4. Procesy v C a make

Vytváření procesů v C a překlad make

Domácí příprava

Předpokládáme, že máte základní znalosti jazyka C a víte, jak funguje překlad ze zdrojových kódů jazyka C do binární spustitelné aplikace (v obecném případě, kdy je zdrojových souborů více).

Dále byste měli mít alespoň minimální povědomí o použití překladače `gcc` a jeho základních parametrech.

Nastudujte si použití nástroje `make` pro překlad programu v jazyku C/C++: make.

Dále je pro absolvování cvičení nutné mít přehled o systémových voláních fork, pipe, dup, open, kill, wait a execve, tzn. měli byste vědět jak vzniká nový proces a jak lze přesměrovat standardní vstup a výstup. Potřebné informace se dozvíte na některé z předchozích přednášek.

Zadání úlohy

Vytvořte v jazyce C, C++ nebo Rust program `forkpipe`, který:

- Vytvoří dva procesy (potomky) voláním funkce `fork`. Prvně vytvořený proces budeme níže nazývat *GEN*, druhý *NSD*. Původní (hlavní) proces bude označován jako *MAIN*. (POZOR, záleží na pořadí vytvoření potomků).
- Tyto potomky propojí rourou (funkce `pipe` a `dup2`) tak, aby standardní výstup *GEN* byl napojen na standardní vstup *NSD*.

- Poté počká 5 sekund pomocí volání `sleep(5)`.
- Poté pošle procesu `GEN` signál `SIGTERM`, který způsobí ukončení obou potomků.

> *GEN* skončí díky obsluze signálu – viz níže, *NSD* by pak měl skončit automaticky také, neboť při ukončení *GEN* OS uzavře rouru a *NSD* je naprogramován tak, že skončí při uzavření standardního vstupu.

- Následně počká na ukončení procesů *GEN* a *NSD* (např. pomocí funkce `wait`).
- Pokud libovolný z potomků skončil chybou (návratový kód `!= 0`), vypíše na standardní výstup řádku "`ERROR`" a skončí s návratovým kódem `1`, v opačném případě vypíše tamtéž řádku "`OK`" a skončí s kódem `0`.
- Pokud libovolné systémové volání vrátí chybu, aktuální proces okamžitě skončí s návratovým kódem `2`.
- *GEN* bude na standardní výstup vypisovat řádky obsahující dvě mezerou oddělená náhodná čísla vygenerovaná funkcí `rand()` (např. `printf("%d %d\n", rand(), rand())`).

> Pozor, při generování příliš vysokých čísel může program nsd počítat i několik sekund. Můžete předpokládat, že při testech v BRUTE nevrátí funkce `rand()` větší číslo než `4095` a váš program urychlit použitím `rand() % 4096` místo samotného `rand()`.

- Mezi výpisem řádek bude *GEN* čekat jednu sekundu pomocí volání `sleep(1)`.
- *GEN* bude reagovat na signál `SIGTERM`. Při zaslání tohoto signálu *GEN* vypíše na standardní chybový výstup řádku "`GEN TERMINATED`" a skončí s návratovým kódem `0`.
- *NSD* zavolá po vytvoření funkci `execl`, aby začal vykonávat program `nsd` (viz níže). Žádný jiný proces by neměl `execl` volat.

Dále vytvořte `Makefile`, který:

- Při spuštění příkazu `make` bez parametrů zkompiluje váš program `forkpipe` a program `nsd` pro výpočet největšího společného dělitele, jehož zdrojové kódy si stáhněte z nsd.tgz. Obě binárky program umístí do stejného adresáře jako soubor `Makefile`.
- Při změně libovolného zdrojového souboru `make` překompiluje pouze soubory, které jsou z daného souboru generovány (ať přímo či nepřímo). Ostatní generované

soubory měněny nebudou.
- Překladači jazyka C/C++ vždy předá volbu `-Wall`, která způsobí výpis užitečných varování, a dále volby v proměnné `EXTRA_CFLAGS`, která je ve výchozím stavu prázdná.

Poznámky k implementaci

- Rouru musíte vytvořit dříve než potomky, abyste je s ní mohli propojit.
- Po vytvoření potomků rodičovský proces rouru nepotřebuje a měl by ji uzavřít. Pokud ji neuzavře, *NSD* neskončí automaticky po ukončení *GEN*, jak je popsáno výše, protože roura bude stále otevřená.
- Uzavírejte důsledně všechny file descriptory, které již nebudete potřebovat. Předejdete tím záludným chybám.
- Z funkce obsluhující signál (tzv. *signal handler*) není možné volat libovolnou funkci. Viz man signal-safety. Zejména by se nemělo používat "buffered I/O", tedy např. funkce `printf`. Pokud nepodporovanou funkci zavoláte, program může (ale nemusí) zhavarovat (jedná se o tzv. chybu souběhu neboli "*race condition*").
- Pro ladění programu se vám můžete hodit příkaz `strace -f`. Vyzkoušejte také argumenty `-e process`, `-e file`, `-e trace=dup2`, apod.
- Odevzdávejte všechny soubory zabalené v archivu. Můžete použít příkaz

C Rust

```
tar czf osy04.tgz Makefile *.[ch]
```

- Pokud program implementujete v Rustu můžete používat následující Crates a to tak, že v `Cargo.toml` uvedete:

```
[dependencies]
nix = { version = "0.29", features = ["signal", "fs"] }
libc = "0.2"
anyhow = "1.0"
lazy_static = "1.5"
```

Ne všechny poskytnuté crates je nutné využít, správné řešení v Rustu může jít napsat i bez nich. Dostupné verze crates jsou dané tím, jaké verze byly použity při vytváření obrazu pro BRUTE a mohou být starší než jaké budete používáte lokálně.

Program překládejte pomocí `cargo build --offline` (v Makefile).

Náhodná čísla generujte pomocí `libc::rand()`. Standardně se v Rustu používá crate `rand`, v rámci AE ale používáme vlastní generátor náhodných čísel, aby byly výsledky reprodukovatelné, a ten aktuálně funguje pouze s `libc::rand()`.

Aplikace nsd

Zdrojové kódy aplikace `nsd` se skládají ze souborů `nsd_main.c`, `nsd.c` a `nd.c` a hlavičkových souborů `nsd.h` a `nd.h`. Jejich přeložením vytvoříte binární soubor '`nsd`'.

Zdrojové soubory naleznete v archivu nsd.tgz.

Ukázkový kód

Jak vytvořit proces a rouru se můžete inspirovat ukázkovým kódem z manuálové stránky systémového volání pipe.

Domácí příprava na další cvičení

Pro další cvičení budete potřebovat vědět

- co jsou vlákna,
- jaké problémy mohou nastávat při paralelních běhů vláken,
- jaké synchronizační prostředky máme k dispozici a
- jak je vytvoříme v jazyce C s využitím knihovny **pthread**.

Potřebné informace byste měli získat na přednáškách, případně se podívejte na

- Pthread tutorial
- Manuálové stránky funkcí pthread