

Vlákna podle standardu POSIX

POSIX

- Portable Operating System Interface
- S názvem přišel Richard Stallman v odpovědi na žádost IEEE o lepší označení než IEEE-IX
- API kompatibilní s UNIX-like operačními systémy
 - Obecně ho může implementovat kterýkoliv OS
 - Např. MS Windows
- Aktuálně se dělí na tři části:
 - API jádra OS (Real-Time, vlákna, bezpečnost, IPC)
 - Příkazová řádka a utility
 - Validace
- Kompatibilní se standardem POSIX
 - HP-UX
 - OpenSolaris
 - Windows NT/2003/Vista Enterprise
- Většinou kompatibilní, bez certifikátu
 - FreeBSD
 - Linux (většina distribucí)

PThreads – POSIX Threads

- Obvyklý název pro knihovny implementující standard POSIX pro práci s vlákny
- Obsahují
 - Definice datových typů
 - Funkce pro manipulaci s vlákny
 - Synchronizační funkce

```
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>

void *thread_func( void *vptr_args );

int main( void ){
    int i, j;
    pthread_t thread;

    pthread_create( &thread, NULL,
                   &thread_func, NULL );

    for( j= 0; j < 20; ++j ){
        fprintf( stdout, "a\n" );
        sleep(10);
    }

    pthread_join( thread, NULL );

    exit( EXIT_SUCCESS );
}

void *thread_func( void *vptr_args ){
    int i, j;

    for( j= 0; j < 20; ++j ){
        fprintf( stderr, "  b\n" );
        sleep(10);
    }
    return NULL;
}
```

http://en.wikipedia.org/wiki/POSIX_Threads

Filozofie vláken v POSIXu

- V řeči C je vlákno funkce, která se vykonává nezávisle na main
- Vlákna může existovat několik a k synchronizaci používají zámky, mutexy a podmínkové proměnné, nebo jiné, dostupné mechanismy
 - Je-li třeba, programátor si může dopsat vlastní, aplikačně-specifické, za pomoci stávajících
- Lze napsat efektivní program, ale programátor sám ručí za správnou synchronizaci
- Pthreads mají odstínit, co je vespod – tj. co a jak plánuje vlákna a jak jsou realizovány synchronizační funkce
 - Napiš a zkompilej pro cílovou platformu
 - Existují i implementace, které podporují víceprocesorové systémy
 - Záleží na OS, který musí povolit spuštění procesu na více než jednom procesoru
 - pthread_setaffinity_np
 - sched_setaffinity
 - <http://www.linuxjournal.com/article/6799>
 - Jiné implementace si zase kladou za cíl pokrýt výpočet v clusteru, tj. s distribuovanou pamětí
 - SISI Pthreads
- Původním cílem bylo snížit režii spuštění několika spolupracujících procesů – viz KIV/OS
- Většinou se vyskytují jako C knihovna, ale lze je najít i jinde – Kylix

Objekty v POSIXu

- jsou reprezentovány pomocí handle
 - pthread_t
 - pthread_mutex_t
 - pthread_cond_t
- objekty mají sadu atributů, které lze měnit
 - pthread_attr_t
 - pthread_mutexattr_t
 - pthread_condattr_t
- funkce pro manipulaci s objekty buď vrátí 0 jako OK, nebo jinou hodnotu jako chybový kód

Vlákno

- má svůj zásobník
- má svůj kontext a prioritu
- prostředky „jsou psané na“ proces
- vytvoří se voláním

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

 - thread – handle nového vlákna
 - attr – NULL pro default, nebo vlastní
 - start_routine – adresa funkce, kde začíná kód vlákna
 - arg – uživatelské data pro nově vytvořené vlákno, např. číslo vlákna, nějaké ID, ukazatel na data, apod.

- stav
 - ready – může být naplánováno, pak je ready nebo terminated
 - running – běží, pak je ready, waiting, nebo terminated
 - waiting – čeká na splnění nějaké podmínky, pak je ready
 - terminated – RIP, handle je sice stále platný, ale použít lze už jenom `pthread_join`, nebo `pthread_detach` – zneplatní handle vlákna a uvolní jeho kontext & co.
- Atributy
 - Plánování (řazeno podle priority kategorie)
 - FIFO – běží vlákna s nejvyšší prioritou, v pořadí FIFO, dokud se všechny nezablokují; pak se přepne na RR
 - RR (Round Robin)
 - vlákna běží podle priority v pořadí FIFO
 - plánují se po časových kvantech – střídají se ve výpočtu
 - když dojdou plánovatelná vlákna s nejvyšší prioritou, začnou se plánovat ty s nižší prioritou
 - až dojdou všechna, přepne se na Other
 - Other
 - vlákna se střídají bez ohledu na prioritu,
 - čím větší priorita, tím více se přidělí strojového času
 - default, označuje se i jako FG

- BG (background)
 - Vlákno poběží na pozadí
 - Vlákna, které mají podíl na rychlosti reakce UI dostávají o něco více strojového času – u vláken na pozadí toho není třeba
 - Existují další background varianty
 - Low FIFO
 - Low RR
- Priorita – viz KIV/OS
- Velikost zásobníku – programátor si může zadat vlastní potřebnou velikost zásobníku
- Guardsize – hlídá, že ne bude překročena zadaná velikost zásobníku, aby se zabránilo poškození dat v důsledku přetečení
- Ukončí se
 - buď ukončením funkce – return v C, dobrovolné ukončení
 - pthread_exit – nevrací návratovou hodnotu, protože už není komu, dobrovolné ukončení
 - pthread_cancel
 - externí žádost o ukončení vlákna
 - vlákno má možnost se bránit pomocí pthread_setcancelstate
 - enable/disable
 - existují tzv. cancellation points – PTHREAD_CANCEL_DEFERRED
 - mají za úkol zajistit odemknutí zámků
 - lze je potlačit – PTHREAD_CANCEL_ASYNCHRONOUS

- existuje destruktork vlákna
 - vlákno může po přijetí požadavku na likvidaci provést úklidové akce – např. odemknutí zámků
 - `pthread_cleanup_push`
 - `pthread_cleanup_pop`

Mutexy

- `pthread_mutex_lock (&mutex_handle);`
- `state = pthread_try_lock (&mutex_handle);`
- `pthread_mutex_unlock (&mutex_handle);`
- `pthread_mutexattr_settype()`
- `PTHREAD_MUTEX_NORMAL`
 - `PTHREAD_MUTEX_DEFAULT`
 - Vlákno jej může uzamknout pouze jednou, další volání už způsobí deadlock
 - Vícenásobné zamykání je pro programátora pohodlnější aneb menší šance na chybu
 - tohle se jim moc nepovedlo
- `PTHREAD_MUTEX_RECURSIVE`
 - Vlákno ho může zamknout několikrát, ale také ho musí tolikrát odemknout
- `PTHREAD_MUTEX_ERRORCHECK`
 - Slouží pro ladění
 - Funguje jako normal, ale zná vlastníky a pozná pokus o druhé zamčení tím samým vláknem
 - Nebylo by lepší mít jenom recursive?

Podmínkové proměnné

- vlákno se uspí do té doby, dokud se nesplní nějaká podmínka, která se pak signalizuje pomocí podmínkové proměnné
- podmínková proměnná je svázána s mutexem
- `pthread_cond_init`
- `pthread_cond_wait`
- `pthread_cond_timedwait`
- `pthread_cond_signal`
- `pthread_cond_destroy`


```
pthread_mutex_lock (&barrier_lock);
/* uzamknutí dat podmínky */

barrier_cnt++;
/* změna dat podmínky, dělají všichni */
/* test podmínky - dělají všichni */

if (barrier_cnt < N) {
    /* dělají všichni až na posledního */
    pthread_cond_wait(&cond_barrier,
                     &barrier_lock);
    /* blokující operace, tj. vlákno přejde
       do stavu waiting, ale odemkne se
       zámek - kvůli tomu se do operace
       dává jeho adresa */

    /* tady se vlákno probudí, ale už zase
       s uzamčeným barrier_loc */
} else {
    /* poslední vlákno */
    barrier_cnt = 0;

    /* nějaká změna dat a dále vyslání
       signálu "probuzení" pro všechna
       vlákna čekající v podmínkové
       proměnné cond_barrier */

    pthread_cond_broadcast(&cond_barrier);
}

/* sem už přijdou všichni, a pokaždé je
   zde zamčený zámek, musí se tudíž
   odemknout */
pthread_mutex_unlock (&barrier_lock);
```

Semafor

```
#include <semaphore.h>

int sem_init(sem_t *sem,
             int pshared,
             unsigned int value);

int sem_wait(sem_t *sem);
/* P(sem), wait(sem) */

int sem_post(sem_t *sem);
/* V(sem), signal(sem) */

int sem_getvalue(sem_t *sem, int *sval);
int sem_trywait(sem_t *sem);

int sem_destroy(sem_t *sem);
/* undo sem_init() */

/* named semaphores
   these are less useful here */

sem_t *sem_open( ... );
int sem_close(sem_t *sem);
int sem_unlink(const char *name);
```

Srovnání s Adou a Javou

- pthreads nejsou tak vysokoúrovňový prostředek jako tasky Ady a vlákna Java
- je možné tak provádět věci, které by jinak nešly
- je možné napsat efektivnější program
- programátor je však zodpovědný za spoustu věcí, o které by se s vysokoúrovňovým prostředkem nemusel starat
 - nereentrantní funkce
 - alokace paměti – každý paměťový manažer není thread-safe, záleží na runtime libraries prostředí
 - vždy korektní opuštění kritické sekce a to i v případě výjimky