Αναφορά για το Πρόγραμμα “Το Πρόβλημα των Καπνιστών”

**1. Εισαγωγή**

Το πρόγραμμα αυτό υλοποιεί το κλασικό πρόβλημα συγχρονισμού γνωστό ως “Πρόβλημα των Καπνιστών” (Smokers’ Problem), προσαρμοσμένο σε τρεις πελάτες (Clients) που χρειάζονται τρεις διαφορετικούς πόρους. Στην πρωτότυπη εκδοχή, οι καπνιστές χρειάζονταν καπνό, χαρτί και σπίρτα· εδώ όμως χρησιμοποιούμε αφηρημένους πόρους r1r1​, r2r2​ και r3r3​.

Στόχος του προβλήματος είναι ο σωστός συγχρονισμός, ώστε κάθε πελάτης να μπορεί να αποκτά με ασφάλεια τους δύο πόρους που χρειάζεται και να μην προκύπτουν φαινόμενα όπως αδιέξοδα (deadlocks) ή λιμοκτονία (starvation).

**2. Περιγραφή του Προβλήματος**

Υπάρχουν τρεις τύποι οντοτήτων στο σύστημά μας:

1. **Servers (S12, S13, S23)**: Παρέχουν στο τραπέζι δύο πόρους τη φορά.
   * Ο Server S12 παρέχει τους πόρους r1r1​ και r2r2​.
   * Ο Server S13 παρέχει τους πόρους r1r1​ και r3r3​.
   * Ο Server S23 παρέχει τους πόρους r2r2​ και r3r3​.
2. **Clients (C12, C13, C23)**: Χρειάζονται δύο συγκεκριμένους πόρους για να προχωρήσουν.
   * Ο Client C12 χρειάζεται r1r1​ και r2r2​.
   * Ο Client C13 χρειάζεται r1r1​ και r3r3​.
   * Ο Client C23 χρειάζεται r2r2​ και r3r3​.
3. **Pushers**: Ενδιάμεσες οντότητες (νήματα) που ανιχνεύουν ποιος πόρος προστέθηκε στο τραπέζι (μέσω σημαφορέων) και, ανάλογα με τον συνδυασμό των πόρων, «ξυπνούν» τον κατάλληλο Client.

Το τραπέζι απεικονίζεται από μια δομή (κλάση Table), η οποία περιέχει σημαφορέ (semaphores) για τον συγχρονισμό των πόρων, των Clients, καθώς και ένα κεντρικό σημαφορό tableEmpty για τον συγχρονισμό των Servers.

**3. Τεχνική Επισκόπηση της Υλοποίησης**

1. **Table**
   * Περιέχει τα semaphore r1, r2, r3 (ένα για κάθε πόρο).
   * Περιέχει τα semaphore client12, client13, client23 (ένα για κάθε Client), καθώς και μεταβλητές-σημαίες (hasR1, hasR2, hasR3) που υποδεικνύουν ότι υπάρχει ένας συγκεκριμένος πόρος στο τραπέζι.
   * Χρησιμοποιεί το tableEmpty για να εξασφαλίσει ότι μόνο ένας Server τη φορά θα τοποθετήσει πόρους στο τραπέζι.
2. **Servers**
   * Κάθε Server (π.χ. S12) εκτελεί περιοδικά το acquire στο tableEmpty (για να «κλείσει» το τραπέζι) και στη συνέχεια κάνει release τους δύο πόρους που του αντιστοιχούν (r1 και r2 για τον S12).
   * Ακολουθεί μικρή καθυστέρηση (π.χ. Thread.sleep(...)) ώστε να δοθεί χρόνος στους υπόλοιπους μηχανισμούς να ενεργοποιηθούν.
3. **Pushers**
   * Υπάρχει ένας Pusher για κάθε πόρο (Pusher(1), Pusher(2), Pusher(3)).
   * Κάθε Pusher περιμένει στο αντίστοιχο σημαφορό (π.χ. ο Pusher(1) στο r1). Όταν ανιχνεύσει ότι απελευθερώθηκε ο πόρος (π.χ. r1.acquire()), θα ελέγξει σε συνδυασμό με τις boolean μεταβλητές (hasR2, hasR3 κ.λπ.) ποιον Client πρέπει να ειδοποιήσει.
   * Στην πράξη, ο Pusher(1) φροντίζει να «δέσει» το r1r1​ είτε με το r2r2​ (άρα ξυπνάει τον C12), είτε με το r3r3​ (ξυπνάει τον C13), κ.ο.κ.
4. **Clients**
   * Κάθε Client περιμένει το δικό του semaphore (client12, client13 ή client23) μέχρι να αποκτήσει τους δύο απαιτούμενους πόρους του.
   * Μόλις αποκτήσει τους πόρους, εκτυπώνει μήνυμα επιτυχίας και αμέσως απελευθερώνει το tableEmpty (με table.tableEmpty.release()), ώστε ο επόμενος Server να μπορεί να τοποθετήσει κι άλλους πόρους.

**4. Παράδειγμα Σωστής Εκτέλεσης**

Μία τυπική ροή εκτέλεσης που επιδεικνύει τη σωστή λειτουργία είναι η εξής:

1. Ο S12 αποκτά το tableEmpty, τοποθετεί r1 και r2.
2. Οι Pusher(1) και Pusher(2) ξυπνούν, βλέπουν ότι υπάρχει r1r1​ και r2r2​, και ειδοποιούν τον C12.
3. Ο C12 ξυπνάει, αποκτά τους δύο πόρους, εκτελεί την «εργασία» του (κατανάλωση) και απελευθερώνει το tableEmpty.
4. Ο S13 αποκτά το tableEmpty, τοποθετεί r1 και r3. Ακολουθεί παρόμοια διαδικασία όπου ο C13 αποκτά τους πόρους του.
5. Ο S23 αποκτά το tableEmpty, τοποθετεί r2 και r3, και ο C23 αποκτά τους πόρους του.

Καθώς όλα τα νήματα τρέχουν σε αέναο βρόχο (while(true)), η διαδικασία επαναλαμβάνεται συνεχώς, χωρίς να δημιουργείται αδιέξοδο.

**5. Παρατηρήσεις και Συμπεράσματα**

* **Συγχρονισμός:** Το κλειδί της επιτυχίας είναι η σωστή χρήση των σημαφορέ και του tableEmpty. Η ορθή διαδοχή acquire(tableEmpty) → release(rX, rY) στους Servers και η χρήση του mutex στους Pushers προστατεύει τις κοινόχρηστες μεταβλητές hasR1, hasR2, hasR3.
* **Διαύγεια Κώδικα:** Η κλάση Table λειτουργεί ως κοινή «αποθήκη» σημαφορέ. Η ονοματοδοσία (π.χ. r1, client12, κλπ.) διευκολύνει την κατανόηση και τη συντήρηση του κώδικα.
* **Αποφυγή Deadlock:** Εφόσον χρησιμοποιείται μόνο ένα tableEmpty που είναι διαθέσιμο σε έναν Server τη φορά, δεν προκύπτει συνθήκη αμοιβαίου αποκλεισμού όλων. Οι πελάτες δεν προβαίνουν σε «επιθετική» δέσμευση πόρων — απλά περιμένουν στο δικό τους semaphore.
* **Αποφυγή Starvation:** Κάθε πελάτης λαμβάνει τελικά τους πόρους του, καθώς οι Servers φέρνουν περιοδικά κάθε πιθανό ζεύγος (r1,r2r1​,r2​, r1,r3r1​,r3​, r2,r3r2​,r3​).

**6. Συνολική Αξιολόγηση**

Το πρόγραμμα επιδεικνύει επιτυχώς μια λύση στο **Πρόβλημα των Καπνιστών**. Μέσω της απλής αρχιτεκτονικής (Servers, Pushers, Clients) και της σωστής χρήσης των σημαφορέ, διασφαλίζεται ότι οι πελάτες θα λαμβάνουν έγκαιρα τους πόρους τους χωρίς να μπλοκάρουν αμοιβαία. Η λύση αυτή είναι επεκτάσιμη και σε περισσότερους πόρους ή πελάτες, αν και η πολυπλοκότητα του κώδικα θα αυξηθεί ανάλογα.