Synchronizace v jádře, symetrický multiprocesor, atomické operace.

Thursday, May 30, 2013 8:29 AM

Problém kritické sekce - uvedení dat do nekonzistentního stavu nebo přerušení v okamžiku výpočtu

Kód jádrových funkcí se vykonává:

- Při systémovém volání
- Při obsluze výjimek a přerušení

Synchronizace v jádře

- Atomické operace
  - Atomická instrukce, jen jeden procesor aktivní v době jejího vykonání
- Spinlock
  - Klasický spinlock, u uniprocesoru řešen jen zákazem preempce jádra, uvnitř spinlocku je preempce jádra též zakázána, u multiprocesoru v čekací smyčce je preemplce jádra povolena a kernel tak místo toho může naplánovat jiný proces
- Read/Write spinlock
  - Oproti klasickému spinlocku zde readeři mohou číst neustále, writer musí ale čekat až všichni vše přečtou, pak to zamknout a zapsat; to samé u readerů, kteří všichni musejí čekat až jim to writer zas uvolní
- Seqlock

  When using read/write spin locks, requests issued by kernel control paths to perform a read_lock or a write_lock operation have the same priority: readers must wait until the writer has finished and, similarly, a writer must wait until all readers have finished.
  Seqlocks introduced in Linux 2.6 are similar to read/write spin locks, except that they give a much higher priority to writers:
  in fact a writer is allowed to proceed even when readers are active. The good part of this strategy is that a writer never waits (unless another writer is active);
  the bad part is that a reader may sometimes be forced to read the same data several times until it gets a valid copy.
  Each seqlock is a seqlock_t structure consisting of two fields:
  a lock field of type spinlock_t and an integer sequence field.
  This second field plays the role of a sequence counter. Each reader must read this sequence counter twice, before and after reading the data, and check whether the two values coincide.
  In the opposite case, a new writer has become active and has increased the sequence counter, thus implicitly telling the reader that the data just read is not valid.
- Kernel Semaphore

  However, whenever a kernel control path tries to acquire a busy resource protected by a kernel semaphore, the corresponding process is suspended. It becomes runnable again when the resource is released.
  Therefore, kernel semaphores can be acquired only by functions that are allowed to sleep:
  interrupt handlers and deferrable functions cannot use them.
- BKL (Big Kernel Lock)
- http://www.ibm.com/developerworks/linux/library/l-linux-synchronization/index.html

Systémová volání sdílejí jednotlivé struktury v jádře OS – snaha o zabránění soutěže nad systémovými daty prostřednictvím:

Nepreemptivnost procesů jádra

Jádrový proces nemůže být přerušen a nahrazen procesem s vyšší prioritou (pokud samotný proces neudělá yield). I tak ale může být přerušen výjimkou nebo přerušením, a obsluha přerušení může být zase přerušena přerušením.

Problém blokujících operací – možnost deadlocku -> uvolnit procesor před blokující operací

Atomické operace

Instrukce provádějící celou v jedné instrukci (inc, dec...)

Nemůže vzniknout nekonzistence dat – není přerušení

Pouze pro jednoprocesorový stroj, pro multiprocesorový je speciální instrukce zamykající sběrnici

Zákaz přerušení

Kritická oblast začíná zákazem přerušení a končí obnovením přerušení

KS musí být krátká a rychlá -> blokování I/O a procesoru

Velmi nebezpečné a nefunguje na víceprocesorových systémech

Nesmí se čekat na oznámení události přerušením

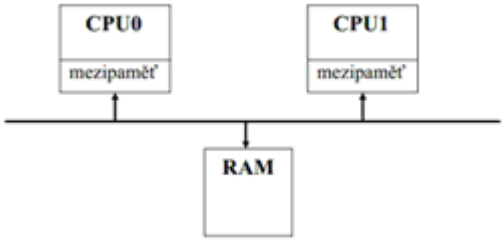
Zamykání

Pro víceprocesorové systémy velmi nákladné (režie)

Semafory v jádru, spinlock

Zabránění uvíznutí kdy více procesů žádá více zdrojů (přiřazení A->1, 2 B-> 2, 1) – prostředky se VŽDY alokují v pořadí adres semaforů

Symetrický multiprocesing SMP



- Procesory (identické) a sdílená hlavní pamět jsou připojeny ke společné sběrnici, tu je nutno zamykat pro výhradní přístup jednoho procesoru (aby nemohly 2 procesy zapisovat na stejné místo v paměti naráz).
- Klasické atomické instrukce (dec, inc) nejsou atomické, nutná speciální instrukce zamykající sběrnici
- Nutná synchronizace mezipaměti (cache) procesorů: když procesor modifikuje svou mezipamět, musí kontrolovat jestli stejná data nejsou v mezipaměti jiného procesoru a když ano, musí je aktualizovat nebo zneplatnit.
- Úloha může být zpracovávána postupně různými procesory, je umístěna ve sdílené hlavní paměti
- Na více procesorech naráz mohou být současně provedena systémová volání
- Obsluha přerušení – obsluhuje procesor, který má k dispozici nejvíce volných prostředků (obsluhuje

proces s nejnižší prioritou)

Nutno řešit pokročilým způsobem synchronizaci jádra – semafory + spinlock

spinlock – efektivní pro multiprocesory – blokující skončí na jiném procesoru a nemusí se nic přepínat

spinlock nesmí chránit data přístupná při obsluze přerušení - přerušení modifikuje data – problém řešený zákazem přerušení v KS

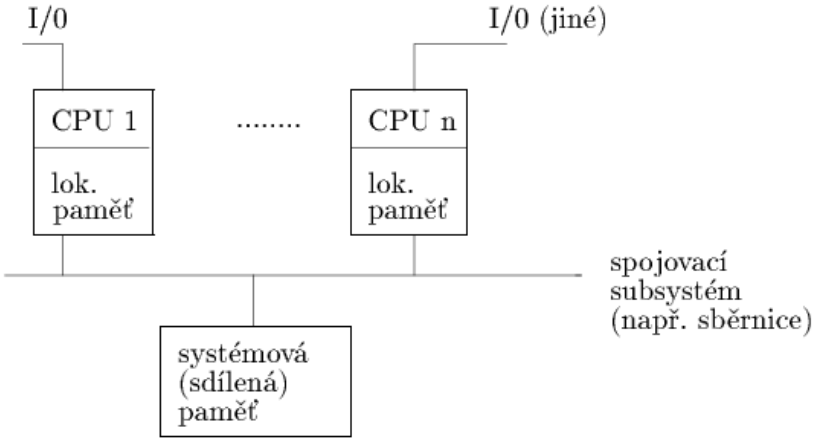
Procesy mohou využívat zámky dvěma způsoby:

sdílené - pro čtení (lepší propustnost)

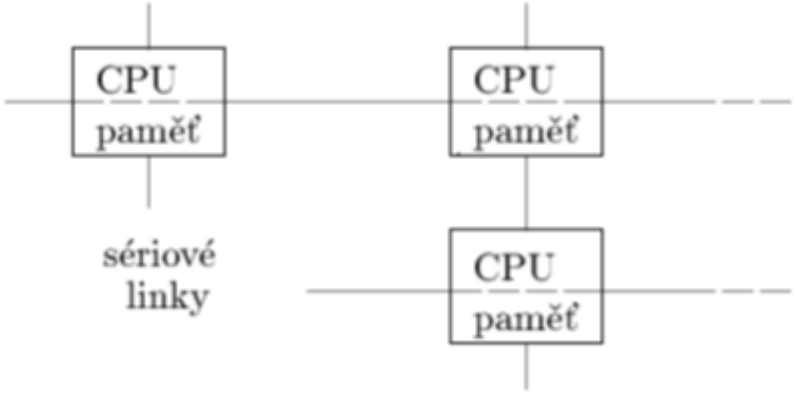
výhradní – pro zápis

Kromě SMP existují další technologie pracující na podobném principu:

- **asymetrické multiprocesory ASMP** - navíc vlastní lokální paměti a I/O připojení u procesorů, které tak mohou mít různé instrukční sady – každý procesor má speciální funkce – nelze libovolný proces na libovolný CPU



- **multiprocesor s distribuovanou pamětí** - procesory mají jen lokální paměti, není sdílená paměť, komunikace zasíláním zpráv, topologie 2D mřížky nebo N-rozměrné krychle, výhoda odstranění společné sběrnice jako úzkého hrdla



Atomické operace

- operace vykonávaná procesorem v jedné instrukci (inc, dec)
- problém u SMP – instrukce již není atomická v rámci procesů – nutno navíc uzamknout sběrnici instrukcí lock*
- atomická instrukce pro podporu semaforů a spinlocku – TSL (TestAndSetLock)

```
mov edx, DWORD(-1) //-1 zamčeno
//otestujeme stav zámku, 0 = odemčeno
spin: mov eax, [lockState]
test eax, eax
jnz spin
//zkusíme ho zamknout s -1
lock cmpxchg [lockState], edx
//nepředběhl nás jiný procesor?
//původní lockState je v eax
test eax, eax
jnz spin
```

Neoptimální verze:

```
while(!acquire_lock())
{ Sleep( 0 ); }
do_work();
release_lock();
```

```
spin_lock:
    TSL R, lock ;; atomicky provede R:=lock a lock:=1
    CMP R, 0 ;; byla v lock 0?
    JNE spin_lock ;; pokud nebyla (R<>0), byl zámek nastaven ->
    cyklus
    RET ;; návrat, tj. vstup do KS

spin_unlock:
    LD lock, 0 ;; ulož hodnotu 0 do lock
    RET
```

Z <https://d.docs.live.net/67cbdb2faf0647ea/Dokumenty/FAV/A11N0110P/Státní%20závěrečná%20zkouška/PPR/PPR_Karfik_v2.docx>

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