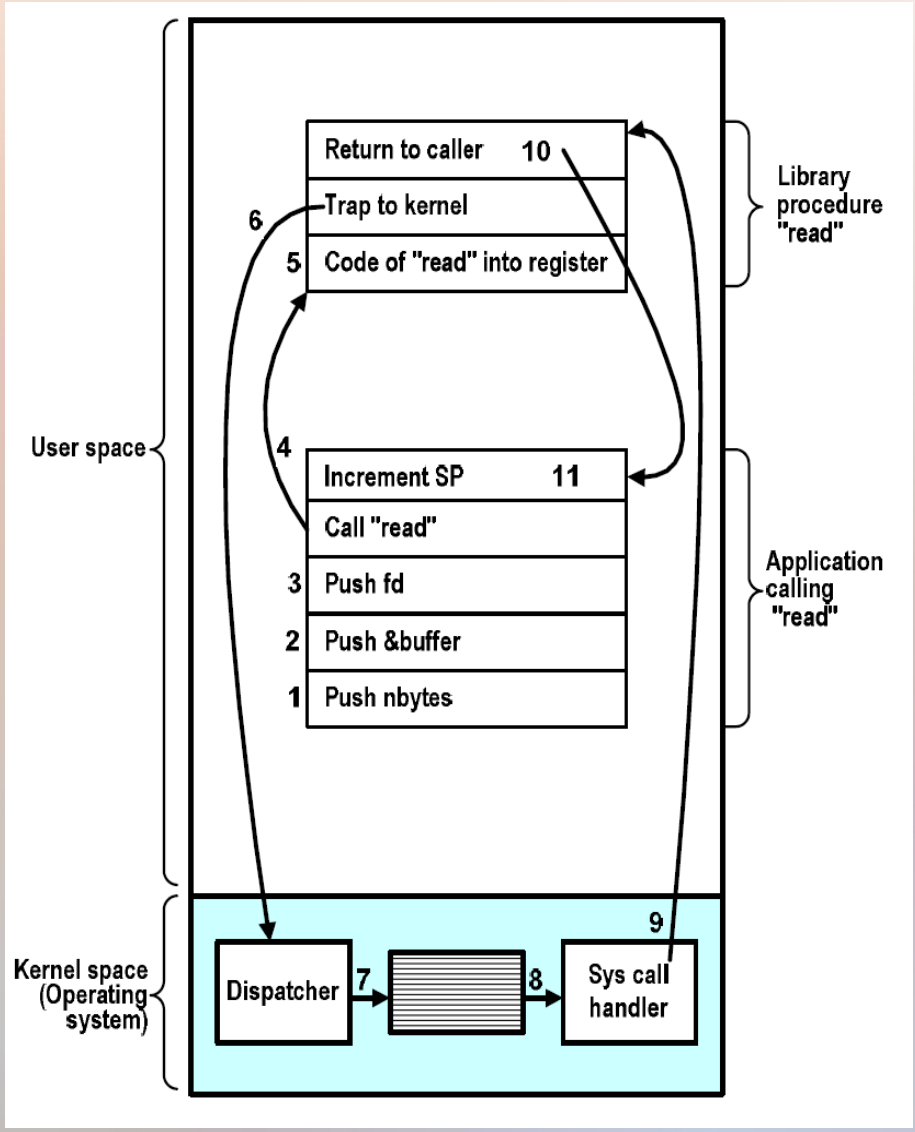
API

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
- Win API
  - Specifikace volání základních služeb systému v M$ Windows
- Standard Template Library pro C++
- Java API

Volání služeb jádra OS přes API

- 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



POSIX

Portable **O**perating **S**ystem **I**nterface for Uni**x** – IEEE standard pro systémová volání i systémové programy

Standardizační proces začal 1985 – důležité pro přenos programů mezi systémy

1988 POSIX 1 Core services – služby jádra

1992 POSIX 2 Shell and utilities – systémové programy a nástroje

1993 POSIX 1b Real-time extension – rozšíření pro operace reálného času

1995 POSIX 1c Thread extension – rozšíření o vlákna

Po roce 1997 se spojil s ISO a byl vytvořen standard POSIX:2001 a POSIX:2008

Ukázka POSIX definice

NAME

abort - generate an abnormal process abort **SYNOPSIS**

#include <stdlib.h>

void abort(void);

DESCRIPTION

[CX] The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2008 defers to the ISO C standard.

The *abort*() function shall cause abnormal process termination to occur, unless the signal SIGABRT is being caught and the signal handler does not return.

[CX] The abnormal termination processing shall include the default actions defined for SIGABRT and may include an attempt to effect *fclose()* on all open streams.

The SIGABRT signal shall be sent to the calling process as if by means of *raise()* with the argument SIGABRT.

[CX] The status made available to *wait()*, *waitid()*, or *waitpid()* by *abort*() shall be that of a process terminated by the SIGABRT signal. The *abort*() function shall override blocking or ignoring the SIGABRT signal.

Ukázka POSIX definice

FUTURE DIRECTIONS

None.

SEE ALSO

exit , *kill* , *raise* , *signal* , *wait* , *waitid*

XBD *<stdlib.h>*

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 6

Extensions beyond the ISO C standard are marked.

Changes are made to the DESCRIPTION for alignment with the ISO/IEC 9899:1999 standard.

The Open Group Base Resolution bwg2002-003 is applied.

IEEE Std 1003.1-2001/Cor 1-2002, item XSH/TC1/D6/10 is applied, changing the DESCRIPTION of abnormal termination processing and adding to the RATIONALE section.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/9 is applied, changing ``implementation-defined functions'' to ``implementation-supplied functions'' in the APPLICATION USAGE section.

Základní služby jádra OS – POSIX

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 správa procesů

- Primitivní shell:

```
while (TRUE) {                                  /* nekonečná smyčka */
   type_prompt( );                              /* zobraz výzvu (prompt) */
   read_command (command, parameters)       /* 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 */
      execve (command, parameters, 0); /* vykonej příkaz command  */
   }
}
```

Základní služby jádra OS – POSIX

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

Windows API

Nebylo plně popsáno, skrytá volání využívaná pouze spřátelenými stranami

MS developers mají privilegovaný přístup k popisu systémových volání

Win16 – 16-ti bitová verze rozhraní pro Windows 3.1

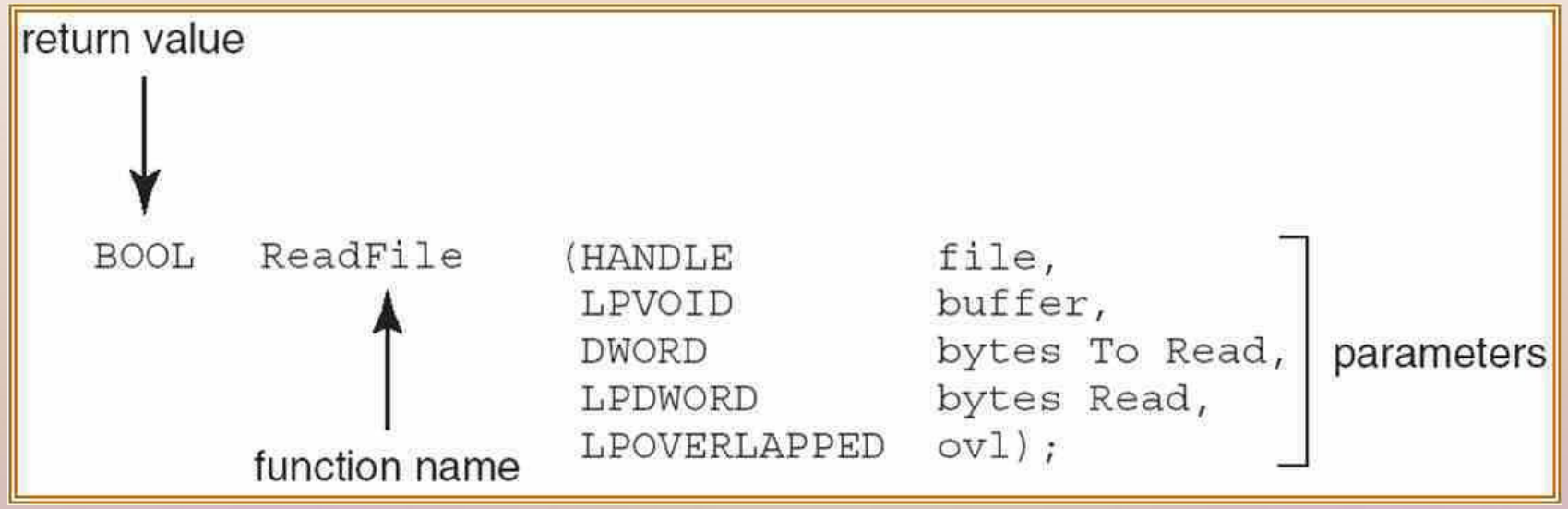
Win32 – 32 bitová verze od Windows NT

Win32 for 64-bit Windows – 64 bitová verze rozhraní Win32

Nová window mohou zavést nová volání, případně přečíslování starých služeb.

Příklad Win API

- Funkce ReadFile() z Win32 API – funkce, která čte z otevřeného souboru



- Parametry předávané funkci ReadFile()
  - HANDLE file – odkaz na soubor, ze kterého se čte
  - LPVOID buffer – odkaz na buffer pro zapsání dat ze souboru
  - DWORD bytesToRead – kolik bajtů se má přečíst
  - LPDWORD bytesRead – kolik bajtů se přečetlo
  - LPOVERLAPPED ovl – zda jde o blokované čtení

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

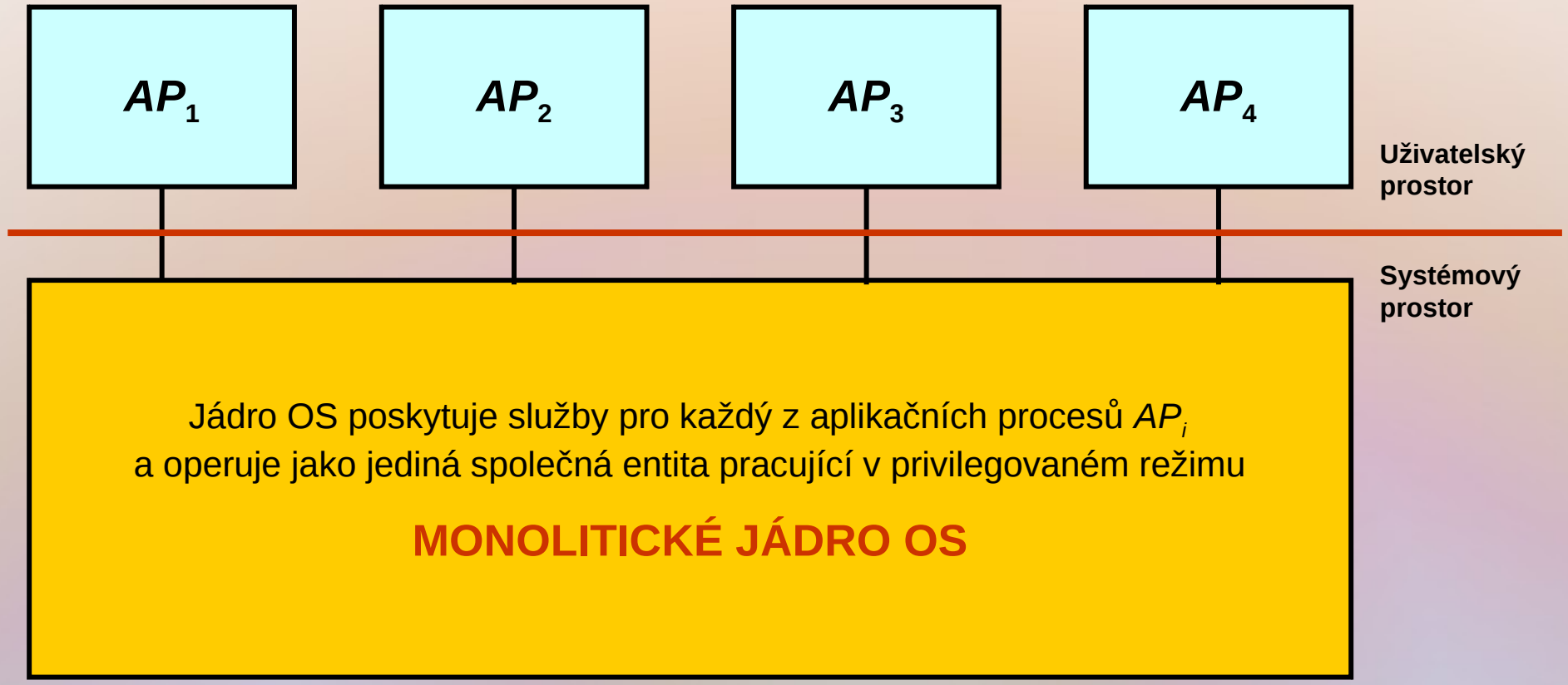
POSIX služby mount, umount, kill, chmod a další nemají ve Win32 přímou obdobu a analogická funkcionalita je řešena jiným způsobem

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 jádře v rámci běhu procesu
  - Obecně lze realizaci služeb provádět v kontextu uživ. procesu
    - tj. jako jeho podprogram běžící při zamaskovaném přerušení a ležící v adresním prostoru uživatelského procesu – užito relativně zřídka
- 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

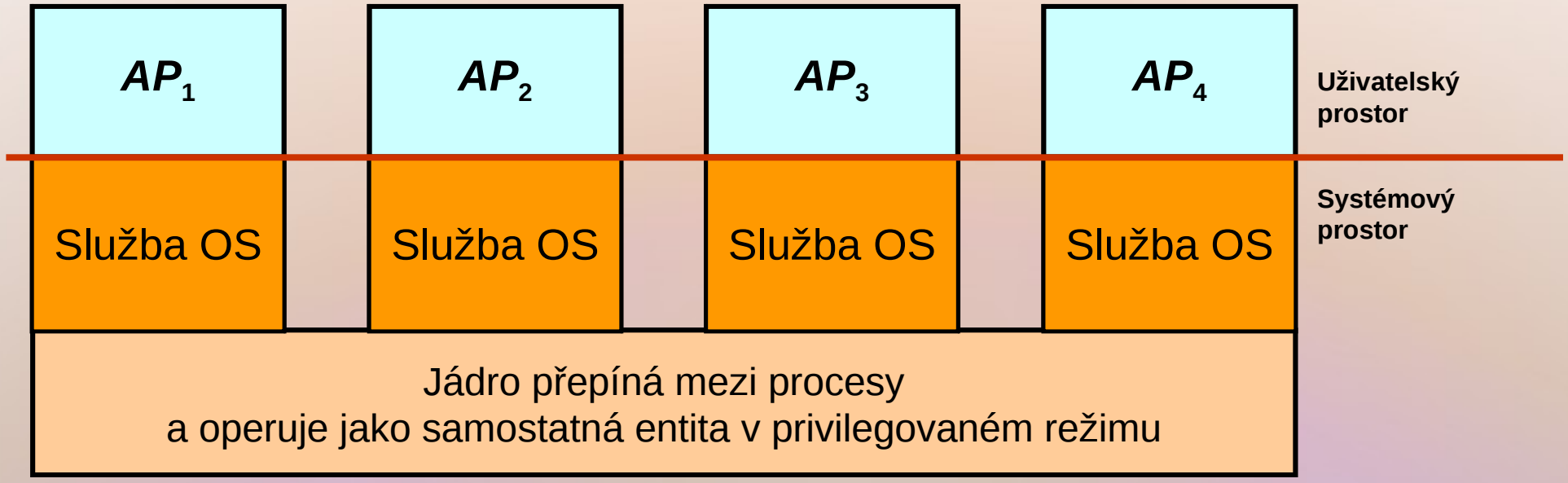
Služba OS plně jako součást JOS

- Tradiční řešení



Služba OS jako součást procesu

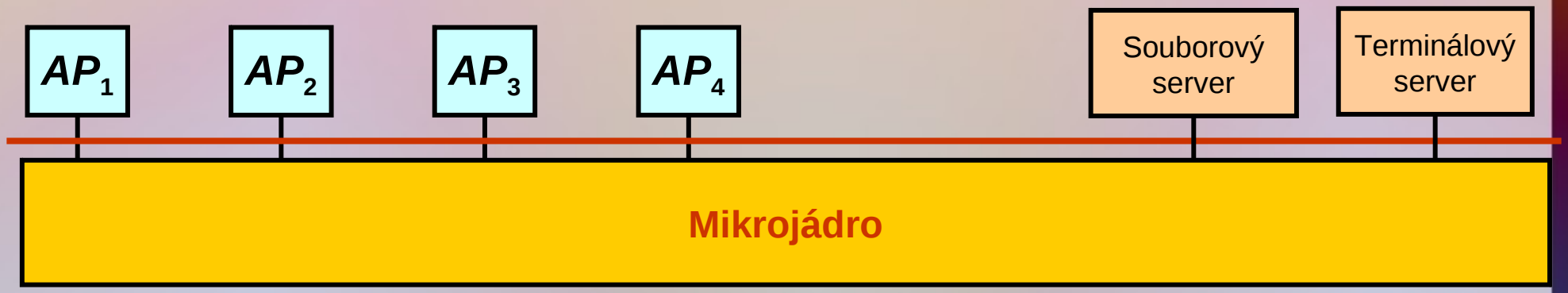
- Alternativní řešení



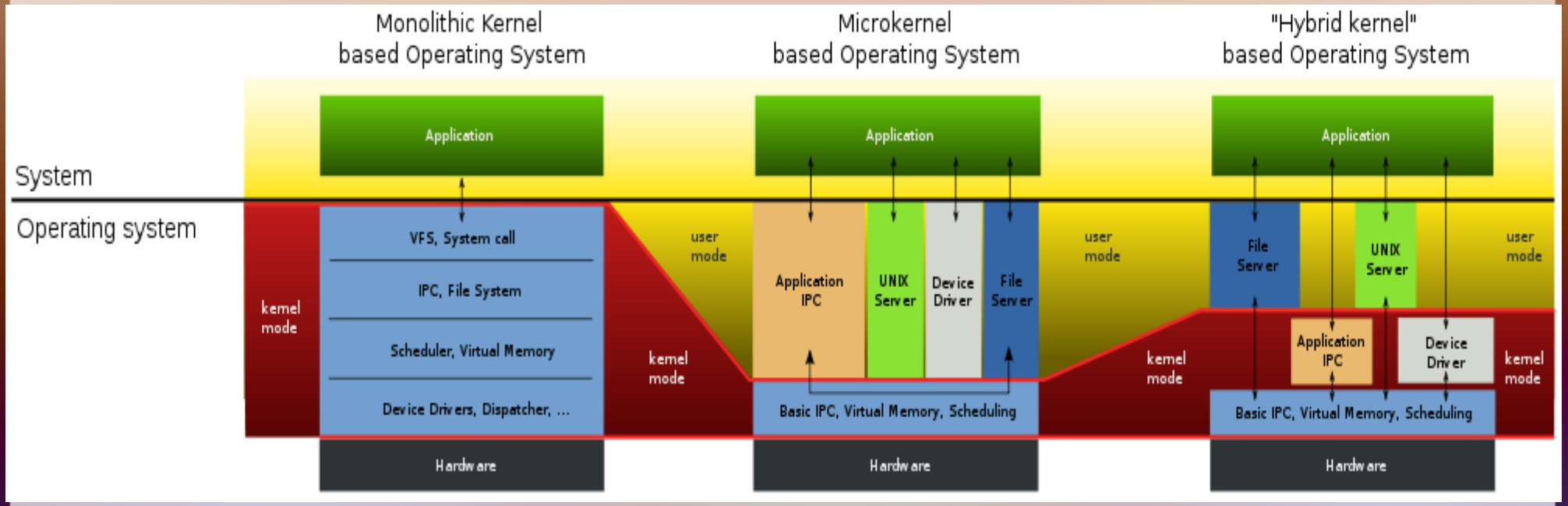
  - Synchronní přerušení se obsluhuje v režii procesu → minimalizace přepínání mezi procesy.
    - Používáno např. v UNIX SVR4
  - Uvnitř JOS používá každý proces samostatný zásobník
  - Kód a data JOS jsou ve sdíleném adresovém prostoru a jsou sdílena všemi procesy

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 pouze ústředna pro přepojování zpráv
  - Řešení snadno implementovatelné i na multiprocesorech
- Malé jádro => mikrojádro (µ-jádro) – (microkernel)



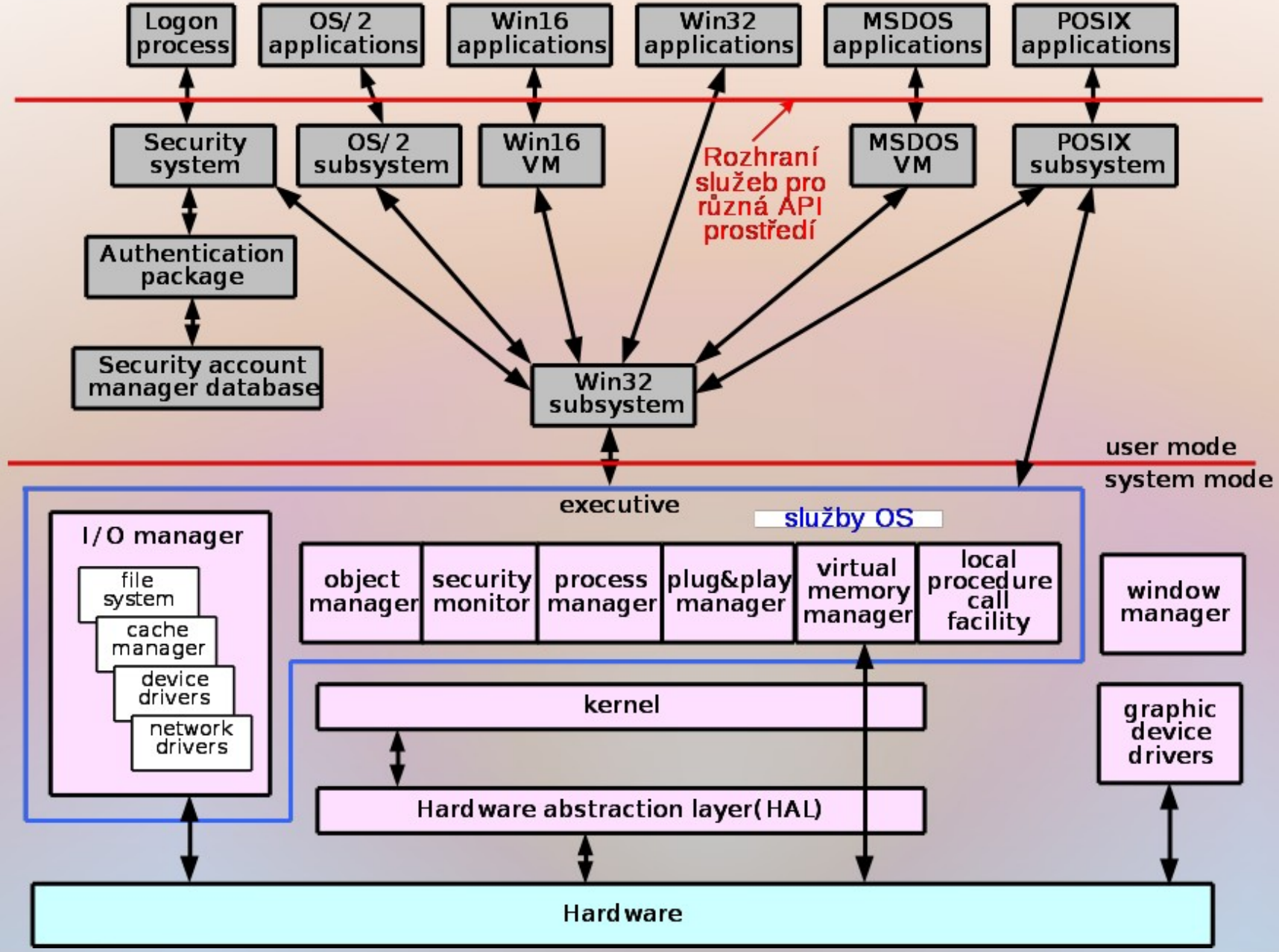
Porovnání mikrojádra a monolitického jádra



OS s µ-jádrem – výhody

- 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 v síti i uvnitř systému
- 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



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
  - Údaje jsou jen orientační, Microsoft data nezveřejňuje
    - SLOC (*Source Lines of Code*) je velmi nepřesný údaj: Tentýž programový příkaz lze napsat na jediný nebo celou řadu řádků.

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	30 mil.
Windows XP SP2	2005	**48 mil.**
Windows 7	2010	**???** (není známo)

To je dnes vše.

Otázky?