Operační systémy 1

9. Cvičení

Radek Janoštík

KATEDRA
INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Úkoly z minula – jak to šlo?

- Kolik se vám toho podařilo udělat?
  1. Změna a ověření reprezentace znaků
  2. WinAPI – čtení souboru pomocí `ReadFile`
  3. Spusťte vhodnou aplikaci (notepad, kalkulačku), v TaskManageru zjistěte jeho pid.
  4. Pomocí funkce `GetProcessImageFileName` zjistěte cestu ke spuštěnému souboru.
  5. Pomocí funkce `TerminateProcess` jej ukončete.

- Dalo něco zabrat?

- Nějaké dotazy?

- Do písemek můžete nahlédnout po probraní látky.

Procesy v unixech

- *Proces* = základní entitou vykonávající (nějaký) kód, program
- Unixová filozofie – malé, krátké, jednoúčelové programy
  - Skládají se do větších celků pomocí skriptů *shellu*
  - Případně „řetězení“ vstupů a výstupů pomocí *rour (pipe)*
- Procesy mají stromovou hierarchii
  - První proces – *init* – spouští (tranzitivně) všechny ostatní
  - Jednoznačný identifikátor: `proces id - pid`
- Programové zjištění *pid*: funkce `pid_t getpid()` z `unistd.h`
- Příkazy `ps` a `pstree`

Vytvoření nového procesu

- Systémové volání `fork` dostupné jako funkce `pid_t fork()` z `unistd.h`
  - Vytvoří nový proces jako *potomka* procesu, který toto systémové volání zavolal
  - Potomek je klonem *rodičovského* procesu – bude vykonávat stejný kód nad kopií dat
  - Potomek má jiný `pid` a také vlastní paměťový prostor
- Musíme tyto dva procesy nějak rozlišit $\Rightarrow$ návratová hodnota funkce `fork()`
  - U rodiče vrátí `fork()` `pid` potomka
  - U potomka vrátí `fork()` hodnotu 0
  - Nastane-li chyba vrátí `fork()` zápornou hodnotu
- Jak ale spustit úplně jiný program?

Spuštění jiného programu

- Vytváření pouze kopií není moc praktické ⇒ systémové volání `exec`
  - Načte do paměti kód programu
  - Začne jej vykonávat

- Sada funkcí v `unistd.h`

```
1 int execl(const char *path, const char *arg, ...);
2 int execlp(const char *file, const char *arg, ...);
3 int execle(const char *path, const char *arg , ..., char *const envp[]);
4 int execv(const char *path, char *const argv[]);
5 int execvp(const char *file, char *const argv[]);
6 int execve(const char *filename, char *const argv [], char *const envp[]);
```

- Rozdíly ve způsobu předávání argumentů (proměnlivý počet vs. pole)
  - Konec pole nebo poslední argument signalizován hodnotou NULL (nepředává se nikde počet)
  - První prvek argumentů je vždy název programu

- Udává se celá cesta k programu
  - Kromě variant s `p` – hledá program v `PATH`

Ukončení procesu

- Aktuální běžící proces ukončíme systémovým volání `exit` – funkce `void exit(int)`
  - Obsažena v `stdlib.h`
  - Argument – návratová hodnota, která se předá rodiči
  - Doporučený rozsah 0 – 127
- Funkce `void abort()` – ukončí daný proces a uloží na disk obraz paměti (`core dump`)
  - Možné pak analyzovat program, hledat chyby
- Nástroj `kill` – ukončí jiný běžící proces (zašle signál, aby se ukončil)
  - `pid` jako argument

Čekání na procesy

- U souběžně běžících procesů nemůžeme vynutit ani predikovat vzájemné pořadí běhů
- Pro testování můžeme procesy uspávat pomocí funkce

```
unsigned int sleep(unsigned int seconds)
```

- Čekání na potomka – funkce z `sys/wait.h`
  - `pid_t wait(int *status)` – zastaví program a počká dokud neskončí některý potomek
  - `pid_t waitpid(pid_t pid, int *status, int options)` – počká na konkrétní `pid`
  - Vrací `pid` potomka
  - Pokud není ukazatel `status` `NULL` vrací informace o ukončení potomka
- Sada maker pro informaci o ukončení potomka:
  - `WIFEXITED(status)` vrací nenulové číslo, pokud potomek skončil normálně,
  - `WEXITSTATUS(status)` vrací návratový kód potomka, smí se použít, pokud je `WIFEXITED(status)` nenulová hodnota,
  - `WIFSIGNALED(status)` vrací nenulové číslo, pokud byl potomek ukončen signálem,
  - `WTERMSIG(status)` vrací číslo signálu, který proces ukončil, smí se použít, pokud je `WIFSIGNALED(status)` nenulová hodnota.

Úkol (1/3)

1. Napište program, který po svém spuštění vypíše své pid a bude provádět nějakou činnost, nedojde k jeho ukončení.
2. Identifikujte program pomocí nástroje `ps`.
3. Identifikujte program pomocí nástroje `pstree`.
4. Program ukončete pomocí kombinace kláves `ctrl+c`.

Úkol (2/3)

5. Vytvořte program, který zavolá fork(), a ověřte chování návratových hodnot.

6. S použitím `exec` spusťte program `uname --all`.

7. Ověřte, že pokud volání `exec` neselže, je nahrazen aktuální kód programu.

8. Spojte volání `fork` a `exec` tak, aby rodič vykonával nějakou činnost, zatímco potomek zavolá `uname --all` a ukončí se.

Úkol (3/3)

9. Upravte program(y) z předchozích úkolů tak, aby na neúspěšné volání `fork` nebo `exec` zareagovaly voláním `exit` nebo `abort`.

10. Ukončete nějaký běžící (ideálně nějaký méně důležitý) proces s pomocí nástroje `kill`.

11. Upravte předchozí úkol(y) tak, aby čekaly na dokončení potomka.