

B4B35OSY: Operační systémy

Lekce 2. Systémové volání

Petr Štěpán

stepan@fel.cvut.cz



September 18, 2017

Outline

- 1 Složení OS
- 2 Služby OS
- 3 Struktura OS

Outline

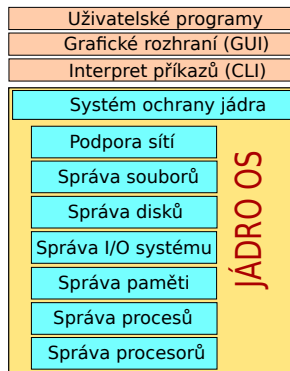
1 Složení OS

2 Služby OS

3 Struktura OS

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

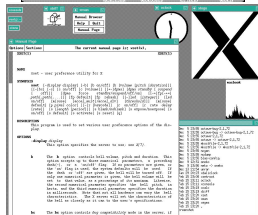
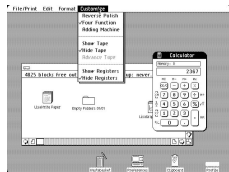


Interpret příkazů

- Většina zadání uživatele je předávána operačnímu systému řídicími 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)



Systemové 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
- Systemové programy jsou v rámci OS řešeny formou výpočetních/uživatelských procesů, ne jako služby OS

Jádro OS

- Poskytuje ochranu/izolaci
 - Aplikačních programů mezi sebou
 - Hardwaru před škodlivými aplikacemi
 - Dat (souborů) před neoprávněnou manipulací
- Řídí přidělování zdrojů aplikacím
 - Paměť, procesorový čas, přístup k HW, síti, ...
- Poskytuje aplikacím služby
 - Jaké?

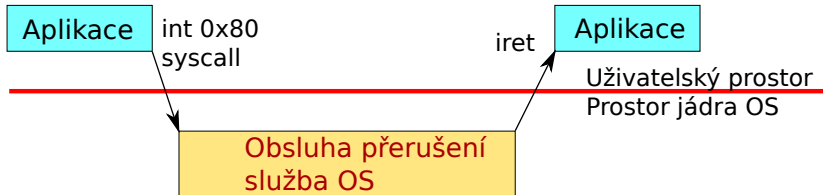
Ochrana jádra OS

- Ochrana jádra
 - 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 x86 rozlišuje 4 úrovně ochrany (privilege level): 0 – jádro OS, 3 – uživatelský mód
 - Jiné architektury mají většinou jen dva módy (jeden bit ve stavovém slově)
 - V uživatelském módu jsou některé instrukce zakázané (opakování - jaké?)
- Přechod z uživatelského módu do systémového
 - pouze programově vyvolaným přerušením
 - speciální instrukce (trap, int, sysenter, swi, ...)
 - nejde spustit cokoliv, spustí se pouze kód povolený operačním systémem
 - Systémové volání – služby jádra (system calls)
- Přechod ze systémového módu do uživatelského
 - Speciální instrukce či nastavení odpovídajících bitů ve stavovém slově FLAGS
 - návrat z přerušení

Ochrana jádra OS



Outline

1 Složení OS

2 Služby OS

3 Struktura OS

Služby jádra OS

x86 System Call Example - Hello World on Linux

```
.section .rodata
greeting:
    .string "Hello_\World\n"

.text
.global \_start
\_start:
    mov $4,%eax /* write is syscall no. 4 */
    mov $1,%ebx /* file descriptor – 1 == stdout */
    mov $greeting,%ecx /* address of the data */
    mov $12,%edx /* length of the data */
    int $0x80 /* call the system */
```

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 mezi pamětí jádra a uživatelským prostorem
 - malá data lze přenést v registrech – návratová hodnota funkce
 - velká data – uživatel musí připravit prostor, jádro z/do něj nakopíruje data, předává se pouze adresa (ukazatel)
- Volání služby jádra na strojové úrovni není komfortní
 - Je nutné použít assembler, musí být dodržena volací konvence
 - Zapouzdření pro programovací jazyky – API
 - Základem je běhová knihovna jazyka C (libc, C run-time library)
- 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/>

ABI

- Application Binary Interface
- Definuje rozhraní na úrovni strojového kódu:
 - V jakých registrech se předávají parametry
 - V jakém stavu je zásobník
 - Zarovnání vícebytových hodnot v paměti
- ABI se liší nejen mezi OS, ale i mezi procesorovými architekturami stejného OS.
 - Např: Linux i386, amd64, arm, ...
 - Možnost podpory více ABI: int 0x80, systenter, 32/64 bit

ABI Linuxu

32 bitový systém (i386):
instrukce `int 0x80`

Popis	Registr
číslo <code>syscall</code>	<code>eax</code>
první argument	<code>ebx</code>
druhý argument	<code>ecx</code>
třetí argument	<code>edx</code>
čtvrtý argument	<code>esi</code>
pátý argument	<code>edi</code>
šestý argument	<code>ebp</code>

64 bitový systém (amd64):
instrukce `syscall`

Popis	Registr
číslo <code>syscall</code>	<code>rax</code>
první argument	<code>rdi</code>
druhý argument	<code>rsi</code>
třetí argument	<code>rdx</code>
čtvrtý argument	<code>r10</code>
pátý argument	<code>r9</code>
šestý argument	<code>r8</code>

API

Standardy pro soustavy služeb OS (system calls)

- Rozhraní systémových služeb – API (Application Programming Interface)
- Definuje rozhraní na úrovni zdrojového kódu
 - Jména funkcí, parametry, návratové hodnoty, datové typy
- POSIX (IEEE 1003.1, ISO/IEC 9945)
 - Specifikuje nejen system calls ale i rozhraní standardních knihovnických 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 MS Windows
- Nesystémová API:
 - 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 systémové ABI a vykoná instrukci pro systémové volání
- Ta přepne CPU do privilegovaného režimu a předá řízení do vstupního bodu jádra
- Podle kódu požadované služby jádro zavolá funkci implementující danou službu (tabulka ukazatelů)
- Po provedení služby se řízení vrací aplikačnímu programu s případnou indikací úspěšnosti

POSIX

- Portable Operating System Interface for Unix – 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

UNIX

- Operační systém vyvinutý v 70. letech v Bellových laboratořích
- Protiklad tehdejšího OS Multix
- Motto: **V jednoduchosti je krása**
- Ken Thompson, Dennis Ritchie
- Pro psaní OS si vyvinuli programovací jazyk C
- Jak UNIX tak C přežilo do dnešních let
- Linux, FreeBSD, *BSD, GNU Hurd, VxWorks, ...

Unix v kostce

- Všechno je soubor
- Systémová volání pro práci se soubory:
 - `open(pathname, flags)` → file descriptor (celé číslo)
 - `read(fd, data, délka)`
 - `write(fd, data, délka)`
 - `ioctl(fd, request, data)` – vše ostatní co není read/write
 - `close(fd)`
- Souborový systém:
 - `/bin` – aplikace
 - `/etc` – konfigurace
 - `/dev` – přístup k hardwaru
 - `/lib` – knihovny

POSIX dokumentace

- Druhá kapitola manuálových stránek
- Příkaz (např. v Linuxu): `man 2 ioctl`

`ioctl(2)` - Linux man page

Name

`ioctl` - control device

Synopsis

```
#include <sys/ioctl.h>
```

```
int ioctl(int d, int request, ...);
```

Description

The `ioctl()` function manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g., terminals) may be controlled with `ioctl()` requests. The argument `d` must be an open file descriptor. The second argument is a device-dependent request code. The third argument is an untyped pointer to memory. It's traditionally `char *argp` (from the days before `void *` was valid C), and will be so named for this discussion.

POSIX dokumentace

Pokračování

An `ioctl()` request has encoded in it whether the argument is an in parameter or out parameter, and the size of the argument `argp` in bytes. Macros and defines used in specifying an `ioctl()` request are located in the file `<sys/ioctl.h>`.

Return Value

Usually, on success zero is returned. A few `ioctl()` requests use the return value as an output parameter and return a nonnegative value on success. On error, `-1` is returned, and `errno` is set appropriately.

Errors

`EBADF` `d` is not a valid descriptor.

`EFAULT` `argp` references an inaccessible memory area.

`EINVAL` Request or `argp` is not valid.

`ENOTTY` `d` is not associated with a character special device.

`ENOTTY` The specified request does not apply to the kind of object that the descriptor `d` references.

Notes

In order to use this call, one needs an open file descriptor. Often the `open(2)` call has unwanted side effects, that can be avoided under Linux by giving it the `O_NONBLOCK` flag.

See Also

`execve(2)`, `fcntl(2)`, `ioctl_list(2)`, `open(2)`, `sd(4)`, `tty(4)`

Přehled služeb jádra

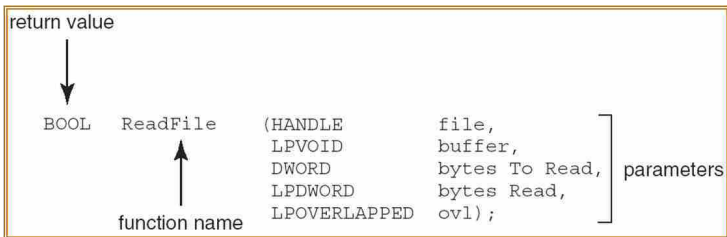
- Práce se soubory
 - open, close, read, write, lseek, stat
- Správa souborů a adresářů
 - mkdir, rmdir, link, unlink, mount, umount, chdir, chmod
- Správa procesů
 - fork, waitpid, execve, exit, kill, signal

Windows API

- Nebylo plně popsáno, skrytá volání využívaná pouze spřátelenými stranami
- MS developers mají privilegovaný přístup k popisu systémových volání
- Win16 – 16 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 přečíslovat staré služby.

Windows API příklad

- 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 blokové čtení

Porovnání POSIX a Win32

POSIX	Win32	Popis
fork	CreateProcess	Vytvoř nový proces
waitpid	WaitForSingleObject	Čeká 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
seek	SetFilePointer	Posouvá ukazatel v souboru
stat	GetFileAttributesExt	Vrací informace o souboru
mkdir	CreateDirectory	Vytvoří nový adresář
rmdir	RemoveDirectory	Smaže adresář souborů
link	–	Win32 nepodporuje symbolické odkazy
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 funkcionality je řešena jiným způsobem.

Outline

1 Složení OS

2 Služby OS

3 Struktura OS

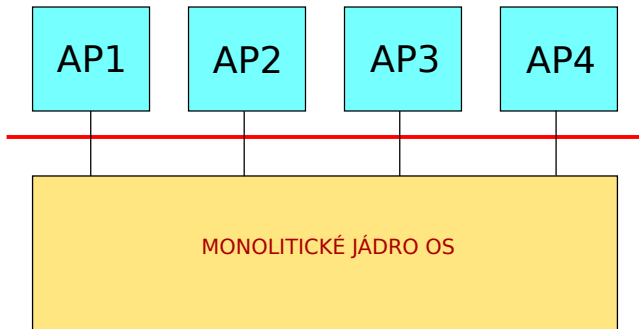
Vykonání služeb jádra OS

- 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
 - „USB MIDI má přístup ke klíči k šifrování disku :)“ CVE-2016-2384
- Služba OS je typicky implementována jako kód v jádře, běžící „v kontextu“ daného procesu/vlákná
- Některé služby jsou částečně (nebo i úplně) implementovány v uživatelském prostoru systémovou knihovnou.
- Služba/systémové volání
 - Procesor se přepne do systémového režimu, nepřepíná se však kontext volajícího procesu/vlákná
 - K přepnutí kontextu (přechodu od jednoho procesu k jinému: proces1 – OS – proces2) dochází jen, je-li to nutné z hlediska plánování procesů po dokončení služby (např. volání „sleep“).

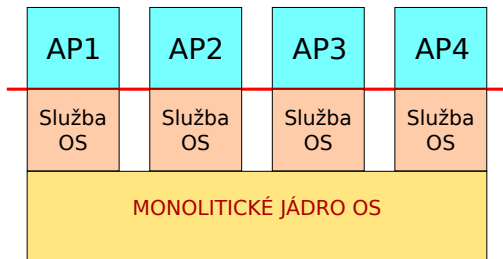
Služba OS plně součást JOS

Tradiční řešení

Jádro OS poskytuje služby pro každý z aplikačních programů AP_i a operuje jako jediná společná entita pracující v privilegovaném režimu.



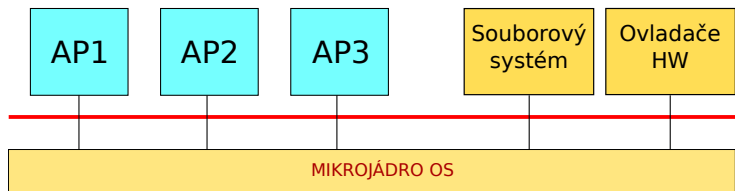
Služba OS jako součást procesu



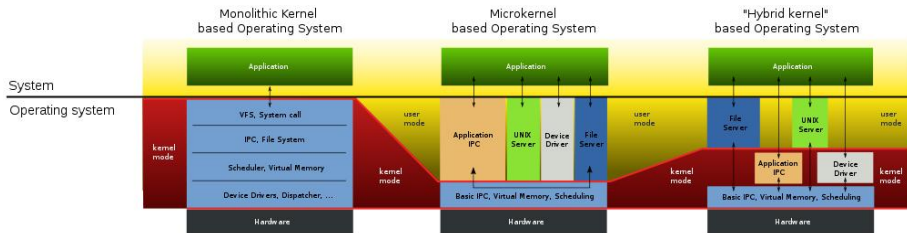
- Synchronní přerušení se obsluhuje v režii procesu → minimalizace přepínání mezi procesy.
- Používáno např. v UNIX SVR4
- Uvnitř JOS používá každý proces samostatný zásobník
- Kód a data JOS jsou ve sdíleném adresovém prostoru a jsou sdílena všemi procesy
- Nutnost implementovat bezpečně s ohledem na paralelní běh procesů - thread safe

Procesově orientované jádro OS

- 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 multiprocesech
- Malé jádro \Rightarrow mikrojádro (μ -jádro) – (microkernel)



Porovnání JOS

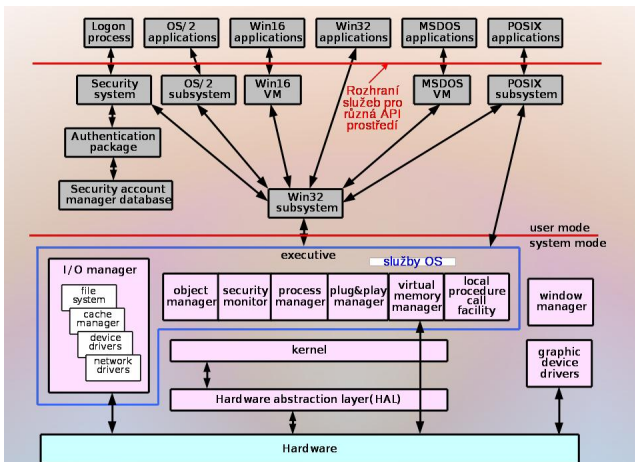


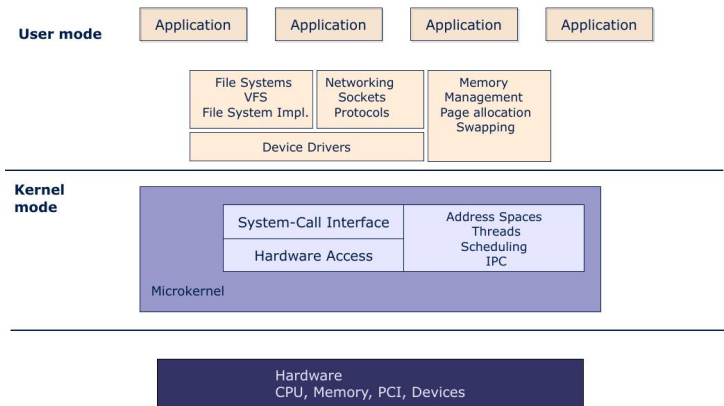
Mikrojádro - vlastnosti

- 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 -jádro
- To vše za cenu
 - zvýšené režie, volání služeb je nahrazeno výměnou zpráv mezi aplikačními a systémovými procesy

Příklad OS s μ -jádre

Windows XP (podle marketindových letáků MS)



L4Re - OS s μ -jádrém

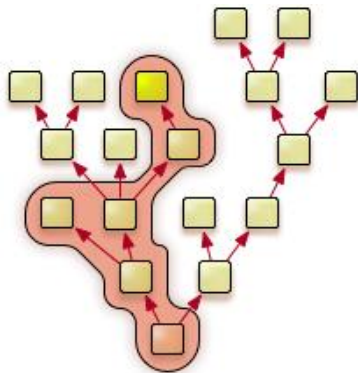
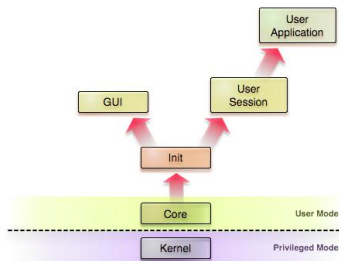
<http://os.inf.tu-dresden.de>

<http://www.kernkonzept.com/>

Genode - OS s μ -jádrem

Jeden z cílů: Omezit velikost “Trustued computing base”

<http://genode.org/>



Nova - OS s μ -jádroem

Systemová volání OS Nova:

- call
- reply
- create_pd
- create_ec
- create_sc
- create_pt
- create_sm
- revoke
- lookup
- ec_ctrl
- sc_ctrl
- pt_ctrl
- sm_ctrl

- assign_pci
- assign_gsi

Výukový OS - bude používán na cvičení

- Víc jich není
- PD - protection domain - proces
- EC - execution context
- SC - scheduling context
- PT - portal
- SM - semafor

Závěr

OS jsou funkčně velmi složité

OS	Rok	Počet služeb JOS
Unix	1971	33
Unix	1979	47
Sun OS4.1	1989	171
4.3 BSD	1991	136
Sun OS4.5	1992	219
Sun OS5.6 (Solaris)	1997	190
Linux 2.0	1998	229
WinNT 4.0	1997	3443

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

Závěr

OS jsou velmi rozsáhlé

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	SLOC
Windows 3.1	1992	3mil.
Windows NT 3.5	1993	4mil.
Windows 95	1995	15mil.
Windows NT 4.0	1997	16mil.
Windows 98 SR-2	1999	18mil.
Windows 2000 SP5	2002	30mil.
Windows XP SP2	2005	48mil.
Windows 7	2010	není známo