Základní meziprocesová komunikace v Linuxu I: Signály

4

1 Signály

Mezi nejzákladnější nástroje pro meziprocesovou komunikaci v unixových operačních systémech patří tzv. signály. Jedná se o sadu zpráv, které mohou být procesu (nebo některému z jeho vláken) asynchronně zaslány, a proces na ně může zareagovat. Signály se používají k ukončení procesu, k ošetření chybových stavů nebo základní správě procesů. Do jisté míry se podobají přerušením, jak je používá hardware.

Je nanejvýš pravděpodobné, že každý, kdo pracoval v Linuxu s programy v příkazové řádce, se signály již setkal. Příkladem budiž kombinace kláves `ctrl+c`, která pošle procesu signál, který se označuje `SIGINT` a znamená, že by měl proces ukončit prováděnou činnost. Vychozí chování je, že se program ukončí. Toto chování lze ale změnit a některé programy toho využívají. Například máme-li interpreter programovacího jazyka a pracujeme-li s ním přes rozhraní typu, REPL[1] může kombinace `ctrl+c` resp. zaslání signálu `SIGINT` přerušit prováděný výpočet, aniž by došlo k ukončení samotného interpretru.

Pro explicitní ukončení procesu slouží signál `SIGKILL`, který obstará ukončení procesu na úrovni jádra OS. Avšak tato forma ukočení je často příliš radikální, protože daný proces nemá šanci na své ukončení jakkoliv zareagovat, například nemůže uložit rozpracovaná data. Proto by se tato forma ukončení procesu měla používat až v případě, že všechny ostatní možnosti selžou. Pro korektní ukončení procesů existuje signál `SIGTERM`, po jehož obdržení by proces měl provést nezbytné kroky pro své ukončení a sám se ukončit.

Další běžný signál, se kterým se setkávají zejména programátoři, kteří vytváří programy v jazyce C/C++ je signál `SIGSEGV`,[2] kterým je procesu oznámeno, že provedl neplatný přístup do paměti a měl by na to zaragovat.[3] Dalším signálem, se kterým se můžeme setkat, je např. signál `SIGSTOP`, který způsobí zastavení (uspání) procesu do doby než obdrží signál `SIGCONT`. Signál `SIGSTOP` je možné procesu zaslat pomocí kombinace kláves `ctrl+z`.

Z uživatelského pohledu se k zasílání signálů používají buď klávasové zkratky (viz předchozí odstavce) nebo nástroj (program) `kill`. Program `kill` akceptuje jako své argumenty signál a id procesu (pid), kterému má být signál zaslán. Signál je určen buď svým číslem nebo pojmenováním. Seznam signálů lze získat příkazem `kill -L`.

Zaslání signálů:

```
kill -s <nazev-signalu> <process-id>
```

[1] Read-Eval-Print-Loop

[2] Segmentation violation

[3] Výchozí chování je vypsání chybové hlášky *Segmentation fault* a ukončení procesu.

```
kill -n <cislo-signalu> <process-id>
```

Příklady:

```
kill -s SIGTERM 42
kill -n 9 7475
```

Identifikátor procesu můžeme získat např. s pomocí nástroje `ps` nebo `top`. Případně můžeme použít nástroj `killall`, kde se proces identifikuje pomocí svého názvu a signál je zaslán všem procesům daného jména.

Úkol č. 1: Vytvořte program, který bude v nekonečné smyčce vypisovat na standardní výstup přirozená čísla. Mezi výpisy jednotlivých hodnot vložte pauzu jednu až dvě sekundy pomocí funkce `sleep`. Tento program spusťte a zkuste zaslat tomuto procesu různé signály `SIGTERM`, `SIGQUIT`,[4] `SIGKILL`, `SIGSTOP`, využijte pro tyto účely klávesové zkratky i příkaz `kill`.

Úkol č. 2: Spusťte nástroj `ping`, který bude testovat dostupnost počítače na nějaké IP adrese (např. 127.0.0.1). Zkuste tomuto procesu zaslat signály jako v předchozím úkolu a sledujte rozdíly.

2 Definice vlastních reakcí na signály

Většina signálů má definované své implicitní chování (např. ukončení procesu) nebo je ignorována. Avšak s výjimkou některých konkrétních signálů, jako jsou `SIGKILL`, `SIGSTOP` nebo `SIGCONT`,[5] je možné definovat funkce, které jsou zavolány, pokud proces obdrží zadaný signál. Např. můžeme proces bezpečně ukončit po obdržení signálu `SIGTERM` nebo reagovat jiným způsobem, např. nástroj `ping` po obdržení signálu `SIGQUIT` vypíše průběžné statistiky, ale neukončí se.

K určení, co se má provést po obdržení signálu slouží funkce `sigaction`, jejíž prototyp a datavé struktury, se kterými pracuje, najdeme v hlavičkovém souboru `<signal.h>`. Funkce, má tři parametry, kde první parametr je číslo signálu, jehož obsluhu chceme definovat, dále následuje odkaz na strukturu, která definuje chování při obdržení signálu, a pomocí třetího parametru můžeme získat předchozí nastavení obsluhy daného signálu.

K definici obsluhy signálu slouží struktura `struct sigaction`, která ma atribut `sa_handler`, což je ukazatel na funkci typu `void foo(int)` a jedná se o funkci, která bude zavolána po obdržení signálu. Jako argument je této funkci předán obdržený signál.

Dále struktura obsahuje atribut `sa_flags`, kde jsou uloženy příznaky upravující chování obslahy signálu. Protože obsluha příchozího signálu je kód programu, který je prováděný jako každý jiný, může proces obdržet další signál, na který by mělo být zareagováno. V takovém případě ale hrozí chyba souběhu.[6] Aby se tomu dalo zabránit, je možné pomocí atributu `sa_mask` určit, které signály jsou tzv. zamaskovány a jejich zpracování je odloženo do doby, než bude dokončena právě probíhající obsluha signálu.

Použití ukazuje následující příklad:

[4] Tento signál standardně slouží k ukončení procesu a získání obrazu paměti (core dumpu) a je možné jej vyvolovat pomocí `ctrl+\`.

[5] Tyto signály jsou de facto určené pro operační systém.

[6] race condition

```
void my_handler(int id)
{
        printf("signal: %i\n", id);
}

// nastaveni obsluhy signalu
struct sigaction sa;
sa.sa_handler = my_handler;
sa.sa_flags = 0;
sigemptyset(&sa.sa_mask);
sigaddset(&sa.sa_mask, SIGINT);

if (sigaction(SIGINT, &sa, NULL) == -1) {
        printf("error: handler not installed\n");
}
```

Úkol č. 3: Do programu z úkolu č. 1 vhodným způsobem začleňte výše zmíněný kód a ověřte jeho funkčnost.

Pokud proces čeká na blokující operaci (např. je uspán funkcí `sleep` nebo čeká na nějakou I/O operaci) a obdrží signál, který není ignorován, je možné, že daná operace bude přerušena, např. čeká se méně než předepsaný počet sekund, operace není dokončena.[7]

Úkol č. 4: Ověřte, že v případě příchozího signálu, funkce `sleep` nedokončí zadanou dobu čekání.

3 Další funkce pro práci se signály

- `int kill(pid_t pid, int sig)`

  – zašle signál `sig` procesu s `pid`

- `int raise(int sig)`

  – proces zašle signál `sig` sám sobě

- `int pthread_kill(pthread_t thread, int sig)`

  – vláknu `thread` je zaslán signál `sig`

- `int pause(void)`

  – pozastaví běh procesu, dokud neobdrží signál

- `int alarm(unsigned seconds)`

[7] V takovém případě je přerušení operace signalizováno návratovou hodnotou. Toto chování je nutné, aby procesy mohly korektně reagovat na příchozí signály. U jednotlivých funkcí OS je vhodné konzultovat s dokumentací, jak se chovají v případě příchozího signálu.

– za `seconds` zašle aktuálnímu procesu signál `SIGALRM`.

Úkol č. 5: Rozšiřte řešení z úkolu č. 1 (resp. 3) tak, aby po deseti sekundách došlo k záslání signálu `SIGALRM` a vypsání informace o tom, že uplynulo již deset sekund.

Úkol č. 6: Vytvořte program, který jako svého potomka spustí nástroj `ping` (viz úkol č. 2), třikrát po sobě pět sekund vyčká a pošle mu signál `SIGQUIT`, a nakonec mu pošle signál `SIGTERM`.