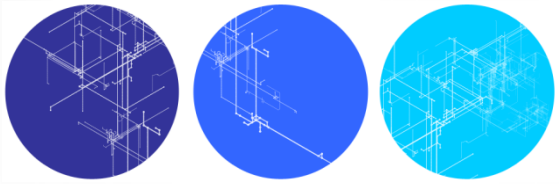
PB153
OPERAČNÍ SYSTÉMY A JEJICH ROZHRANÍ



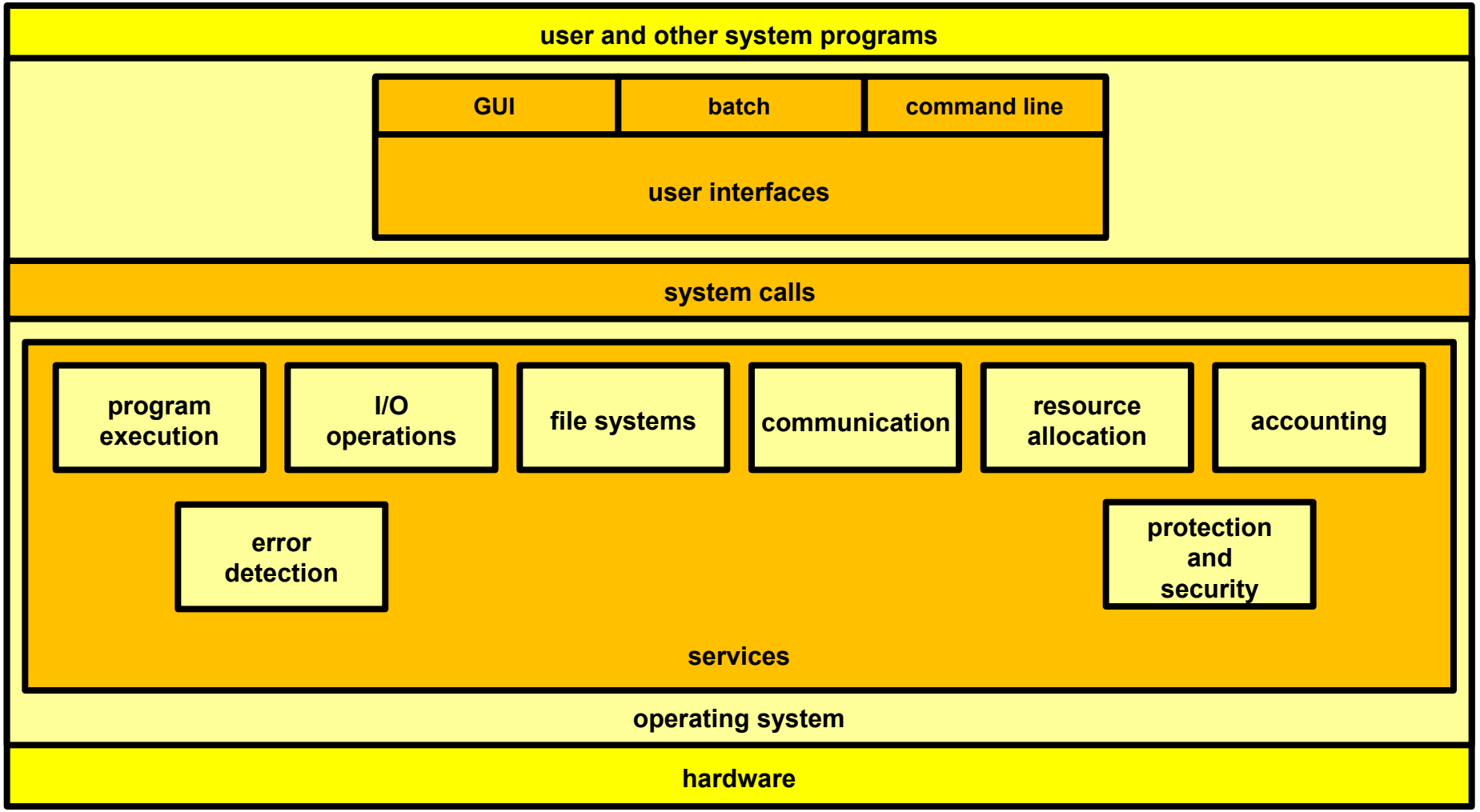
Struktura a rozhraní OS

03

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 nejjednotnější přístup k jednotlivých 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 pomocí 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 součást 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

WINDOWS

- Windows API rozděluje funkce do přibližně 100 kategorií

Functions by Category

msdn

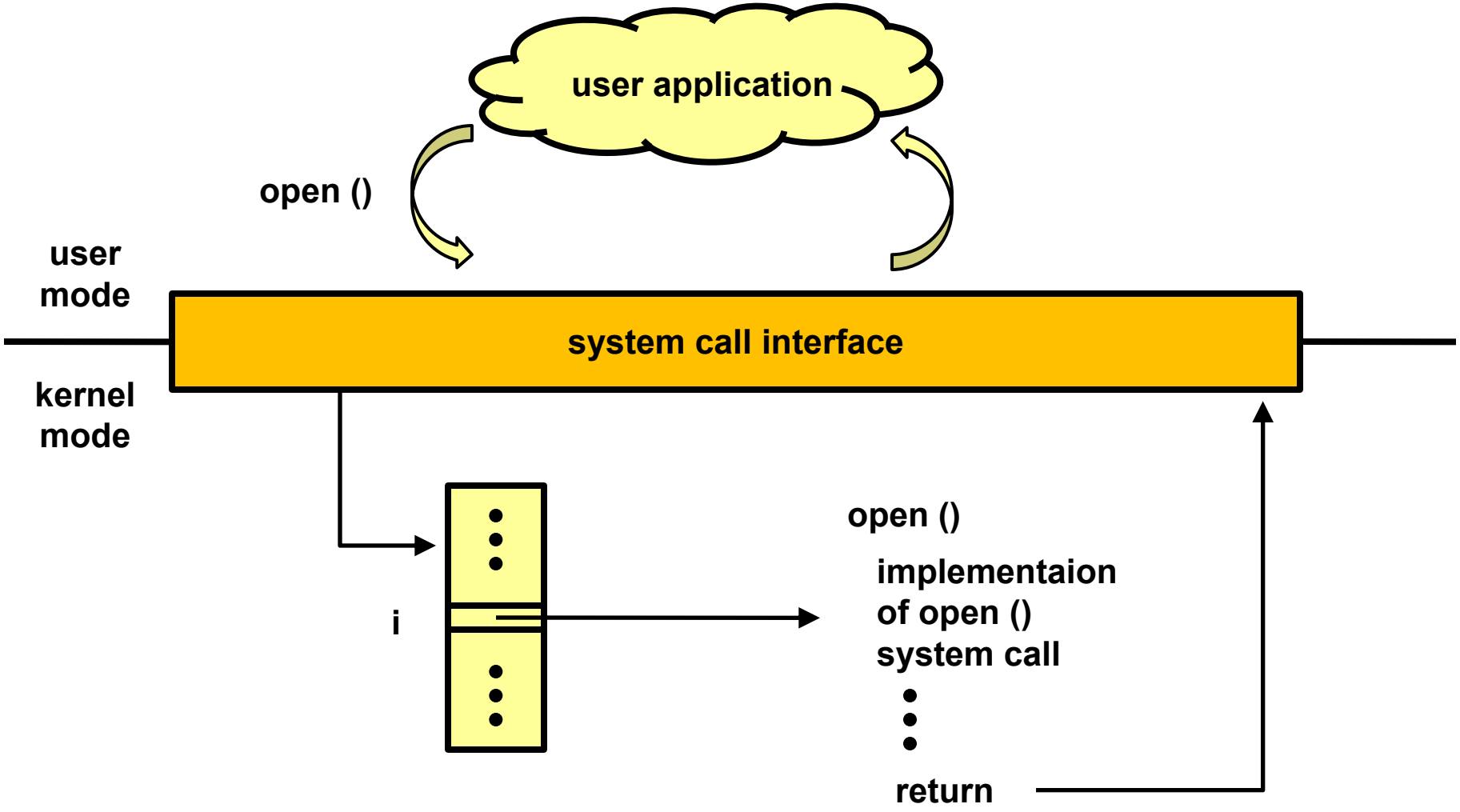
The following is a list of the categories of functions in the Windows application programming interface (API). For a list of functions in alphabetical order, see Functions in Alphabetical Order. For a list of functions by release, see Functions by Release.

- Application Recovery and Restart
- Atom
- Backup
- Bitmap
- Brush
- Button
- Caret
- Clipboard
- Clipping
- Color
- Combo Box
- Common Dialog Box
- Common Log File System
- Communication
- Console
- Coordinate Space and Transformation
- Cursor
- Debug Help Library
- Debugging
- Device Context
- Device Management
- Dialog Box
- Directory Management
- Disk Management
- Dynamic Data Exchange
- Dynamic Data Exchange Management
- Dynamic-Link Library
- Edit Control
- Error Handling
- Event Logging
- File Management
- Filled Shape
- Font and Text
- Handle and Object

SYSTÉMOVÁ VOLÁNÍ

- Systémová 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).

SYSTÉMOVÁ VOLÁNÍ



SYSTÉMOVÁ 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 (Portable Operating System Interface)
  - V oblasti Windows: Win32 (Windows API)
- 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

POSIX

- Od POSIX.1 (1988) po POSIX:2008
  - IEEE Std 1003.1-2008
  - ISO/IEC/IEEE 9945:2009
  - The Open Group Technical Standard Base Specifications, Issue 7
- V části „System interfaces“ je popsáno 1191 položek.

POSIX PŘÍKLAD - READDIR

NAME

readdir, readdir_r - read a directory

SYNOPSIS

```
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
int readdir_r(DIR *restrict dirp, struct dirent *restrict entry,
       struct dirent **restrict result);
```

DESCRIPTION

The type **DIR**, which is defined in the *<dirent.h>* header, represents a *directory stream*, which is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files; files may be removed from a directory or added to a directory asynchronously to the operation of *readdir*().

The *readdir*() function shall return a pointer to a structure representing the directory entry at the current position in the directory stream specified by th argument *dirp*, and position the directory stream at the next entry. It shall return a null pointer upon reaching the end of the directory stream. The structure **dirent** defined in the *<dirent.h>* header describes a directory entry. The value of the structure's *d_ino* member shall be set to the file serial number of the file named by the *d_name* member. If the *d_name* member names a symbolic link, the value of the *d_ino* member shall be set to the file serial number of the symbolic link itself.

The *readdir*() function shall not return directory entries containing empty names. If entries for dot or dot-dot exist, one entry shall be returned for dot ar one entry shall be returned for dot-dot; otherwise, they shall not be returned.

The pointer returned by *readdir*() points to data which may be overwritten by another call to *readdir*() on the same directory stream. This data is not overwritten by another call to *readdir*() on a different directory stream.

If a file is removed from or added to the directory after the most recent call to *opendir()* or *rewinddir()*, whether a subsequent call to *readdir*() returns a entry for that file is unspecified.

The *readdir*() function may buffer several directory entries per actual read operation; *readdir*() shall mark for update the last data access timestamp of the directory each time the directory is actually read.

POSIX - DETAILY

```
EPERM  The system does not allow unlinking of directories, or unlinking of directo-
       ries  requires  privileges  that the current process doesn't have.  (This is
       the POSIX prescribed error return; as noted above, Linux returns EISDIR  for
       this case.)

EPERM (Linux only)
       The filesystem does not allow unlinking of files.
```

```
ENVIRONMENT
       POSIXLY_CORRECT
              This environment variable is examined by the getopt(3) routines.  If it is set, parsing stops  as  soon
              as  a parameter is found that is not an option or an option argument. All remaining parameters are also
              interpreted as non-option parameters, regardless whether they start with a '-'.
```

```
       The behavior of execlp() and execvp() when errors occur while  attempt-
       ing to execute the file is historic practice, but has not traditionally
       been documented and is not specified by the POSIX standard.   BSD  (and
       possibly  other  systems) do an automatic sleep and retry if ETXTBSY is
       encountered.  Linux treats it as a hard error and returns immediately.
```

POSIX KOMPATIBILITA

Fully POSIX-compliant

The following operating systems conform (i.e., are 100% compliant) to one or more of the various POSIX standards.

- A/UX
- AIX
- BSD/OS
- DSPnano
- HP-UX
- INTEGRITY
- IRIX
- LynxOS
- Mac OS X[16]
- MINIX
- MPE/iX
- QNX [17]
- RTEMS (POSIX 1003.13-2003 Profile 52)
- Solaris
- Tru64
- Unison RTOS
- UnixWare
- velOSity
- VxWorks [17]

Mostly POSIX-compliant

The following, while not officially certified as POSIX compatible, conform in large part:

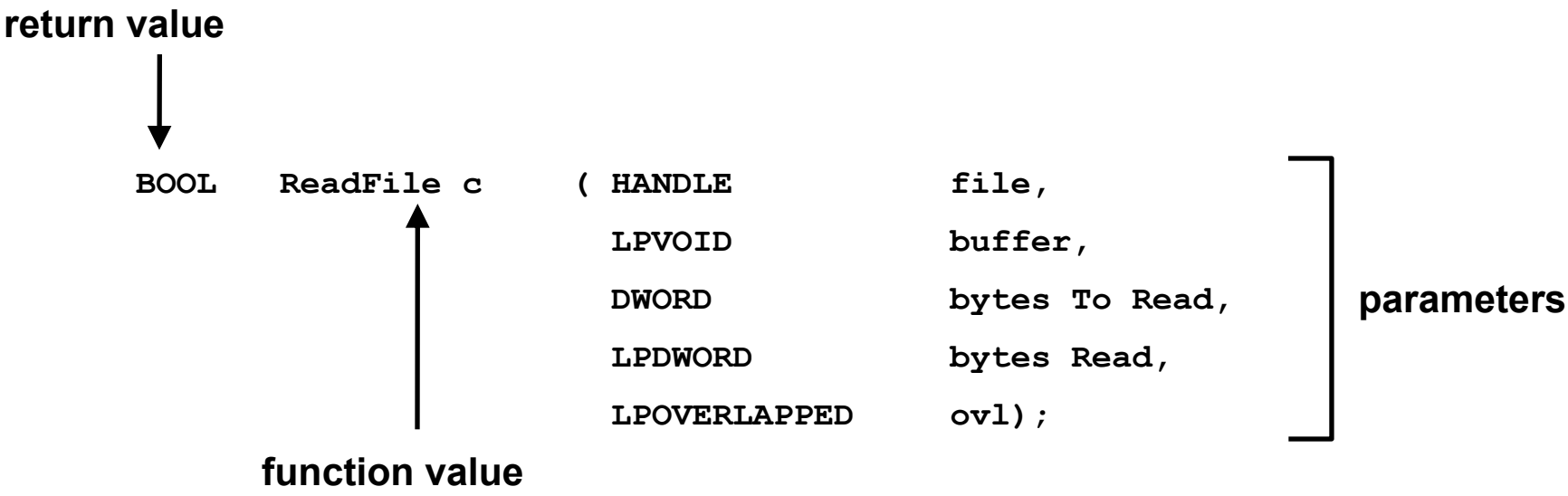
- BeOS / Haiku
- FreeBSD[18]
- Linux (most distributions — see Linux Standard Base)
- Contiki
- NetBSD
- Nucleus RTOS
- OpenBSD
- OpenSolaris
- PikeOS RTOS for embedded systems with optional PSE51 and PSE52 partitions;
- RTEMS – POSIX API support designed to IEEE Std. 1003.13 -2003 PSE52
- Sanos
- SkyOS
- Syllable
- VSTa

Zdroj: Wikipedia, heslo POSIX

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)



- 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

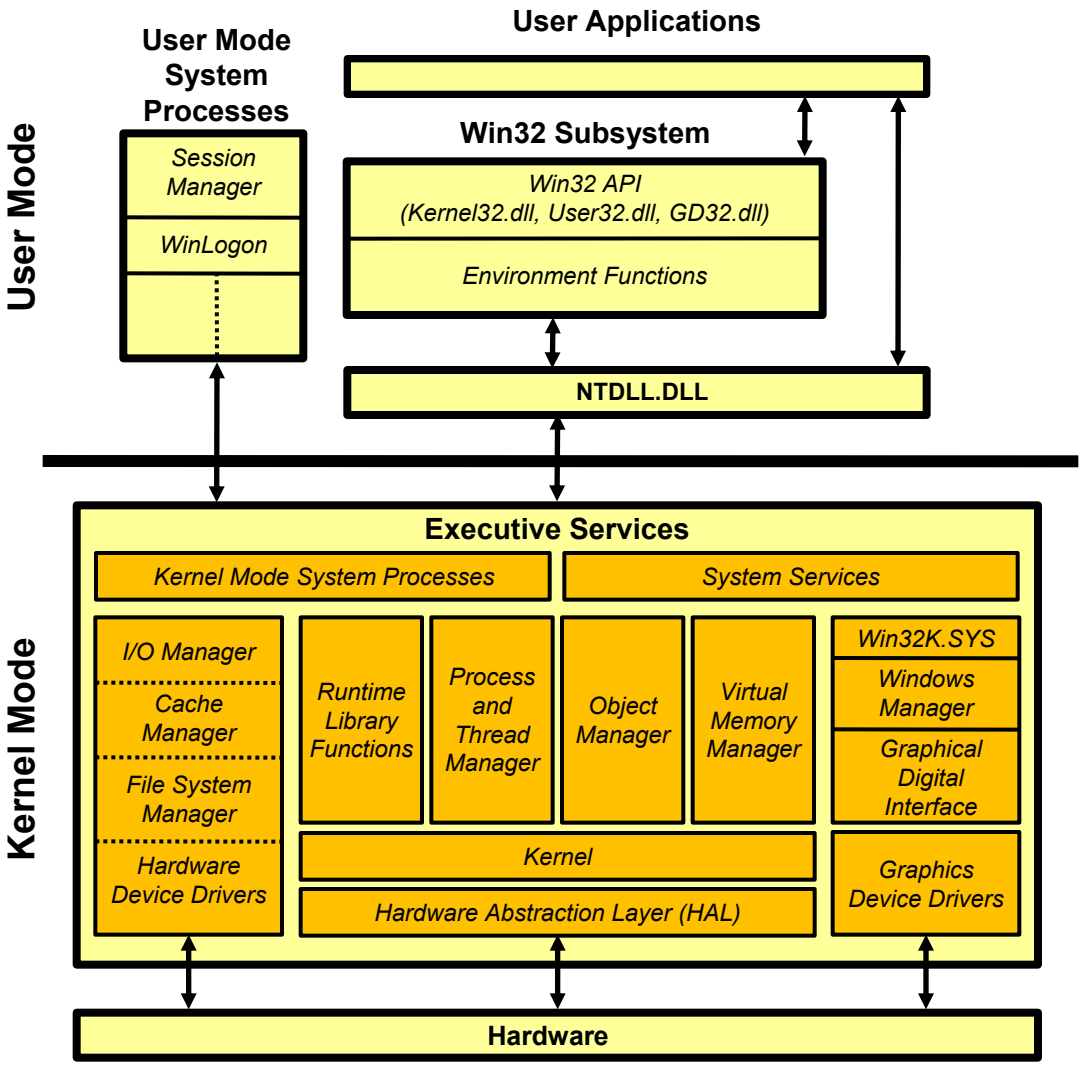
PŘÍKLAD VOLÁNÍ API (WIN32)

- 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

SYSTÉMOVÁ 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 aex, 112h
mov edx, 7FFE0300h
call dword prt 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)

SYSTÉMOVÁ 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	0x0011
NtAllocateUuids	0x0009	0x0009	0x0009	0x0009	0x000F	0x000F	0x000F	0x000F	0x000F	0x0010	0x0010	0x0010	0x0010	0x0011	0x0011	0x0011	0x0012
NtAllocateVirtualMemory	0x000A	0x000A	0x000A	0x000A	0x0010	0x0010	0x0010	0x0010	0x0010	0x0011	0x0011	0x0011	0x0011	0x0012	0x0012	0x0012	0x0013
NtAlpcAcceptConnectPort																0x0013	0x0014
NtAlpcCancelMessage																0x0014	0x0015
NtAlpcConnectPort																0x0015	0x0016
NtAlpcCreatePort																0x0016	0x0017
NtAlpcCreatePortSection																0x0017	0x0018
NtAlpcCreateResourceReserve																0x0018	0x0019
NtAlpcCreateSectionView																0x0019	0x001A
NtAlpcCreateSecurityContext																0x001A	0x001B
NtAlpcDeletePortSection																0x001B	0x001C
NtAlpcDeleteResourceReserve																0x001C	0x001D
NtAlpcDeleteSectionView																0x001D	0x001E
NtAlpcDeleteSecurityContext																0x001E	0x001F
NtAlpcDisconnectPort																0x001F	0x0020
NtAlpcImpersonateClientOfPort																0x0020	0x0021
NtAlpcOpenSenderProcess																0x0021	0x0022
NtAlpcOpenSenderThread																0x0022	0x0023
NtAlpcQueryInformation																0x0023	0x0024
NtAlpcQueryInformationMessage																0x0024	0x0025
NtAlpcRevokeSecurityContext																	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

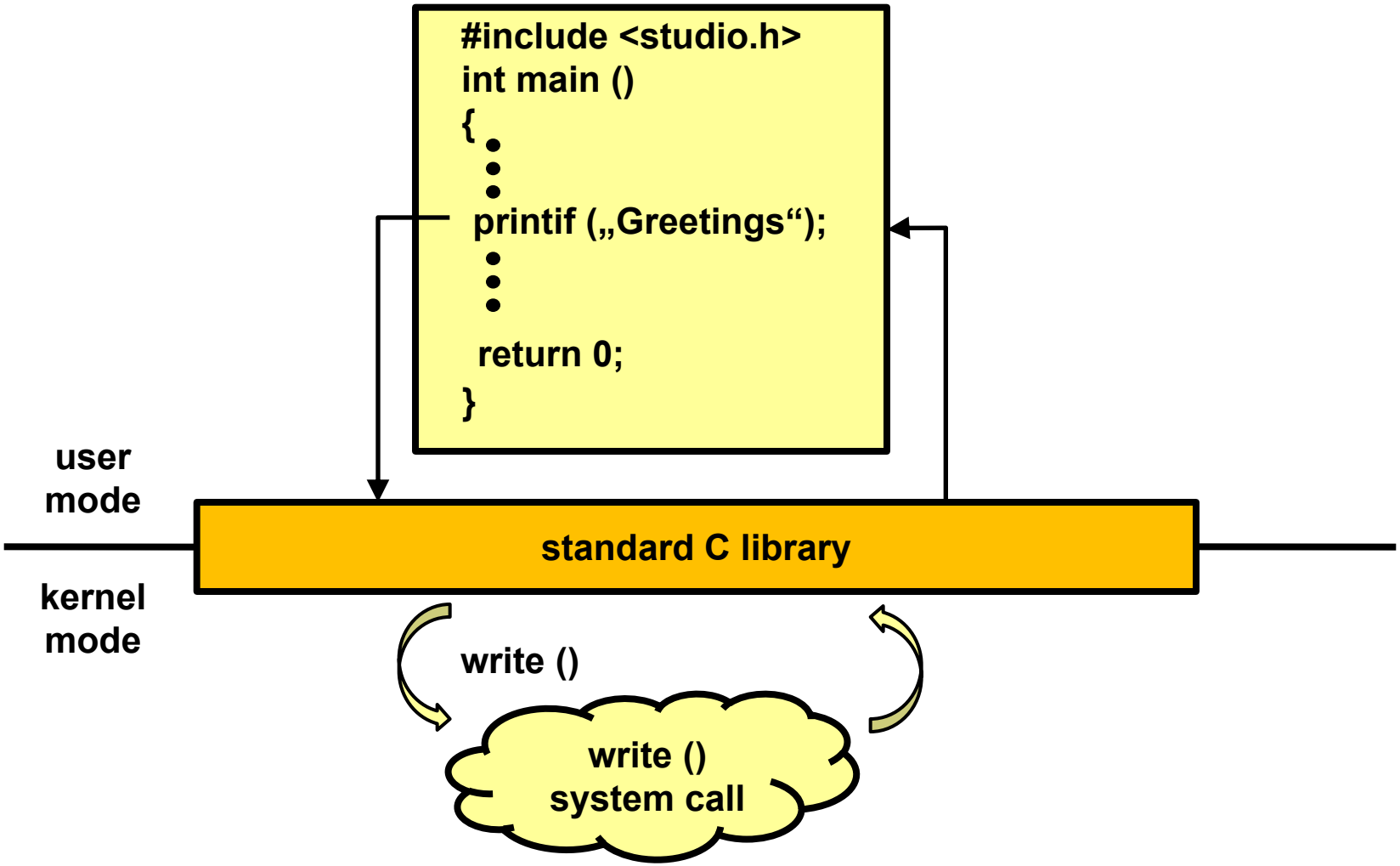
- Systémová volání
  - standardně přes `int 0x80`
  - nově i přes instrukci `syscall (sysenter)`
- Číslo systémového volání
  - v registru `eax`
  - v jádře 2.2 přibližně 200 volání, v jádře 2.6 přes 300 volání
- Parametry systémového volání
  - v registrech `ebx, ecx, edx, esi, edi, ebp`
  - kromě fce 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\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	exit	terminate a current process	*kernel/exit.c*
2	fork	create a child process	*arch/i386/kernel/proces.c*
3	read	read from a file descriptor	*fs/read_write.c*
4	write	write to a file descriptor	*fs/read_write.c*
5	open	open a file or device	*fs/open.c*
6	close	close a file descriptor	*fs/open.c*
7	waitpid	wait for process termination	*kernel/exit.c*
8	creat	create a file or device ("man 2 open" for information)	*fs/open.c*
9	link	make a new name for a file	*fs/namei.c*
10	unlink	delete a name and possibly the file it refers to	*fs/namei.c*
11	execve	execute program	*arch/i386/kernel/proces.c*
12	chdir	change working directory	*fs/open.c*
13	time	get time in seconds	*kernel/time.c*
14	mknod	create a special or ordinary file	*fs/namei.c*
15	chmod	change permissions of a file	*fs/open.c*
16	lchown	change ownership of a file	*fs/open.c*

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	1985	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

Výukovou pomůcku zpracovalo

Servisní středisko pro e-learning na MU

CZ.1.07/2.2.00/28.0041
Centrum interaktivních a multimediálních studijních opor pro inovaci výuky a efektivní učení

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ