Teoria Współbieżności

Laboratorium 2 - Synchronizacja wątków

Magdalena Pastuła

Data laboratorium: 13.10.2020

1. Zadania do wykonania.

Na laboratorium do wykonania były następujące zadania:

1. Implementacja semafora binarnego przy pomocy metod wait i notify oraz użycie go do synchronizacji programu Race.
2. Pokazanie, że do zadania z punktu 1. potrzebne jest użycie pętli while, a użycie warunku if jest niewystarczające.
3. Implementacja semafora licznikowego poprzez semafory binarne. Zastanowienie się, czy semafor binarny jest szczególnym przypadkiem semafora licznikowego.

2. Koncepcja rozwiązania.

Zadanie 1

Zaimplementowany semafor binarny posiada dwie zmienne prywatne: `_stan`, czyli informację, czy semafor został zabrany czy nie, oraz `_waiting` oznaczającą ilość wątków oczekujących na zwolnienie dostępu do zmiennej. Dodatkowo, klasa ta zawiera metody `P` i `V` wykonujące odpowiednio zabranie oraz oddanie semafora.

Zadanie 2

Aby móc zauważyć problem przy użyciu pojedynczego sprawdzania zmiennej `_stan`, stworzono po 50 wątków inkrementujących i dekrementujących zmienną Count, przy czym zmniejszono inkrementację i dekrementację do 1000 kroków z 100 000 000.

Zadanie 3

Zaimplementowany semafor licznikowy, podobnie jak binarny, zawiera zmienną `_stan`, informującą o maksymalnej liczbie wątków z dostępem, zmienną `_waiting`, która zawiera liczbę czekających wątków oraz dodatkowo zmienną `_using`, która zawiera informację, ile wątków aktualnie blokuje dostęp do zasobu. Ponadto klasa ta zawiera tablicę semaforów binarnych u długości zadanej w konstruktorze i równej zmiennej `_stan`.
Aby móc przetestować działanie zaimplementowanego semafora, napisano uproszczony program realizujący problem pięciu filozofów, gdzie jednocześnie maksymalnie tylko czterech z nich może zająć semafor. Każdy z wątków wypisuje na konsolę informację o zajęciu semafora, czeka sekundę, a następnie wypisuje informację o jego zwolnieniu.

3. Implementacja i wyniki.

Zadanie 1

Metody klasy semafor:

```
public synchronized void P() {
    while (!_state) {
        _waiting += 1;
        try {
            wait();
        } catch (Exception ex) {
            System.out.println("error");
        }
    }
    _state = false;
    _waiting -= 1;
}

public synchronized void V() {
    if (_waiting != 0) {
        notify();
    }
    _state = true;
}
```

Cały kod znajduje się w folderze `ex1` w załączonym zipie.

Zadanie 2

Fragment metody main klasy Race udowadniający konieczność ciągłego sprawdzania zmiennej `_stan` przy oczekiwaniu na dostęp:

```
IThread it[] = new IThread[50];
DThread dt[] = new DThread[50];

for (int i=0;i<50;i++) {
    it[i] = new IThread(cnt, sem);
    dt[i] = new DThread(cnt, sem);
    it[i].start();
    dt[i].start();
}

try {
    for (int i=0;i<50;i++) {
        it[i].join();
        dt[i].join();
    }
} catch(InterruptedException ie) { }
```

Wartość zmiennej Counter po wykonaniu wszystkich wątków wyniosła 4.

Cały kod znajduje się w folderze `ex2` w załączonym zipie.

Zadanie 3

Metody zaimplemetowanego semafora licznikowego na bazie semafora binarnego:

```
public SemaforLicz (int number) {
    _state = number;
    _sems = new Semafor[number];

    for (int i=0;i<number;i++) {
        _sems[i] = new Semafor();
    }
}

public synchronized void P() {
    while (_using >= _state) {
        _waiting += 1;
        try {
            wait();
        } catch (Exception ex) {
            System.out.println("error");
        }
    }
    _sems[_using].P();
    _using += 1;
}

public synchronized void V() {
    if (_waiting != 0) {
        _waiting -= 1;
        notify();
    }
    _using -= 1;
    _sems[_using].V();
}
```

Wynikiem programu testującego zaimplementowany semafor liczący jest następujący zapis w konsoli:

```
Semaphore taken.
Semaphore taken.
Semaphore taken.
Semaphore taken.
Semaphore returned.
Semaphore returned.
Semaphore taken.
Semaphore returned.
Semaphore returned.
Semaphore returned.
```

Cały kod do tego zadania znajduje się w folderze `ex3` w załączonym zipie.

4. Wnioski z ćwiczenia.

Wnioski wyciągnięte z poszczególnych zadań:

Zadanie 1

Zaimplementowany semafor binarny poprawnie chroni dostęp do zmiennej Counter.

Zadanie 2

W przypadku wykorzystania funkcji `wait()` należy używać pętli while zamiast instrukcji warunkowej if, ponieważ wywołanie `wait()` jest równoważne z wywołaniem funkcji `wait(0,0)`, co oznacza, że czas oczekiwania na otrzymanie informacji o przerwaniu oczekiwania jest równy 0. Oznacza to, że w przypadku nie otrzymania informacji o przerwaniu oczekiwania praktycznie od razu oczekiwanie zostanie przerwane. Dodatkowo, wątek może wyjść ze stanu oczekiwania również z powodu przerwania lub wyjątku.

Zadanie 3

Semafor binarny nie jest szczególnym przypadkiem semafora licznikowego. Wynika to z faktu, że semafor liczący zlicza ilość wątków zajmujący dany zasób, natomiast semafor binarny zawiera tylko informację, czy dany zasób jest zajęty, czy też nie.

Po wypisanych napisach w konsoli programu testującego wyraźnie widać, że jeden wątek czeka na zwolnienie semafora przez którykolwiek inny wątek. Oznacza to, że napisana implementacja semafora liczącego spełnia swoje zadanie.

5. Bibliografia.

1. Dokumentacja funkcji wait() w Javie
2. Dokumentacja funkcji wait(long, int) w Javie
3. Z. Weiss, T. Gruźlewski, "Programowanie współbieżne i rozproszone w przykładach i zadaniach", WNT