Προηγμένες Τεχνικές Προγραμματισμού
Producer Consumer using Monitor

Δασούλας Ιωάννης – 1053711

Δομή Προγράμματος

Η άσκηση αφορά τη λύση του προβλήματος producer – consumer χρησιμοποιώντας synchronized μεθόδους (υλοποίηση Monitor). Η δομή του monitor ελέγχει την πρόσβαση των νημάτων στον κοινό πόρο (στην προκειμένη περίπτωση, στον αποθηκευτικό χώρο) βάσει κάποιων κριτηρίων που τίθενται (στην προκειμένη περίπτωση το αν ο χώρος είναι γεμάτος ή άδειος. Ο αποθηκευτικός χώρος αναπαρίσταται από την κλάση Buffer.

```
public class Buffer {
    private int value = 50;
    private boolean empty = true;

    public synchronized void put(int val) {
        while(!empty) {
            try {
                wait();
            }
            catch (InterruptedException e) {
                System.err.println("Exception:" + e.toString());
            }
        }
        value = val;
        empty = false;
        notifyAll();
    }

    public synchronized int get() {
        int value;
        while(empty) {
            try {
                wait();
            }
            catch (InterruptedException e) {
                System.err.println("Exception:" + e.toString());
            }
        }

        value = this.value;
        empty = true;
        notifyAll();
        return value;
    }
}
```

Η μέθοδος put (int val) τοποθετεί το στοιχείο val στον αποθηκευτικό χώρο και η μέθοδος get () το επιστρέφει σε μορφή int ξανά. Ορίζοντας τις μεθόδους ως synchronized εξασφαλίζεται ότι ένα νήμα θα μπορεί να έχει πρόσβαση κάθε φορά. Με τη μέθοδο wait() το νήμα το νήμα «περιμένει» όσο η συνθήκη που έχει τεθεί δεν ικανοποιείται και με την notify()/notifyAll() ειδοποιείται ότι μπορεί να συνεχίσει. Η συνθήκη τίθεται με την Boolean μεταβλητή empty που είναι αληθής όταν έχει καταναλωθεί το στοιχείο του αποθηκευτικού χώρου και ψευδής όταν έχει παραχθεί ένα στοιχείο.

Με τις synchronized εξασφαλίζεται ο αμοιβαίος αποκλεισμός και δεν υπάρχει κίνδυνος για παράνομο interleaving οπότε και αφαιρέθηκε η μέθοδος get_string() που είχε υλοποιηθεί στα προηγούμενα εργαστήρια καθώς και η μέθοδος μέτρησης παράνομων interleaving που υλοποιήθηκε σε αυτά.

Στις κλάσεις Producer και Consumer υλοποιούνται τα νήματα του παραγωγού και του καταναλωτή, αντιστοίχως.

```
public class Producer extends Thread{
    private Buffer buffer;
    Random random = new Random();
    public int rand;
    public String pos;
    private int threadNumber;

    static int reps = 10;

    public Producer(Buffer buffer, int threadNumber){
        this.buffer = buffer;
        this.threadNumber = threadNumber;
    }

    public void run() {
        for(int i = 1; i<reps; i++) {
            try {
                Thread.sleep(1000);
            }
            catch(InterruptedException e) {
                System.err.println(e.toString());
            }

            System.out.println("Ο παραγωγός "+ threadNumber + " παρήγαγε το αγαθό " + i);
            buffer.put(i);

        }
    }
}
```

```java
public class Consumer extends Thread{
    private Buffer buffer;

    private int val;
    private int threadNumber;
    static int reps = 10;

    public Consumer(Buffer buffer, int threadNumber){
        this.buffer = buffer;
        this.threadNumber = threadNumber;
    }

    public void run() {
        for (int i = 1; i < reps ; i++) {
            try {
                Thread.sleep(1000);
            }
            catch(InterruptedException e) {
                System.err.println(e.toString());
            }

            val = buffer.get();
            System.out.println("Ο καταναλωτής " + threadNumber + " κατανάλωσε το αγαθό " + val);

        }
    }
}
```

Στον δημιουργό των κλάσεων περνιούνται ως ορίσματα ένα αντικείμενο τύπου Buffer (ο κοινός πόρος) και ο αριθμός του νήματος. Με την μεταβλητή reps ορίζονται οι φορές που θα γίνει put ή get αντιστοίχως, μετά το sleep που υπάρχει για την εναλλαγή των νημάτων. Στο τέλος τυπώνεται κάθε φορά το αποτέλεσμα.

1 Producer & 1 Consumer

```java
public class ProducerConsumer {
    public static void main(String[] args){
        Buffer buffer = new Buffer();

        Producer p1 = new Producer(buffer, 1);
        Consumer c1 = new Consumer(buffer, 1);
        //Producer p2 = new Producer(buffer, 2);
        //Consumer c2 = new Consumer(buffer, 2);

        p1.start();
        c1.start();
        //p2.start();
        //c2.start();
    }

}
```

Ο παραγωγός και ο καταναλωτής εναλλάσσονται στην πρόσβαση του αποθηκευτικού χώρου. Βασική προϋπόθεση για να λειτουργήσει και να τερματιστεί ομαλά το πρόγραμμα είναι να έχουν τα δύο νήματα ίδιο αριθμό επαναλήψεων (p1.reps = c1.reps). Αλλιώς καταλήγει σε deadlock.

Αποτέλεσμα:

```
0 παραγωγός 1 παρήγαγε το αγαθό 1
0 καταναλωτής 1 κατανάλωσε το αγαθό 1
0 παραγωγός 1 παρήγαγε το αγαθό 2
0 καταναλωτής 1 κατανάλωσε το αγαθό 2
0 παραγωγός 1 παρήγαγε το αγαθό 3
0 καταναλωτής 1 κατανάλωσε το αγαθό 3
0 παραγωγός 1 παρήγαγε το αγαθό 4
0 καταναλωτής 1 κατανάλωσε το αγαθό 4
0 παραγωγός 1 παρήγαγε το αγαθό 5
0 καταναλωτής 1 κατανάλωσε το αγαθό 5
0 παραγωγός 1 παρήγαγε το αγαθό 6
0 καταναλωτής 1 κατανάλωσε το αγαθό 6
0 παραγωγός 1 παρήγαγε το αγαθό 7
0 καταναλωτής 1 κατανάλωσε το αγαθό 7
0 παραγωγός 1 παρήγαγε το αγαθό 8
0 καταναλωτής 1 κατανάλωσε το αγαθό 8
0 παραγωγός 1 παρήγαγε το αγαθό 9
0 καταναλωτής 1 κατανάλωσε το αγαθό 9
```

Η παραγωγή και η κατανάλωση, καθώς και η εκτύπωση εκτελείται ομαλά και με σωστή σειρά.

2 Producers & 2 Consumers

```java
public class ProducerConsumer {
    public static void main(String[] args){
        Buffer buffer = new Buffer();

        Producer p1 = new Producer(buffer, 1);
        Consumer c1 = new Consumer(buffer, 1);
        Producer p2 = new Producer(buffer, 2);
        Consumer c2 = new Consumer(buffer, 2);

        p1.start();
        c1.start();
        p2.start();
        c2.start();
    }

}
```

Οι παραγωγοί και ο καταναλωτές εναλλάσσονται στην πρόσβαση του αποθηκευτικού χώρου. Κι εδώ, βασική προϋπόθεση για να λειτουργήσει και να τερματιστεί ομαλά το πρόγραμμα είναι να έχουν τα δύο νήματα ίδιο αριθμό επαναλήψεων (p1.reps = c1.reps). Αλλιώς καταλήγει σε deadlock.

Αποτέλεσμα:

```
0 παραγωγός 1 παρήγαγε το αγαθό 1
0 παραγωγός 2 παρήγαγε το αγαθό 1
0 καταναλωτής 1 κατανάλωσε το αγαθό 1
0 καταναλωτής 2 κατανάλωσε το αγαθό 1
0 παραγωγός 2 παρήγαγε το αγαθό 2
0 παραγωγός 1 παρήγαγε το αγαθό 2
0 καταναλωτής 1 κατανάλωσε το αγαθό 2
0 καταναλωτής 2 κατανάλωσε το αγαθό 2
0 παραγωγός 1 παρήγαγε το αγαθό 3
0 παραγωγός 2 παρήγαγε το αγαθό 3
0 καταναλωτής 1 κατανάλωσε το αγαθό 3
0 καταναλωτής 2 κατανάλωσε το αγαθό 3
0 παραγωγός 1 παρήγαγε το αγαθό 4
0 παραγωγός 2 παρήγαγε το αγαθό 4
0 καταναλωτής 1 κατανάλωσε το αγαθό 4
0 καταναλωτής 2 κατανάλωσε το αγαθό 4
0 παραγωγός 2 παρήγαγε το αγαθό 5
0 παραγωγός 1 παρήγαγε το αγαθό 5
0 καταναλωτής 1 κατανάλωσε το αγαθό 5
0 καταναλωτής 2 κατανάλωσε το αγαθό 5
0 παραγωγός 2 παρήγαγε το αγαθό 6
0 παραγωγός 1 παρήγαγε το αγαθό 6
0 καταναλωτής 1 κατανάλωσε το αγαθό 6
0 καταναλωτής 2 κατανάλωσε το αγαθό 6
0 παραγωγός 1 παρήγαγε το αγαθό 7
0 καταναλωτής 1 κατανάλωσε το αγαθό 7
0 παραγωγός 2 παρήγαγε το αγαθό 7
0 καταναλωτής 2 κατανάλωσε το αγαθό 7
0 παραγωγός 1 παρήγαγε το αγαθό 8
0 παραγωγός 2 παρήγαγε το αγαθό 8
0 καταναλωτής 1 κατανάλωσε το αγαθό 8
0 καταναλωτής 2 κατανάλωσε το αγαθό 8
```

Η παραγωγή και η κατανάλωση, καθώς και η εκτύπωση εκτελείται ομαλά και με σωστή σειρά.

2 Producers & 1 Consumer or 1 Producer & 2 Consumers

```
public class ProducerConsumer {
    public static void main(String[] args){
        Buffer buffer = new Buffer();

        Producer p1 = new Producer(buffer, 1);
        Consumer c1 = new Consumer(buffer, 1);
        Producer p2 = new Producer(buffer, 2);
        //Consumer c2 = new Consumer(buffer, 2);

        p1.start();
        p2.start();
        c1.start();
        //c2.start();
    }

}
```

Τα νήματα εναλλάσσονται στην πρόσβαση του αποθηκευτικού χώρου. Βασική προϋπόθεση για να λειτουργήσει και να τερματιστεί ομαλά το πρόγραμμα είναι η κλάση που υλοποιείται μόνο από ένα νήμα, να έχει διπλάσιο αριθμό επαναλήψεων (c1.reps = 2*p1.reps ή p1.reps = 2*c1.reps). Αλλιώς καταλήγει σε deadlock. Έστω ότι υπάρχουν δύο παραγωγοί και δύο καταναλωτές.

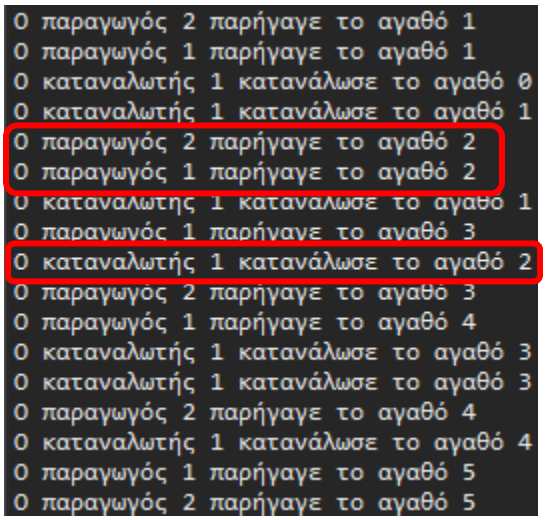
Αποτέλεσμα:

```
0 παραγωγός 2 παρήγαγε το αγαθό 0
0 παραγωγός 1 παρήγαγε το αγαθό 0
0 καταναλωτής 1 κατανάλωσε το αγαθό 0
0 παραγωγός 2 παρήγαγε το αγαθό 1
0 παραγωγός 1 παρήγαγε το αγαθό 1
0 καταναλωτής 1 κατανάλωσε το αγαθό 0
0 καταναλωτής 1 κατανάλωσε το αγαθό 1
0 παραγωγός 1 παρήγαγε το αγαθό 2
0 παραγωγός 2 παρήγαγε το αγαθό 2
0 καταναλωτής 1 κατανάλωσε το αγαθό 1
0 παραγωγός 2 παρήγαγε το αγαθό 3
0 καταναλωτής 1 κατανάλωσε το αγαθό 2
0 παραγωγός 1 παρήγαγε το αγαθό 3
0 καταναλωτής 1 κατανάλωσε το αγαθό 2
0 καταναλωτής 1 κατανάλωσε το αγαθό 3
0 παραγωγός 1 παρήγαγε το αγαθό 4
0 καταναλωτής 1 κατανάλωσε το αγαθό 3
0 παραγωγός 2 παρήγαγε το αγαθό 4
0 καταναλωτής 1 κατανάλωσε το αγαθό 4
0 παραγωγός 1 παρήγαγε το αγαθό 5
0 καταναλωτής 1 κατανάλωσε το αγαθό 4
0 παραγωγός 2 παρήγαγε το αγαθό 5
0 καταναλωτής 1 κατανάλωσε το αγαθό 5
0 καταναλωτής 1 κατανάλωσε το αγαθό 5
```

Η παραγωγή και η κατανάλωση γίνονται σωστά, αλλά η εκτύπωση δεν γίνεται με τη σωστή σειρά. Αυτό συμβαίνει λόγω interleavings που γίνονται πριν ή μετά τις εκτυπώσεις, όταν ο αριθμός των νημάτων του Producer δεν είναι ίδιος με αυτόν του Consumer, από την στιγμή που ο αποθηκευτικός χώρος της άσκησης έχει χωρητικότητα ενός στοιχείου. Για να λυθεί αυτό το πρόβλημα, μπορεί να γίνεται η εκτύπωση μέσα στις synchronized μεθόδους της κλάσεις Buffer και τότε η σειρά θα είναι σωστή, όμως με αυτόν τον τρόπο μεγαλώνει ο κρίσιμος τομέα του προγράμματος για μία λειτουργία που δεν είναι αναγκαία να γίνει. Για αυτό και αποφεύγεται!

If(condition) vs While(condition)

Αν χρησιμοποιείται το if block, αντί του while block, τότε δεν ελέγχεται η περίπτωση ξανά πριν μία παραγωγή ή κατανάλωση στοιχείου. Έτσι, ενδέχεται να υπάρξει overwrite, δηλαδή να παραχθούν δύο στοιχεία συνεχόμενα προτού το πρώτο καταναλωθεί, όπως φαίνεται στο παρακάτω αποτέλεσμα έπειτα από χρήση if block:



Το αγαθό 2 παράχθηκε από τους δύο παραγωγούς, αλλά ο καταναλωτής το κατανάλωσε μία μόνο φορά και δεν πρόλαβε δεύτερη καθώς έγινε overwrite.

Notify() vs NotifyAll()

Έστω μία υλοποίηση με 2 consumers. Αν ο ένας όταν τελειώσει καλεί notify() αντί για notifyAll(), τότε ενδέχεται να «ξυπνήσει» τον δεύτερο consumer. Αυτός με τη σειρά του, θα μπει στον while έλεγχο, ο οποίος θα είναι αληθής, αφού μόλις έγινε κατανάλωση, οπότε το νήμα θα κάνει για αόριστο χρόνο wait() και θα δημιουργηθεί αλγοριθμικό αδιέξοδο (deadlock). Αντίθετα, με την notifyAll(), θα αφυπνιστεί και ο producer, οπότε ακόμα κι αν μπει σε wait() ο δεύτερος consumer, κάποια στιγμή ο producer θα τον «ξυπνήσει» και θα συνεχιστεί κανονικά η ροή του προγράμματος. Το αποτέλεσμα της χρήσης του notify() σε υλοποίηση με 2 producers και 2 consumers:

```
ProducerConsumer (2) [Java Application] C:\Program Files\Java\jre
Ο παραγωγός 2 παρήγαγε το αγαθό 25
Ο καταναλωτής 1 κατανάλωσε το αγαθό 25
Ο καταναλωτής 2 κατανάλωσε το αγαθό 25
Ο παραγωγός 1 παρήγαγε το αγαθό 26
Ο καταναλωτής 1 κατανάλωσε το αγαθό 26
Ο παραγωγός 2 παρήγαγε το αγαθό 26
Ο καταναλωτής 2 κατανάλωσε το αγαθό 26
Ο παραγωγός 2 παρήγαγε το αγαθό 27
Ο παραγωγός 1 παρήγαγε το αγαθό 27
Ο καταναλωτής 2 κατανάλωσε το αγαθό 27
Ο καταναλωτής 1 κατανάλωσε το αγαθό 27
Ο παραγωγός 2 παρήγαγε το αγαθό 28
Ο παραγωγός 1 παρήγαγε το αγαθό 28
Ο καταναλωτής 1 κατανάλωσε το αγαθό 28
Ο καταναλωτής 2 κατανάλωσε το αγαθό 28
Ο παραγωγός 1 παρήγαγε το αγαθό 29
Ο παραγωγός 2 παρήγαγε το αγαθό 29
Ο καταναλωτής 1 κατανάλωσε το αγαθό 29
Ο καταναλωτής 2 κατανάλωσε το αγαθό 29
Ο παραγωγός 2 παρήγαγε το αγαθό 30
Ο παραγωγός 1 παρήγαγε το αγαθό 30
Ο καταναλωτής 1 κατανάλωσε το αγαθό 30
Ο καταναλωτής 2 κατανάλωσε το αγαθό 30
Ο παραγωγός 2 παρήγαγε το αγαθό 31
Ο καταναλωτής 1 κατανάλωσε το αγαθό 31
Ο παραγωγός 1 παρήγαγε το αγαθό 31
Ο καταναλωτής 2 κατανάλωσε το αγαθό 31
Ο παραγωγός 2 παρήγαγε το αγαθό 32
Ο παραγωγός 1 παρήγαγε το αγαθό 32
Ο καταναλωτής 1 κατανάλωσε το αγαθό 32
Ο παραγωγός 2 παρήγαγε το αγαθό 33
Ο παραγωγός 2 παρήγαγε το αγαθό 34
```

Για το πρόγραμμα είχε οριστεί να παραχθούν οι αριθμοί από το 0 έως το 100 αλλά λόγω του deadlock, η λύση σταμάτησε στο 34, χωρίς να τερματιστεί το πρόγραμμα.

Σχόλια

Συνοψίζοντας, μέσα από τα διάφορα παραδείγματα γίνεται εμφανής η επιτυχία του Monitor όσον αφορά τον αμοιβαίο αποκλεισμό (object locks) και την αρμονική συνύπαρξη των νημάτων (wait() & notify()).

Επίσης, μέσα από τα παραδείγματα με την χρήση του if block και του notify(), αντίστοιχα, φαίνεται και πόσο σημαντικός είναι ο τρόπος που ο προγραμματιστής συγχρονίζει τα νήματα, αλλά και το πόσο εύκολα ενδέχεται να οδηγηθεί σε λάθος υλοποιήσεις που φαινομενικά μπορεί να μοιάζουν σωστές.

UML Diagram