Perterson's, & Bakery Algorithm

29 de setembro de 2023 18:37

- Peterson's Algorithm

Algoritmo de exclusão mútua entre dois processos.
Existe um parte de *flags* e uma variável partilhada entre as threads chamada *turn* para coordenar o acesso à *critical section.*

A `flag[i]` é colocada a *true* caso o processo *i* esteja interessado em interferir na *critical section*.

A desvantagem está quando existe apenas uma *thread* interessada na critical section, pois esta ficará à espera para dar o lugar à outra que não existe.
O desempate é decido garantidamente pela *variável turn* - pela ordem em que os processos indicam a sua intenção de entrar.

```
public class PetersonAlgorithm implements Lock {
    3 usages
    private final boolean[] flag = new boolean[2];
    2 usages
    private volatile int turn;

    1 usage  new *
    @Override
    public void lock(int i) {
        int j = 1 - i; // other process
        flag[i] = true; // "I'm interested"
        turn = j; // "it's your turn"
        while (flag[j] && turn == j) { // while the other is intere
            // wait
        }
    }

    1 usage  new *
    @Override
    public void unlock(int i) { flag[i] = false; // "I'm no longer
}
```

- Bakery Algorithm

Number of threads = 3
Array choosing[] = [false, false, false]
Array ticket[] = [0, 0, 0]

Lock thread with id = 0;

choosing[0] = true;
⇒ current thread is interested in getting into the critical section

ticket[0] = findMax() + 1;
⇒ get the next available ticket (finding the current max value is needed

```java
private int findMax() {
    int m = ticket[0];
    for (int i = 1; i < ticket.length; i++)
        if (ticket[i] > m)
            m = ticket[i];
    return m;
}
```

É possível que duas threads escolham o mesmo número de ticket, mas nunca têm o mesmo thread ID

```java
for (int j = 0; j < numberOfThreads; j++) {
    // If the thread j is the current thread go the next thread.
    if (j == id)
        continue;

    // Wait if thread j is choosing right now.
    while (choosing[j]) { /* nothing */ }

    while (ticket[j] != 0 && (ticket[id] > ticket[j] || (ticket[id] == ticket[j] && id > j))) { /* nothing */ }

}
```

Para cada thread (exceto a thread atual):

- esperar caso essa thread esteja interessada na critical section
- esperar caso:
    - o ticket number dessa thread seja diferente de 0,
    - e caso o número de ticket da thread atual que chamou o lock seja maior do que o número de ticket dessa thread
        - ou caso o número de ticket da thread atual que chamou o lock seja igual ao número de ticket dessa thread e o id seja maior do que o dessa thread

```java
@Override
public void unlock(int id) {
    ticket[id] = 0;
}
```

A thread deixa de estar interessada na critical section.

Testing:

```java
Thread[] threads = new Thread[Bakery.numberOfThreads];

for (int i = 0; i < threads.length; i++) {
    threads[i] = new Thread(new Bakery(i));
    threads[i].start();
}
```