Vlákna

Tématicky zaměřený vývoj aplikací v jazyce C
skupina Systémové programování – Linux

Martin Dražar, Martin Husák, Petr Velan

Fakulta informatiky
Masarykova univerzita
drasar|husak|velan@ics.muni.cz

Brno, 2. listopadu 2015

Vlákna

motivace, použití a práce s vlákny

Motivace

Vlákna jako nástroj pro

- load balancing
- rozdělení problému a urychlení výpočtu
- komunikace (interakce s uživatelem)
- pokračování práce během potenciálně velmi dlouhého blokování procesu (např. IO)
- potřeba synchronně reagovat na typicky asynchronní podněty

Co je to vlákno

Proces (pro připomenutí)

Instance běžícího programu – objekt pracující podle kódu programu, má vlastní adresní paměťový prostor, využívá prostředky jádra a komunikuje s ostatními procesy.

Co je to vlákno

Proces (pro připomenutí)

Instance běžícího programu – objekt pracující podle kódu programu, má vlastní adresní paměťový prostor, využívá prostředky jádra a komunikuje s ostatními procesy.

Vlákno

Objekt pracující podle kódu programu, který je ale součástí procesu a sdílí jeho prostředky s ostatními vlákny procesu.

Co je to vlákno

- Z hlediska jádra je důležitý pojem úloha - pro ni se plánuje čas CPU - úlohou je každé vlákno procesu (mapování 1:1) a každý proces má alespoň jedno vlákno.
- Vlákno je abstrakcí toku výpočtu – aktivity procesu. Jde o samostatně proveditelný tok instrukcí, který lze využít ke strukturování programu.
- Co se životního cyklu týče, jsou na tom vlákna stejně jako procesy.
- Z pohledu programátora jde o funkci, která běží samostatně v rámci procesu. Tyto funkce běží v rámci programu souběžně.

Vlastnosti vláken

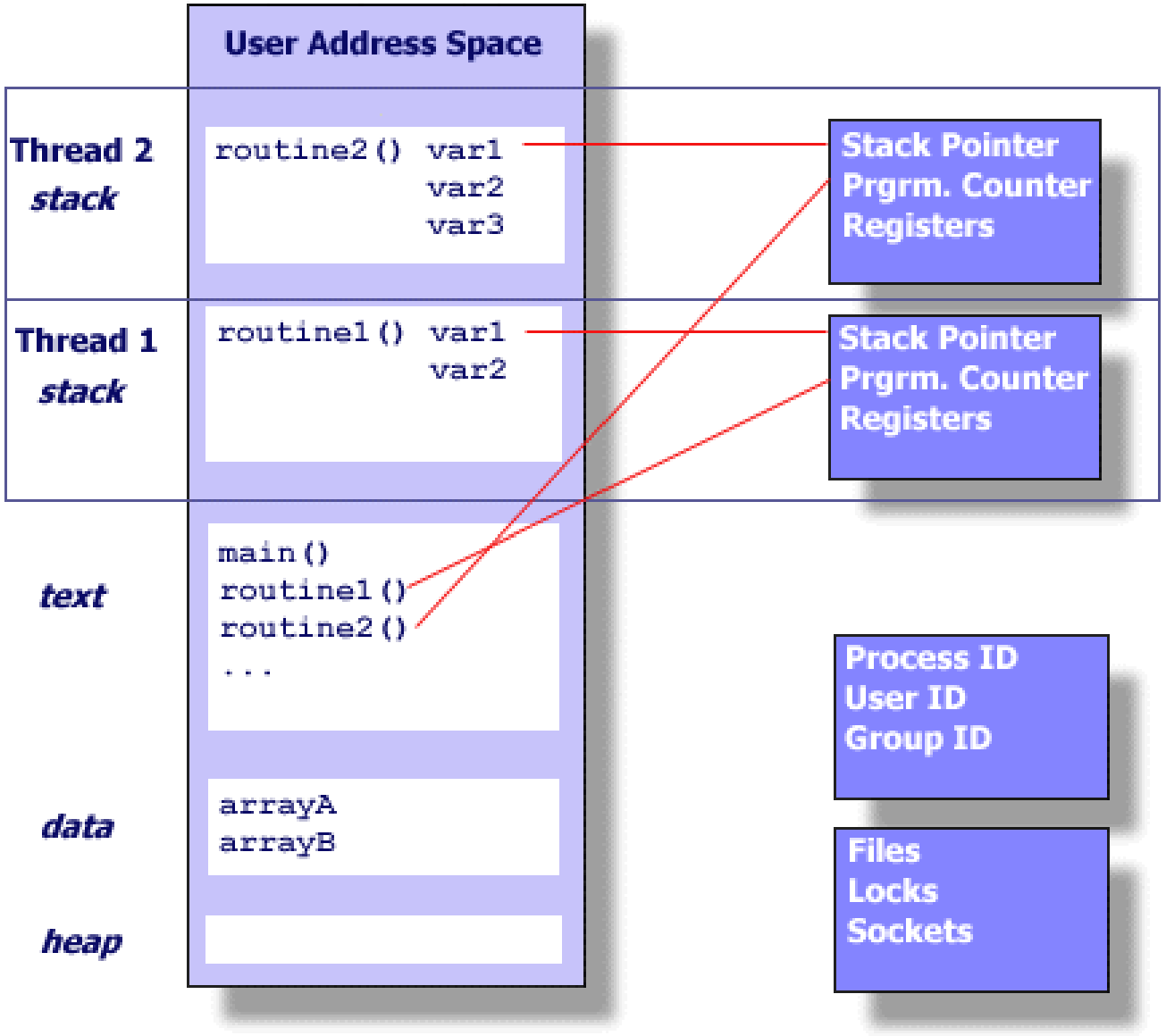
V rámci procesu vlákna **sdílí**

- obsluhu signálů
- kontext paměti
- kontext prostředí
- otevřené soubory
- $\rightarrow$ nutnost synchronizace přístupu ke zdrojům (příští téma)

Naopak vlákna v rámci procesu **nesdílí**

- kontext procesoru (CPU se přiděluje vláknům)
- zásobník
- *signály*
- lokální data
- proměnné specifické pro vlákna

Vlákna v rámci procesu



Implementace vláken v Linuxu

- **LinuxThreads** – původní, dnes již nepodporovaná, částečná implementace vláken podle normy POSIX
- **Native POSIX Thread Library (NPTL)** – od jádra 2.6 plnohodnotná implementace POSIX Threads[1]
- obě implementace jsou ve skutečnosti poskytovány přes knihovnou glibc, nicméně podpora vláken je přímo v kernelu
- Zjištění používané verze implementace:

  `getconf GNU_LIBPTHREAD_VERSION`

Použití

- `#include <pthread.h>`
- `gcc -pthread`

Další informace

- `man 7 pthreads`

[1] IEEE POSIX 1003.1c

Vytvoření vlákna

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
```

- každé vlákno může vytvářet nová vlákna
- atributy vlákna je nutné nastavit před jeho vytvořením
- defaultní nastavení vlákna se deklarují pomocí NULL

ID vlákna

```
pthread_t pthread_self(void);
int pthread_equal(pthread_t t1, pthread_t t2);
(pid_t) syscall(SYS_gettid);
```

```
tid != pthread_t
```

Pozor na souběh a konstrukce typu:

```
for(i = 0; i < N; i++) {
    pthread_create(&tid, attr, start_routine, &i);
}
```

úkol

Napište program, který vytvoří určitý počet vláken, která vytisknou nějaký svůj identifikátor

úkol

Napište program, který vytvoří určitý počet vláken, která vytisknou nějaký svůj identifikátor
Možnosti: TID, arg (znak abecedy), . . . , (pthread_t)

Spojování vláken

```
int pthread_join(pthread_t thread, void **retval);
int pthread_detach(pthread_t thread);
```

- mechanismus čekání na dokončení vlákna
- nelze čekat sám na sebe
- mnohonásobné volání `pthread_join()` má nedefinované chování
- na detachované vlákno již nelze volat `pthread_join()` a detach nelze vzít zpět

Spojování vláken

```
int pthread_join(pthread_t thread, void **retval);
int pthread_detach(pthread_t thread);
```

- mechanismus čekání na dokončení vlákna
- nelze čekat sám na sebe
- mnohonásobné volání `pthread_join()` má nedefinované chování
- na detachované vlákno již nelze volat `pthread_join()` a detach nelze vzít zpět

úkol

Upravte předchozí program tak, aby hlavní vlákno skončilo vždy poslední

Atributy vlákna

- nastavení vlastností vlákna – lze je ale nastavit pouze při vytváření vlákna
- `man -k pthread_attr`

Použití

1. `pthread_attr_init()`
2. `pthread_attr_set*`
3. použití ve funkci `pthread_create()`
4. `pthread_attr_destroy()`

Příklady

- `pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED)`
- `pthread_attr_setstacksize()`

clean-up handlery (*destruktory*)

```
void pthread_cleanup_push(void (*routine)(void *), void *arg);
void pthread_cleanup_pop(int execute);
```

Příklad:

```
void on_cancel(void** data) {
    if(data != NULL) {
        free(*data);
        *data = NULL;
    }
}

void* ptr = NULL;
pthread_cleanup_push(on_cancel, &ptr);
ptr = malloc(100);
...
pthread_cleanup_pop(1);
```

Data specifická pro vlákna

- Proměnná je duplikována pro každé vlákno v jeho datové oblasti.
- S těmito proměnnými (klíči) je třeba pracovat jinak než s běžnými proměnnými.

```
int pthread_key_create(pthread_key_t *key, void (*destructor)(void*));
void *pthread_getspecific(pthread_key_t key);
int pthread_setspecific(pthread_key_t key, const void *value);
int pthread_key_delete(pthread_key_t key);
```

Příklad – logování do samostatného souboru v každém vlákně, ale společnou funkcí (typicky callback funkce).

Ukončení vlákna

Vlákno končí

- návratem z funkce vlákna
- voláním funkce `pthread_exit`
- zrušením vlákna jiným vláknem (`pthread_cancel`)

Rušení vlákna

Vlákno může být

- **asynchronně zrušitelné** – lze ho okamžitě zrušit
- **synchronně zrušitelné** – požadavky na zrušení se ukládají do fronty, zrušení proběhne až v místě, které to dovoluje (cancellation points)
- **nezrušitelné** – pokusy o zrušení se ignorují

Rušení vláken je na Linuxu implementováno pomocí real-time signálů.

Rušení vláken

```
int pthread_setcancelstate(int state, int *oldstate);
int pthread_setcanceltype(int type, int *oldtype);
```

Vždy uložte původní stav (typ)!

Stavy

- `PTHREAD_CANCEL_ENABLE`
- `PTHREAD_CANCEL_DISABLE`

Typy

- `PTHREAD_CANCEL_ASYNCHRONOUS`
- `PTHREAD_CANCEL_DEFERRED`

Body zrušení

- funkce `pthread_testcancel`
- seznam všech funkcí fungujících jako *cancellation points* je v `pthreads(7)`

Poznámky

- Pozor na ošetřování chyb – `pthread_*` funkce nenastavují errno, ale vracejí přímo chybový kód odpovídající errno, které je ale na Linuxu thread-safe.
- Vlákna a fork()
  - nový proces je kopií původního vlákna
  - ostatní vlákna neexistují, ale jimi naalokovaná paměť zůstává alokovaná (ztracená)
  - zůstávají zamčené mutexy (viz synchronizace příště)
  - má smysl pouze se současným použitím `exec`
- Vlákna a signály
  - `pthread_kill`, `pthread_sigqueue`
  - nastavení obsluhy signálů je stejné pro všechna vlákna
  - `pthread_sigmask`

Závěr

domácí úkoly a zdroje

Domácí úkol

- tanky se z procesů změní na vlákna world aplikace
- bude potřeba použít pthread_kill pro signály
- bude potřeba předat správně konec pipe a další parametry (jako parametr vlákna)
- na ukončení tanku se nebude čekat pomocí wait, ale pomocí pthread_join.
- vlákna musí správně maskovat signály, které chodí celému procesu od worldclienta

Zdroje

Vlákna

- computing.llnl.gov/tutorials/pthreads/
- www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html