Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
http://courses.softlab.ntua.gr/pl2/

Γλώσσες Προγραμματισμού II

Οι ασκήσεις πρέπει να παραδοθούν στους διδάσκοντες σε ηλεκτρονική μορφή μέσω του συνεργατικού συστήματος ηλεκτρονικής μάθησης moodle.softlab.ntua.gr. Η προθεσμία παράδοσης θα τηρείται αυστηρά. Έχετε δικαίωμα να καθυστερήσετε το πολύ μία άσκηση.

Άσκηση 9 Ταυτοχρονισμός στη Java

Προθεσμία παράδοσης: 19/2/2008

Ένας πεπερασμένος καταχωρητής (bounded buffer) είναι μια FIFO δομή δεδομένων. Έστω η παρακάτω υλοποίηση σε Java ενός πεπερασμένου καταχωρητή που μπορεί να προσπελαστεί από πολλά νήματα.

```
class BoundedBuffer {
  // designed for multiple producer threads and multiple consumer threads
  protected int numSlots = 0;
  protected int[] buffer = null;
  protected int putIn = 0, takeOut = 0;
  protected int count = 0;

  public BoundedBuffer(int numSlots) {
    if (numSlots <= 0)
      throw new IllegalArgumentException("numSlots <= 0");
    this.numSlots = numSlots;
    buffer = new int[numSlots];
  }

  public synchronized void put(int value) throws InterruptedException {
    while (count == numSlots) wait();
    buffer[putIn] = value;
    putIn = (putIn + 1) % numSlots;
    count++;
    notifyAll();
  }

  public synchronized int get() throws InterruptedException {
    int value;
    while (count == 0) wait();
    value = buffer[takeOut];
    takeOut = (takeOut + 1) % numSlots;
    count--;
    notifyAll();
    return value;
  }
}
```

(a) Για ποιό λόγο υπάρχει η εντολή `while (count == numSlots) wait()` στη μέθοδο `put`?

(b) Τα συγχρονισμένα αντικείμενα (synchronized objects) της Java έχουν σημασιολογία wait/notify η οποία πρωτοχρησιμοποιήθηκε στη γλώσσα Mesa.[1] Η σημασιολογία της Mesa είναι τέτοια που δεν εγγυάται ότι μετά από μια κλήση της `notify()` ή της `notifyAll()` κάποιο νήμα που εκείνη τη στιγμή περιμένει θα εκτελεστεί αμέσως. Η σημασιολογία είναι ότι η εκτέλεση της `notify` απλώς σημασιοδοτεί ότι κάποιο νήμα που εκτέλεσε κάποιο `wait()` στο συγκεκριμένο συγχρονισμένο αντικείμενο θα ξαναρχίσει την εκτέλεσή του σε κάποιο χρονικό σημείο στο μέλλον. Αντιθέτως, η σημασιολογία του Hoare για ταυτόχρονη εκτέλεση εγγυάται ότι ένα από τα νήματα που περιμένουν για μια συνθήκη θα τρέξει αμέσως μόλις κάποιο άλλο νήμα σημασιοδοτήσει τη συγκεκριμένη συνθήκη.

Με δεδομένη τη σημασιολογία της Mesa, γιατί πρέπει η κλήση της `wait()` στην `put` να βρίσκεται μέσα σε ένα `while` loop; Τι λάθος μπορεί να συμβεί αν αντικαταστήσουμε το βρόχο με την εντολή ελέγχου: `if (count == numSlots) wait();`

(c) Τι κάνει η κλήση της `notifyAll()` στον παραπάνω κώδικα; Μπορεί να αντικατασταθεί από μια κλήση της `notify()`; Εξηγείστε με συντομία.

(d) Περιγράψτε κάποιο σενάριο στο οποίο ο καταχωρητής δε θα δουλέψει σωστά εάν αφαιρέσουμε όλον τον κώδικα συγχρονισμού από την `put`.

(e) Έστω ότι ένας προγραμματιστής θελήσει να τροποποιήσει την υλοποίηση έτσι ώστε κάποιο νήμα να μπορεί να καλεί την `put` την ίδια στιγμή που ένα άλλο νήμα καλεί (ή εκτελεί) την `get`. Αυτό δημιουργεί πρόβλημα σε κάποιες περιπτώσεις αλλά δουλεύει μια χαρά σε κάποιες άλλες. Υποθέστε ότι κάποιου είδους κλείδωμα (locking) μπορεί να γίνει τη στιγμή της εισόδου στην `put` και `get` το οποίο μπορεί να εγγυηθεί ότι οι δύο αυτές μέθοδοι μπορούν να τρέξουν ταυτόχρονα. Μπορείτε επίσης να υποθέσετε ότι η αύξηση και η μείωση κατά ένα μιας ακέραιας μεταβλητής είναι αδιαίρετη (atomic) λειτουργία και ότι μόνο μία κλήση της `put` και μία της `get` θα τρέχουν ταυτόχρονα. Τι είδους έλεγχος (με βάση τις `putIn` και `takeOut`) μπορεί να χρησιμοποιηθεί για να αποφασίσουμε κατά πόσο η `put` και η `get` επιτρέπεται να εκτελεστούν ταυτόχρονα;

[1] Η γλώσσα Mesa αναπτύχθηκε στα τέλη της δεκαετίας του 1970 στο ερευνητικό κέντρο Xerox PARC.