Синхронизација на повеќе процеси: Задачи

Оперативни системи 2014
Аудиториски вежби

Задача: Producer – Controller

- Проблем: Producer – Controler, со ограничен број проверки
- Потребно е да направите оптимизација на додавањата и проверките на податоците од одреден бафер според следните услови:
- Кога се додава податок во баферот, во истиот момент:
  - Не може да има додавање на други податоци
  - Не може да се прави проверка на податоци
- Кога се прави проверка на податоци, во истиот момент:
  - Може да има максимум 10 активни проверки
  - Не може да има додавање пред да завршат сите започнати проверки
- Иницијално во баферот има податоци

Задача: Producer – Controller

- Баферот е претставен со инстанцата `buffer` од класата `Buffer`. Притоа може да ги користите следните методи:
  - `state.produce()`
    - Додава елемент во баферот.
    - Фрла RuntimeException со соодветна порака доколку во истиот момент се врши додавање или проверка на друг податок.
  - `state.check()`
    - Врши проверка на податок од баферот.
    - Проверува дали во истиот момент се врши додавање или паралелна проверка на повеќе од 10 податоци.

Задача: Producer – Controller

- Имплементирајте ги методите `execute()` од класата `Producer` и `Controller`, кои ќе функционираат според претходните правила. Тие треба да ги користат методите `state.produce()` и `state.check()` за додавање и проверка на податоци од баферот, соодветно. Сите семафори и глобални променливи треба да ги дефинирате самите, а нивната иницијализација да ја направите во методот `init()`. При имплементацијата на методите, не смеете да додадете try-catch блокови во методите.
- При извршувањето има повеќе инстанци од класите `Producer` и `Controller`, кои вршат повеќе од едно додавање и проверка, соодветно. Додавањата и проверките се стартуваат (скоро) истовремено и паралелно се извршуваат.

Решение I: Initialization

```
static Semaphore accessBuffer;
static Semaphore canCheck;
static Semaphore lock;
static int numChecks;

public static void init() {
    accessBuffer = new Semaphore(1);
    canCheck = new Semaphore(10);
    lock = new Semaphore(1);
    numChecks = 0;
}
```

Решение I: execute() methods

```
/* Producer  */
accessBuffer.acquire();
state.produce();
accessBuffer.release();
```

```
/* Controller  */
lock.acquire();
if (numChecks == 0) {
    accessBuffer.acquire();
}
numChecks++;
lock.release();
canCheck.acquire();

state.check();
lock.acquire();
numChecks--;
canCheck.release();
if (numChecks == 0) {
    accessBuffer.release();
}
lock.release();
```

Решение II: Initialization

```
static Semaphore proizveduvac;
static Semaphore kontrolor;

public static void init() {
    proizveduvac = new Semaphore(1);
    kontrolor = new Semaphore(10);
}
```

Решение II: execute() methods

```
/* Producer  */
proizveduvac.acquire();
// Acquires the given number of permits from
// this semaphore,blocking until all are available
kontrolor.acquire(10);
state.produce();
kontrolor.release(10);
proizveduvac.release();

/* Controller    */
kontrolor.acquire();
state.check();
kontrolor.release();
```

Задача: SiO_2

- Во процесот на производство на EPROM меморија, потребен е слој на силициум диоксид (SiO_2). За да се формира оксидниот слој потребно е во ист момент да бидат присутни два атоми на кислород и еден атом на силициум.
- Со користење на семафори напишете програма која ќе помогне во процесот на производство на EPROM меморија.

Задача: SiO_2

- Секој од атомите е посебен процес.
- Силициумовите атоми (процеси) го извршуваат методот `proc_Si()`, а кислородните атоми методот `proc_O()`.
- Откако ќе „се сретнат“ сите три атоми, секој од нив го повикува виртуелниот метод `bond()`, за да се формира SiO_2.

Задача: SiO_2

- Ограничувања:
- Доколку до “бариерата” пристигне атом на силициум, истиот мора да чека да се соберат два атоми на кислород.
- Доколку пристигне атом на кислород, мора да чека на еден атом на силициум и еден атом на кислород.
- Бариерата треба да ја напушти еден атом на силициум и два атоми на кислород, кои формираат молекул на силициум диоксид (SiO_2).

Решение (дискусија):

- Поставување на семафорите:
  - Семафор кој што ќе врши контрола на атомот за силициум

```
Semaphore si = new Semaphore(1);
```

  - Семафор кој што ќе врши контрола на двата атоми на кислород (поради тоа што бројот на дозволи е 2, тоа значи дека може да се направат максимум 2 повици на `acquire()`)

```
Semaphore o = new Semaphore(2);
```

  - Семафор кој што ќе врши контрола за тоа дали дошол атом на силициум (кислород) до бариерата (поради тоа што истиот е поставен на 0, тоа значи дека при повик на функцијата `acquire()` прва, ќе настане блокирање, па мора да се направи барем еден `release()` за да бројот на дозволи стане 1)

```
Semaphore siHere = new Semaphore(0);

Semaphore oHere = new Semaphore(0);
```

  - Семафор кој врши контрола на бариерата

```
Semaphore ready = new Semaphore(0);
```

Решение I:

- Имплементација на методот `proc_Si()`:

```
proc_Si() {
   si.acquire();          // дозволува само еден атом на Si
   siHere.release();      // notify the first O atom for the arrival
   siHere.release();      // notify the second O atom for the arrival
   oHere.acquire();       // wait for the first O atom
   oHere.acquire();       // wait for the second O atom

   // all the atoms needed are here
   ready.release();       // for the first O atom
   ready.release();       // for the second O atom
   bond();                // create the molecule
   si.release();          // another Si atom can bond now
}
```

Решение I:

- Имплементација на методот `proc_O()`:

```
proc_O() {
   o.acquire();          // Only two atoms can interact simultaneously
   siHere.acquire();     // wait for the Si atom to arrive
   oHere.release();      // Notify that an O atom is here
   ready.acquire();      // Wait for the ready signal
   bond();
   o.release();          // The process is done
}
```

Решение II:

- Имплементација на методот `proc_Si()`:

```
proc_Si() {
   si.acquire();          // дозволува само еден атом на Si
   oHere.acquire(2);      // wait for the O atoms

   // all the atoms needed are here
   ready.release(2);      // for the first O atom
   bond();                // create the molecule
   si.release();          // another Si atom can bond now
}
```

Решение II:

- Имплементација на методот `proc_O()`:

```
proc_O() {
   o.acquire();          // Only two atoms can interact simultaneously
   oHere.release();      // Notify that an O atom is here
   ready.acquire();      // Wait for the ready signal
   bond();
   o.release();          // The process is done
}
```

Задача: Синхронизација на тоалет

- Во рамките на еден универзитет, за наставниот кадар се воведуваат заеднички тоалети во кои е дозволен влез и за жени и за мажи, според следните правила:
  - доколку во тоалетот има жена, дозволен е влез за други жени, но не и за мажи, и обратно (кога има маж, само други мажи може да влегуваат).
- На влезот од тоалетот има лизгачки знак кој кажува во која состојба се наоѓа тоалетот:
  - празен, жена внатре, маж внатре.
- За опишаната ситуација да се напишат следните процедури:
  - `zena_vleguva();`
  - `zena_izleguva();`
  - `maz_vleguva();`
  - `maz_izleguva();`

Задача: Синхронизација на тоалет

- Може да се користат бројачи и техники за синхронизација по желба.
- За кога некој влегува во тоалетот треба да се повика `wc.vlezi()`, а кога некој излегува, да се повика `wc.izlezi()`. Променливата wc е веќе дефинирана.
  - методите `vlezi()` и `izlezi()` не се атомични и треба да се погрижите за нивна синхронизација.
- При извршувањето има многу мажи и жени (паралелни нитки) кои се обидуваат да пристапат до тоалетот.

Решение:

```
Semaphore toalet = new Semaphore(1);
final Object mLock = new Object();
final Object zLock = new Object();
int maziVnatre = 0, zeniVnatre = 0; // глобални променливи

maz_vleguva() {
   synchronized (mLock) {
      if (maziVnatre == 0) {
         toalet.acquire(); // тоалетот се поставува на зафатен
      }
      maziVnatre++;
      wc.vlezi();
   }
}
```

Решение:

```
maz_izleguva() {
   synchronized (mLock) {
         wc.izlezi();
         maziVnatre--;
         if (maziVnatre == 0) {
             toalet.release(); // тоалетот е слободен
         }
    }
   }
```

Решение:

```
zena_vleguva() {
   synchronized (zLock) {
      if (zeniVnatre == 0) {
         toalet.acquire(); // тоалетот е зафатен
      }
      zeniVnatre++;
      wc.vlezi();
   }
}
```

Решение:

```
zena_izleguva() {
  synchronized (zLock) {
        wc.izlezi();
        zeniVnatre--;
        if (zeniVnatre == 0) {
            toalet.release(); // тоалетот се ослободува
        }
   }
  }
```

Задача: Уписи на ФИНКИ

- Во процесот на запишување студенти на еден факултет, запишувањето го изведуваат обучени членови на Конкурсната комисија, кои ги запишуваат заинтересираните кандидати.
- На денот на запишувањето има една просторија на располагање во која истовремено можат да работат најмногу 4 членови на Комисијата.
- Секој член од Комисијата запишува 10 кандидати, еден по еден, по што ја напушта просторијата и остава можност да влезе нов член на Комисијата.
- Еден кандидат се опслужува само од еден од членовите на Комисијата.
- Еден член на Комисијата може да опслужува само еден кандидат во даден временски момент.

Задача: Уписи на ФИНКИ

- Кога кај еден член на комисија е присутен еден кандидат, членот на Комисијата го прави запишувањето со повик на виртуелниот метод `zapishi()`. За време на извршување на овој метод, кандидатот мора да биде присутен во просторијата. Потоа е слободен.
- Со користење на семафори, напишете програма која ќе помогне во процесот на запишување студенти.
  - Секој член на Комисијата претставува посебен thread во системот и секој студент исто така претставува посебен thread.
  - Членовите на Комисијата го извршуваат методот `komisijaUpis()`, а кандидатите методот `studentUpis()`.
  - Напишете ги двата метода: `komisijaUpis()` и `studentUpis()`.

Решение:

```
Semaphore slobodnoUpisnoMesto = new Semaphore(4);
Semaphore studentEnter = new Semaphore(0);
Semaphore studentHere = new Semaphore(0);
Semaphore studentDone = new Semaphore(0);

komisijaUpis() {
   slobodnoUpisnoMesto.acquire(); //cekame da se oslobodi mesto za komisijata

   int i=NUM_STUDENTS; //go inicijalizirame brojot na studenti koi treba da
   //bidat upisani od kandidatot od komisijata. (LOKALNA PROMENLIVA)

   while(i>0){
         studentEnter.release();      //Kazuvame deka moze da vleze student
         studentHere.acquire();       //Cekame studentot da gi ostavi dokumentite
                                      //i da ni signalizira deka e ovde
         zapishi();                   //Go zapisuvame studentot
         studentDone.release();       //Kazuvame deka e upisan i moze da zamine
         i--;                         //Go namaluvame brojot na preostanati
                                      studenti
   }
   slobodnoUpisnoMesto.release();     //stom sme upisale 10 studenti, zaminuvame
}
```

Решение:

```
studentUpis() {
   studentEnter.acquire(); //studentot ceka da bide povikan
   ostaviDokumenti();
   studentHere.release(); //kazuva deka e ovde (gi ostavil
                                  dokumentite)
   studentDone.acquire(); //cekame da bide zapisan
   //studentot e zapisan
}
```

Задача: Синхронизација на пушачи

- Во една соба има 3 пушачи и еден агент. Секој пушач витка цигара и ја пуши според следниве правила:
  - За да ја свитка и испуши цигарата потребни му се 3 состојки: тутун, ризла и кибрит.
  - Едниот од пушачите има неограничена количина на тутун, другиот неограничена количина на ризли, додека, пак, третиот неограничена количина на кибрит.
  - Агентот има неограничена количина од трите состојки.
- Процесот е следен:
  - Еден агент по случаен избор одбира 2 различни состојки и ги става на маса.
  - Пушачот кој ја има останатата состојка ја прави и пуши цигарата.
  - Откако ќе ја испуши цигарата му сигнализира на агентот дека може да стави нови две состојки на масата.
  - Процесот се повторува отпочеток
- Иницијално масата е празна.

Решение:

```
static Semaphore accessTable;
static Semaphore emptyTable;
static Semaphore wait[];
static boolean waiting[];

public static void init() {
    emptyTable = new Semaphore(1);
    accessTable = new Semaphore(0);
    wait = new Semaphore[3];
    wait[0] = new Semaphore(0);
    wait[1] = new Semaphore(0);
    wait[2] = new Semaphore(0);
    waiting = new boolean[3];
```

Решение:

```
/* Agent */
emptyTable.acquire();
state.putItems();
for (int i = 0; i < 3; i++) {
    if (waiting[i]) {
        waiting[i] = false;
        wait[i].release();
    }
}
accessTable.release();
```

```
/* Smoker */
accessTable.acquire();
if (state.hasMyItems(type)) {
    state.consume(type);
    emptyTable.release();
} else {
    // wait until new items are added
    waiting[type] = true;
    accessTable.release();
    wait[type].acquire();
}
```