



Pokročilé operační systémy

# Systemová volání I & Binární soubory

Petr Krajča



Katedra informatiky  
Univerzita Palackého v Olomouci



- několik úrovní abstrakce
- příklad otevření souboru:

## unix

- 1 proces volá funkci `fopen` standardní knihovny C, tj.  
`FILE *fopen(const char *path, const char *mode);`
- 2 `fopen` volá funkci `open`, která představuje rozhraní jádra:  
`int open(const char *path, int oflag, ...);`  
širší možnosti (viz `man 3p open`)
- 3 funkce `open` provede systémové volání (předá argumenty jádru, vyvolá službu v privilegovaném režimu)



## Windows

- 1 proces volá funkci `fopen` standardní knihovny C, tj.

```
FILE *fopen(const char *path, const char *mode);
```

- 2 `fopen` volá funkci (`kernel32.dll`)

```
CreateFileA(  
    [in]          LPCSTR          lpFileName,  
    [in]          DWORD           dwDesiredAccess,  
    [in]          DWORD           dwShareMode,  
    [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
    [in]          DWORD           dwCreationDisposition,  
    [in]          DWORD           dwFlagsAndAttributes,  
    [in, optional] HANDLE         hTemplateFile);
```

v API ještě použito makro `CreateFile`, které se expanduje buď na `CreateFileA` nebo `CreateFileW` podle toho, jestli se pracuje s ASCII nebo UNICODE řetězci

## Windows (dokončení)

- 3 funkce `CreateFile[AW]` volá `NtCreateFile` (`ntdll.dll`), která provede systémové volání (předá argumenty jádru, vyvolá službu v privilegovaném režimu)

```
__kernel_entry NTSTATUS NtCreateFile(  
    [out]          PHANDLE          FileHandle,  
    [in]           ACCESS_MASK      DesiredAccess,  
    [in]           POBJECT_ATTRIBUTES ObjectAttributes,  
    [out]          PIO_STATUS_BLOCK IoStatusBlock,  
    [in, optional] PLARGE_INTEGER   AllocationSize,  
    [in]           ULONG             FileAttributes,  
    [in]           ULONG             ShareAccess,  
    [in]           ULONG             CreateDisposition,  
    [in]           ULONG             CreateOptions,  
    [in]           PVOID             EaBuffer,  
    [in]           ULONG             EaLength);
```



- rozhraní OS často popsáno formou volání funkcí vyššího prog. jazyka (C), např. `open`, `NtCreateFile`
- jednotné ABI (např. `cdecl`) kompatibilní se zbytkem platformy
- snadná integrace do dalších prog. jazyků
- lepší přenositelnost
  - hardwarová (různě implementovaná systémová volání)
  - softwarová (různé rozhraní jádra OS)
- možnost emulovat rozhraní jiného OS (WindowsNT a POSIX)

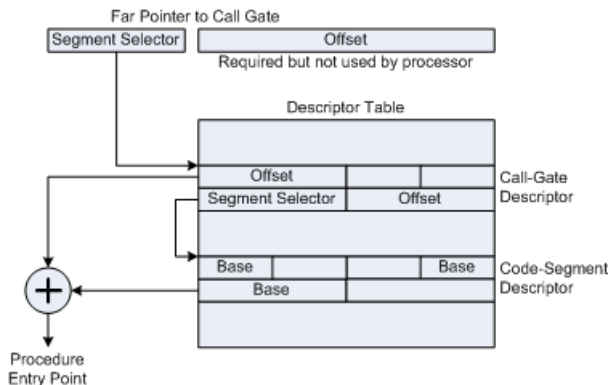


- nutné přepnout procesor z neprivilegovaného režimu do privilegovaného (z ring3 do ring0)
- identifikovat požadovanou službu jádra OS
- předat argumenty

## přerušení

- tradiční řešení
- číslo služby OS je uloženo do registru EAX
- argumenty v registrech EBX, ECX, ...
- služba jádra vyvolána softwarovým přerušením, tj. instrukcí `int`, např.
  - `int 0x80` (Linux)
  - `int 0x2e` (WindowsNT)
- návratová hodnota uložena v EAX

- obecný mechanismus postavaný na segmentaci
- DPL kódového segmentu (CS) udává úroveň oprávnění
- přechod mezi různými kódovými segmenty (pomocí vzdálených skoků)
- využití GDT (musí být chráněný přístup)
- umožňuje přechod i do dalších úrovní oprávnění (ring1, ring2)





- dedikované páry instrukcí pro přepnutí do/z privilegovaného režimu
- Intel: SYSENTER/SYSENTER
- AMD: SYSCALL/SYSRET
- adresa vstupního bodu jádra určena pomocí MSR (Model Specific Register)
- přístup k MSR přes instrukce `rdmsr` a `wrmsr`



## Nejmenší program v Linuxu (32bit)

```
mov eax, 1    ;; sys_exit
mov ebx, 42   ;; vysledek
int 0x80     ;; systemove volani
```

```
./min-app; echo $?
```

## další systémová volání

```
;; int sys_read(unsigned int fd, const char *buf, unsigned int size);
sys_read:
```

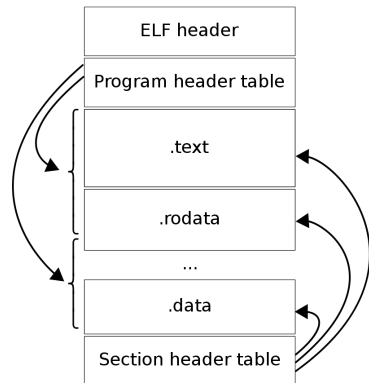
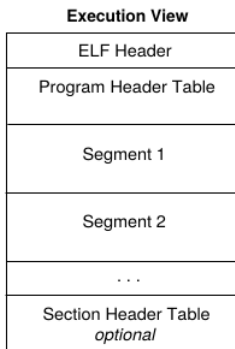
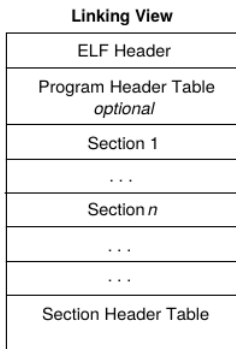
```
    push ebx
    mov eax, SYS_READ_NR    ;; 3
    mov ebx, [esp + 8]      ;; file descriptor
    mov ecx, [esp + 12]    ;; buffer
    mov edx, [esp + 16]    ;; size
    int 0x80               ;; čtení souboru
    pop ebx
    ret
```



- viz příložené soubory (sys-calls/\*)
- přehledný seznam: [https://chromium.googlesource.com/chromiumos/docs/+master/constants/syscalls.md#x86-32\\_bit](https://chromium.googlesource.com/chromiumos/docs/+master/constants/syscalls.md#x86-32_bit)

- binární soubory obsahují různé typy dat
  - spustitelný kód (.text)
  - data pouze pro čtení (.rodata, např. konstanty, řetězcové literály)
  - inicializovaná data (.data, např. globální a statické proměnné)
  - neinicializovaná data (.bss, např. globální a statické proměnné)
- binární soubory obsahují různé typy metadat
  - umístění jednotlivých částí v paměti
  - vstupní bod programu
  - oprávnění k jednotlivým oblastem (např. povolení zápisu, provádění kódu)
  - informace pro přesun kódu, dynamické načítání
- různé formáty s přibližně stejným obsahem
  - Executable and Linkable Format – Linux
  - Portable Executable (PE) – Window
  - Mach-O – OS od Applu

- binární a objektové soubory obsahují velmi podobná data, proto se používá stejný formát pro oba účely (spuštění programu i kompilace)





- <https://refspecs.linuxfoundation.org/elf/elf.pdf>
- viz příložené zdrojové kódy
- pěkný rozbor formátu ELF: <https://raw.githubusercontent.com/corkami/pics/28cb0226093ed57b348723bc473cea0162dad366/binary/elf101/elf101-64.svg>

- 1 vyberte si minimálně jedno systémové volání jádra Linuxu, naprogramujte jeho volání a ověřte, že funguje správně
- 2 s pomocí různých voleb nástroje objdump se podívejte na obsah binárních a objektových souborů
  - užitečné volby `objdump -x`
  - užitečné volby `objdump -d -M intel`
  - srovnajte výstup staticky a dynamicky přeloženého programu