KATEDRA
INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Systémová volání

KMI/OS1 Operační systémy I

Mgr. Markéta Trnečková, Ph.D.
www.marketa-trneckova.cz

Systémová volání

Učební text ke cvičení:
https://phoenix.inf.upol.cz/~krajcap/courses/2025LS/OS1/tutorial07.pdf

Systémová volání

- Zatím jsme psali v assembleru pouze části kódu
- Jazyk C a jeho standardní knihovna nás odstiňoval od volání OS
- Např. výpisy do konzole – `printf()` (sestavení řetězce, zápis na standardní výstup)
- Sestavení řetězce – obvykle vyřešena v rámci standardního kódu v jazyce C
- Zápis na výstup – řeší jádro operačního systému.
- Ukážeme si jak jsou v Linux (AMD64) řešena systémová volání
- Jak napsat program celý assembleru

Minimální program

Operační systém

1. Každý OS poskytuje uživatelským procesům sadu služeb:
   - vytvoření souboru
   - zápis/čtení souboru
   - spuštění nového procesu
   - ...
2. Funkce nabízí přes jednoznačně definované prostředí

Systémová volání

Linux (platforma AMD64)

- Každá služba OS (např. otevření souboru, změna adresáře) je identifikována číslem, které je uloženo v registru `rax`
- Argumenty předávané jádru (např. název souboru, příznaky) jsou uloženy v registrech `rdi`, `rsi`, `rdx`, `r10`, `r8`, `r9` (v tomto pořadí)
- Služba OS je zavolána instrukcí `syscall`
- Návratová hodnota je uložena v registru `rax` (záporné hodnoty indikují chybu), obsah registrů `rcx` a `r11` může být změněn, obsah dalších registrů je zachován

Implementace minimálního programuí

- každý program by měl obsahovat alespoň jedno systémové volání – `exit` (`sys_exit`)
- `exit` se postará o ukončení aktuálně běžícího procesu
- má jeden parametr – udává s jakým kódem byl proces ukončen (0 bez chyby, ostatní označují chybu)
- `exit` má v Linuxu na platformě AMD64 přiřazený kód 60 (dekadicky)

Kompletní výčet služeb a jejich čísel: (např.)
`https://www.chromium.org/chromium-os/developer-library/reference/linux-constants/syscalls/`

Příklad

Příklad

```
global _start

SYS_EXIT equ 60

section .text
_start:
  mov rax, SYS_EXIT
  mov rdi, 42
  syscall
```

- Systémové volání: do `rax` je přiřazeno číslo služby, do `rdi` návratová hodnota
- `syscall` provede samotné systémové volání – ukončení programu

Příklad

Příklad

```
global _start

SYS_EXIT equ 60

section .text
_start:
  mov rax, SYS_EXIT
  mov rdi, 42
  syscall
```

- direktiva equ slouží k definici konstant (vlevo název, vpravo hodnota)
- konstanty zlepšují čitelnost kódu, v případě potřeby lepší nahrazení výskytu konstant
- návěští `_start` představuje vstupní bod programu, je nutné ji deklarovat jako `global`

Příklad
Překlad

Příklad

```
tutorial07: tutorial07.o
      ld -o tutorial07 tutorial07.o
tutorial07.o: tutorial07.asm
      nasm -f elf64 tutorial07.asm
```

- pro získání objektového souboru použijeme `nasm`, jako dříve
- pro slinkování `ld`
- nepoužíváme žádné další knihovny

Příklad
Spuštění

- `./tutorial07`
- program neudělá nic
- návratovou hodnotu můžeme v shellu získat `$?`

Příklad

```
$ ./tutorial07
$ echo $?
```

Příklad

Jakou hodnotu bychom měli dostat?

Hello World!

Příklad

```
global _start
;
; deklarace konstant
;
SYS_WRITE equ 1 ; systemove volani pro zapis do souboru
SYS_EXIT equ 60 ; systemove volani pro ukonceni programu
STDOUT equ 1 ; deskriptor souboru standardniho vystupu
STR_HELLO_LEN equ 13 ; delka vypsaneho retezce
;
; spustitelny kod
;
section .text
_start:
  mov rax, SYS_WRITE ; vypsani retezce Hello World
  mov rdi, STDOUT
  mov rsi, str_hello
  mov rdx, STR_HELLO_LEN
  syscall

  mov rax, SYS_EXIT ; ukonceni programu
  mov rdi, 42
  syscall

;
; (inicializovana) data programu
;
section .data
str_hello:
  db "Hello World!", 10
```

Hello World!

write

- systémové volání `write`
- parametry:
  - deskriptor souboru, do kterého se bude zapisovat
  - řetězec, který se má zapsat do souboru
  - délka řetězce
- standardní výstup – soubor s deskriptorem 1 (to přiřazujeme do `rdi`)
- řetězec, který chceme vypsat – data
- délku řetězce přiřadíme do `rdx`

Hello World!
Data

- Data mají samostatné sekce:
  - `.data` – obecná data
  - `.rodata` – data jen pro čtení
  - `.bss` – neinicializovaná data
- v příkladu se použily `.data` a pseudoinstrukcí `db` jsme přiřadili hodnotu
- Pseudoinstrukce db umožňuje definovat a alokovat místo pro jednobytové hodnoty, případně řetězce, jak lze vidět v našem příkladu
- pseudoinstrukce `dw`, `dd` a `dq` – vytvoří místo pro hodnoty o velikostech 2, 4 a 8 bytů
- Odkaz na dané místo v paměti je v assembleru řešen standardním návěštím
- Při spuštění jsou hodnoty ze sekcí `.data` a `.rodata` načtena ze souboru do paměti a program k nim může přistupovat pomocí instrukcí pro práci s pamětí.

Příklad
Čtení dat

- Načteme data ze standardního vstupu
  Volání `read`, kde deskriptor souboru bude 0 (standardní vstup)
- Data načteme do bufferu (specifikujeme velikost bufferu `BUFFER_SIZE`)
- Tento buffer bude umístěn v sekci neinicializovaných dat `.bss`, označíme ho návěštím `input_buffer`
- K alokaci použijeme pseudoinstrukci `resb n` – vyhradí místo v paměti o velikosti `bytů`
- Po provedení operace čtení se ověří, zda nedošlo k chybě
  `read` v takovém případě vrací zápornou hodnotu, jinak vrací počet úspěšně přečtených bytů
- Pokud dojde k chybě nastavíme chybový návratový kód programu
- Úspěšně načtená data hned vypíšeme pomocí systémového volání `write` a program ukončíme s návratovým kódem 0

Příklad
Čtení dat

Příklad

```
global _start
; deklarace konstant
SYS_READ equ 0 ; systemove volani pro cteni ze souboru
SYS_WRITE equ 1 ; systemove volani pro zapis do souboru
SYS_EXIT equ 60 ; systemove volani pro ukonceni programu
STDIN equ 0 ; deskriptor souboru standardniho vstupu
STDOUT equ 1 ; deskriptor souboru standardniho vystupu
BUFFER_SIZE equ 64 ; velikost bufferu
EOK equ 0 ; konstanta signalizujici, ze program skoncil v poradku
EINPUT equ 1 ; konstanta signalizujici, ze program skoncil chybou
```

Příklad
Čtení dat

Příklad

```
; spustitelny kod
section .text
_start:
  mov rax, SYS_READ ; nacte data ze standardniho vstupu
  mov rdi, STDIN
  mov rsi, input_buffer
  mov rdx, BUFFER_SIZE
  syscall
  cmp rax, 0
  jl fail  ; pokud je vysledek zaporny => chyba
  mov rdx, rax ; rax obsahuje pocet nactenych bytu (predavame jako 3. argument)
  mov rax, SYS_WRITE
  mov rdi, STDOUT ; vypisujeme na standardni vystup
  mov rsi, input_buffer
  syscall ; vypsani obsahu bufferu
  jmp success ; korektni ukonceni programu
fail: ; chyba pri cteni dat
  mov rdi, EINPUT
  jmp exit
success: ; uspesne ukonceni programu
  mov rdi, EOK
exit:   ; predpoklada, ze v rdi je navratovy kod, a ukonci program
  mov rax, SYS_EXIT
  syscall

section .bss
input_buffer:
  resb BUFFER_SIZE
```

Příklad
Spuštění

Příklad

Vyzkoušejte přeložit a spustit předchozí příklad.

```
$ echo "hello"| ./cteni
```

Příklad
Čtení dat

- K alokaci místa pro buffer jsme použili pseudoinstrukci `resb n` – vyhradí místo v paměti o velikosti `bytů`
- Analogicky máme pseudoinstrukce `resw`, `resd`, `resq`, které vyhradí místo o n 16bitových, 32bitových a 64bitových slovech
- Protože hodnoty v sekci `.bss` jsou neinicializované, nezabírají žádné místo v binárním souboru, tím se tato sekce liší od `.data` nebo `.rodata`

Ladění

- Ladit program, který přistupuje přímo ke službám jádra operačního systému nemusí být úplně pohodlné.
- Nástroj `strace` – pro spuštěný program ukazuje, jaká systémová volání byla zavolána, s jakými parametry a jaké byly návratové hodnoty

Příklad

```
$ echo "hello"| strace ./cteni
```

```
execve("./cteni", ["./cteni"], 0x7ffde0720ec0 /* 27 vars */) = 0
read(0, "hello\n", 64)                  = 6
write(1, "hello\n", 6hello
)                       = 6
exit(0)                                 = ?
+++ exited with 0 +++
```

Systémová volání

Úkoly k procvičení

Všechny následující programy vytvořte v assembleru bez použití kódu v jazyce C.

1. Vytvořte program `rect`, který na terminál vypíše obdélník složený ze znaků `'*'` o stranách 20×5.
2. Vytvořte program `mypwd`, který se bude chovat podobně jako standardní unixový příkaz `pwd` a vypíše na standardní výstup plnou cestu k aktuálnímu adresáři. Jaký je aktuální adresář zjistíte pomocí systémového volání `getcwd`.
3. Vytvořte program `mypwd2`, který vypíše jméno aktuálního adresáře, tj. jméno za posledním znakem `'/'`.
4. Upravte poslední ukázkový příklad tak, aby vracel počet řádků přečtených ze standardního vstupu. Tyto úpravy provádějte postupně.
   - Spočítejte řádky na vstupu a výsledek vraťte v návratovém kódu.
   - Spočítejte řádky a jejich počet vypište na standardní výstup.
   - Upravte program, aby pracoval s libovolně velkým vstupem, tj. zpracovával vstup, dokud systémové volání `read` nevrátí 0 nebo zápornou hodnotu.

Systémová volání

Úkoly k procvičení – poznámky

- Vložení souboru na vstup:
  `echo "$(cat text.txt)"| ./cteni`
- výstup bere pouze textový řetězec – čísla je nutné převést na řetězce