Typologie, funkční skladba a architektury OS, příklady z Windows, Unix, Linux, Android

PB 152 ◇ Operační systémy

Jan Staudek
http://www.fi.muni.cz/usr/staudek/vyuka/

Verze: jaro 2017

Osnova dodatku přednášky

- Windows
- Unix
- Linux
- Android

Modulové architektury

- Soudobé metodologie návrhu OS používají pro vytvoření (modulárního) jádra OS OO programovací techniky
- komponenty jádra jsou samostatné jednotky – moduly
- moduly mezi sebou komunikují přes známá rozhraní komunikace přitom není omezena na přísnou hierarchii
- každý modul je samostatně zaveditelný modul jádra, zavádí se, je-li potřebný
  - ✓ do jádra se přidá pro jistý hardware – driver sběrnice, IO zařízení
  - ✓ jako zaveditelné moduly se mohou doplnit podpory pro různé souborové systémy, programovací prostředí, ...

Zkušenosti z vývoje OS Windows

- Systematický přehled viz
  http://windows.microsoft.com/cs-cz/windows/history
- MS-DOS 1.0, 1981
  - ✓ 4000 řádků v assembleru
  - ✓ Intel 8086 microprocessor, provozovatelný i v 8 KB paměti
  - ✓ textově orientovaný jazyk příkazů pro OS, žádné GUI
  - ✓ monoprogramový, monouživatelský systém
- Windows 3.0, 1990
  - ✓ 16 bitová architektura
  - ✓ rozhraní GUI, implementace – vrstva nad MS-DOS

Zkušenosti z vývoje OS Windows

- Windows NT (3.1), 1993
  - ✓ 32 bitová architektura, mikrojádro, multitasking
  - ✓ podpora starších aplikací pro MS-DOS a Windows, pro OS/2 (IBM) a pro POSIX (Unix)
- Windows 95, pokračování Windows 3.0, 1995
  - ✓ 32 bitová architektura, monolitické jádro – poskytoval se vyšší výkon než NT
  - ✓ následný vývoj – Windows 98 a Windows Me a tím končí tato větev

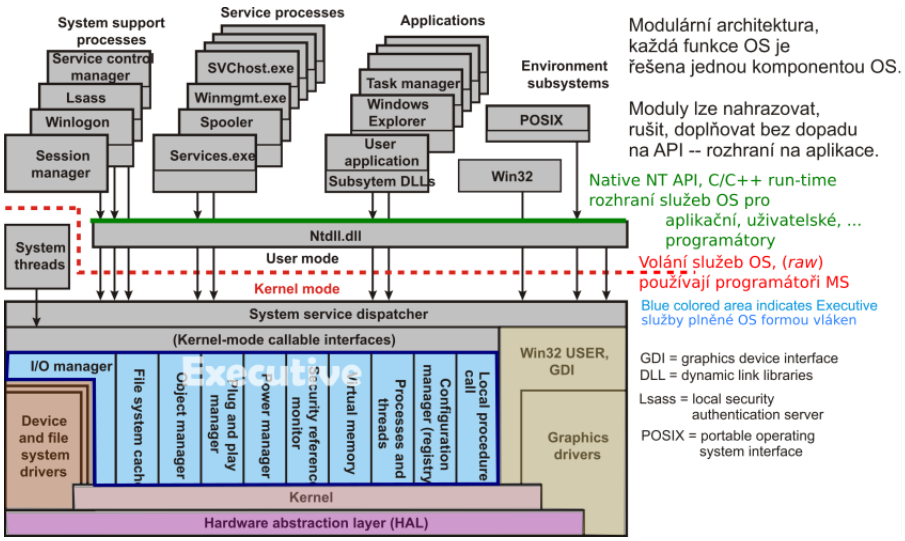
Zkušenosti z vývoje OS Windows

- Windows 2000, v linii NT
  - ✓ podpora distribuovaného zpracování dat
  - ✓ Active Directory – distribuovaný adresář *plug-and-play* a *power-management*
- Windows XP, 2001 – 2005
  - ✓ náhrada verzí Windows založených na MS-DOS verzí založenou na NT
  - ✓ návrat ke spíše monolitické architektuře
- Windows Vista, 2007, kosmetické změny
- Windows Server, 2008 – víceuživatelský systém
- Windows 7, 2009, kosmetické změny
- Windows 8, 2012, + cloud computing
- Windows 10, 2015, kosmetické změny

Windows Architecture

- Všechny verze Windows budované na bázi NT mají na námi uvedené úrovni shodnou strukturu
- Separace jádra a aplikačně orientovaného software
  - ✓ Jádro: privilegovaný režim, vlastní jádro (Kernel), Executive, drivery, vrstva abstrakce hardware
  - ✓ ostatní software běží v uživatelském režimu
- Modulární architektura
  - ✓ Každou funkci (službu) OS plní jedna komponenta OS
  - ✓ OS a aplikace funkci OS se zpřístupňují přes relevantní komponenty pomocí standardních rozhraní
  - ✓ Systémová data lze zpřístupňovat pouze přes příslušné služby OS
  - ✓ Každý modul lze odstranit, nahradit, inovovat bez přepisování celého systému a API

Windows Architecture



Windows Architecture, komponenty v režimu jádra

- Executive:
  - ✓ Základní služby poskytované operačním systémem
  - ✓ Správa paměti, procesů, vláken, IO
  - ✓ Bezpečnost
  - ✓ Výměna zpráv mezi procesy
  - ✓ Vláknová struktura, je možná souběžnost běhů služeb
- Kernel
  - ✓ správa procesorů
  - ✓ plánování vláken, přepínání kontextu procesů
  - ✓ správa výjimek a přerušení
  - ✓ synchronizace multiprocesoru
  - ✓ monolitické řešení,
    žádná vlákna jako v Executive a v uživatelské oblasti

Windows Architecture, komponenty v režimu jádra

- Hardware abstraction layer (HAL)
  - ✓ Konverze generické hardwarové platformy na konkrétně použitou hardwarovou platformu
  - ✓ Jednotný pohled komponent Executive a jádra na systémovou sběrnici, řadič DMA, řadič přerušení, časovač, řadič paměti a podporu SMP ve všech hardwarových platformách
- Device drivers
  - ✓ Dynamické knihovní podprogramy rozšiřující funkce Executive na konkrétní IO zařízení
  - ✓ Implementace softwarové podpory systému souborů
  - ✓ Síťové protokoly
- Windowing and graphics system
  - ✓ Implementace funkcí GUI

Windows Architecture, Executive modules

- I/O manager
  - ✓ framework pro zpřístupňování IO zařízení, navazování ovladačů
  - ✓ implementace IO API,
  - ✓ podpora bezpečnosti a pojmenovávání zařízení, síťových protokolů a systému souborů (se správcem objektů)
- File system cache manager
  - ✓ cache zvyšující výkon IO se soubory
  - ✓ dočasné uchovávání posledně modifikovaných dat v hlavní paměti
- Object manager
  - ✓ správce objektů pro Executive
  - ✓ objekty – reprezentace procesů, vláken, semaforů . . .

Windows Architecture, Executive modules

- Plug-and-play manager
  - ✓ Určuje drivery /ovladače nutné pro podporu konkrétních zařízení a zavádí je
- Power manager
  - ✓ správa energie při prostojích, vypínání, . . .
- Security reference monitor
  - ✓ prosazování pravidel pro řízení přístupu k ob jektů a generování zpráv pro audit
  - ✓ objekty – soubory, procesy, adresové prostory, IO zařízení, . . .
- Virtual memory manager
  - ✓ implementace konceptu virtuální paměti

Windows Architecture, Executive modules

- Process/thread manager
  - ✓ vytváří, ovládá, ruší objekty procesů a vláken
- Configuration manager
  - ✓ správce databáze *registry* uchovávající parametry systémových i uživatelských objektů
- Advanced local procedure call (ALPC) facility
  - ✓ volání procedur mezi procesy – komunikační nástroj mezi lokálními procesy, které implementují služby a subsystémy
  - ✓ ekv. RPC (*remote procedure call*) v distribuovaném prostředí

Windows Architecture, procesy v uživatelském režimu

- Special system processes, system support processes
  - ✓ udržování relací s uživateli (*sessions*)
  - ✓ autentizace, přihlašování, . . .
- Service processes
  - ✓ výpis na systémovou tiskárnu (spooler), záznamenávání událostí
  - ✓ uživatelská část ovladačů, . . .
  - ✓ prostor pro rozšiřování funkčnosti OS
- Environment subsystems
  - ✓ rozhraní služeb konkrétních operačních systémů
  - ✓ Win32 (Windows), POSIX (Unix)
  - ✓ překlad volání služeb na ALPC volání (*Advanced Local Procedure Call*) nebo na Native NT volání služeb

Windows Architecture, procesy v uživatelském režimu

- User applications
  - ✓ provedníschopné programy (.EXE, DLL), *Executables*

Windows Architecture, model klient – server

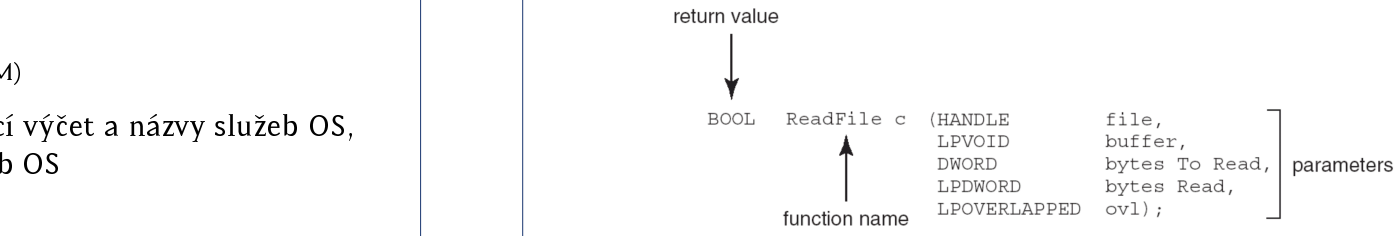
- *Windows OS services*, *environment subsystems* (POSIX, WIN32) a aplikace jsou strukturované do modelu klient–server
  - ✓ klienti komunikují se servery pomocí RPC
  - ✓ asymetrický model, server obsluhuje klienty (správa paměti, síťové služby, . . .)
- Klient
  - ✓ aplikace nebo jiný server
  - ✓ klient požádá o službu zasláním zprávy serveru, zprávu *Executive* doručí správnému serveru, server provede požadovanou službu a vrátí klientovi výsledek jinou zprávou
- Přínosy
  - ✓ zjednodušení *Executive*, zvýšení spolehlivosti, jednotné rozhraní na služby

Volání služeb systému, System Calls

- příklady API služeb OS
  - ✓ Win32 API pro Windows,
  - ✓ POSIX API (UNIX, Linux, Mac OS X),
  - ✓ Java API pro Java virtual machine (JVM)
- Neexistuje žádná norma specifikující výčet a názvy služeb OS, každý OS má svoji sestavu služeb OS
- Java – platforma nezávislá na OS
  - ✓ nelze volat služby OS přímo z javovských programů
  - ✓ řeší se nepřímo, voláním C/C++ funkcionality nativní pro daný OS

Ilustrace standardního API – Win32

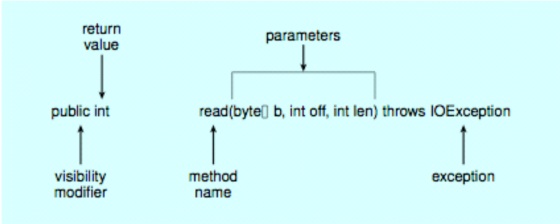
- ✓ Funkce ReadFile()
- ✓ čtení ze souboru dat



- ✓ HANDLE file – jméno souboru, ze kterého se čte
- ✓ LPVOID buffer – cílová vyrovnávací paměť
- ✓ DWORD bytesToRead – délka vyrovnávací paměti
- ✓ LPDWORD bytesRead – délka přečtených dat
- ✓ LPOVERLAPPED ovl – čekat / nečekat na konec operace

Ilustrace standardního Java API

- ✓ metoda read() z třídy java.io.InputStream
- ✓ metoda vrací int reprezentující počet přečtených bytů



- ✓ IOException – odbočka pro řešení IO chyby
- ✓ byte [] b – cílový buffer
- ✓ int off – počáteční offset v b, kam se zapisují data
- ✓ int len maximum čtených bytů

Příklady služeb POSIX (knihovna C)

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

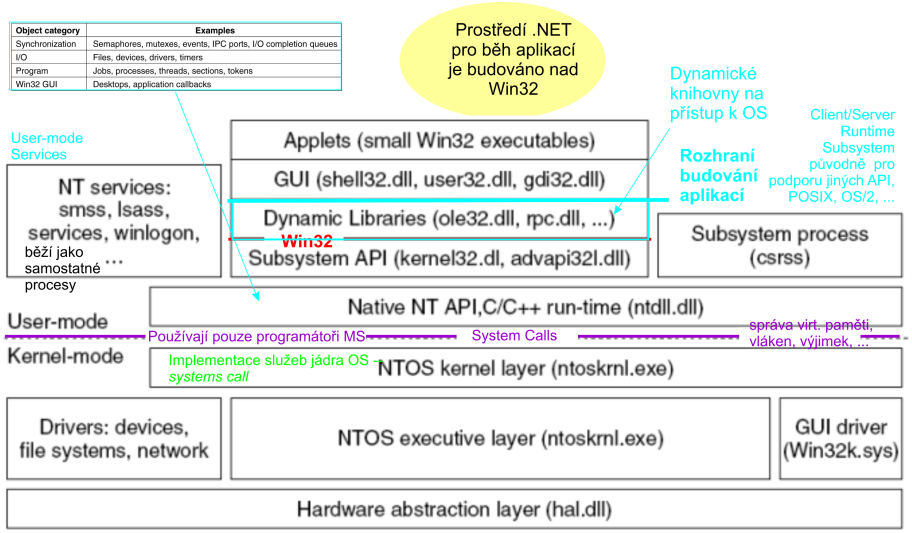
Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Příklady služeb POSIX (knihovna C)

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Rozhraní programů Windows



Subsystémy, DLL, služby

- Subsystémy
  - ✓ původní řešení emulací rozhraní služeb POSIX a OS/2
- DLL, *Dynamic Link Library*
  - ✓ dynamicky, při běhu procesu, zaváděné knihovní programy, nikoli při kompilaci či sestavování
- User mode services, také *NT Services*
  - ✓ analogie služeb implementovaných v jádru
  - ✓ rozšíření funkcionality systému
  - ✓ např. *lsass.exe*, *local service authentication service taskmgr.exe*, generuje tabulkový seznam bežících služeb a aplikací
  - ✓ snadno se napadají, jsou dostupné vzdáleně
  - ✓ množství trvale běžících služeb je ,,obrovské"
  - ✓ představují režii

Příklady volání Native NT API

NtCreateProcess(&ProcHandle, Access, SectionHandle, DebugPortHandle, ExceptPortHandle, ...)
NtCreateThread(&ThreadHandle, ProcHandle, Access, ThreadContext, CreateSuspended, ...)
NtAllocateVirtualMemory(ProcHandle, Addr, Size, Type, Protection, ...)
NtMapViewOfSection(SectHandle, ProcHandle, Addr, Size, Protection, ...)
NtReadVirtualMemory(ProcHandle, Addr, Size, ...)
NtWriteVirtualMemory(ProcHandle, Addr, Size, ...)
NtCreateFile(&FileHandle, FileNameDescriptor, Access, ...)
NtDuplicateObject(srcProcHandle, srcObjHandle, dstProcHandle, dstObjHandle, ...)

- Příklady volání používající *handles* pro manipulaci s objekty mezi procesy
  - ✓ madlo, reprezentace jiné složitější struktury

Rozhraní Win32 API

- veřejně dostupné, plně publikované funkční rozhraní pro tvorbu aplikací
- knihovní podprogramy
  - ✓ buď problém řeší přímo nebo pomocí služeb Native NT calls
- V původním řešení Windows se podporovala další dvě rozhraní
  - ✓ POSIX – volání služeb identické s prostředím Unix
  - ✓ OS2 – volání služeb identické s prostředím OS/2

Rozhraní Win32 API, příklady

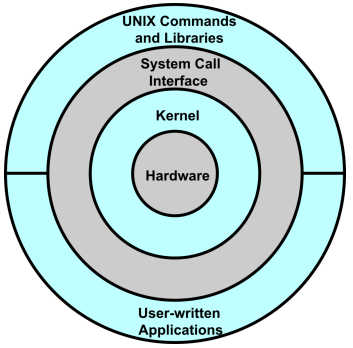
Win32 call	Native NT API call
CreateProcess	NtCreateProcess
CreateThread	NtCreateThread
SuspendThread	NtSuspendThread
CreateSemaphore	NtCreateSemaphore
ReadFile	NtReadFile
DeleteFile	NtSetInformationFile
CreateFileMapping	NtCreateSection
VirtualAlloc	NtAllocateVirtualMemory
MapViewOfFile	NtMapViewOfSection
DuplicateHandle	NtDuplicateObject
CloseHandle	NtClose

Platforma .NET (.NET Framework)

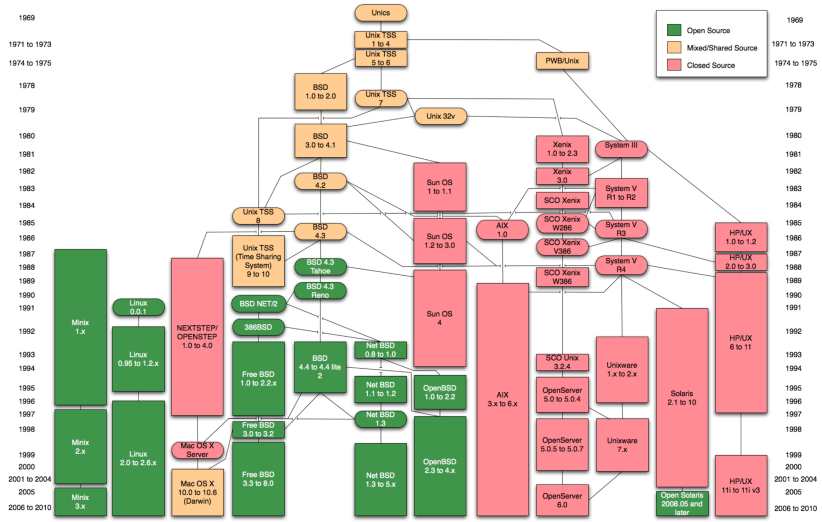
- Microsoft
- virtuální stroj, pro který lze psát programy nezávislé na architektuře systému tento stroj hostujícího
- program napsaný pro .NET Framework se nestará o to, na jakém počítači s jakým OS běží
- .NET virtuální stroj – CLR, *Common Language Runtime*
- programy napsané v C# nebo ve VB.NET jsou kompilovány na tzv. assemblies
- při spuštění programu jsou assemblies v CLR (just-in-time compiler) přeloženy do nativního kódu hostujícího systému

Tradiční systémy typu Unix

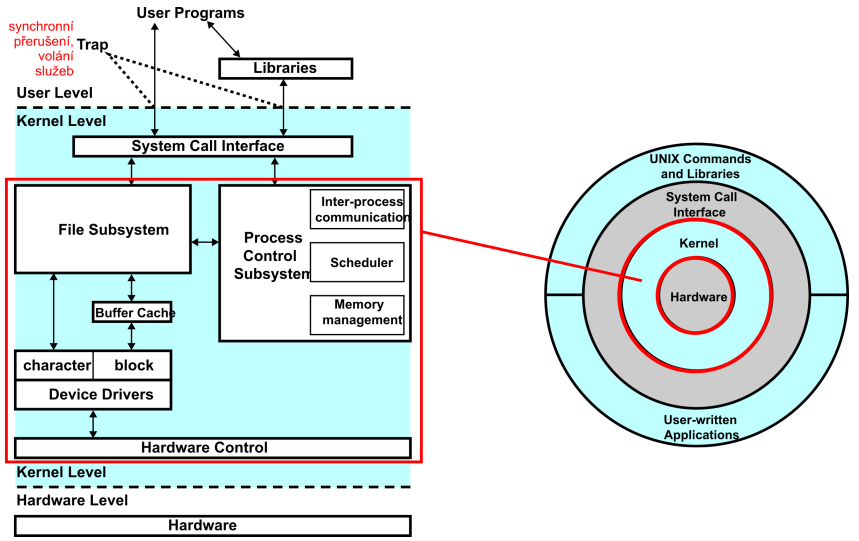
- Vznik Bell Labs PDP-7, 1970, . . . , 1978 Verze 7 – základ dnešních Unixů, . . . , UNIX System V
- Paralelní větev – University of California at Berkeley, UNIX BSD (*Berkeley Software Distribution*)



Tradiční Unix, historie

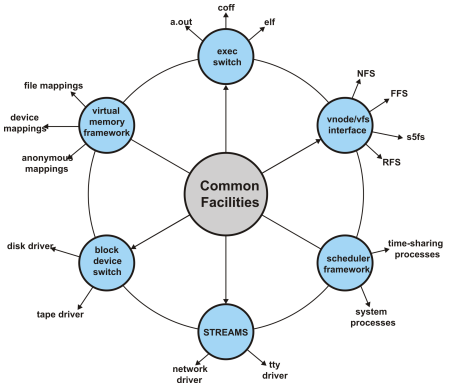


Tradiční Unix, jádro



Soudobé Unixy, jádro

- System V Release 4 (SVR4) / Solaris,
  Berkeley Software Distribution (BSD)
- Na FreeBSD 5.0 a mikrojádru Mach 3.0 je založený Mac OS X



Modulová koncepce

Moduly řeší funkce a služby požadované procesy OS a odpovídající rozhraní

Moduly mohou být implementované různými způsoby

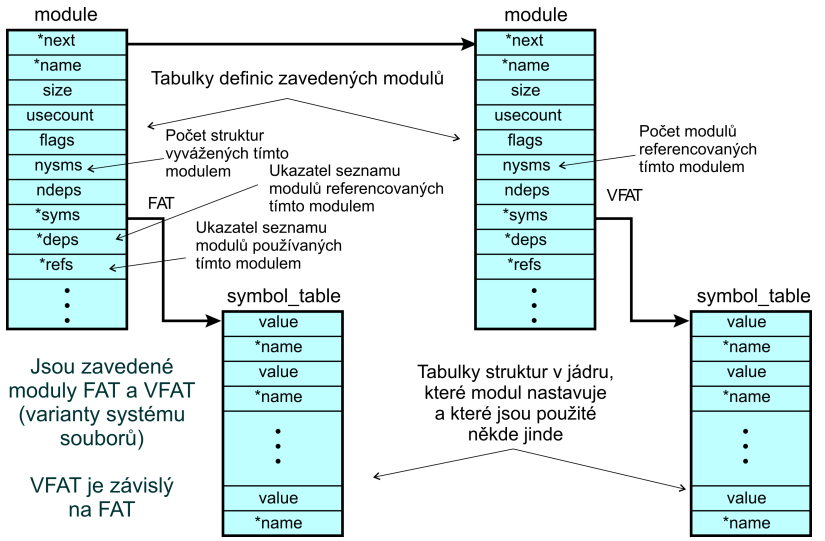
Linux

- Původně Unix pro IBM PC s procesorem Intel 80386
- Od. r. 1991 otevřený projekt,
  jsou dostupné zdrojové programy
  - ✓ pod záštitou *Free Software Foundation* (*FSF*).
- V současnosti plnohodnotná varianta Unixu
- Dostupná na více platformách
  - ✓ Intel Pentium, Itanium, Motorola, IBM PowerPC, . . .
- Vysoce modulární koncepce, snadná konfigurovatelnost

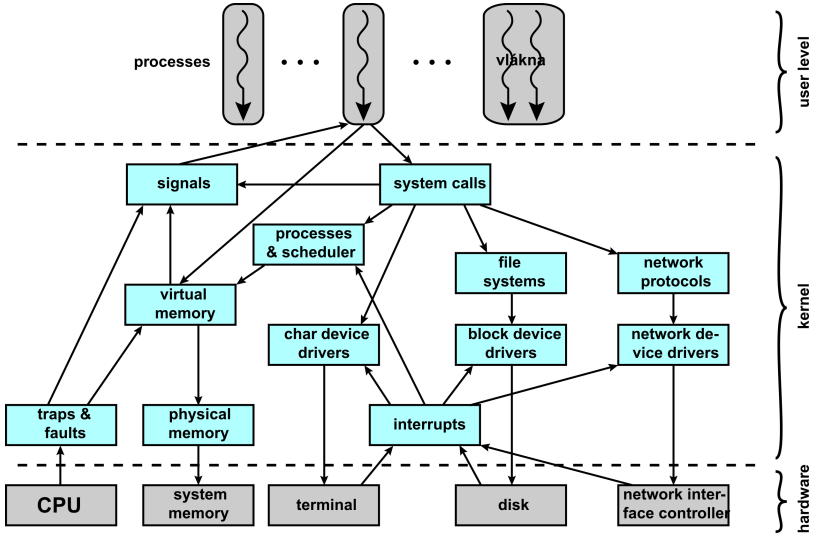
Linux, modulární monolitické jádro

- Není aplikován koncept mikrojádra
  - ✓ Jádro virtuálně obsahuje veškerou funkčnost OS v jednom velkém bloku programu, běží jako jeden proces s jedním adresovým prostorem
  - ✓ Všechny funkční komponenty mají přístup ke všem vnitřním datovým strukturám a programům
- Moduly jádra lze zavádět automaticky a na žádost odstraňovat, jsou relativně nezávislými bloky
  - ✓ moduly jádra – zaveditelné moduly (*Loadable Modules*)
  - ✓ modul je objekt (soubor), jehož kód lze při běhu dynamicky navazovat a odstraňovat do / z jádra
  - ✓ moduly jsou uspořadatelné hierarchicky (*Stackable Modules*)
- Modul jádra je řešený na základě pokynu běžícího procesu

Ilustrace seznamu zavedených modulů jádra Linuxu



Komponenty jádra Linuxu (implementace na arch. IA-64)



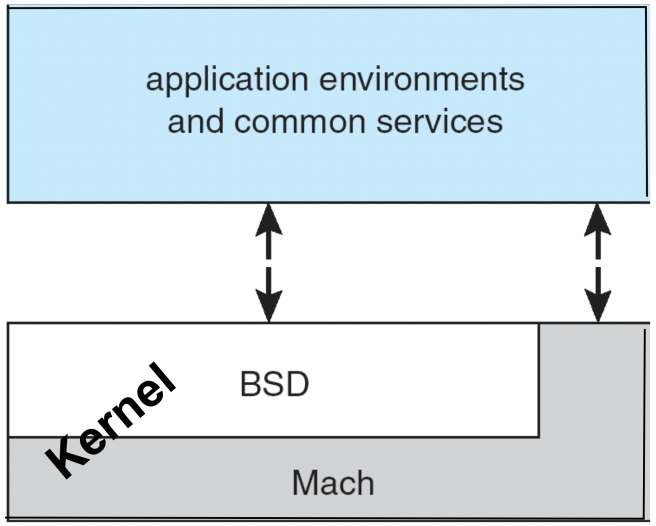
Linuxovské signály

SIGHUP	Terminal hangup	SIGCONT	Continue
SIGQUIT	Keyboard quit	SIGTSTP	Keyboard stop
SIGTRAP	Trace trap	SIGTTOU	Terminal write
SIGBUS	Bus error	SIGXCPU	CPU limit exceeded
SIGKILL	Kill signal	SIGVTALRM	Virtual alarm clock
SIGSEGV	Segmentation violation	SIGWINCH	Window size unchanged
SIGPIPT	Broken pipe	SIGPWR	Power failure
SIGTERM	Termination	SIGRTMIN	First real-time signal
SIGCHLD	Child status unchanged	SIGRTMAX	Last real-time signal

Modulová architektura MAC OS X (*Darwin*)

- hybridní struktura
  - ✓ vrstvová struktura
  - ✓ spodní vrstva – mikrojádro Mach
  - ✓ horní vrstvy –
    aplikační prostředí a obecné služby grafického rozhraní pro aplikace
- Jádro
  - ✓ mikrojádro Mach – volání vzdálených procedur (RPC, *Remote Procedure Call*), meziprocesová komunikace (IPC, *InterProcess Communication*), správa paměti, výměna zpráv, dispečer
  - ✓ jádro BSD – rozhraní na unixovské (BSD) příkazy, podpora síťování (sockets), systém souborů, API definovaná v POSIX
    vč. vláken Pthreads
  - ✓ rozšíření jádra –
    IO kit pro vývoj driverů a dynamicky zaváděných modulů
- Aplikace a obecné služby si zpřístupňují vlastnosti BSD a Mach přímo

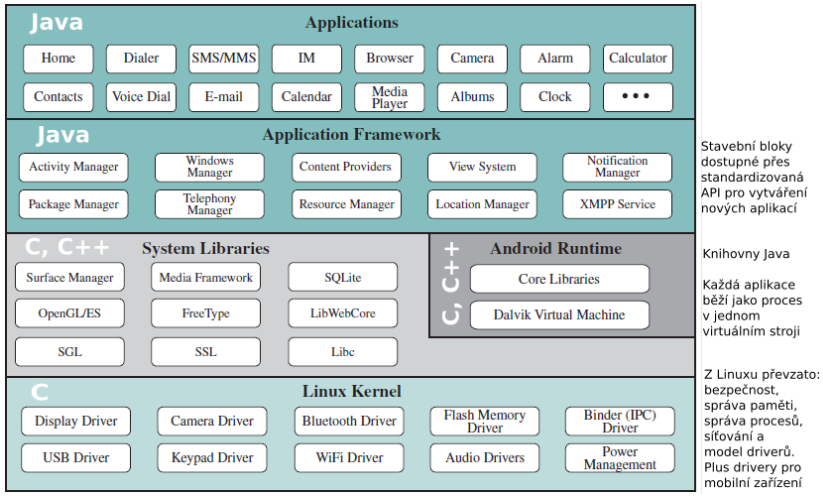
Modulová architektura MAC OS X (*Darwin*)



Android

- OS pro počítače s dotykovou obrazovkou na bázi Linuxu
- Protokolový zásobník nad Linuxem, nikoli úplný OS
  - ✓ Android je v podstatě forma vestavění Linuxu
    do prostředí mobilního výpočetního zařízení
  - ✓ Zjednodušený Linux, zachovaný preemptivní multitasking

Android, protokolový model, *protocol stack*



Android

- Application Framework
  - ✓ Activity Manager: start, zastavení, obnovení aplikace
  - ✓ Window Manager: Javovské rozhraní ovládání oken
  - ✓ Package Manager: Instalace, rušení aplikací
  - ✓ Telephony Manager: Interakce s telefonem, se službami SMS MMS
  - ✓ Content Providers: Sdílení dat mezi aplikacemi
  - ✓ Resource Manager: Práce s lokalizovanými řetězci a bitmapami
  - ✓ View System: Gesta, tlačítka, ...
  - ✓ Location Manager: Napojení na služby GPS, WiFi, ...
  - ✓ Notification Manager: Správce událostí, příchozí zprávy, akce, ...
  - ✓ XMPP: Standardizovaný messaging (např. Chat)

Android

- System Libraries
  - ✓ Surface Manager: správce oken na nízké úrovni
  - ✓ OpenGL: API pro ztvárnění 2D a 3D grafiky
  - ✓ Media Framework: podpora formátů pro záznam a přehrávání
  - ✓ SQL Database: uchovávání persistentních dat
  - ✓ Browser Engine: zobrazování HTML obsahů
  - ✓ Bionic LibC: varianta standardní knihovny C systému se standardním rozhraním Java Native Interface (JNI)

Android, pohled vývojáře aplikace

- Vývojáři postačí API z aplikačního frameworku, umožní mu přístup ke službám nižších vrstev
- Pro volání systémových sluýeb Androidu má nástroje IPC (Interprocess Communication)
- Většina funkčností viditelná přes API aplikačního frameworku vyvolává systémové služby
- K driverům se přistupuje přes HAL (Hardware Abstraction Layer), standardizované rozhraní driverů v jádru

Android, pohled vývojáře aplikace