PB173 – Binární programování Linux

VIII. C bez libc

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

8. 11. 2016

C a `libc`

- `libc` se stará o komunikaci s OS (jádrem)
  - Jádro poskytuje pouze „základní” funkcionalitu
- `libc` poskytuje spoustu nadstaveb
  - Soubory: `fopen`, `fread`, `fwrite`, `fclose`, ...
  - Síť: `getaddrinfo`, `ntoh*`, `hton*`
  - Vlákna (`libpthread`): `pthread_*` (včetně zamykání)
  - A spoustu dalšího (celý POSIX), ...

Bez `libc` nic z toho nemáme

Jak teď ale např. otevřu soubor?

Komunikace s jádrem

- Sdílená paměť
  - Jádro zapisuje, uživatel čte a naopak
- Systémová volání
  - Podobné jako zavolání funkce
  - Ale program a jádro běží v jiných kontextech
  - Přepnutí kontextu
    - Uložení a změna stavu procesoru
    - Relativně drahá operace
  - Uvidíme dnes dále

Systémová volání

- Volání funkce
- Instrukcí procesoru
  - Závislé na architektuře
  - Softwarové přerušení (x86_32: číslo `0x80`)
  - Speciální instrukce (x86_32: `sysenter`, x86_64: `syscall`)
  - **Demo:** implementace `syscall` z `glibc`
- Systémová volání jsou očíslovaná
  - Do jednoho registru číslo
  - Čísla `__NR_*` definovaná jádrem (pro každou architekturu)
  - `sys/syscall.h`
  - Např. (x86_64): `__NR_write` (1), `__NR_exit` (60)

Úkol

`libc` o úroveň níže

1. Zavolejte systémová volání
   - `write` s nějakým textem
   - `exit`
2. Proveďte ale pomocí funkce `syscall` z `libc`
   - Tedy nepište přímo `write(...)`, `exit(...)`
   - Viz `man 2 syscall`
   - První argument: číslo systémového volání
   - Další argumenty: odpovídají už danému volání
   - Např. `syscall(__NR_kill, -1, SIGKILL)`
3. Přeložte a vyzkoušejte

C bez `libc`

- Žádné hlavičkové soubory z `libc`, ani jeho funkce
- Jen to, co poskytuje jádro (`/usr/include/linux/`)
- Při překladu: `gcc -nostdinc ...`
- Při linkování: `gcc -nostdlib ...`
  - `gcc` stále může vkládat volání `memset` apod.

Volací konvence

Parametr	Registry			
	Uživatelský prostor		Systémová volání	
	`x86_32`	`x86_64`	`x86_32`	`x86_64`
1.	EAX	RDI	EBX	RDI
2.	EDX	RSI	ECX	RSI
3.	Zásobník	RDX	EDX	RDX
4.	Zásobník	RCX	ESI	R10
5.	Zásobník	R8	EDI	R8
6.	Zásobník	R9	EBP	R9
7.	Zásobník	Zásobník	*Jen 6 parametrů*	
8.	Zásobník	Zásobník		
Návratová	EAX	RAX	EAX	RAX

Pozn.: `x86_32` a `-mregparm`

Úkol

Systémová volání bez `libc`

1. Otevřete a projděte si `pb173-bin/08/`
2. Zavolejte
   - `fork`
   - Z potomka: `write` na standardní výstup
   - Z rodiče: `read` 16 bytů a jejich `write` na standardní výstup
   - Z obou potom: `exit`
3. Přeložte a spusťte
   - `gcc -nostdlib ...`

Data z jádra

- Jádro předává
  - Parametry programu
  - Proměnné prostředí
  - Rozšiřující vektor
- Vše na zásobníku
  - Formát zásobníku je pevně daný

Formát zásobníku

Počet parametrů	`long`
1. parametr	`char *`
...	`char *`
m-tý parametr	`char *`
Konec parametrů	`OUL`
1. proměnná prostředí	`char *`
...	`char *`
n-tá proměnná prostředí	`char *`
Konec proměnných prostředí	`OUL`
Rozšiřující vektor 1	`struct { long type, val; }`
...	`struct { long type, val; }`
Rozšiřující vektor o	`struct { long type, val; }`
Konec rozšiřujících vektorů	`{AT_NULL, ? }`

Rozšiřující vektor

Typ záznamů

```
struct {
  long type;
  long value;
};
```

- Typy: `AT_NULL`, `AT_ENTRY`, `AT_RANDOM`, ...
- Pole struktur zakončuje typ `AT_NULL`

Úkol

Zjištění názvu programu a platformy

1. Pište do programu `pb173-bin/08/nostd.c`
2. Vypište první parametr programu
   - Tj. název samotného programu
3. Dále iterujte přes zásobník až k rozšiřujícímu vektoru
4. Tam najděte typ `AT_PLATFORM`
5. Vypište hodnotu jako řetězec
   - Použijte `write`
6. Přeložte (bez `libc`) a spusťte

Virtuální systémová volání

`vsyscall`

- Jen na některých architekturách
  - A konfiguracích jádra
- Neprobíhá přepnutí kontextu
- Speciální stránka(y) s kódem namapovaným jádrem
- Podpora jen 3 funkcí
  - `gettimeofday`: aktuální čas (`0xffffffffff600000`)
  - `time`: čas v sekundách od Epochy (`0xffffffffff600400`)
  - `getcpu`: číslo CPU a NUMA uzlu (`0xffffffffff600800`)
- **Demo:** `/proc/self/maps`

Úkol

Volání `time` z `vsyscall`

1. Zavolejte adresu s `time`
2. Vypište hodnotu jako řetězec
3. Přeložte a spusťte několikrát

Virtuální dynamická knihovna

`vdso`

- Speciální knihovna přilinkovaná jádrem
- Podpora také závislá na architektuře
- Podobná `vsyscall`, ale více flexibilní
  - Navíc obsahuje ale jen `clock_gettime`
- **Demo:** `ldd` a `readelf`
- Knihovna v ELF formátu
  - Adresa začátku v auxv: `getauxval(AT_SYSINFO_EHDR)`
  - Dále se přečte ELF pomocí `libelf`
  - Ukázkový kód: `tools/testing/selftests/vDSO/parse_vdso.c`

Úkol: domácí úkol