Programowanie współbieżne i rozproszone

mgr inż. Marcin Mrukowicz

Instrukcja laboratoryjna VII

Klasyczne problemy synchronizacji i ich rozwiązania w języku Java

2. Problem czytelników i pisarzy (ang. readers–writers problems)

W problemie tym występuje podział procesów (wątków) na dwie klasy: czytelników, którzy jedynie odczytują wartość jakiegoś zasobu oraz pisarzy, którzy poza tym, że odczytują wartość, to również ją modyfikują. Sytuacja ta często występuje np. podczas korzystania z pliku, albo bazy danych przez wiele procesów (wątków) jednocześnie. Zwykle dąży się do sytuacji, w której odczyt zasobu jest zablokowany jedynie w przypadku, gdy jest on aktualnie modyfikowany. Oznacza to, że czytelnicy nie powinni blokować się wzajemnie, natomiast czytelnicy powinni blokować pisarzy. Pisarz powinien blokować innych pisarzy i wszystkich czytelników.

1. Przepisz, następującą klasę w języku Java:

```
public class ReaderWriterLock {

    public static void main(String[] args) {
        String lineSep = System.lineSeparator();
        File f = new File("temp.txt");
        ReentrantLock lock = new ReentrantLock(true);
        Condition fileExist = lock.newCondition();

        class Reader extends Thread {
            String name;

            public Reader(String name) {
                this.name = name;
            }

            @Override
            public void run() {
                while (lock.isLocked());
                lock.lock();

                try {
                    if (!f.exists()) {
                        fileExist.await();
                    }
                    FileReader fileReader = new FileReader(f);
                    String fileContent;
                    Scanner scanner = new Scanner(fileReader);
                    StringBuilder sb = new StringBuilder();
                    while (scanner.hasNextLine()) {
                        sb.append(scanner.nextLine());
                        sb.append(lineSep);
                    }
                    fileContent = sb.toString();
                    fileReader.close();

                    System.out.println("Reader " + name + " reads: ");
                    System.out.print("\u001B[32m");
                    System.out.print(fileContent);
                    System.out.print("\u001B[0m");

                } catch (InterruptedException | IOException e) {
                    e.printStackTrace();
                } finally {
                    lock.unlock();
```

```
                }
            }
        }

        class Writer extends Thread {
            String name;

            public Writer(String name) {
                this.name = name;
            }

            @Override
            public void run() {
                lock.lock();
                boolean fileExst = f.exists();
                try {
                    Random r = new Random();
                    String[] words = {"aaa", "bbb", "ccc", "ddd", "eee", "fff"};
                    String s = words[r.nextInt(words.length)];
                    String fileContent = "";
                    if (fileExst) {
                        FileReader fileReader = new FileReader(f);
                        Scanner scanner = new Scanner(fileReader);
                        StringBuilder sb = new StringBuilder();
                        while (scanner.hasNextLine()) {
                            sb.append(scanner.nextLine());
                            sb.append(lineSep);
                        }
                        fileContent = sb.toString();
                    }
                    FileWriter fileWriter = new FileWriter(f);

                    fileWriter.write(fileContent + "Writer " + name + " writes: " + s + lineSep);
                    System.out.print("\u001B[34m");
                    System.out.println("Writer " + name + " reads: ");
                    System.out.print(fileContent);
                    System.out.print("\u001B[0m");

                    System.out.print("\u001b[31m");
                    System.out.println("Writer " + name + " appends: " + s);
                    System.out.print("\u001B[0m");
                    fileWriter.close();
                    if (!fileExst) {
                        System.out.println("Number of readers waiting for create file: " +
                                lock.getWaitQueueLength(fileExist));
                        fileExist.signalAll();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                } finally {
                    lock.unlock();
                }
            }
        }

        new Reader("I").start();
        new Reader("II").start();
        new Writer("I").start();
        new Reader("III").start();
        new Reader("IV").start();
        new Reader("V").start();
        new Reader("VI").start();
        new Writer("II").start();

    }
}
```

klasa *ReaderWriterLock.java*

W powyższej klasie występuje synchronizacja dostępu do pliku przez wiele wątków, jednak klasa rozwiązuje problem czytelników i pisarzy w taki sposób, że zachowana zostaje spójność danych (nie ma możliwości, aby pisarz rozpoczął modyfikację pliku, gdy czytelnik go odczytuje), ale czytelnicy blokują się nawzajem. Rozwiązanie to zatem nie jest optymalne, a nawet może być uważane za łamiące założenia wstępne dla problemu. Co ważne, jeżeli plik jeszcze nie istnieje,

czytelnicy poczekają, aż któryś pisarz zdąży utworzyć ten plik i wyśle powiadomienie do wszystkich czytelników (czytelnicy mogą odczytywać plik niezależnie od siebie, zatem sensowne jest wybudzenie wszystkich wątków czytelników, za pomocą metody *signalAll*). Również pisarze blokują się tutaj nawzajem, co akurat jest pożądane (nie może nastąpić *race condition* pomiędzy pisarzami).

- **przenieś instrukcję `boolean fileExst = f.exists();` w procesie pisarza powyżej wywołania metody `lock.lock();` Usuń plik *temp.txt*, jeżeli istnieje (i usuwaj go po każdym uruchomieniu), uruchom klasę co najmniej 10 razy, co się stało?**
- **Czy potrafisz wyjaśnić, w jaki sposób czytelnicy są blokowani przez pisarzy?**

Przepisz poniższą klasę:

```
public class ReaderWriterOptimized {
    public static void main(String[] args) throws InterruptedException {
        String lineSep = System.lineSeparator();
        File f = new File("temp.txt");
        ReentrantLock writing = new ReentrantLock(true);
        Condition fileExist = writing.newCondition();
        Semaphore reading = new Semaphore(10);

        class Reader extends Thread {
            String name;

            public Reader(String name) {
                this.name = name;
            }

            @Override
            public void run() {
                boolean printOnce = true;
                while (writing.isLocked()) {
                    if (printOnce) {
                        System.out.println("Some writer is writing, i have to wait!");
                        printOnce = false;
                    }
                }

                try {
                    reading.acquire();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                try {
                    if (!f.exists()) {
                        writing.lock();
                        reading.release();
                        fileExist.await();
                        writing.unlock();
                        reading.acquire();
                    }
                    FileReader fileReader = new FileReader(f);
                    String fileContent = "";
                    Scanner scanner = new Scanner(fileReader);
                    StringBuilder sb = new StringBuilder();
                    while (scanner.hasNextLine()) {
                        sb.append(scanner.nextLine());
                        sb.append(lineSep);
                    }
                    fileContent = sb.toString();
                    fileReader.close();

                    System.out.println("Reader " + name + " reads: ");
                    System.out.print("\u001B[32m");
                    System.out.print(fileContent);
                    System.out.print("\u001B[0m");

                } catch (InterruptedException | IOException e) {
                    e.printStackTrace();
                } finally {
                    reading.release();
                }
            }
```

```
    }

    class Writer extends Thread {
        String name;

        public Writer(String name) {
            this.name = name;
        }

        @Override
        public void run() {
            boolean printOnce = true;
            while (reading.availablePermits() < 10) {
                if (printOnce) {
                    System.out.println("Some reader is reading now, i have to wait");
                    printOnce = false;
                }
            }

            boolean fileExst = f.exists();
            writing.lock();
            try {
                Random r = new Random();
                String[] words = {"aaa", "bbb", "ccc", "ddd", "eee", "fff"};
                String s = words[r.nextInt(words.length)];
                String fileContent = "";
                if (fileExst) {
                    FileReader fileReader = new FileReader(f);
                    Scanner scanner = new Scanner(fileReader);
                    StringBuilder sb = new StringBuilder();
                    while (scanner.hasNextLine()) {
                        sb.append(scanner.nextLine());
                        sb.append(lineSep);
                    }
                    fileContent = sb.toString();
                }
                FileWriter fileWriter = new FileWriter(f);

                fileWriter.write(fileContent + "Writer " + name + " writes: " + s + lineSep);
                System.out.print("\u001B[34m");
                System.out.println("Writer " + name + " reads: ");
                System.out.print(fileContent);
                System.out.print("\u001B[0m");

                System.out.print("\u001b[31m");
                System.out.println("Writer " + name + " appends: " + s);
                System.out.print("\u001B[0m");
                fileWriter.close();
                if (!fileExst) {
                    fileExist.signalAll();
                }

            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                writing.unlock();
            }
        }
    }

    new Writer("I").start();
    new Reader("I").start();
    new Reader("II").start();
    new Writer("II").start();

    new Reader("III").start();
    new Reader("IV").start();
    new Reader("V").start();
    new Reader("VI").start();

    int maxIterationsNo = 100000;
    int i = 0;
    while (i < maxIterationsNo) {
        Thread.sleep(1000);
        new Writer("I").start();
        new Writer("II").start();
        new Writer("III").start();
```

```
            new Reader("I").start();
            new Reader("II").start();
            new Reader("III").start();
            new Reader("IV").start();
            new Reader("V").start();
            new Reader("VI").start();
            i++;
        }
    }
}
```

klasa ReaderWriterOptimized.java

Powyższa klasa rozwiązuje problem czytelników i pisarzy w następujący sposób: z jednej strony występuje blokada *writing*, która uniemożliwia pisarzom jednoczesną modyfikację pliku oraz sprawia, że w momencie w którym któryś pisarz pisze, czytelnicy muszą poczekać na zakończenie tego procesu; z drugiej strony czytelnicy **nigdy** nie używają jej wprost (jedynie sprawdzają, czy pisarz nie pisze), zatem nie blokują się wzajemnie. Jednakże czytelnicy **korzystają z semafora reading**, który pozwoli maksymalnie 10 czytelnikom na jednoczesne odczytywanie pliku; równocześnie pisarz sprawdza, czy jacyś czytelnicy nie odczytują aktualnie pliku i powstrzymuje się przed jego modyfikacją, dopóki wszyscy czytelnicy nie zakończą czytania. To rozwiązanie można uznać za poprawne i zgodne z założeniami początkowymi problemu.

- **uruchom powyższą klasę jeden raz i obserwuj jej działanie przez około minutę. Wypowiedz się, czy obserwacja jej działania spełnia wszystkie założenia poprawnego rozwiązania problemu czytelników i pisarzy. Możesz wprowadzić dłuższy czas uśpienia wątku głównego dla ułatwienia analizy.**
- **Zauważ, że jak na razie liczba czytelników, mogących jednocześnie odczytywać plik nie przekracza 10 (ustalonego arbitralnie limitu). Zwiększ ten limit do 100 i zmodyfikuj pętle while w wątku głównym, tak aby utworzyć jednocześnie 1000 czytelników (oczywiście za pomocą kolejnej pętli, nie przejmuj się ich nazwami, możesz użyć jednej). Co się stało?**

Plik temp.txt jest zapisywany naprawdę na dysku, zatem warto usunąć go po zakończonej pracy. Ograniczono co prawda ilość iteracji programu, ale plik może zająć stosunkowo dużo miejsca na dysku.