

ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

Semestrální práce z předmětu KIV/PPR

Simulátor paralelně běžících vláken

Martin Sloup, A08N0111P

msloup@students.zcu.cz

Zadání

V základní verzi semestrální práce, uznatelné jen v rámci předmětu KIV/PPR, individuálně vytvořte simulátor paralelně běžících vláken, která budou implementovat a demonstrovat využití dále uvedených synchronizačních prostředků. Vlákno simulujte objektem, jehož třída bude implementovat rozhraní, nebo dědit z abstraktní třídy, následující metody:

```
ISimulatedThread = interface
  procedure Init(const Data:pointer);
  procedure Step(const SysCall:ISysCall);
end;
```

Metoda Init bude sloužit k inicializaci vlákna, metoda Step pak jeho krokování. Adresa, či reference, objektu a návratová hodnota metody Step je ekvivalentem intelovského **cs:rip** – tj. **Program Status Word**. Jejich implementace pak může vypadat následovně:

```
procedure TSimulatedThread.Init(const Data:pointer);
begin
  ...
  FNextStep:=0;
end;


procedure TSimulatedThread.Step(const SysCall:ISysCall);
begin
  inc(FNextStep);

  case FNextStep of
    1: FEvent:=SysCall.CreateEvent(nil, true, false, nil);
    2: SysCall.WaitForSingleObject(FEvent, 1000);
    3: SysCall.CloseHandle(FEvent);
  else
    FNextStep:=0;
  end;
end;
```

Simulovaný hardware, na němž vlákna poběží, omezte počtem procesorů – tj. kolik vláken může běžet zároveň. Dále simulujte preemptivní operační systém, který sám přepíná vlákna. Tj. musíte napsat alespoň základní plánovač. Další funkce implementujte dle potřeby. Nápověda: x86 procesory mají instrukci **HLT**, která až do přerušení pozastaví vykonávání kódu na daném procesoru. Ze synchronizačních prostředků nabízených zvoleným programovacím jazykem, nebo jeho knihovnami, které sami nenapíšete, můžete použít jen minimální potřebnou konstrukci k realizaci instrukce **TestAndSet**, případně k jedné výhradní operaci. Tj. ekvivalenty instrukcí x86: **cmpxchg** a **lock**.

Implementujte následující funkce WinAPI:

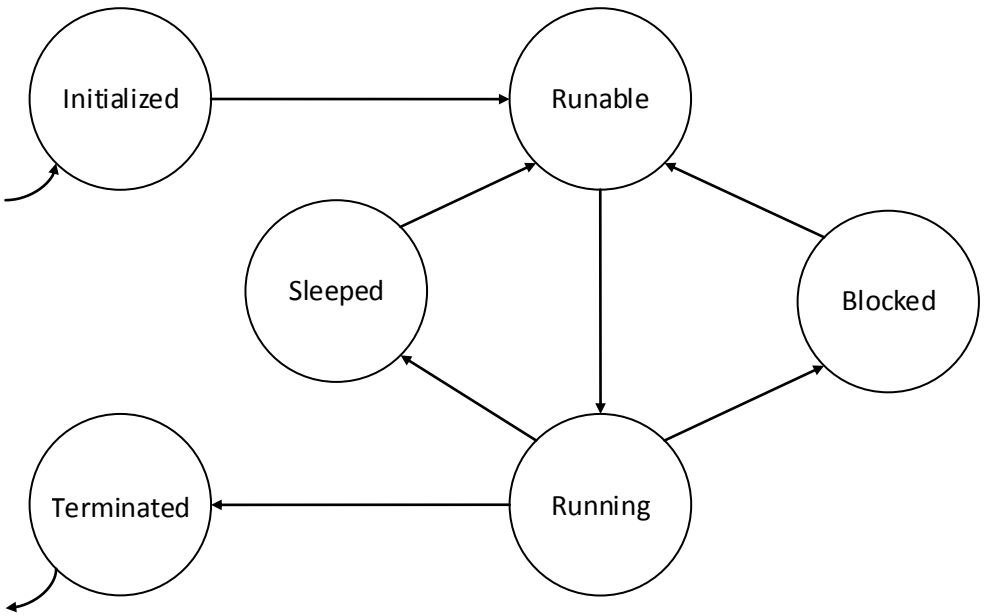
- CreateEvent, stačí se omezit na variantu CreateEvent(nil, true, false/true, nil)
- SetEvent
- ResetEvent
- WaitForMultipleObjects s tím, že objekty budou pouze ty, které si vytvoříte pomocí CreateEvent; implementujte všechny parametry až na bAlertable
- CloseHandle se stejným omezením jako WaitForMultipleObjects

Výsledný program napište tak, aby bylo možné zvolit počet vláken větší než je počet procesorů a zároveň, aby bylo možné nastavit počet objektů, na které se bude čekat pomocí implementace **WaitForMultipleObjects**. Pokud si někdo troufá na x86 implementaci, **VirtualBox**, **Qemu** a **Bosch** umí emulovat intelovské **SMP**. Ačkoliv by takové řešení bylo patřičně ohodnoceno, nepouštějte se do toho, co je nad vaše síly.

Řešení

K vytvoření simulátoru běžících vláken (fiberů) byl použit programovací jazyk Java s využitím vláken. Základní princip simulátoru staví na formě virtuálních procesorů (**CPU**), které jsou implementovány pomocí Javovských vláken a v cyklu provádí instrukce fiberů.

CPU vykonává instrukce fiberu jen po určitou dobu, a pak vybere jiný fiber, který je připraven k běhu. Dosavadnímu vláknu přiřadí stav plánovaný, pokud během svého běhu nezískal jiný stav. Umožní tak, že plánovač ho může později znovu naplánovat.



Obrázek 1 - Stavy fiberů

Fibery mohou nabývat těchto stavů (na obrázku je znázorněna jejich provázanost):

- **Initialized** – Fiber byl právě vytvořen. Je připraven být převeden do stavu Runable (spustitelný)
- **Runable** – Fiber je ve stavu spustitelný. Nyní je jen potřeba, aby ho vybral plánovač pro CPU.
- **Running** – Fiber běží na jednom z CPU. CPU vykonává postupně jeho instrukce.
- **Blocked** – Fiber čeká na seznam událostí, než budou zasignalizovány.
- **Sleeped** – Fiber je uspán na zadanou dobu. Po této době může být znovu naplánován. Jedná se o obdobu sleep funkce v klasických operačních systémech.
- **Terminated** – Fiber doběhl. Je připraven na odstranění ze seznamu fiberů.

Plánovač vybírá vlákna k běhu náhodně z podmnožiny seznamu všech fiberů, které mají stav **Runable**. Pokud se povede takový fiber vybrat, předá jej CPU, které následně vykonává instrukce tohoto fiberu. Toto se provádí do té doby, dokud fiberu nevyprší přidělený čas běhu nebo se nezmění stav fiberu. Pokud se fiberu nezměnil stav, přidělí fiberu stav **Runable**. Nakonec CPU zavolá znovu plánovač, který vybere další fiber. Nepovede-li se najít volný fiber k běhu, CPU vykoná halt instrukci (uspí se) a po chvilce to zkusí znovu.

Plánovač pravidelně prohlíží seznam fiberů a hlídá, zda fibery, které čekají na event, nevypršel čas čekání na signalizaci události. Také hlídá u fiberů stav **Sleeped**. Tedy, zda již čas uspání není vyšší než aktuální čas.

Synchronizace mezi fibery je zajištěna pomocí událostí (eventů), nad kterým jeden event může čekat a jiný event provede nastavení signalizace eventu, čímž zruší čekání nad eventem.

V simulátoru jsou implementovány dva typy eventů: **manuálně resetující** a **automaticky resetující**. Manuálně resetujícímu event stačí nastavit signalizaci pouze jednou. Signalizace je u eventu držena, dokud není zavolána metoda **resetEvent**. Pokud není event signalizován, tak fibery nad těmito eventy jsou uspány do doby signalizace. V opačném případě fibery nad těmito eventy již nečekají. U automaticky resetujícího eventu se probouzí pouze již spící fibery. Žádná signalizace se nenastavuje. Pouze pokud nikdo nad eventem nečeká, je nastavena signalizace. Ale ta je platná jen pro první fiber, který bude chtít nad eventem čekat. Taktéž je možné u obou typů eventů nastavit dobu čekání (timeout). Pokud vyprší tato doba čekání, jsou u fiberu vykonány další instrukce bez ohledu na stav eventů.

Dle zadání byly implementovány všechny požadované funkce **WinAPI**. Následuje jejich zběžný popis:

- **createEvent** – Vytvoří nový event. Je možné u eventu definovat jeho jméno (metoda createEventEx). V případě již existujícího eventu se stejným jménem je pouze vrácena reference již

existujícího eventu. Z definice této metody ve WinAPI byl pouze vypuštěn parametr lpEventAttributes.

- **setEvent** – Nastaví signalizaci eventu (u manuálně resetujícího) nebo jen probudí spící fibery (u automaticky resetujícího).
- **resetEvent** – Zruší nastavení signalizace u eventu.
- **waitForSingleObject** – Fiber čeká na signalizaci nad daným eventem. Je možné specifikovat maximální dobu čekání, případně nastavit dobu čekání na nekonečno (bude čekat stále). Z definice té samé metody ve WinAPI byl pouze vypuštěn parametr bAlertable.
- **waitForMultipleObjects** – Prakticky dělá to samé jako předchozí funkce, ale čekání je provedeno nad více eventy. Je možné nastavit, že se bude čekat jen na první signalizovaný event v seznamu.
- **closeHandle** – Uzavře zadaný event. Od této doby již není instance otevřeného / vytvořeného eventu pomocí metody createEvent (případně createEventEx) k dispozici.

U simulátoru musela být zajištěna podpora konkurenčního přístupu ke sdíleným položkám, aby si vlákna ve formě CPU nepřepisovala data navzájem. To bylo dosaženo vytvořením systému zámků (**spin locků**). Tento zámek je tvořen metodou **TSL** (Test and set). Dle zadání nebylo možné použít synchronizační prostředky jazyka Java (výjimku tvoří metoda **TSL**).

Programátorská příručka

V následujících několika odstavcích bude popsána struktura stěžejních tříd a metod programu. Pro přesnější popis je možné zabrousit do přiložené **javadoc dokumentace** v podadresáři API.

Balík cz.zcu.kiv.ppr.msloup

Třída Computer

Vytváří dojem abstrakce počítače. Z konstruktoru přebírá počet procesorů (instancí třídy CPU), jenž následně vytváří. Pomocí metody **executeFiber** je možné na stroji spustit jakýkoliv fiber.

Třída Main

Slouží ke spuštění simulátoru. Vytváří instanci třídy **Computer** a spouští tak ukázkové fibery.

Balík cz.zcu.kiv.ppr.msloup.kernel

V tomto balíku je obsaženo veškeré jádro simulátoru.

Interface IEvent

Slouží na zakrytí implementace Eventu pro Fiber.

Třída CPU

Reprezentuje virtuální procesor v simulaci. Implementace procesoru je provedena oddědění z Javovské třídy **Thread** a doimplementováním metody run. Tato metoda obsahuje cyklus, ve kterém od plánovače získává fiber a vykonává instrukce (**stepy**) tohoto fiberu, pro daný časový úsek.

Třída Event

Třída implementuje eventy sloužící k synchronizaci vláken. Jsou instancí nad třídou **EventStructure**, která je společná pro eventy se stejným jménem. Je tak ošetřena nemožnost použití eventu po jeho uzavření. Obsahuje veškerou logiku pro nastavení signalizace a zrušení signalizace eventu.

Třída EventStructure

Obsahuje vlastní stav eventu, seznam fiberů, které nad příslušným eventem čekají. Eventy se stejným názvem používají tu samou instanci **EventStructure**.

Třída Fiber

Abstraktní třída, kterou musí dědit veškeré uživatelské Fibery, které chtějí běžet na simulátoru paralelních vláken. Pro uživatelské Fibery je potřeba doimplementovat metody **getStepCount** a **step**. První z nich vrací celkový počet instrukcí stepu (počet case bloků uvnitř switche), druhá simuluje pomocí case bloků uvnitř switche jednotlivé instrukce fiberu. Celkový počet instrukcí je potřeba k určení konce běhu fiberu. Do **step** metody se předává objekt systémového volání. V zadání je uvedena povinná metoda init, která byla nahrazena konstruktorem třídy.

Kromě toho třída Fiber obsahuje také některé pomocné metody pro řízení skoků usnadňující implementaci cyklů a rozhodování uvnitř programového kódu Fiberu. Jedná se o metody **ifGoto** a **goTo**.

Třída Memory

Shromažďuje veškeré společné struktury simulátoru. Mezi ně patří například seznam fiberů v simulaci, seznam eventů (**EventStructure**) nebo taká zámky, pro zajištění konkurenčního přístupu k těmto strukturám.

Třída Scheduler

Obsahuje implementaci plánovače fiberů pro procesory. Kromě výběru fiberů k běhu na procesorech, kontroluje také vypršení timeoutů při čekání na signalizaci eventu nebo běžného uspání fiberu. Při stavu **Terminated** u fiberu, odebírá fiber ze seznamu fiberů.

Třída SysCall

Definuje pomocí metod systémová volání (**API**) pro fibery. U některých metod pouze předává volání do jiných tříd (např. **setEvent**, **resetEvent**). Instance této třídy je v simulátoru společná pro všechny procesory.

Rozlišení, který fiber provedl systémové volání, je provedeno dotazem na CPU. Ten vrací referenci na CPU, na které běží fiber vykonávající toto systémové volání. Z CPU již není problém získat referenci na příslušný fiber.

Enum FiberState

Popisuje všechny stavy fiberu, které mohou nastat.

Balík cz.zcu.kiv.ppr.msloup.kernel.util

Obsahuje pomocné třídy, jako je například třída **Lock**.

Třída Lock

Implementuje zamykání metodou **spin lock**, které je potřeba k synchronizaci přístupu u sdílených objektů v paměti. Stěžejní metodou je zde **TSL** (Test and Set), která zajišťuje atomičnost nastavení zámku.

Balík cz.zcu.kiv.ppr.msloup.userspace

Tento balík slouží k uložení tříd uživatelských fiberů. Zde se nacházejí dvě třídy **FarmerFiber** a **WorkerFiber**, které demonstrují schopnosti implementace simulátoru. Činnost těchto dvou tříd je popsána níže v Uživatelské dokumentaci nebo pomocí vývojových diagramů v příloze.

Uživatelská dokumentace

Simulátor vyžaduje ke svému běhu **Java Runtime Environment** (JRE) ve verzi 1.6 nebo vyšší. Simulátor je možné spustit následujícím příkazem:

```
java –jar simcpu.jar  [pocet_CPU [pocet_fiberu [pocet_eventu]]]
```

Kde **pocet_CPU** zde udává počet virtuálních procesorů, na kterém budou spuštěny fibery. **Pocet_fiberu** zde udává počet vytvořených instancí **WorkerFiber**. A nakonec parametr **pocet_eventu** říká, kolik eventů bude vytvořeno v simulaci. Pokud nebudou uvedeny příslušné parametry, vezmou se výchozí hodnoty, tj.: **2, 100 a 10** (počet CPU, počet fiberů a počet eventů).

Testovací fibery v simulaci provádí následující činnost, nejprve **FarmerFiber**:

1. Vytvoření zadaného počtu eventů.
2. Vytvoření zadaného počtu instancí **WorkerFiber**, kterému je v parametru předán náhodný počet eventů. Spuštění vytvořených instancí.
3. Náhodný výběr eventu ze seznamu nezasignalizovaných eventů. Zasignalizování vybraného eventu. Prodleva alespoň 200 ms do dalšího výběru.
4. Zrušení všech vytvořených eventů.

Nyní pro **WorkerFiber**:

1. Čekání nad zadaným počtem eventů (zadáno v konstruktoru).
2. Ohlášení, o dokončení čekaní nad zadanýma eventama včetně jejich vypsání.

Jednotlivé činnosti vláken jsou vypisovány na obrazovku včetně ladících informací, jako například na kterém procesoru běží který fiber.

Závěr

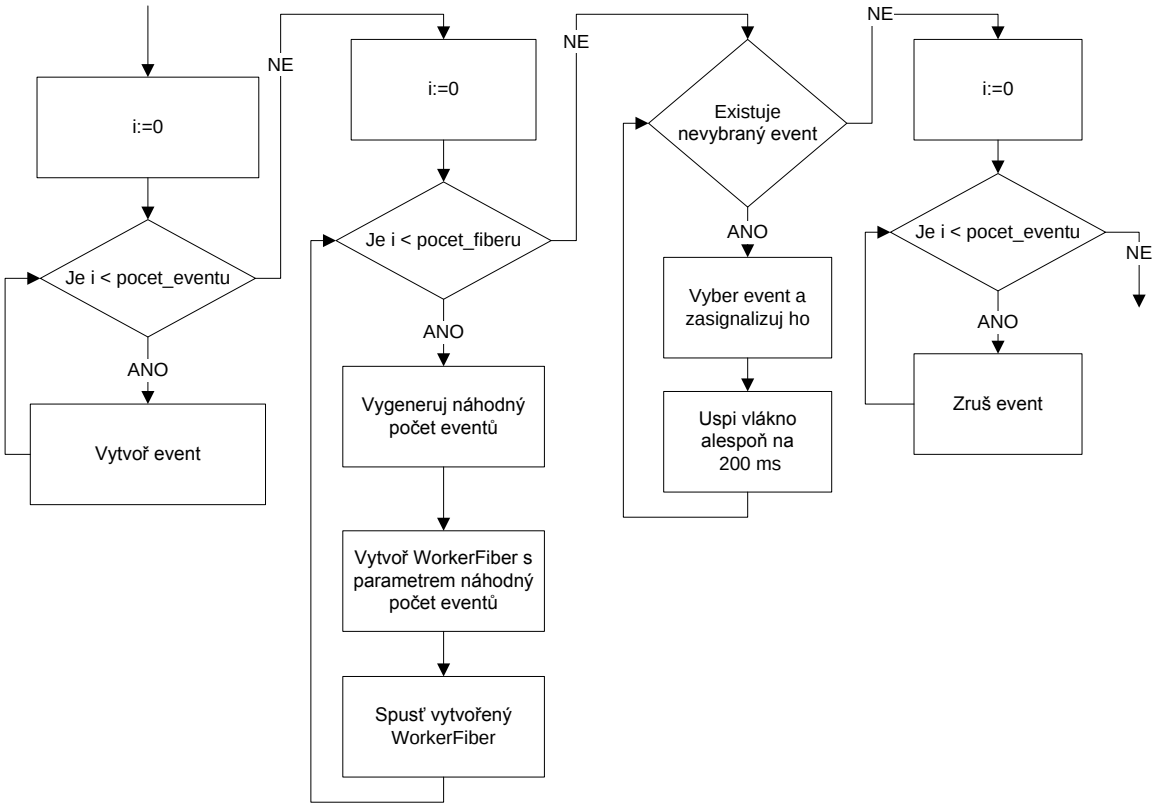
Dle zadání byl vytvořen simulátor paralelně běžících vláken s podporou synchronizace prostřednictvím eventů. API simulátoru umožňuje jednoduché vytvoření vláken (fiberů) a simulování těchto vláken v simulátoru. Programování simulátoru zabralo docela dost času a hromady přepsání (refaktoringu) kódu. V některých situacích se nedalo odhadnout, ve které třídě by se měla příslušná vlastnost nacházet. Ze začátku dělalo problém větvení instrukčního kódu fiberu, ale to bylo nakonec vyřešeno elegantním vytvořením pomocných metod pro skoky. Popisování jednotlivých instrukcí fiberu přineslo zpřehlednění výsledného kódu. Určitě bych viděl v simulátoru ještě nějaké možnosti vylepšení, ale bylo nutné již vytvořenou práci (tedy simulátor) odevzdat.

Použité zdroje

- Manuál MSDN k vybraným funkcím
- Přednášky předmětu KIV/PPR
- Selský rozum

Přílohy

Vývojový diagram FarmerThread



Vývojový diagram WorkerThread

