# Vlákna

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

Jiří Novosad

Fakulta informatiky
Masarykova univerzita
novosad@fi.muni.cz

Brno, 19. října 2014

## Vlákna

motivace, použití a práce s vlákny

# Motivace

**Vlákna jako nástroj pro**

- rozdělení problému a urychlení výpočtu
- prolínání výpočtu a I/O
- komunikace (interakce s uživatelem)
- potřeba synchronně reagovat na typicky asynchronní podněty (např. signály)

# Co je to vlákno

## Vlákno

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

- Z hlediska jádra je důležitý pojem úloha – pro ni se plánuje čas CPU – úlohou je každé vlákno procesu (mapování 1:1).
- Každý proces má alespoň jedno vlákno – při spuštění procesu je to vlákno provádějící `main()` (tzv. hlavní 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 podobně jako procesy.
- Z pohledu programátora jde o proceduru, která běží samostatně v rámci procesu.
- Vlákna / procedury se provádí souběžně.

# Vlastnosti vláken

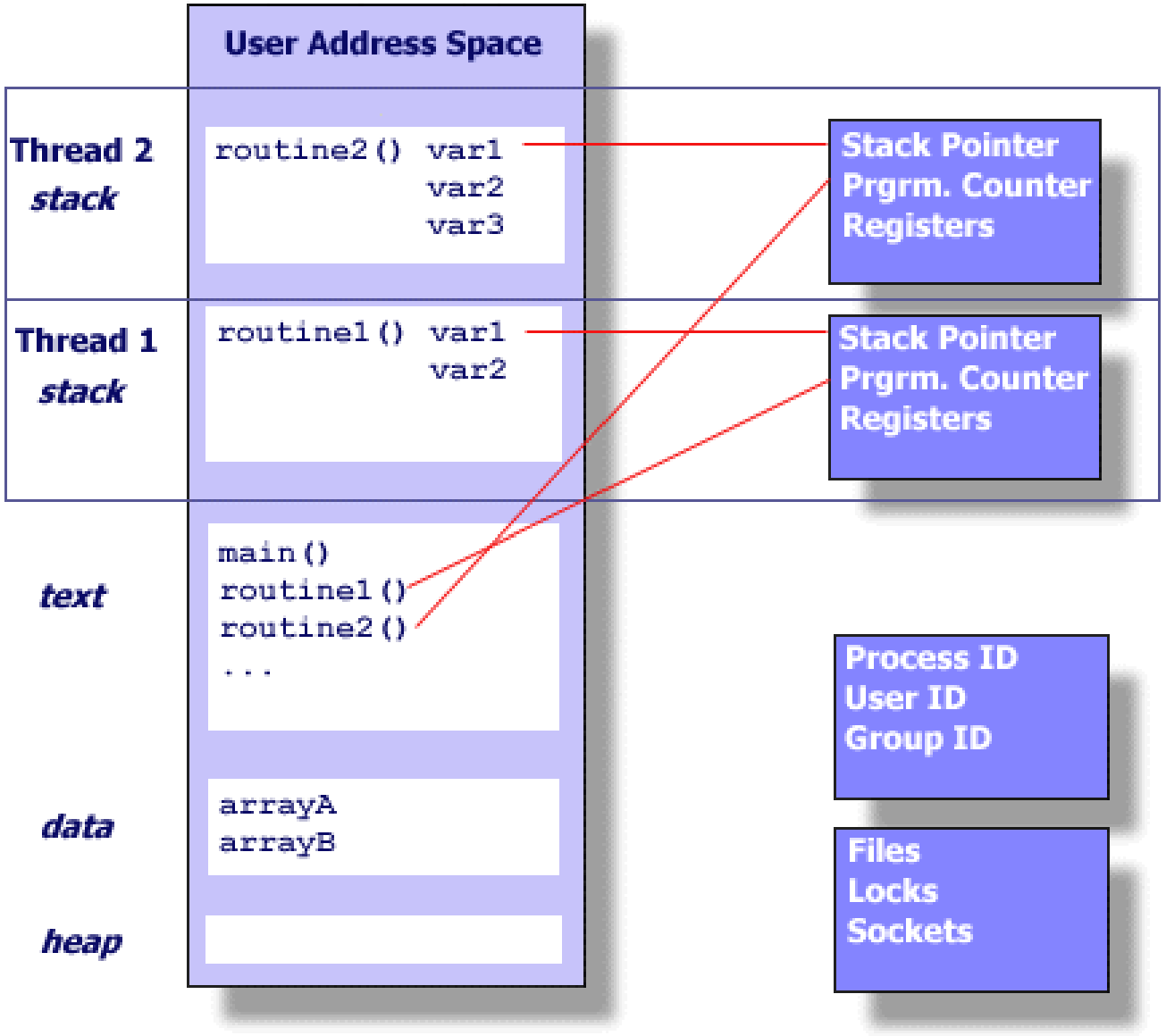
V rámci procesu vlákna **sdílí**

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

Naopak vlákna v rámci procesu **nesdílí**

- kontext procesoru (CPU se přiděluje vláknům)
- zásobník
- blokované a čekající signály
- data specifická 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

# Ošetření chyb

- funkce `pthread_*` nenastavují `errno`
- návratovou hodnotou je 0 v případě úspěchu
- v případě neúspěchu pak chybový kód

# 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); // man 2 syscall
```

```
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 počet vláken zadaný jako argument. Vlákna 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);
```

Mohou to být makra $\rightarrow$ stejné zanoření (scope).

# 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 se ukládají do fronty, zrušení proběhne až po nastavení vlákna na zrušitelné

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)`

# Procesy vs. vlákna

Lze nalézt ekvivalenci mezi některými funkcemi pro provádění operací nad procesy a nad vlákny:

| | | |
|---|---|---|
| `pthread_create` | $\sim$ | `fork` |
| `pthread_exit` | $\sim$ | `exit` |
| `pthread_join` | $\sim$ | `waitpid` |
| `pthread_cleanup_push` | $\sim$ | `atexit` |
| `pthread_self` | $\sim$ | `getpid` |
| `pthread_cancel` | $\sim$ | `abort` |

# Poznámky

- 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 v podstatě pouze s následným použitím `exec`
- Vlákna a signály
  - `pthread_kill`, `pthread_sigqueue`
  - nastavení obsluhy signálů je stejné pro všechna vlákna
  - `pthread_sigmask`
  - `sigwait`
- Pozor na ošetřování chyb – `pthread_*` funkce nenastavují `errno`, ale vracejí přímo chybový kód odpovídající `errno`. (pozn.: `errno` je na Linuxu thread-safe.)

## Závěr

domácí úkoly a zdroje

# Domácí úkol

- Vytvořte aplikaci, která v několika zadaných textových souborech (počet není omezen) paralelně hledá specifikovaný řetězec.
- Řetězec je case sensitive a kromě escape sekvencí (včetně nového řádku) může obsahovat libovolné znaky.
- Prvním argumentem je řetězec, který se hledá
- Další argumenty (aspoň 1) jsou soubory, které se prohledávají.
- Délka řádku ani velikost souborů nejsou omezeny.
- Výstupem jsou řádky souborů, které daný řetězec obsahují, prefixované jménem souboru.
- Informace o výskytu se tiskne průběžně.
- Chování odpovídá příkazu `grep retezec soubor1 ... souborN`

# Zdroje

## Vlákna

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