Vlákna a synchronizace

Na tomto cvičení byste si měli vyzkoušet, jak vytvořit vlákno v jazyce C, C++ či Rust s využitím knihovny `pthread` a jak vlákna synchronizovat tak, aby nedošlo k poškození dat, se kterými se pracuje z více vláken.

Domácí příprava

Pro toto cvičení budete potřebovat znalosti o tom

- co jsou vlákna, mutexy a semafory,
- jak tyto prostředky vytvoříte v jazyce C s využitím knihovny `pthread`,
- jak se vlákna vytvářejí a ukončují,
- jaké problémy mohou nastávat při paralelním běhů vláken a
- jak psát programy tak, aby tyto problémy nenastaly.

Potřebná teorie byla vyložena na přednášce, včetně ukázek použití funkcí knihovny `pthread`. Před cvičením je doporučeno podívat se na manuálové stránky potřebných funkcí, zejména:

- `pthread_create`, `pthread_join`,
- `pthread_mutex_init`, `pthread_mutex_destroy`, `pthread_mutex_lock`, `pthread_mutex_unlock`
- `sem_init`, `sem_destroy`, `sem_wait`, `sem_post`.

Také se můžete podívat na videa Unix Threads in C.

Zadání úlohy

Implementujte v jazyce C, C++, nebo Rust vícevláknový program `prod-cons` splňující následující požadavky:

- V hlavním vlákně (funkce `main()`) se vytvoří jedno vlákno, kterému budeme říkat *producent* a dále `N` vláken *konzument*.
- Hodnota `N` bude zadávána jako parametr při spouštění programu (`argv[1]`). Pokud žádný parametr nebude zadán, bude `N` rovno `1`.
- `N` musí být v rozmezí 1 až počet CPU v systému (`sysconf(_SC_NPROCESSORS_ONLN)`). Pokud bude zadána jiná hodnota, program skončí s návratovým kódem `1`.
- Producent bude splňovat následující:
  - Bude číst ze standardního vstupu "příkazy" ve formě dvojic `<X> <slovo>`, kde `<X>` je celé nezáporné číslo a `<slovo>` je libovolná neprázdná sekvence znaků (kromě whitespace). `X` je od slova odděleno jednou mezerou, jednotlivé příkazy jsou od sebe vždy odděleny koncem řádku (`\n`). Na konci vstupu může, ale nemusí být konec řádku (`\n`).

    Platný vstup tedy může vypadat například takto:

    ```
    20 foo
    5 bar
    1 baz
    ```

    Délka slova je omezená pouze velikostí dostupné paměti. To znamená, že **slovo musíte mít v paměti uloženo jen jednou**. Na víc kopií nemusíte mít dost paměti.

    K načítání příkazů doporučujeme použít následující kód:

C/C++ on GNU/Linux | C/C++ on MacOS | Rust

```
int ret, x;
char *text;
while ((ret = scanf("%d %ms", &x, &text)) == 2) {
    ...
}
```

Direktiva `%ms` (malloc string) způsobí, že `scanf` dynamicky alokuje takové množství paměti, které je potřeba pro uložení načítaného slova. Nezapomeňte potom tuto paměť uvolnit funkcí `free()`.

  - Pokud bude zadán neplatný příkaz (tj. neodpovídající předchozímu bodu), program skončí s návratovým kódem `1`, ale až poté, co budou zpracovány předchozí, platné příkazy (viz také požadavky níže).
  - Pro každý přečtený příkaz dynamicky alokuje (`malloc()`, `new`, …) datovou strukturu, uloží do ní `X` a `slovo` a zařadí ji na konec spojového seznamu.
- Každý konzument bude splňovat následující:
  - Bude ze začátku spojového seznamu vybírat položky vkládané producentem.
  - Pokud v seznamu žádná položka není, bude čekat, až tam producent něco přidá (bez spotřeby výpočetního času, žádný polling).
  - Pokud producent přidá `P` položek, vzbudí se maximálně `P` konzumentů, ostatní budou dále čekat.
  - Pro každou vyzvednutou položku konzument vypíše na standardní výstup řetězec "`Thread n: slovo slovo slovo...`", kde `n` je číslo konzumenta (pořadí vytvoření konzumenta v rozsahu 1– `N`) a slovo se opakuje X-krát (informace od producenta). Tento řetězec bude celý na jedné řádce ukončené `\n`.
- Pouze producent bude číst ze standardního vstupu.
- Pouze konzumenti budou zapisovat na standardní výstup.
- Standardní chybový výstup můžete použít k ladicím výpisům.
- Uzavření standardního vstupu je požadavkem na ukončení programu. Pokud není

řečeno jinak, návratový kód bude `0`.

- Všechny platné "příkazy" zaslané na standardní vstup budou mít odpovídající řádku na standardním výstupu (nic se neztratí).
- Žádné čekání ve vašem programu by nemělo být implementováno formou pollingu (periodická kontrola, že se něco stalo).
- Program před ukončením uvolní všechnu dynamicky alokovanou paměť.

Do odevzdávacího systému nahrajte:

C/C++ Rust

Svůj zdrojový kód a `Makefile`, který vygeneruje program `prod-cons` ve stejném adresáři jako `Makefile`.

Nenahrávejte zkompilované binární soubory (.o apod.).

Program překládejte s příznaky `-Wall -g -O2` a navíc s příznaky v proměnné `EXTRA_CFLAGS` (evaluátor ji bude nastavovat podle potřeby). Pokud tato proměnná není definována na příkazové řádce `make`, nastavte její hodnotu na "`-fsanitize=address -fno-omit-frame-pointer`" (viz např. operátor `?=`). Pokud provádíte překlad a linkování odděleně, používejte příznaky v `EXTRA_CFLAGS` také při linkování.

Překladač nesmí generovat žádná varování.

Testovací vstup

Testovat váš program můžete např. následovně:

```
(echo "20 foo"; echo "3 bar"; echo "5 baz") | ./prod-cons 4
```

Program by měl vypsat zhruba toto (čísla threadů a pořadí řádků může být jiné):

```
Thread 1: foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo foo 1
Thread 2: bar bar bar
Thread 1: baz baz baz baz baz
```

Zkuste i chování při neplatném vstupu:

```
echo "invalid" | ./prod-cons 4
echo "Exit code: $?"
```

Výsledkem by mělo být:

```
Exit code: 1
```

Domácí příprava na další cvičení

Nastudujte si použití podmínkových proměnných a k tomu příslušné funkce v knihovně `pthread`:

- `pthread_cond_init`
- `pthread_cond_destroy`
- `pthread_cond_signal`
- `pthread_cond_broadcast`
- `pthread_cond_wait`