Programové prostředky pro multithreading - Java, POSIX, WinAPI, OpenMP.

Thursday, May 30, 2013 8:32 AM

Java

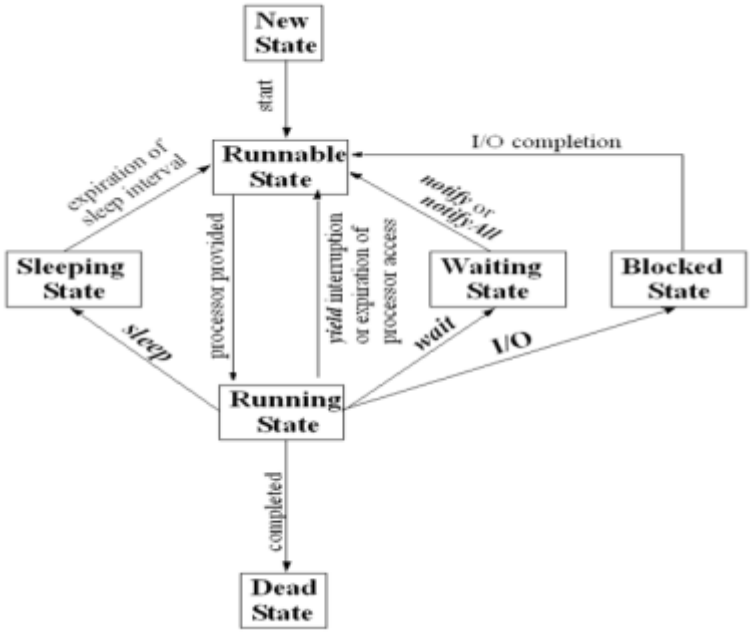
- V programovacím jazyce Java jsou objekty a metody potřebné pro práci s vlákny dostupné ve standardních knihovnách.
- Základem je třída Thread

Thread

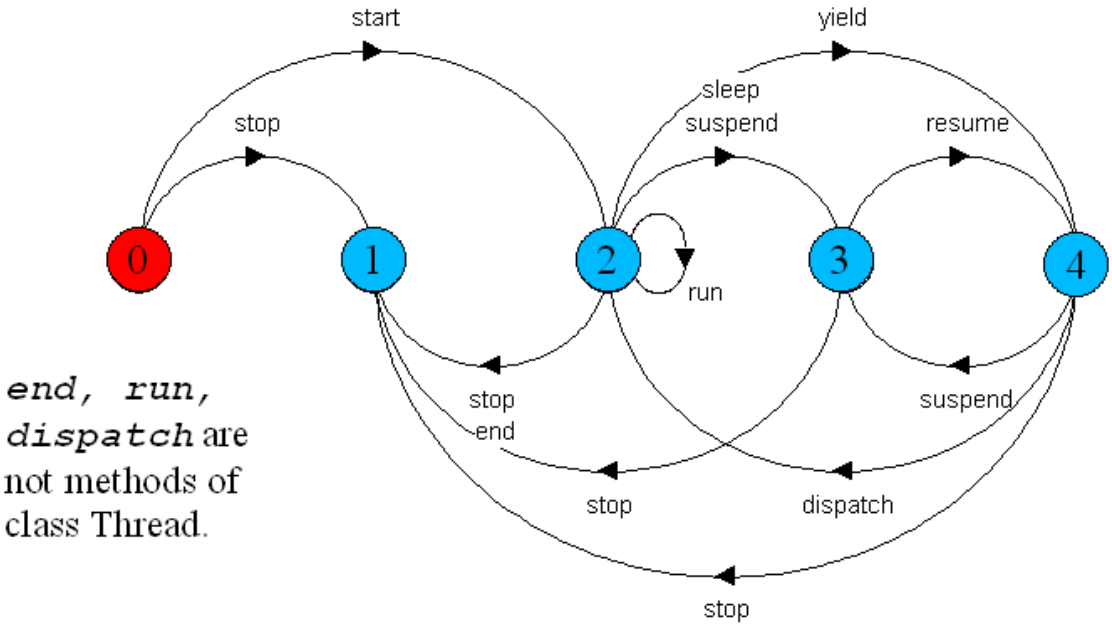
- tvoří základ všech paralelních programů v Javě
- pro jednoduché programy stačí oddědit od této třídy a překrýt metodu *run()*, do které se napíše výkonný kód vlákna.
- protože někdy je potřeba, aby naše třída dědila od jiné a zároveň měla vlastnosti vlákna, existuje ještě rozhraní *Runnable*, které stačí implementovat a třída rovněž získá vlastnosti vlákna
- pro napsání paralelního programu stačí vytvořit potřebné třídy, napsat jejich výkonné kódy do metod *run(),* a pak vlákna spustit (vlákna se spouští metodou *start()*

- *Stop, resume, suspend* - ***deprecated***
    - Stop - po jejím volání vlákno uvolní všechny držené monitory --> může zanechat uvolněné objekty v nekonzistentním stavu

Stavy vlákna



V javě:



Monitory v Javě

- komunikace vláken je řešena přes sdílenou paměť
- kritické sekce jsou ošetřeny klíčovým slovem ***synchronized***
- monitor je součástí každého objektu
- pro implementaci monitorů jsou uvnitř objektu monitoru skryté atributy:
  - **1 zámek** – pro všechny synchronizované metody
  - **1 fronta** – pro blokovaná vlákna čekající na vstup do synchronizované metody (kritická sekce)
  - nad frontou fungují privátní metody monitoru (objektu) *wait()*, *notify()* a *notifyAll()*
  - kromě těchto implicitních monitorů objektu lze v metodě ještě vytvořit synchronizační blok, který může použít i jiný zámek, než je implicitní zámek objektu, což dovoluje jemnější členění vzájemného vyloučení (např. jen na část metody)

POSIX

Struktury v posixu jsou reprezentovány pomocí *handle*. Struktury jsou **pthread_t, pthreat_mutex_t** a **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 vracejou 0 jako OK, nebo jinou hodnotu jako chybový kód.

Vlákna

Program vlákna je vlastně funkce C s typem void* prog_name(void* arg)

vlákno se vytvoří následujícím způsobem:

```
int status; //0 uspech, jinak chyba
status = pthread_create(&worker, NULL, prog_name,…);
```

- vlákno běží hned po vytvoření
- stavy vlákna jsou **ready, running, waiting** a **terminated**
- vlákno se ruší pomocí *pthread_exit* (ze stejného vlákna) nebo *pthread_cancel* (z cizího)
- Atributy:
  - **způsob plánování**: FIFO – nejprioritnější kategorie (nejdříve se plánují vlákna tétokategorie), RR – round-robin, FG – foreground, implicitní kategorie, střídání vláken, ty s vyšší prioritou mají více času, BG – background, všechna vlákna se střídají, ale dostávají méně času než FG
  - **priorita**
  - **rozměr zásobníku**
  - **hlídač zásobníku** – jak daleko se můžeme od konce zásobníku dostat, jinak vznikne výjimka
  - **konec vlákna** - vlákno dojde na konec svého programu - na toto ukončení se lze

synchronizovat z jiného vlákna pomocí pthread_join(na_koho_se_čeká, kam_přijde_výsledek) nebo zabito z vnějšku – pokud možno nepoužívat, základní funkce pro likvidaci pthread_cancel(oběť); oběť se může bránit pthread_setcancelstate(…), k likvidaci nemůže dojít kdekoliv v kódu vlákna, ale jen v předem připravených místech (volání blokující funkce, volání pthread_testcancel()), popisovaný způsob je synchronní, existuje i asynchronní)

Mutex

Je to synchronizační primitivum, umožňuje implementaci vzájemného vyloučení v kritické sekci. Jen jedno vlákno vlastnící mutex může být v kritické sekci, ostatní vlákna čekají.

```
Pthread_mutex_lock(mutex_handle)
Pthread_mutex_unlock(mutex_handle)
State = pthread_try_lock(mutex_handle)
```

Podmínková proměnná

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 - blokující operace, vlákno se převede do stavu waiting, a odemkne se zámek
Pthread_cond_timedwait
Pthread_cond_signal
Pthread_cond_broadcast - jako notifyAll()
Pthread_cond_destroy
```

WINAPI

Process

- Má virtuální paměťový prostor, systémové zdroje, bezpečnostní kontext (kdokoliv nemůže provádět cokoliv), jedinečný identifikátor, spustitelný kód
- Prioritní třída – vlákna pak mohou mít prioritu pouze v rozsahu prioritní třídy procesu.
- Má alespoň jedno vlákno (primární - to hlavní), které může vytvářet další vlákna – threads a fibers

Thread

- Entita v rámci procesu, kterou **plánuje OS**. Všechny vlákna sdílí paměťový prostor a zdroje svého procesu. Vlastní handler výjimek. Priorita: OS zajišťuje inverze priorit, dynamic boost, foreground/input včetně realtime.
- Jedinečný identifikátor
- TLS – thread local storage, data, která jsou specifická/lokální pro daný thread. Možnost, jak si thread může zabezpečit jedinečný přístup ke svým datům. Každý proces má k dispozici několik TLS slotů, které mohou být použity jeho thready. Možnost využití ke zpracování výjimek.

Fiber

- Běží v kontextu vlákna, **plánuje ho thread procesu**, jeden thread může naplánovat několik fibers. FLS – fiber local storage =analogie k TLS, FLS je asociováno s threadem.
- Má malý kontext (v porovnání s kontextem threadu) – zásobník, podmnožinu registrů a inicializační data.
- Má přístup do TLS threadu, v jehož kontextu běží.
- Nemá prioritu.

Synchonizační objekty WinApi

Mutex – vyžaduje přepnutí do režimu jádra, CreateMutex, OpenMutex, ReleaseMutex

Event – stavy signaled/nonsignaled, může být buď pulsní, tj, překlopí se do nonsignaled jakmile propustí jednoho čekajícího, nebo zůstane signaled (Manual reset Event, Auto Reset Event). CreateEvent, SetEvent, ResetEvent.

Pozn. txkoutny říkal, že event je rychlejší kvůli tomu, že nevyžaduje přepnutí do režimu jádra - lhal. Mutex, Event i Semaphore vyžadují přepnutí, protože jsou implementovány na úrovni jádra a umožňují synchronizaci napříč různými procesy (mohou být pojmenované). Opět v přednáškách je jaksi uvedeno, že kritická sekce má velkou režii - není to pravda, kritická sekce jako jediná neumožňuje synchronizaci mezi procesy, ale pouze v rámci jednoho procesu. Díky tomu při běžném vstupu/výstupu z/do ní (pokud je volná) není nutné nikam přepínat. Do režimu jádra se přepne pouze v případě, že je kritická sekce obsazená a je nutné čekat na uvolnění - pak se přepne do režimu jádra a čeká se nad nějakým synchronizačním primitivem, např. EVENT nebo MUTEX.

http://www.codeproject.com/Articles/7953/Thread-Synchronization-for-Beginners?fid=89682&select=3191727&fr=31#xx0xx

For Windows, critical sections are lighter-weight than mutexes.
Mutexes can be shared between processes, but always result in a system call to the kernel which has some overhead.
Critical sections can only be used within one process, but have the advantage that they only switch to kernel mode in the case of contention - Uncontended acquires, which should be the common case, are incredibly fast. In the case of contention, they enter the kernel to wait on some synchronization primitive (like an event or semaphore).
I wrote a quick sample app that compares the time between the two of them. On my system for 1,000,000 uncontended acquires and releases, a mutex takes over one second. A critical section takes ~50 ms for 1,000,000 acquires.
Here's the test code, I ran this and got similar results if mutex is first or second, so we aren't seeing any other effects.

From <http://stackoverflow.com/questions/800383/what-is-the-difference-between-mutex-and-critical-section>

Semafor – klasický semafor, CreateSemaphore, OpenSemaphore, ReleaseSemaphore

Kritická sekce – Critical section, má velkou režii, EnterCriticalSection (blokující), TryEnterCriticalSection (neblokující), LeaveCriticalSection

Na signalizaci objektů lze čekat pomocí WaitForSingleObject nebo WaitForMultipleObjects.

Knihovny pro tvorbu paralelních aplikací bez nutnosti tvoření vláken v kódu:

OpenMP

Idea je taková, že vezmu sériový program a jen s minimem změn a s pomocí direktiv preprocesoru určím, které úseky kódu se budou vykonávat paralelně. Pokud se překladač vyvolá s direktivou OpenMP přeloží se jako paralelní program, jinak jako sériový. Pomocí direktiv pak lze překladači říci, které proměnné jsou sdílené a vyžadují tedy synchronizaci (kritickou sekci) a které nikoli.

From <https://d.docs.live.net/e3534876709763a3/Dokumenty/ZCU/Statnice/Statnice.docx>