Základní práce s procesy v unixech 9

Vznik operačního systému Unix se datuje do přelomu šedesátých a sedmdesátých let minulého století. Omezený výkon počítačů té doby se propsal i do architektury tohoto operačního systému[1] a mimo jiné i do návrhu jeho rozhraní pro práci s procesy, které je ve srovnání s Windows výrazně jednodušší, přičemž pro drtivou většinu situací zcela dostačující.

1 Procesy v unixech

Historicky je v unixech základní entitou vykonávající program proces. Jinými slovy na proces můžeme nahlížet jako na instanci běžícího programu. K tradici unixových operačních systémů patří, že jednotlivé programy jsou malé, plnící jeden účel, a je možné je skládat do větších celků pomocí skriptů v unixovém *shell*u. Shell slouží jednak jako rozhraní pro interaktivní práci uživatelů formou příkazového řádku, ale protože se velice často jedná o plnohodnotný programovací jazyk, je možné v shellu sestavit program, který spouští a propojuje menší programy do větších programů.

Poznámka: V unixových operačních systémech se dá setkat s podrobnou dokumentací k jednotlivým programům/nástrojům. Ta je k dispozici jako program `man`, který obsahuje popis nástrojů[2] (např. `man ls` vypíše dokumentaci k příkazu `ls`) nebo funkcí standardní knihovny, např. `man fork` zobrazí popis funkce `fork()`.

1.1 Identifikace a organizace procesů

V unixových operačních systémech tvoří procesy stromovou hierarchii, přičemž v kořeni tohoto stromu je proces označovaný jako `init`, který je spuštěn jako první proces po spuštění operačního systému. Podobně jako na platformě Microsoft Windows je každý proces identifikován svým číslem, které se označuje jako *proces id* (zkráceně pid).

Aktuální pid procesu lze získat pomocí funkce `pid_t getpid()`, jejíž prototyp je v hlavičkovém souboru `unistd.h`.

Pro získání přehledu o běžících procesech se v unixech používá příkaz `ps`. Ten bez parametrů vypíše seznam procesů spojených s aktuálním terminálem. Pomocí dalších přepínačů lze tento výpis rozšířit, např. `ps -u` vypíše všechny procesy daného uživatele, `ps aux` vypíše podrobné informace o všech procesech. Více viz dokumentace programu `ps`.

Pokud by nás zajímala hierarchie procesů, můžeme použít nástroj `pstree`, který jednotlivé procesy zobrazí jako strom.

[1] A do operačních systémů z něj odvozených jako je např. GNU/Linux.

[2] Záleží na tom, jaká dokumentace je nainstalována.

Úkoly:

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`.

1.2 Vytvoření nového procesu

V unixových operačních systémech k vytvoření nového procesu slouží systémové volání `fork`, které je programům v jazyce C k dispozici jako funkce `pid_t fork()`, jejíž prototyp se nachází v hlavičkovém souboru `unistd.h`. Toto systémové volání vytvoří nový proces, který je potomkem procesu, jenž zavolal `fork`[3] a je to klon rodičovského procesu. To znamená, že rodič i potomek vykonávají stejný kód a každý má vlastní kopii dat. Rozdíl je v tom, že potomek má přiřazené nové unikátní id a rodič i potomek mají svůj oddělený paměťový prostor. Protože po zavolání `fork()` existují v paměti dva procesy vykonávající stejný kód, je potřeba je nějakým způsobem rozlišit. K tomu slouží návratová hodnota této funkce. Pokud se vykonávaný kód nachází v rodiči, funkce `fork` vrací pid potomka, pokud se vykonávaný kód nachází v potomkovi, vrací funkce `fork()` hodnotu 0. Pokud `fork()` vrátí zápornou hodnotu, došlo k chybě.

Úkol: Vytvořte program, který zavolá `fork()`, a ověřte výše popsané chování.

1.3 Spuštění jiného programu

Vytvoření identické kopie procesu má pouze omezené použití a v praxi často potřebujeme spustit jiný program. K tomu slouží systémové volání `exec`, které do paměti nahraje kód programu a začne jej vykonávat. Toto systémové volání je k dispozici jako sada funkcí s prototypy v hlavičkovém souboru `unistd.h`.

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

Funkce `execl` a `execv` se liší ve formě argumentů, kdy první tři jsou funkce s proměnlivým počtem argumentů a poslední tři funkce své argumenty přebírají formou ukazatele na pole hodnot. V obou případech je konec pole signalizován hodnotou `NULL`. S výjimkou funkcí `execlp` a `execvp` se uvádí plná cesta k souboru s programem. Funkce `execlp` a `execvp` umí pracovat jen s jménem programu a vyhledávají daný program podle nastavení proměnné prostředí `PATH`. Funkce `execle` a `execve` předávají spouštěnému programu proměnné prostředí ve tvaru `promenna=hodnota`.

[3] Tj. rodičovského procesu.

Ve všech případech, pokud došlo k selhání funkce, je vrácena záporná hodnota.

Pozor, jako první prvek pole argumentů je předávaný název programu.

Úkoly:

5. S použitím `exec` spusťte program `uname --all`.
6. Ověřte, že pokud volání `exec` neselže, je nahrazen aktuální kód programu.
7. Spojte volání `fork` a `exec` tak, aby rodič vykonával nějakou činnost, zatímco potomek zavolá `uname --all` a ukončí se.

1.4 Ukončení procesu

Aktuálně běžící proces je možné ukončit systémovým voláním `exit`, které je realizováno jako funkce `void exit(int)`, kde argument odpovídá návratové hodnotě, která je předána rodiči. Používat by se měly hodnoty jen z rozsahu 0 až 127, další hodnoty mají svůj vyčleněný význam. Tuto funkci najdeme deklarovanou v hlavičkovém souboru `stdlib.h`, kde je k dispozici ještě prototyp funkce `void abort()`, která ukončí aktuálně běžící program a uloží na disk obraz paměti (tzv. core dump), který je možné použít další k analýze programu a identifikaci chyb.

K ukončení běžícího programu jiným procesem slouží nástroj `kill`, který zašle signál procesu, aby se ukončil.[4] Používá se ve tvaru `kill <pid>`.

Úkoly:

8. Upravte program(y) z předchozích úkolů tak, aby na neúspěšné volání `fork` nebo `exec` zareagovaly voláním `exit` nebo `abort`.
9. Ukončete nějaký běžící (ideálně nějaký méně důležitý) proces s pomocí nástroje `kill`.

1.5 Čekání

Při souběžné práci s procesy nemáme garantované, že budou prováděny v určitém pořadí, ani jak bude jejich vzájemný souběh řešen. Při testování může být výhodné proces na nějakou dobu uspat. K tomu se dá použít funkce `unsigned sleep(unsigned seconds)`, která uspí aktuální proces na daný počet sekund.

1.5.1 Čekání na potomka

Často potřebujeme provést kód v potomkovi a v rodiči počkat, až tento kód doběhne. K tomu máme dispozici funkce `pid_t wait(int *status)` a `pid_t waitpid(pid_t pid, int *status, int options)` v hlavičkovém souboru `sys/wait.h`. Obě tyto funkce zastaví běh programu, dokud neskončí některý potomek (funkce `wait`) nebo potomek s daným pid (funkce `waitpid`).[5] Obě funkce vrací pid potomka, a pokud ukazatel `status` není NULL, jsou vráceny informace o ukončení potomka. S těmi se pracuje pomocí sady maker, např.

[4] Toto je výchozí chování, nástroj umožňuje zasílat i další signály.

[5] Může se stát, že potomek dokončí svou činnost dřív, než je zavolána funkce `wait`, v takovém případě není běh programu pozastaven a funkce vrací výsledek okamžitě.

- `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.

1.5.2 Zombie procesy

Pokud potomek skončí a rodič na něj nečeká voláním `wait`, vznikne tzv. *zombie proces*, tj. proces, který skončil, ale ještě v systému existuje, dokud si rodičovský proces nevyzvedne informace o jeho ukončení pomocí `wait`. Pokud si rodič tyto informace nevyzvedne nikdy (ani po svém ukončení), zombie proces je adoptován procesem init, který jej odstraní ze systému.

Úkol:

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