



PB153

Operační systémy a jejich rozhraní

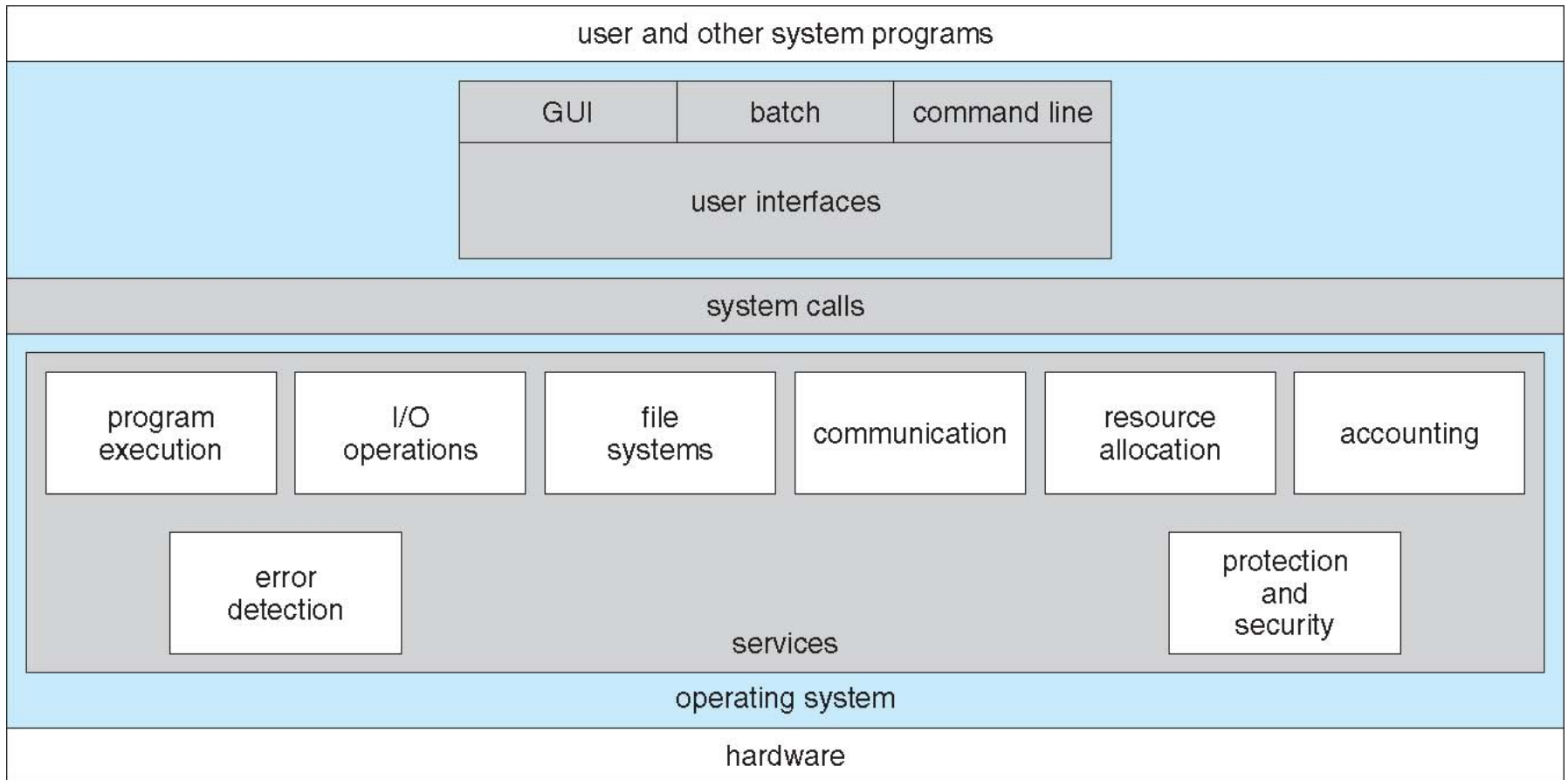
Struktura a rozhraní OS



Komponenty OS

- Správa procesů
- Správa operační paměti
- Správa souborů
- Správa I/O zařízení
- Správa sekundárních pamětí
- Správa síťových služeb
- Ochranný systém
- Interpret příkazů (shell)

Struktura OS





Správa procesů

- Proces (aktivní entita) je spuštěný program (pasivní entita)
 - jeden program může být zároveň spuštěn řadou uživatelů
 - proces je jednotkou práce systému
 - procesy jádra OS
 - uživatelské procesy
- Proces pro svou činnost potřebuje zdroje
 - procesor, paměť, soubory
 - zdroje se alokují při spuštění nebo při běhu

● ● ● | Správa procesů (2)

- Typické aktivity OS v oblasti správy procesů
 - vytváření a ukončování uživatelských a systémových procesů
 - potlačování (suspending) a obnovování (resuming) procesů
 - mechanismy pro synchronizaci procesů
 - mechanismy pro komunikaci mezi procesy
 - mechanismy pro detekci a řešení uváznutí (deadlock)

Správa operační paměti

- Při spuštění procesu musíme program nahrát do paměti
- Později může proces vyžadovat dodatečnou paměť pro data a tuto paměť OS vracet
- Jakmile proces končí, musí OS veškerou paměť užívanou procesem opět uvolnit

Správa operační paměti

- Typické aktivity OS v oblasti správy paměti
 - znalost, která část paměti je využívána a kým (kterým procesem)
 - přidělování (alokace) a uvolňování (dealokace) paměti podle požadavků procesů
 - rozhodování o tom, který proces kdy zavést do paměti



Správa souborů

○ Soubor

- kolekce souvisejících informací definovaná svým tvůrcem
 - může mít strukturu, ale nemusí
 - z hlediska OS typicky jen posloupnost bajtů
- ## ○ I/O zařízení na kterých jsou soubory uloženy mohou být nejrůznějšího typu
- magnetické disky, optické disky, magnetické pásky
 - OS zavádí abstraktní koncept souboru

Správa souborů (2)

- Typické aktivity OS v oblasti správy souborů
 - vytváření a rušení souborů
 - vytváření a rušení adresářů (složek)
 - základní operace pro manipulaci se soubory a adresáři (např. čtení ze souboru, zápis do souboru, seznam souborů v adresáři)
 - zálohování

Správa I/O zařízení

- Snaha o skrytí specifik jednotlivých I/O zařízení
 - co nejjednodušší přístup k jednotlivým I/O zařízením
 - mnoho OS zpřístupňuje I/O zařízení přes speciální soubory
 - /dev/sda, /dev/lp0 v UNIXu
 - \\.\PHYSICALDRIVE2, CONIN\$ ve Win32
- Ovladače jednotlivých HW komponent
- Řízení bufferů, kešování, spooling

Správa sekundární paměti

- Typické sekundární paměti jsou disky
- Správa sekundárních pamětí obvykle formou souborů – souborového systému
- Typické aktivity OS
 - správa volného místa
 - přidělování místa
 - plánování činnosti disku (které požadavky kdy vyřídit)

● ● ● | Správa síťových služeb

- Komunikace je řízena protokolem
 - protokol je několikastranný algoritmus pro dosažení určitého cíle
- Snaha o transparentnost
 - síťové souborové systémy (SMB, NFS, ...)
 - API podobné jako přístup k souborům



Ochranný systém

- V multitaskingovém a multiuživatelském OS musíme jednotlivé procesy navzájem chránit
- **Ochrana** je mechanismus, který řídí přístup programů, procesů a uživatelů ke zdrojům počítačového systému
 - HW za pomoci OS zajišťuje, že proces může používat pouze adresy svého adresového prostoru
 - časovač brání jednomu procesu v získání plného přístupu k CPU
 - režim CPU brání uživatelským procesům spouštět privilegované instrukce
- Ochranný mechanismus rozlišuje autorizované a neautorizované použití prostředků



Interpret příkazů

- Interpret příkazů, neboli command-line interpreter (command line interface - CLI), neboli shell
 - úkolem je získávat příkazy od uživatele a provádět je
- Za tento interpret příkazů můžeme považovat i moderní GUI
- Někdy je interpret příkazů přímo součástí jádra operačního systému, někdy je to jen speciální program

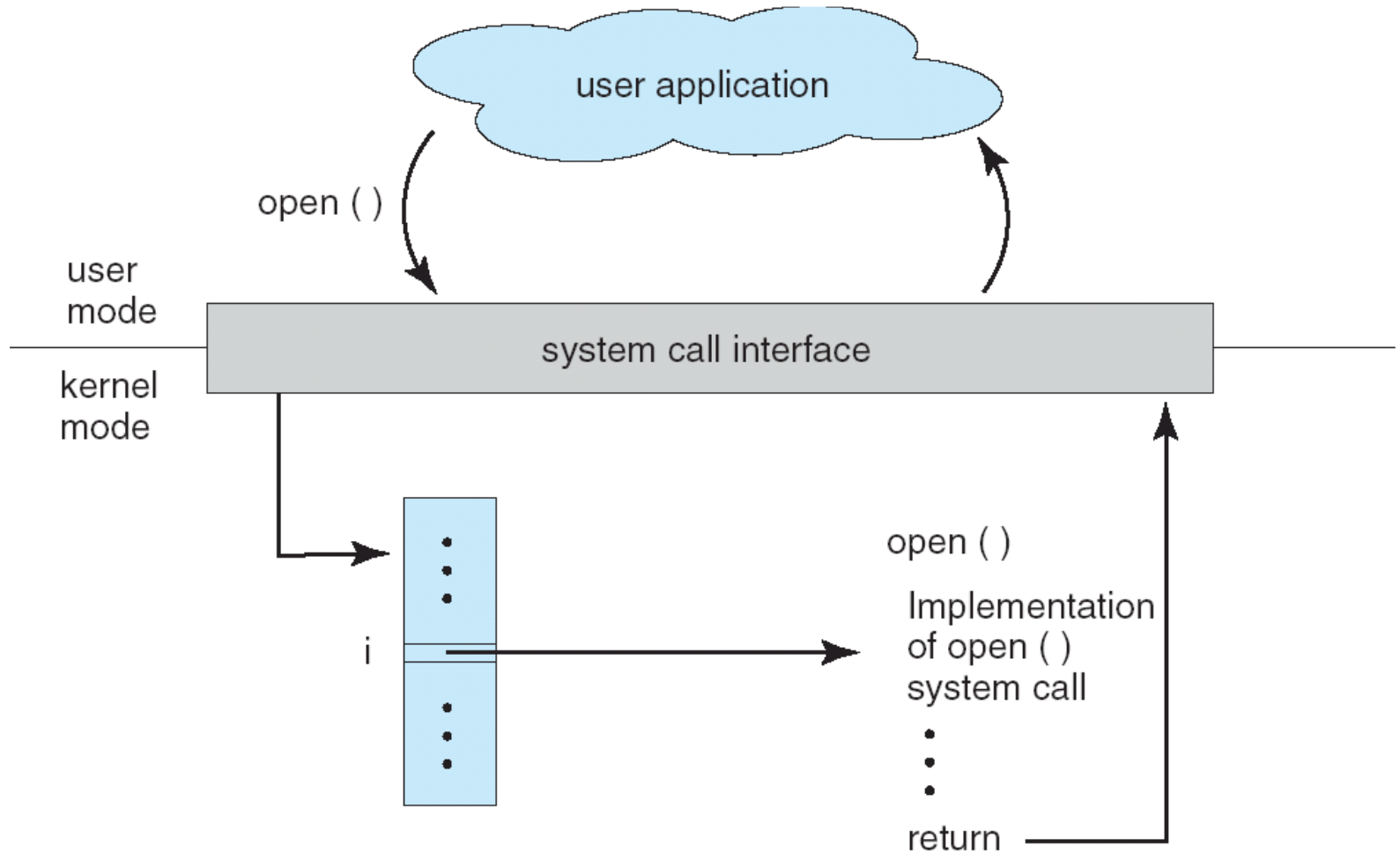
Interní služby OS

- Zabezpečují efektivní provoz samotného OS
- Alokace zdrojů
 - plánovací algoritmy pro přidělování CPU
 - přidělování přístupu k tiskárnám apod.
- Účtování
 - máme přehled o tom, kteří uživatelé kdy využily které zdroje
- Ochrana
 - autentizace, řízení přístupu

Systemová volání

- Systemová volání tvoří rozhraní mezi uživatelským procesem a OS
 - typicky jsou popsána jako instrukce assembleru a jsou uvedena v programátorském manuálu k OS
 - vyšší programovací jazyky obsahují některé funkce, které odpovídají systémovým voláním (např. open, write) a dále knihovní funkce, které poskytují vyšší funkčnost a v rámci této spouští (třeba hned několik) systémových volání (např. fopen, fwrite).

Systemová volání



Systemová volání (2)

- Různé OS a různé HW platformy mívají různé způsoby jak volat služby OS a různou strukturu těchto služeb
- Nicméně existují určité standardy, které usnadňují přenositelnost programového kódu
 - v oblasti UNIXu: POSIX
 - V oblasti Windows: Win32
- Teoreticky kód který bude psán podle standardu bude přeložitelný na kompatibilních platformách, v praxi však existuje celá řada verzí standardu a mnoho výjimek co je a není implementováno

Způsob předání parametrů

○ Registry

- `mov ah,01h`
- `mov cx,2000h`
- `int 10h`

○ Zásobník

- `mov ax,0001h`
- `push ax`
- `int 10h`

○ Blok (struktura, tabulka) v paměti & pointer

- `mov ax,0001h`
- `mov [tabulka1],ax`
- `mov bx,tabulka1`
- `int 10h`

Příklad volání API (Win32)

return value

```
      ↓  
BOOL  ReadFile c (HANDLE      file,  
                LPVOID      buffer,  
                DWORD        bytes To Read,  
                LPDWORD      bytes Read,  
                LPOVERLAPPED ovl);
```

↑
function name

parameters

- Popis parametrů funkce ReadFile()
 - HANDLE file — deskriptor otevřeného souboru
 - LPVOID buffer — buffer pro čtená data
 - DWORD bytesToRead — počet bajtů, které chceme číst
 - LPDWORD bytesRead — kam uložit počet načtených bajtů
 - LPOVERLAPPED ovl — pro asynchronní I/O

Příklad volání API (Win32)

○ Funkce Win32 „CreateProcessAsUser“

```
BOOL CreateProcessAsUser(  
    HANDLE hToken,          // handle to a token representing the logged-on user  
    LPCTSTR lpApplicationName, // pointer to name of executable module  
    LPTSTR lpCommandLine,   // pointer to command line string  
    LPSECURITY_ATTRIBUTES lpProcessAttributes, // process security attributes  
    LPSECURITY_ATTRIBUTES lpThreadAttributes, // thread security attributes  
    BOOL bInheritHandles,   // whether new process inherits handles  
    DWORD dwCreationFlags,  // creation flags  
    LPVOID lpEnvironment,   // pointer to new environment block  
    LPCTSTR lpCurrentDirectory, // pointer to current directory name  
    LPSTARTUPINFO lpStartupInfo, // pointer to STARTUPINFO  
    LPPROCESS_INFORMATION lpProcessInformation // pointer to PROCESS_INFORMATION  
);
```

○ **Windows NT:** Requires version 3.51 or later.

Windows: Unsupported.

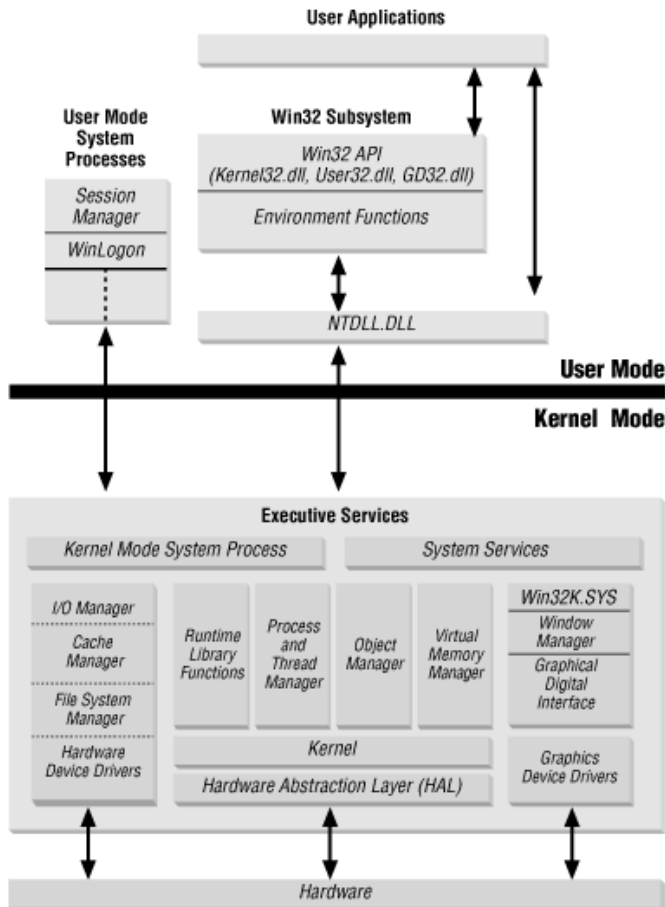
Windows CE: Unsupported.

Header: Declared in winbase.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

Windows API



- Programátorské rozhraní není definováno na úrovni systémových volání jádra
- Windows API
 - dříve Win32
 - kernel32.dll, user.dll
- Nižší vrstva ntdll.dll



Systemová volání Windows

- Přerušení 0x2E
- Příklad ReadFile z ntdll.dll:
Ntdll!ZwReadFile
77f8c552 mov eax, 0xa1 ; the service number
77f8c557 lea edx, [esp+4]
77f8c55b int 2e
77f8c55d ret 0x24
- Příklad používající SYSENTER:
mov eax, 112h
mov edx, 7FFE0300h
call dword ptr ds:[edx]
retn 24
Přičemž voláme funkci s kódem:
mov edx, esp
sysenter

● ● ● | Příklad: SYSENTER

- „The SYSENTER instruction is part of the "Fast System Call" facility introduced on the Pentium® II processor. The SYSENTER instruction is optimized to provide the maximum performance for transitions to protection ring 0 (CPL = 0). The SYSENTER instruction sets the following registers according to values specified by the operating system in certain model-specific registers.
 - CS register set to the value of (SYSENTER_CS_MSR)
 - EIP register set to the value of (SYSENTER_EIP_MSR)
 - SS register set to the sum of (8 plus the value in SYSENTER_CS_MSR)
 - ESP register set to the value of (SYSENTER_ESP_MSR)“
- SYSENTER/SYSEXIT pro procesory Intel (od Pentia II)
- SYSCALL/SYSRET pro procesory AMD (od K7)

Systemová volání Windows

System Call Symbol	NT-SP3	NT-SP4	NT-SP5	NT-SP6	2K-SP0	2K-SP1	2K-SP2	2K-SP3	2K-SP4	XP-SP0	XP-SP1	XP-SP2	XP-SP3	2003-SP0	2003-SP1	VISTA-SP0	SEVEN-SP0
NtAcceptConnectPort	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
NtAccessCheck	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001
NtAccessCheckAndAuditAlarm	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002
NtAccessCheckByType					0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003
NtAccessCheckByTypeAndAuditAlarm					0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004
NtAccessCheckByTypeResultList					0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005
NtAccessCheckByTypeResultListAndAuditAlarm					0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006
NtAccessCheckByTypeResultListAndAuditAlarmByHandle					0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007
NtAddAtom	0x0003	0x0003	0x0003	0x0003	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008
NtAddBootEntry										0x0009	0x0009	0x0009		0x0009	0x0009	0x0009	0x0009
NtAddDriverEntry														0x000A	0x000A	0x000A	0x000A
NtAdjustGroupsToken	0x0004	0x0004	0x0004	0x0004	0x0009	0x0009	0x0009	0x0009	0x0009	0x000A	0x000A	0x000A	0x000A	0x000B	0x000B	0x000B	0x000B
NtAdjustPrivilegesToken	0x0005	0x0005	0x0005	0x0005	0x000A	0x000A	0x000A	0x000A	0x000A	0x000B	0x000B	0x000B	0x000B	0x000C	0x000C	0x000C	0x000C
NtAlertResumeThread	0x0006	0x0006	0x0006	0x0006	0x000B	0x000B	0x000B	0x000B	0x000B	0x000C	0x000C	0x000C	0x000C	0x000D	0x000D	0x000D	0x000D
NtAlertThread	0x0007	0x0007	0x0007	0x0007	0x000C	0x000C	0x000C	0x000C	0x000C	0x000D	0x000D	0x000D	0x000D	0x000E	0x000E	0x000E	0x000E
NtAllocateLocallyUniqueId	0x0008	0x0008	0x0008	0x0008	0x000D	0x000D	0x000D	0x000D	0x000D	0x000E	0x000E	0x000E	0x000E	0x000F	0x000F	0x000F	0x000F
NtAllocateReserveObject																	0x0010
NtAllocateUserPhysicalPages					0x000E	0x000E	0x000E	0x000E	0x000E	0x000F	0x000F	0x000F	0x000F	0x0010	0x0010	0x0010	0x0010
NtAllocateUuids	0x0009	0x0009	0x0009	0x0009	0x000F	0x000F	0x000F	0x000F	0x000F	0x0010	0x0010	0x0010	0x0010	0x0011	0x0011	0x0011	0x0011
NtAllocateVirtualMemory	0x000A	0x000A	0x000A	0x000A	0x0010	0x0010	0x0010	0x0010	0x0010	0x0011	0x0011	0x0011	0x0011	0x0012	0x0012	0x0012	0x0012
NtAlpcAcceptConnectPort																	0x0013
NtAlpcCancelMessage																	0x0014
NtAlpcConnectPort																	0x0015
NtAlpcCreatePort																	0x0016
NtAlpcCreatePortSection																	0x0017
NtAlpcCreateResourceReserve																	0x0018
NtAlpcCreateSectionView																	0x0019
NtAlpcCreateSecurityContext																	0x001A
NtAlpcDeletePortSection																	0x001B
NtAlpcDeleteResourceReserve																	0x001C
NtAlpcDeleteSectionView																	0x001D
NtAlpcDeleteSecurityContext																	0x001E
NtAlpcDisconnectPort																	0x001F
NtAlpcImpersonateClientOfPort																	0x0020
NtAlpcOpenSenderProcess																	0x0021
NtAlpcOpenSenderThread																	0x0022
NtAlpcQueryInformation																	0x0023
NtAlpcQueryInformationMessage																	0x0024
NtAlpcRevokeSecurityContext																	0x0025
																	0x0026



Příklad 1: MS-DOS

- DOS interrupts
 - INT 20H Terminate a program
 - INT 21H DOS Services
 - INT 25H/26H Absolute Disk Read/Write
 - INT 27H Terminate but Stay Resident
 - INT 28H DOS Safe Interrupt / DOS Timeslice (UNDOCUMENTED)
 - INT 2eH Perform DOS Command (UNDOCUMENTED)
 - INT 2fH Multiplex Interrupt (print spooler, TSR control)
 - INT 33H Mouse Support
 - INT 67H Expanded Memory Manager (LIM-EMS)
- Address Pointers (not used as software interrupts)
 - INT 22H Terminate address
 - INT 23H Control-Break address
 - INT 24H Critical Error Handler address
- Volání služeb DOSu:
 - AH = číslo služby
 - ostatní registry (AL, DL, DX, CX, DS:DX, DS:SI, ES:DI, ES:BX) = parametry
 - INT 21H
 - návratová hodnota bývá v AL

● ● ● | Příklad 1: MS-DOS (2)

- Systémové volání pro otevření existujícího souboru
- DOS Fn 3dH: Open a File Handle
 - Expects:
 - AH=3dH
 - DS:DX=address of an ASCIIZ string of a filespec
 - AL=Open Mode
 - AL = 0 to open for reading
 - AL = 1 to open for writing
 - AL = 2 to open for reading and writing
 - Returns: AX=error code if CF is set to CY
=file handle if no error

● ● ● | Příklad 1: MS-DOS (3)

○ Programový kód

```
filename: DATA "c:\autoexec.bat", 0
```

```
mov ax, 3d00h
```

```
mov dx, [filename]
```

```
int 21h
```

● ● ● | Příklad 2: Linux

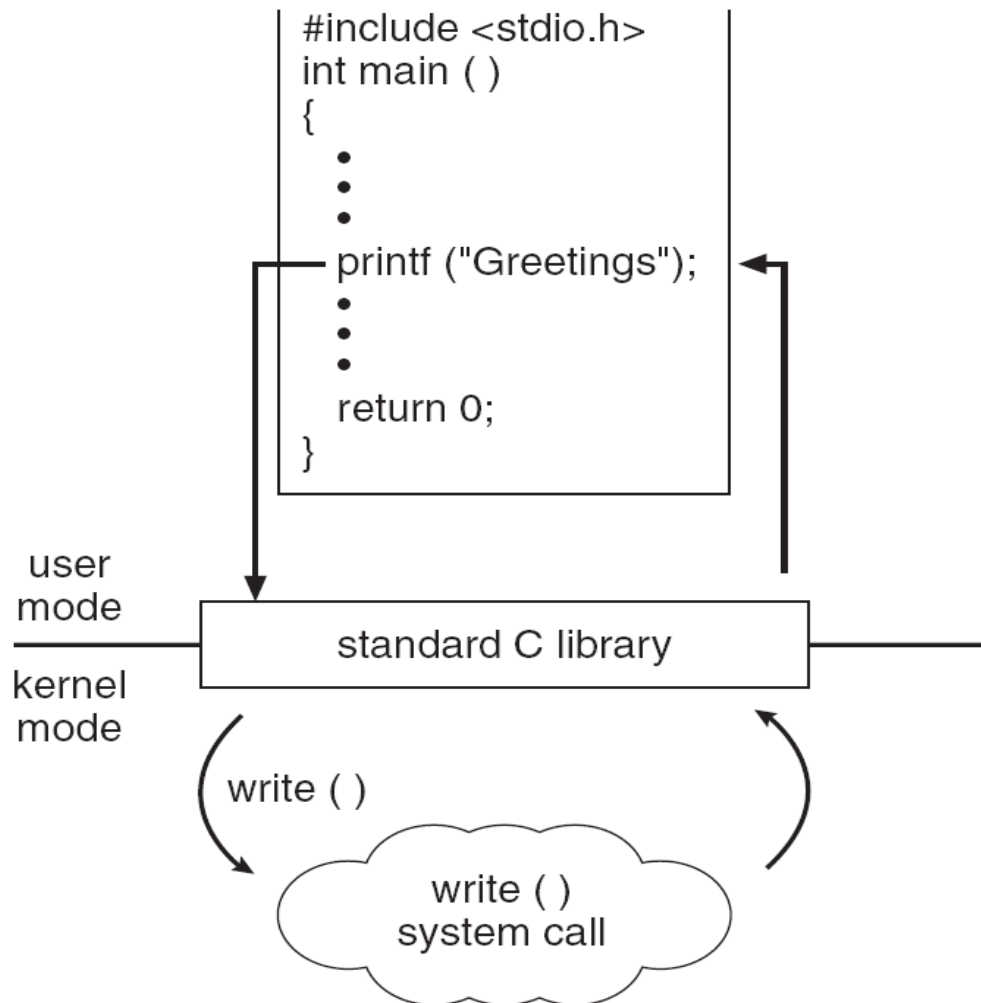
- Systémová volání
 - standardně přes `int 0x80`
 - nově i přes instrukci `syscall` (`sysenter`)
- Číslo systémového volání
 - v registru `eax`
 - v jádře 2.2 přibližně 200 volání, v jádře 2.6 přes 300 volání
- Parametry systémového volání
 - v registrech `ebx`, `ecx`, `edx`, `esi`, `edi`, `ebp`
 - kromě `fce 117` (parametrem je odkaz na strukturu)
- Výsledek
 - uložen v `eax`

Příklad 2: Linux (2)

- Hello world v assembleru pod Linuxem

```
section .text
global _start
msg db 'Hello, world!',0xa
len equ $ - msg
_start:
    mov edx,len ;message length
    mov ecx,msg ;message to write
    mov ebx,1 ;file descriptor (stdout)
    mov eax,4 ;system call number (sys_write)
    int 0x80 ;call kernel
    mov eax,1 ;system call number (sys_exit)
    int 0x80 ;call kernel
```

Příklad 2: Linux (3)



Příklad 2: Linux (4) - strace

anxur:~\$ strace echo "Ahoj"

```
execve("/bin/echo", ["echo", "Ahoj"], [/* 33 vars */]) = 0
brk(0) = 0x1a4fe000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x2b0d6f209000
uname({sys="Linux", node="anxur.fi.muni.cz", ...}) = 0
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=104404, ...}) = 0
close(3) = 0
open("/lib64/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\332\1Y7\0\0\0"... , 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1717800, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x2b0d6f224000
close(3) = 0
arch_prctl(ARCH_SET_FS, 0x2b0d6f225260) = 0
munmap(0x2b0d6f20a000, 104404) = 0
open("/usr/lib/locale/locale-archive", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=56454288, ...}) = 0
mmap(NULL, 56454288, PROT_READ, MAP_PRIVATE, 3, 0) = 0x2b0d6f226000
close(3) = 0
brk(0) = 0x1a4fe000
brk(0x1a51f000) = 0x1a51f000
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 110), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x2b0d727fd000
write(1, "Ahoj\n", 5Ahoj) = 5
close(1) = 0
exit_group(0) = ?
```


Příklad 2: Linux (5)

System Call Quick Reference

No	Func Name	Description	Source
1	<u>exit</u>	terminate the current process	<i>kernel/exit.c</i>
2	<u>fork</u>	create a child process	<i>arch/i386/kernel/process.c</i>
3	<u>read</u>	read from a file descriptor	<i>fs/read_write.c</i>
4	<u>write</u>	write to a file descriptor	<i>fs/read_write.c</i>
5	<u>open</u>	open a file or device	<i>fs/open.c</i>
6	<u>close</u>	close a file descriptor	<i>fs/open.c</i>
7	<u>waitpid</u>	wait for process termination	<i>kernel/exit.c</i>
8	<u>creat</u>	create a file or device ("man 2 open" for information)	<i>fs/open.c</i>
9	<u>link</u>	make a new name for a file	<i>fs/namei.c</i>
10	<u>unlink</u>	delete a name and possibly the file it refers to	<i>fs/namei.c</i>
11	<u>execve</u>	execute program	<i>arch/i386/kernel/process.c</i>
12	<u>chdir</u>	change working directory	<i>fs/open.c</i>
13	<u>time</u>	get time in seconds	<i>kernel/time.c</i>
14	<u>mknod</u>	create a special or ordinary file	<i>fs/namei.c</i>
15	<u>chmod</u>	change permissions of a file	<i>fs/open.c</i>
16	<u>lchown</u>	change ownership of a file	<i>fs/open.c</i>

● ● ● | Příklad 3: FreeBSD

- Systémová volání
 - přes VOLÁNÍ FUNKCE, která obsahuje `int 0x80`
 - nebo přes volání brány `call 7:0`
- Číslo systémového volání
 - v registru `eax`
- Parametry systémového volání
 - na zásobníku (první parametr je ukládán poslední)
- Výsledek
 - v registru `eax`
 - volající proces musí vyčistit zásobník (parametry)

● ● ● | Příklad 3: FreeBSD (2)

- Hello World v assembleru ve FreeBSD

```
section .text
```

```
    global _start
```

```
msg db "Hello, world!",0xa
```

```
len equ $ - msg
```

```
_syscall: int 0x80 ;system call
```

```
    ret
```

```
_start: ;tell linker entry point
```

```
    push dword len ;message length
```

```
    push dword msg ;message to write
```

```
    push dword 1 ;file descriptor (stdout)
```

```
    mov eax,0x4 ;system call number (sys_write)
```

```
    call _syscall ;call kernel
```

```
    add esp,12 ;clean stack (3 arguments * 4)
```

```
    push dword 0 ;exit code
```

```
    mov eax,0x1 ;system call number (sys_exit)
```

```
    call _syscall ;call kernel
```



Komplexita OS

- Počet systémových volání OS / API volání

OS	rok	počet systémových/API volání
UNIX	1971	33
UNIX	1979	47
Windows	1985	450
SunOS	1989	171
4.3 BSD	1991	136
SunOS	1992	219
SunOS	1997	190
Linux	1998	229
NT 4.0	1999	3443
Linux	2009	336



Komplexita OS (2)

- Rozsah zdrojového kódu Windows

Verze	rok	miliony řádků
NT 3.1	1993	4-5
NT 3.5	1994	7-8
95	1995	10
NT 4.0	1996	11-12
98	1998	18
2000	2000	29
XP	2001	40
2003 Svr	2003	50
Vista	2009	64