

Paralelné programovanie

doc. Ing. Michal Čerňanský, PhD.
FIIT STU Bratislava

Vlákná

- Procesy
- Vlákna – odľahčené procesy
- Fibres – odľahčené vlákna
- Všeobecne o POSIX threads
- Rutiny
 - Vytváranie
 - Mutexy

Procesy

- Vlákno (Thread) – nezávislý tok inštrukcií
- Funkcia bežiaca nezávisle od hlavného toku programu
- Proces
 - Objekt operačného systému
 - Informácia o programe, zdrojoch, stave vykonávania programu:

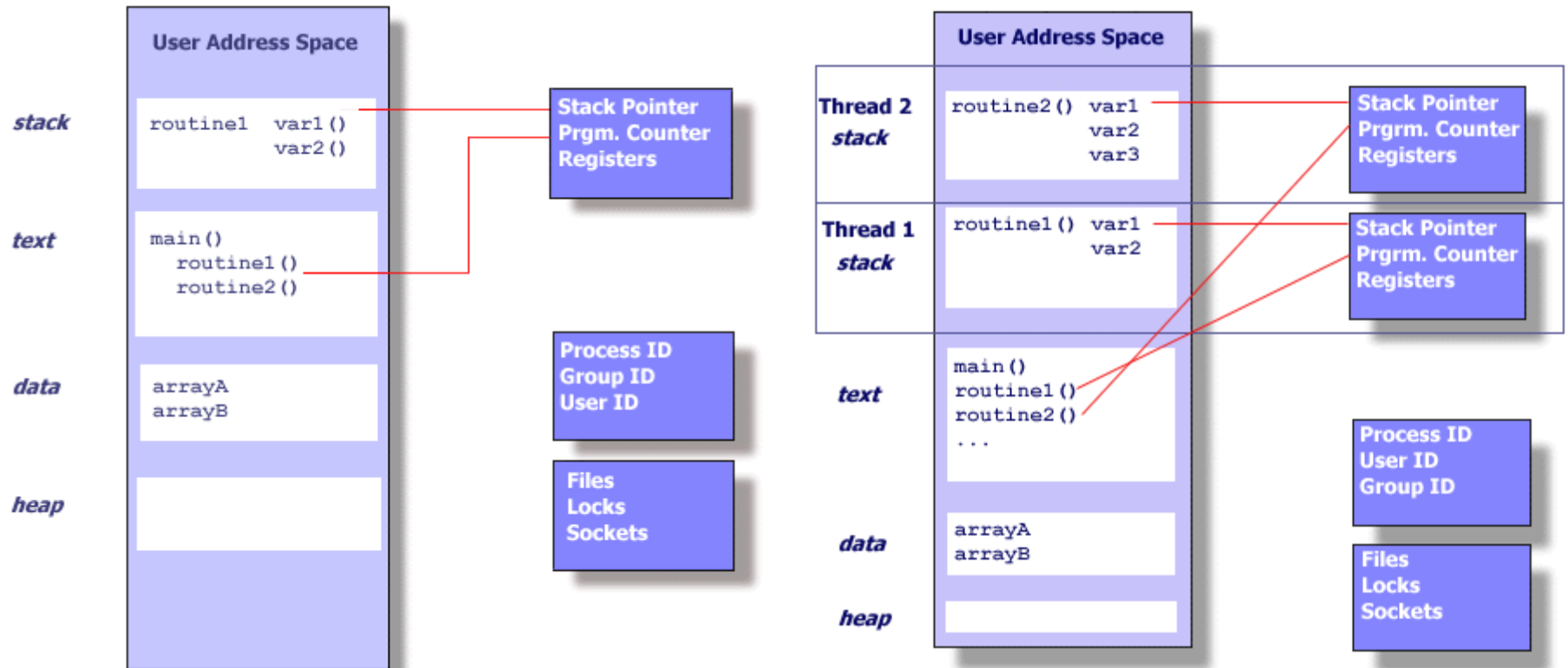
Procesy

- ❑ ID procesu, skupiny, používateľa (Process ID, process group ID, user ID, and group ID)
- ❑ Prostredie (Environment)
- ❑ Pracovný adresár (Working directory)
- ❑ Programové inštrukcie (Program instructions)
- ❑ Registre (Registers)
- ❑ Zásobník (Stack)
- ❑ Halda resp. hromada (Heap)
- ❑ Deskriptory súborov (File descriptors)
- ❑ Akcie na signály (Signal actions)
- ❑ Zdieľané knižnice (Shared libraries)
- ❑ Medziprocesová komunikácia: rady správy, rúry, semaféry, zdieľaná pamäť (Inter-process communication tools: message queues, pipes, semaphores, or shared memory).

Vlákná

- Vlákna existujú v rámci procesov
- Môžu byť plánované a vykonávané OS
- Vykonávané nezávisle
- Duplikujú iba potrebné zdroje:
 - ❑ Ukazateľ na zásobník
 - ❑ Registre
 - ❑ Vlastnosti pre plánovanie (priorita, politika)
 - ❑ Množina signálov
 - ❑ ...

Vlákna



Vlákná

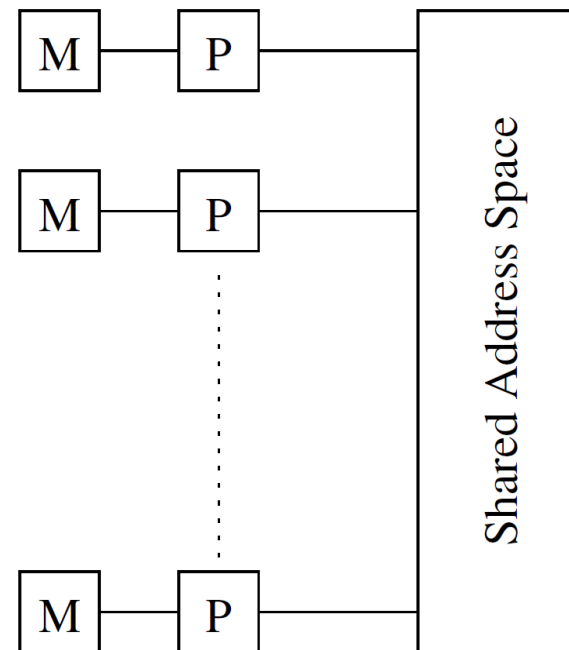
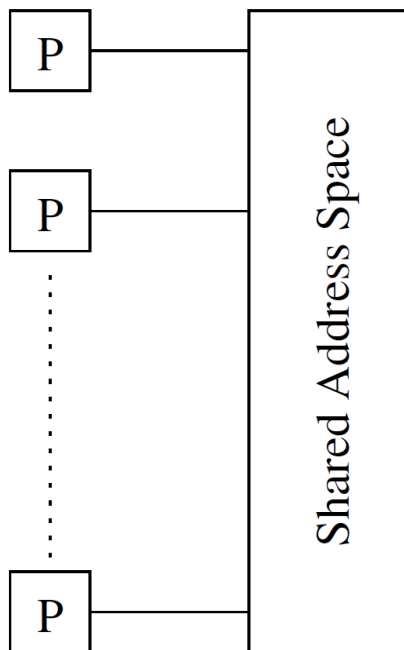
- Vlákno existuje v rámci procesu a používa zdroje procesu
- Má vlastný nezávislý tok inštrukcií
- Duplikuje iba zdroje potrebné na nezávislý beh
- Zdieľa zdroje procesu s ostatnými vláknami
- „Lightweight Process“ – väčšia časť inicializačnej práce už vykonaná procesom

Vlákná

- Vlákno zdieľa zdroje procesu s ostatnými vláknami:
 - ❑ Zmeny vykonané inými vláknami na zdieľaných systémových prostriedkoch sú viditeľné ostatnými vláknami
 - ❑ Dva ukazatele s rovnakou adresou ukazujú na to isté miesto v pamäti
 - ❑ Je možné zapisovať a čítať z toho istého miesta v pamäti, potreba synchronizovať
 - ❑ Vlákňová bezpečnosť (Thread Safeness)

Vlákná

- Zdieľaná pamäť – prístupná z každého vlákna
- Zásobník, súkromná pamäť vlákien – lokálna pamäť
- Logický pamäťový model:



Vlákná

- Portabilita
 - sekvenčné aj paralelné počítačové systémy
- Zakrytie oneskorení pri prístupe (Latency Hiding)
 - pamäť, IO, komunikácia
 - sekvenčné aj paralelné počítačové systémy
- Jednoduché plánovanie a vyrovňovanie záťaže
 - možnosť definovať väčší počet súbežných úloh
 - systém sa stará o dynamické plánovanie vlákien
 - maximalizácia vyťaženia procesorov

Vlákná

- Použitie vlákien:
 - ❑ Práca môže byť vykonaná vo viacerých súbežných úlohách
 - ❑ Údaje je možné spracovávať viacerými súbežnými úlohami
 - ❑ Nerovnomerné zaťaženie procesora v rôznych častiach programu
 - ❑ Potreba vykonávať asynchrónne operácie
 - ❑ Rôzna dôležitosť úloh v programe (priority)
- V paralelných aj sekvenčných prostrediach

Vlákná

- Modely použitia vlákien:
 - Hlavné vláko – pracovné vlákna (Master – Worker)
 - Prúdové spracovanie (Pipeline)
 - Seberovné vlákna (Peer)
-

POSIX Threads - Pthreads

- Množstvo implementácií vlákien
 - POSIX Threads – štandard na UNIX OS
 - Implementované ako knižnica funkcií
 - Na väčšine UNIX aj nie UNIX systémoch
 - Podobnosť konceptov a konštrukcií aj s inými implementáciami vlákien aj v iných jazykoch a prostrediach
-

POSIX Threads - Pthreads

- Menežovanie – riadenie vlákien
- Vzájomné vylučovanie (Mutual Exclusion)
- Podmienené premenné (Condition Variables)

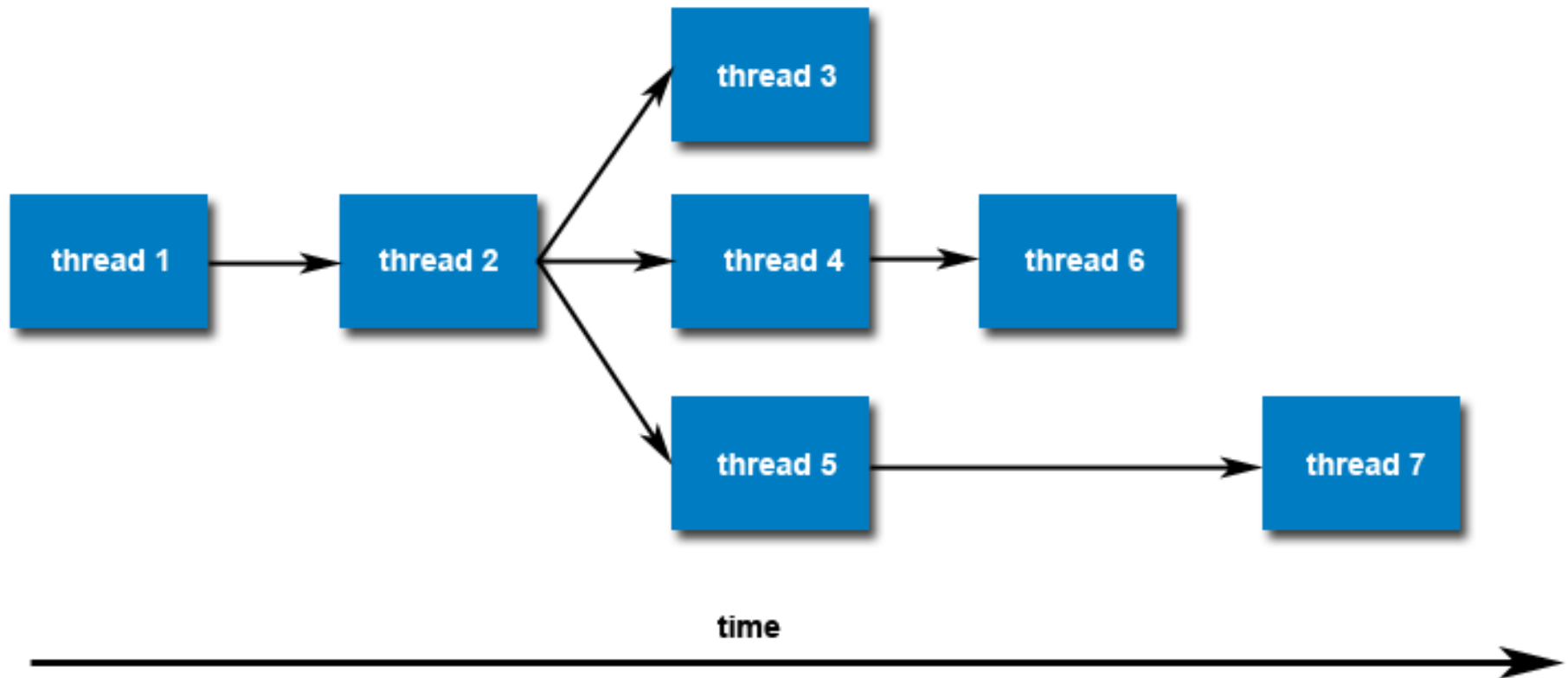
Pthreads – riadiace funkcie

- `pthread_create (thread,attr,start_routine,arg)`
- `pthread_exit (status)`
- `pthread_attr_init (attr)`
- `pthread_attr_destroy (attr)`

Pthreads – riadiace funkcie

- Vytvorenie vlákna
 - Funkcia `main()` je vykonávaná v rámci jedného predvoleného vlákna
 - Ostatné vlákna je potrebné explicitne vytvoriť
 - `pthread_create` vytvorí nové vlákno
 - `thread`: identifikátor vlákna
 - `attr`: atribúty pre vlákno, `NULL` ak štandardné
 - `start_routine`: C funkcia, kt. sa bude vykonávať v rámci vlákna
 - `arg`: jediný argument pre C funkciu

Pthreads – riadiace funkcie



Pthreads – riadiace funkcie

- Atribúty vlákna
 - ❑ pthread_attr_init - inicializácia atribútov
 - ❑ pthread_attr_destroy – odstránenie atribútov
 - ❑ Iné funkcie použité na nastavenie alebo čítanie atribútov
-

Pthreads – riadiace funkcie

- Ukončenie vlákna
 - ❑ Skončí C funkcia vlákna
 - ❑ Vlákno zavolá f. `pthread_exit`
 - ❑ Iné vlákno zavolá `pthread_cancel`
 - ❑ Celý proces je ukončený volaním `exec` alebo `exit`

Pthreads – riadiace funkcie

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid) {
    long tid;
    tid = (long)threadid;
    printf("Hello World! It's me, thread #%ld!\n", tid)
    pthread_exit(NULL);
}

int main (int argc, char *argv[]) {
    pthread_t threads[NUM_THREADS];
    int rc;
    long t;

    for(t=0; t<NUM_THREADS; t++){
        printf("In main: creating thread %ld\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if (rc){
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

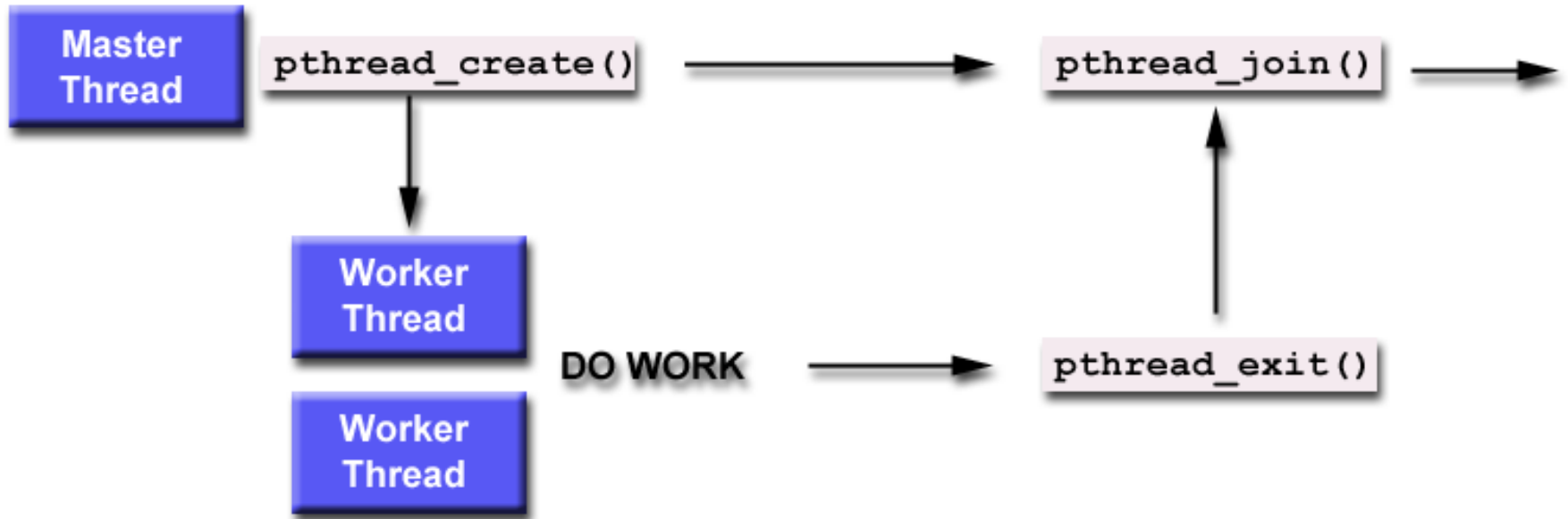
Pthreads – riadiace funkcie

- „Joinable“ vlákno
 - `pthread_join (threadid,status)`
 - `pthread_detach (threadid)`
 - `pthread_attr_setdetachstate (attr,detachstate)`
 - `pthread_attr_getdetachstate (attr,detachstate)`
-

Pthreads – riadiace funkcie

- „Joinable“ vlákno – hlavné vlákno môže počkať na ukončenie vytvoreného vlákna
- Spôsob synchronizácie vlákien
- Možnosť získať výsledok ukončeného vlákna
- Pre dané vlákno je možné iba jedno volanie `pthread_join()`
- Musí byť špecifikované pri vytvorení vlákna v atribútoch
- Možnosť „odpútať“ „joinable“ vlákno - `pthread_detach()`

Pthreads – riadiace funkcie



Pthreads – riadiace funkcie

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

```
void *function( void *ptr ) {
    char *message = (char *) ptr;
    printf("%s \n", message);
    pthread_exit(0);
}
```

```
int main(void) {
    pthread_t thread1, thread2;
    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    int  iret1, iret2;

    iret1 = pthread_create( &thread1, NULL, &function, (void*) message1);
    iret2 = pthread_create( &thread2, NULL, &function, (void*) message2);

    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);

    printf("Thread 1 returns: %d\n",iret1);
    printf("Thread 2 returns: %d\n",iret2);
    exit(EXIT_SUCCESS);
}
```


Pthreads – riadiace funkcie

- Menežovanie zásobníkov
- `pthread_attr_getstacksize (attr, stacksize)`
- `pthread_attr_setstacksize (attr, stacksize)`
- `pthread_attr_getstackaddr (attr, stackaddr)`
- `pthread_attr_setstackaddr (attr, stackaddr)`

Pthreads – vzájomné vylučovanie

- Mutex – Mutual Exclusion
- Synchronizácia vlákien, ochrana premenných v zdieľanej pamäti
- Zámok na ochranu zdieľaných zdrojov
- Iba jedno vlákno môže vlastniť (uzamknúť) mutex v danom čase
- Ak sa viaceré vlákna pokúsia obdržať vlastníctvo, iba jeden ho získa
- Až keď ho vlákno uvoľní, môže ho získať iné

Pthreads – vzájomné vylučovanie

- Úprava globálnej premennej – zabránenie súťaženiu o prostriedky
- Kritická sekcia – takto chránené globálne
- Postup použitia
 - ❑ Vytvorenie a inicializácia mutexu
 - ❑ Vlákna sa pokúšajú uzamknúť mutex
 - ❑ Iba jedno vlákno uspeje a vykonáva množinu akcií
 - ❑ Vlastniace vlákno uvoľní mutex
 - ❑ Iné (alebo aj to isté) vlákno sa zmocní mutexu a vykonáva akcie
 - ❑ ...
 - ❑ Mutex je deštruovaný

Pthreads – vzájomné vylučovanie

- Pri ochrane zdrojov každé vlákno musí používať mutex – úloha programátora
- Vlákna, ktoré nezískajú mutex, sú uspaté, a keď je mutex vláknom čo ho získalo uvoľnený, ďalšie vlákno je prebudené a získa mutex

Pthreads – vzájomné vylučovanie

- `pthread_mutex_init (mutex,attr)`
- `pthread_mutex_destroy (mutex)`

- `pthread_mutexattr_init (attr)`
- `pthread_mutexattr_destroy (attr)`

- `pthread_mutex_lock (mutex)`
- `pthread_mutex_trylock (mutex)`
- `pthread_mutex_unlock (mutex)`

Pthreads – vzájomné vylučovanie

- Mutex musí byť inicializovaný

- ❑ Statická inicializácia pri jeho deklarácii

```
pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
```

- ❑ Dynamická inicializácia

```
pthread_mutex_init(&mymutex, &attr);
```

```
pthread_mutex_destroy(&mymutex);
```

```
pthread_mutexattr_t attr;
```

```
pthread_mutexattr_init(&attr);
```

```
pthread_mutexattr_destroy(&attr);
```

Pthreads – vzájomné vylučovanie

- Atribút
 - ❑ Attr môže byť NULL – štandardné hodnoty
 - ❑ Protokol – politika na zabránenie zmene priorít
 - ❑ Prioc ceiling – maximálna priorita
 - ❑ Process-shared - zdieľanie procesmi
- Nemajú všetky implementácie

Pthreads – vzájomné vylučovanie

- `pthread_mutex_lock (mutex)`
 - `pthread_mutex_trylock (mutex)`
 - `pthread_mutex_unlock (mutex)`
-

Pthreads – vzájomné vylučovanie

- `pthread_mutex_lock()` - pokus získať mutex, ak už získaný iným vláknom, uspatie vlákna (blokovanie), až kým nie je mutex uvoľnený
- `pthread_mutex_trylock()` - pokus získať mutex, ak už získaný iným vláknom, okamžitý návrat z volania funkcie so zodpovedajúcou návratovou hodnotou
- `pthread_mutex_unlock()` - úvoľnenie mutexu

Pthreads – vzájomné vylučovanie

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

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int counter = 0;

void *function() {
    pthread_mutex_lock( &mutex );
    counter++;
    printf("Counter value: %d\n",counter);
    pthread_mutex_unlock( &mutex );
    pthread_exit(0);
}

int main(void) {
    int rc1, rc2;
    pthread_t thread1, thread2;

    pthread_create( &thread1, NULL, function, NULL);
    pthread_create( &thread2, NULL, function, NULL);

    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);

    exit(EXIT_SUCCESS);
}
```

Pthreads – podmienené premenné

- Ďalšia možnosť synchronizovať vlákna
 - Synchronizácia na základe hodnoty dát, testovanie splnenia podmienky
 - Polling – obsadzujúce čakanie
 - Vždy použité súčasne s mutexom
-

Pthreads – podmienené premenné

- Ďalšia možnosť synchronizovať vlákna
 - Synchronizácia na základe hodnoty dát, testovanie splnenia podmienky
 - Polling – obsadzujúce čakanie
 - Vždy použité súčasne s mutexom
-

Pthreads – podmienené premenné

- `pthread_cond_init (condition,attr)`
- `pthread_cond_destroy (condition)`
- `pthread_condattr_init (attr)`
- `pthread_condattr_destroy (attr)`
- `pthread_cond_wait (condition,mutex)`
- `pthread_cond_signal (condition)`
- `pthread_cond_broadcast (condition)`

Pthreads – podmienené premenné

- Podmienená premenná musí byť inicializovaná

- Statická inicializácia pri jej deklarácii

```
pthread_cond_t myconvar = PTHREAD_COND_INITIALIZER;
```

- Dynamická inicializácia

```
pthread_cond_init (condition,attr)
```

```
pthread_cond_destroy (condition)
```

```
pthread_condattr_init (attr)
```

```
pthread_condattr_destroy (attr)
```

Pthreads – podmienené premenné

- Atribút
 - Attr môže byť NULL – štandardné hodnoty
 - Process-shared - zdieľanie procesmi

Pthreads – podmienené premenné

- `pthread_cond_wait()`
 - ❑ Blokuje volajúce vlakno, až kým nie je podmienka signalizovaná
 - ❑ zodpovedajúci mutex musí byť uzamknutý a pri blokovaní je uvoľnený
 - ❑ Po prijatí signálu uzamkne mutex
 - ❑ Mutex musí byť explicitne uvoľnený
-

Pthreads – podmienené premenné

- The `pthread_cond_signal()`
 - ❑ Signalizácia na zobudenie vlákna čakajúceho na podmienenú premennú
 - ❑ Zodpovedajúci mutex musí byť pred volaním uzamknutý
 - ❑ Zodpovedajúci mutex musí byť uvoľnený aby `pthread_cond_wait()` funkcia pokračovala

Pthreads – podmienené premenné

- The `pthread_cond_broadcast()`
 - Ak viac vlákien je v stave čakania na podmienenú premennú
- Volat' `pthread_cond_signal()` predcalling `pthread_cond_wait()` je logická chyba

Pthreads – podmienené premenné

```
#include <pthread.h>
int          conditionMet = 0;
pthread_cond_t  cond  = PTHREAD_COND_INITIALIZER;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
#define NTHREADS 5

void *threadfunc(void *parm) {
    pthread_mutex_lock(&mutex);
    while (!conditionMet) pthread_cond_wait(&cond, &mutex);
    pthread_mutex_unlock(&mutex);
    return NULL;
}

int main(int argc, char **argv) {
    int i;
    pthread_t threadid[NTHREADS];

    for(i=0; i<NTHREADS; ++i) pthread_create(&threadid[i], NULL, threadfunc, NULL);

    pthread_mutex_lock(&mutex);
    conditionMet = 1;
    pthread_cond_broadcast(&cond);
    pthread_mutex_unlock(&mutex);

    for (i=0; i<NTHREADS; ++i) pthread_join(threadid[i], NULL);

    pthread_cond_destroy(&cond);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Pthreads – podmienené premenné

- `if (! conditonMet) cond_wait(cond, mutex);` - nestačí `if`
- `while (! conditonMet) cond_wait(cond, mutex);` - správna konštrukcia
- “Spurious Wakeups” – „falošné prebudenie“
 - ❑ `cond_wait(cond,mutex)` môže skončiť aj keď podmienená premenná nebola signalizovaná
 - ❑ Dôsledok náročnosti implementácie v multiprocessorovom poč. systéme
- Správny štýl programovania
 - ❑ Po obdržaní mutexu môže byť už podmienka neplatná

Pthreads – podmienené premenné

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

pthread_mutex_t count_mutex      = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t condition_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t condition_cond   = PTHREAD_COND_INITIALIZER;

int count = 0;
#define COUNT_DONE 10
#define COUNT_HALT1 3
#define COUNT_HALT2 6

void *functionCount1() {
    ...
}

void *functionCount2() {
    ...
}

int main(void) {
    pthread_t thread1, thread2;

    pthread_create( &thread1, NULL, functionCount1, NULL);
    pthread_create( &thread2, NULL, functionCount2, NULL);
    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);

    exit(EXIT_SUCCESS);
}
```

Pthreads – podmienené premenné

```
void *functionCount1() {
    for(;;) {
        pthread_mutex_lock( &condition_mutex );
        while( count >= COUNT_HALT1 && count <= COUNT_HALT2 ) {
            pthread_cond_wait( &condition_cond, &condition_mutex );
        }
        pthread_mutex_unlock( &condition_mutex );

        pthread_mutex_lock( &count_mutex );
        count++;
        printf("Counter value functionCount1: %d\n",count);
        pthread_mutex_unlock( &count_mutex );

        if(count >= COUNT_DONE) pthread_exit(0);
    }
}

void *functionCount2() {
    for(;;) {
        pthread_mutex_lock( &condition_mutex );
        if( count < COUNT_HALT1 || count > COUNT_HALT2 ) {
            pthread_cond_signal( &condition_cond );
        }
        pthread_mutex_unlock( &condition_mutex );

        pthread_mutex_lock( &count_mutex );
        count++;
        printf("Counter value functionCount2: %d\n",count);
        pthread_mutex_unlock( &count_mutex );

        if(count >= COUNT_DONE) pthread_exit(0);
    }
}
```

Výstup:

```
Counter value functionCount1: 1
Counter value functionCount1: 2
Counter value functionCount1: 3
Counter value functionCount2: 4
Counter value functionCount2: 5
Counter value functionCount2: 6
Counter value functionCount2: 7
Counter value functionCount1: 8
Counter value functionCount1: 9
Counter value functionCount1: 10
Counter value functionCount2: 11
```

Pthreads – podmienené premenné

Init:

```
mutex_init(&count_mutex);  
mutex_init(&cond_mutex);  
cond_init(&cond);
```

Thread A cyklus:

```
mutex_lock(&cond_mutex);  
while (count >= 3 && count <= 6)  
    cond_wait(&cond, &mutex);  
mutex_unlock(&cond_mutex);
```

```
mutex_lock(&count_mutex);  
count := count + 1;  
mycount := count;  
mutex_unlock(&count_mutex);
```

```
// do work with my_count
```

Thread B cyklus:

```
mutex_lock(&cond_mutex);  
if (count < 3 || count > 6)  
    cond_signal(&cond);  
mutex_unlock(&cond_mutex);
```

```
mutex_lock(&count_mutex);  
count := count + 1;  
mycount := count;  
mutex_unlock(&count_mutex);
```

```
// do work with my_count
```

Pthreads – podmienené premenné

- V čase práce s premennou „count“ môže byť podmienka neplatná
- Prerušenie toku vykonávania pred uzamknutím „count_mutex“

Pthreads – podmienené premenné

```
void *functionCount1() {
    for(;;) {
        pthread_mutex_lock( &condition_mutex );
        while( count >= COUNT_HALT1 && count <= COUNT_HALT2 ) {
            pthread_cond_signal( &condition_cond );
            pthread_cond_wait( &condition_cond, &condition_mutex );
        }
        pthread_mutex_unlock( &condition_mutex );

        // Count
        count++;
        printf("Counter value functionCount1: %d\n",count);

        ...
    }
}

void *functionCount2() {
    for(;;) {
        pthread_mutex_lock( &condition_mutex );
        while( count < COUNT_HALT1 || count > COUNT_HALT2 ) {
            pthread_cond_signal( &condition_cond );
            pthread_cond_wait( &condition_cond, &condition_mutex );
        }
        pthread_mutex_unlock( &condition_mutex );

        count++;
        printf("Counter value functionCount2: %d\n",count);

        ...
    }
}
```

Výstup:

```
Counter value functionCount1: 1
Counter value functionCount1: 2
Counter value functionCount1: 3
Counter value functionCount2: 4
Counter value functionCount2: 5
Counter value functionCount2: 6
Counter value functionCount2: 7
Counter value functionCount1: 8
Counter value functionCount1: 9
Counter value functionCount1: 10
```

Pthreads – podmienené premenné

Init:

```
mutex_init(&cond_mutex);  
cond_init(&cond);
```

Thread A cyklus:

```
mutex_lock(&cond_mutex);  
while (count >= 3 && count <= 6) {  
    cond_signal(&cond);  
    cond_wait(&cond, &mutex);  
}  
mutex_unlock(&cond_mutex);
```

```
count := count + 1;  
mycount := count;
```

```
// do work with my_count
```

Thread B cyklus:

```
mutex_lock(&cond_mutex);  
while (count < 3 || count > 6) {  
    cond_signal(&cond);  
    cond_wait(&cond, &mutex);  
}  
mutex_unlock(&cond_mutex);
```

```
count := count + 1;  
mycount := count;
```

```
// do work with my_count
```

Pthreads – podmienené premenné

- V čase práce s premennou „count“ môže byť podmienka neplatná
- Prerušenie toku vykonávania po inkrementácií premennej „count“

Pthreads – podmienené premenné

```
void *functionCount1() {
    for(;;) {
        pthread_mutex_lock( &condition_mutex );
        while( count >= COUNT_HALT1 && count <= COUNT_HALT2 ) {
            pthread_cond_signal( &condition_cond );
            pthread_cond_wait( &condition_cond, &condition_mutex );
        }

        count++;
        printf("Counter value functionCount1: %d\n",count);

        pthread_mutex_unlock( &condition_mutex );
        ...
    }
}

void *functionCount2() {
    for(;;) {
        pthread_mutex_lock( &condition_mutex );
        while( count < COUNT_HALT1 || count > COUNT_HALT2 ) {
            pthread_cond_signal( &condition_cond );
            pthread_cond_wait( &condition_cond, &condition_mutex );
        }

        count++;
        printf("Counter value functionCount2: %d\n",count);

        pthread_mutex_unlock( &condition_mutex );
        ...
    }
}
```

Výstup:

```
Counter value functionCount1: 1
Counter value functionCount1: 2
Counter value functionCount1: 3
Counter value functionCount2: 4
Counter value functionCount2: 5
Counter value functionCount2: 6
Counter value functionCount2: 7
Counter value functionCount1: 8
Counter value functionCount1: 9
Counter value functionCount1: 10
```

Pthreads – podmienené premenné

Init:

```
mutex_init(&cond_mutex);  
cond_init(&cond);
```

Thread A cyklus:

```
mutex_lock(&cond_mutex);  
while (count >= 3 && count <= 6) {  
    cond_signal(&cond);  
    cond_wait(&cond, &mutex);  
  
    count := count + 1;  
    mycount := count;  
}  
mutex_unlock(&cond_mutex);  
  
// do work with my_count
```

Thread B cyklus:

```
mutex_lock(&cond_mutex);  
while (count < 3 || count > 6) {  
    cond_signal(&cond);  
    cond_wait(&cond, &mutex);  
  
    count := count + 1;  
    mycount := count;  
}  
mutex_unlock(&cond_mutex);  
  
// do work with my_count
```

Zdroje

- Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar. Introduction to Parallel Computing, 2nd Edition, Addison-Wesley 2003, „Introduction to Parallel Computing“ <http://www-users.cs.umn.edu/~karypis/parbook/>
 - Blaise Barney, Lawrence Livermore National Laboratory: POSIX Threads Programming. <https://computing.llnl.gov/tutorials/pthreads/>

 - Obrázky prevzaté z:
 - Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar. Introduction to Parallel Computing, 2nd Edition, Addison-Wesley 2003, „Introduction to Parallel Computing“ <http://www-users.cs.umn.edu/~karypis/parbook/>
 - Blaise Barney, Lawrence Livermore National Laboratory: POSIX Threads Programming. <https://computing.llnl.gov/tutorials/pthreads/>
-