OS MMXX

MIT ;)

https://pdos.csail.mit.edu/6.828/2020

Zamky

Preco nas ma tato tema zaujimat?

- Pouzivatel spusta viacero programov naraz
- Kernel musi vediet obsluhovat viacero systemovych volani sucasne
- Takze viacero tokov vykonavania sucasne moze pristupovat k roznym datovym strukturam

Preco nas ma tato tema zaujimat?

- Pouzivatel spusta viacero programov naraz
- Kernel musi vediet obsluhovat viacero systemovych volani sucasne
- Takze viacero tokov vykonavania sucasne moze pristupovat k roznym datovym strukturam

- Zamky nam pomahaju tuto situaciu zvladat (spolocne to zvladneme)
- Zamky znizuju efektivitu paralelneho vykonavania (nie vzdy to spolocne zvladneme)

Preco nas ma tato tema zaujimat?

- Ukazka: zmazme acquire()/release() v kfree()
  - xv6 nastartuje normalne
  - Zbehnu vsetky usertesty, okrem stratenia nejakych pamatovych ramcov… `usertests reparent2`
- Preco pride ku strate ramcov? Obrazok so subehom
- Potrebujeme zamky pre zabezpecenie spravnosti fungovania, ale zaroven stracame vykon (kfree() bezi seriovo)

ADT Lock

- Uz sme o tom hovorili na minulej prednaske

- lock l
- acquire(l)
- x = x + 1 // KO (*critical section*)
- release(l)

ADT Lock

- Ak viacero vlakien (sucasne) vyvola acquire(l)
  - Iba jednemu vlaknu sa podari z funkcie vratit a pokracovat vo vykonavani kodu
  - Ostatne musia cakat na uvolnenie (release()), ostanu v stave “zablokovany”

ADT Lock

- Ak viacero vlakien (sucasne) vyvola acquire(l)
  - Iba jednemu vlaknu sa podari z funkcie vratit a pokracovat vo vykonavani kodu
  - Ostatne musia cakat na uvolnenie (release()), ostanu v stave “zablokovany”
- Zamok nie je automaticky spaty so ziadnou premennou – je na programatorovi, aby samotnym kodom urcil, na co bude zamok sluzit

Kedy pouzit v kode zamok?

- Ak viac tokov vykonavania moze sucasne pristupit k spolocnemu pamatovemu miestu a aspon jeden z tychto tokov vykonava operaciu zapisu na danom pamatovom mieste

Kedy pouzit v kode zamok?

- Ak viac tokov vykonavania moze sucasne pristupit k spolocnemu pamatovemu miestu a aspon jeden z tychto tokov vykonava operaciu zapisu na danom pamatovom mieste

- Nikdy v programe nepristupuj k zdielanym udajom bez pouzitia spravneho zamku (kazdy udaj moze mat vlastny zamok)

Automaticke uzamykanie

- Vyssie programovacie jazyky poskytuju ADT s automatickou podporou uzamykania
- Tento pristup nie je mozne pouzit na vsetky pripady pouzitia

Automaticke uzamykanie

- Tento pristup nie je mozne pouzit na vsetky pripady pouzitia
  - Napr. rename(“dir1/file1”, “dir2/file2”)
    - lock(dir1), erase(file1), unlock(dir1)
    - lock(dir2), add(file2), unlock(dir2)
  - Problem: subor isty casovy interval nejestvuje
  - rename() musi byt atomicka operacia

Automaticke uzamykanie

- Tento pristup nie je mozne pouzit na vsetky pripady pouzitia
  - Napr. rename(“dir1/file1”, “dir2/file2”)
    - lock(dir1), erase(file1), unlock(dir1)
    - lock(dir2), add(file2), unlock(dir2)
  - Problem: subor isty casovy interval nejestvuje
  - rename() musi byt atomicka operacia:
    - lock(dir1), lock(dir2)
    - erase(file1), add(file2)
    - unlock(dir2), unlock(dir1)

Automaticke uzamykanie

- Tento pristup nie je mozne pouzit na vsetky pripady pouzitia
  - Napr. rename(“dir1/file1”, “dir2/file2”)
    - lock(dir1), erase(file1), unlock(dir1)
    - lock(dir2), add(file2), unlock(dir2)
  - Problem: subor isty casovy interval nejestvuje
  - rename() musi byt atomicka operacia:
    - lock(dir1), lock(dir2)
    - erase(file1), add(file2)
    - unlock(dir2), unlock(dir1)
- Programator musi mat kontrolu nad pouzitim

Na co su zamky dobre

1. Aby sme sa vyhli stratam udajov pri ich aktualizacii

Na co su zamky dobre

1. Aby sme sa vyhli stratam udajov pri ich aktualizacii
2. Aby sme dokazali z viac krokovej operacie urobit atomicku operaciu (ukryt interny nekonzistentny medzistav)

Na co su zamky dobre

1. Aby sme sa vyhli stratam udajov pri ich aktualizacii
2. Aby sme dokazali z viac krokovej operacie urobit atomicku operaciu (ukryt interny nekonzistentny medzistav)
3. Vo vseobecnosti na zachovanie invariantov pri roznych ADT
   - Invariant je platny pred zacatim operacie
   - Zamky “ukryju” docasne porusenie invariantu
   - Na konci operacie pred uvolnenim zamku sa platnost invariantu obnovi

Uviaznutie

- Majme verziu rename() s 2 zamkami

vlakno A:	vlakno B:
rename(“a/1”,”b/2”)	rename(“b/3”,”a/4”)
lock(a)	lock(b)
lock(b)	lock(a)
…	…

- Ake moze byt riesenie?

Uviaznutie

- Uzamykanie v tom istom poradi
  - Musime vediet, o ktore zamky sa jedna
  - Zoradit ich, a az potom volat acquire()
  - Prilis komplexne riesenie

- Programator musi poznat kod, vyznam jednotlivych zamkov a miesta ich pouzitia

Zamky versus paralelne vykonavanie

- Zamky ZNEMOZNUJU paralelne vykonavanie

Zamky versus paralelne vykonavanie

- Zamky ZNEMOZNUJU paralelne vykonavanie
- Na umoznenie (ciastocneho) paralelizmu je casto nutne rozdelit datovu mnozinu (ktoru zamok chrani) na mensie casti
- Kazda mensia cast bude mat vlastny zamok
- Vid ulohy cvicenia

- V zmysle delenia datovej mnoziny hovorime o granularite (hruba versus jemna)

Zamky versus paralelne vykonavanie

- Najdenie optimalnej granularity je netrivialna uloha

Zamky versus paralelne vykonavanie

- Najdenie optimalnej granularity je netrivialna uloha
  - Cely FS / per adresar a subor / per diskovy blok
  - Cele jadro systemu / kazdy subsystem / kazdy objekt

Zamky versus paralelne vykonavanie

- Najdenie optimalnej granularity je netrivialna uloha
  - Cely FS / per adresar a subor / per diskovy blok
  - Cele jadro systemu / kazdy subsystem / kazdy objekt

- Uloha 1 z cvicenia
  - 1 zoznam volnych ramcov RAM rozdelit na tolko zoznamov, kolko je CPU
  - Kazde CPU bude vyuzivat svoj zoznam
  - Mozu pristupovat k zoznamom sucasne

Granularita zamkov

- Ako najst spravnu granularitu?

Granularita zamkov

- Ako najst spravnu granularitu?
- Navrh zacneme jednym velkym zamkom (tzv. *big lock*)
  - Mensia sanca uviaznutia
  - Menej dokazovania spravnosti invariantov

Granularita zamkov

- Ako najst spravnu granularitu?
- Navrh zacneme jednym velkym zamkom (tzv. *big lock*)
  - Mensia sanca uviaznutia
  - Menej dokazovania spravnosti invariantov
- Zmeriame efektivitu riesenia
  - Casto riesenie s velkymi zamkami postacuje

Granularita zamkov

- Ako najst spravnu granularitu?
- Navrh zacneme jednym velkym zamkom (tzv. *big lock*)
  - Mensia sanca uviaznutia
  - Menej dokazovania spravnosti invariantov
- Zmeriame efektivitu riesenia
  - Casto riesenie s velkymi zamkami postacuje
- Jemnejsiu granularitu riesime, iba ak je to nutne (s ohladom na zmeranu efektivitu riesenia)

Implementacia zamkov

```
1 struct lock { int locked; }
2 acquire(l) {
3    while(1) {
4        if (l->locked == 0) {
5            l->locked = 1;
6            return;
7        }
8    }
9 }
```

Implementacia zamkov

- Ako vyriesit subeh?
- Potrebujeme atomicku operaciu

Implementacia zamkov

- Ako vyriesit subeh?
- Potrebujeme atomicku operaciu
  - Vymeny obsahu pamatoveho miesta
  - Moze byt aj zlozitejsia CAS (*compare-and-swap*)

Implementacia zamkov

- Ako vyriesit subeh?
- Potrebujeme atomicku operaciu
  - Vymeny obsahu pamatoveho miesta
  - Moze byt aj zlozitejsia CAS (*compare-and-swap*)
- Atomicka instrukcia swap(addr, reg)
  1. lock addr (globalne zamkne adresu (zbernicu))
  2. temp := *addr
  3. *addr := reg
  4. reg := temp
  5. unlock addr

Implementacia zamkov

- Implementacia ADT Spinlock pomocou CAS

Implementacia zamkov

- Implementacia ADT Spinlock pomocou CAS

```
acquire(l) {
  while(__sync_lock_test_and_set(&lk->locked, 1) != 0)
      ;
}
```

Implementacia zamkov

- Implementacia ADT Spinlock pomocou CAS

```
acquire(l) {
  while(__sync_lock_test_and_set(&lk->locked, 1) != 0)
      ;
}
```

  - Ak lk->locked bolo rovne 1, instrukcia znovu nastavi hodnotu 1; zaroven vrati hodnotu 1 (povodnu hodnotu pred zmenou)
  - Ak lk->locked bolo rovne 0, instrukcia nastavi hodnotu 1 a vrati hodnotu 0 (povodnu hodnotu pred zmenou)

Poradie instrukcii

- Doteraz sme predpokladali, ze program sa vykonava sekvencne
  - T.j. majme sekvenciu ASM instrukcii
  - Instrukcie sa vykonavaju za sebou

Poradie instrukcii

- Doteraz sme predpokladali, ze program sa vykonava sekvencne
  - T.j. majme sekvenciu ASM instrukcii
  - Instrukcie sa vykonavaju za sebou

- Toto nie je pravda
  - GCC moze optimalizaciou poprehadzovat instr
  - CPU moze pri vykonavani poprehadzovat instr

Poradie instrukcii

Program 1:

```
locked = 1
x = x + 1
locked = 0
```

Program 2:

```
while(locked == 1)
    ….
locked = 1
x = x + 1
locked = 0
```

==========

```
locked = 1
locked = 0
x = x + 1
```

Poradie instrukcii

- Aby sme zabranili poprehadzovaniu instrukcii narabajucich s pamatou (ukazatelmi), vyuzivame dalsiu gcc *intrinsic* funkciu __sync_synchronize()

Poradie instrukcii

- Aby sme zabranili poprehadzovaniu instrukcii narabajucich s pamatou (ukazatelmi), vyuzivame dalsiu gcc *intrinsic* funkciu __sync_synchronize()

- Kompilator nemoze presunut instrukcie pracujuce s pamatou za volanie tejto “funkcie”
- Kompilator by mal vygenerovat instrukciu “*memory barrier*” pre cielove CPU, aby ani CPU neprehodilo ziadnu pamatovu instrukciu spred bariery za nu

Naco je dobry ADT Spinlock

- Neustale vytazuje na 100% CPU; je potrebny?

Naco je dobry ADT Spinlock

- Neustale vytazuje na 100% CPU; je potrebny?

- Vyuziva sa pri cakani na “kratky” cas
  - Kratky cas znamena kratsie ako je rezia nutna k preplanovaniu vlakna pri ADT Sleeplock
- Zaroven pri ADT Spinlock NESMIE prist ku preplanovaniu
  - Vid minula prednaska
  - Pri moznosti preplanovania by mohlo nastat uviaznutie

Naco je dobry ADT Sleeplock

- Ak je cakanie dlhsie nez rezia nutna k pouzitiu ADT Sleeplock
- Ak je nutne preplanovanie

- Cakajuce vlakno uvolni CPU
- Zvacsa sa caka na externe udalosti (udaje zo zariadeni)
- Klavesnica, disk, sietova karta…

Domace citanie

Chapter 6

Locking

xv6: a simple, Unix-like teaching operating system