Κωδικός Εργασίας (2.2)

**Ομάδα** [13]

**Μέλος A**: [Γεώργιος, Σιδέρης, 1622, sideris@uth.gr]

**Μέλος B**: [Φώτιος, Τσώκος, 1679, tsokos@inf.uth.gr]

1. Περιγραφή σημείων συγχρονισμού με ψευτοκώδικα

|  |
| --- |
| **1\_1 Η τεχνική επίλυσης**  Προκειμένου να λύσουμε το πρόβλημα της γέφυρας ας αναλογιστούμε πως θα λυνόταν στον αναλογικό κόσμο το ίδιο πρόβλημα. Η απάντηση είναι ιδιαίτερα απλή... με ένα φανάρι! Το φανάρι εξασφαλίζει ότι:  1)Δεν θα υπάρχουν πάνω στη γέφυρα οχήματα αντίθετων κατευθύνσεων.  2)Όλα τα αμάξια σε αναμονή κάποια στιγμή θα περάσουν τη γέφυρα.  Η αλλαγή ωστόσο της κατεύθυνσης του φαναριού γίνεται με κριτήριο το χρόνο, κάτι που στη δική μας υλοποίηση μπορούσε εύκολα να παραβιάσει το δεύτερο κριτήριο (Δηλαδή να υπάρχουν περισσότερα αμάξια στη γέφυρα από την χωρητικότητα της). Έτσι χρησιμοποιήσαμε τον αριθμό αμαξιών ,τα οποία έχουν εισέλθει στη γέφυρα, σαν κριτήριο αλλαγής της επιτρεπόμενης κατεύθυνσης. Με αυτό το τρόπο λύνουμε τα κριτήρια (2), (3), θα εξηγήσουμε παρακάτω το γιατί.  **1\_2 Δομή της λύσης**  Με βάση τη τεχνική “φανάρι” καταλήξαμε στο παρακάτω αλγόριθμο:  Για μια αποθήκη μεγέθους Ν  1)Επιλέγουμε μια αρχική κατεύθυνση  2)Αφήνουμε N αμάξια (όσα και η χωρητικότητα της γέφυρας) να περάσουν  3)Εφόσον έχει βγει και το τελευταίο από τη γέφυρα δεχόμαστε αμάξια αντίθετης κατεύθυνσης  4)Επαναλαμβάνουμε  **1\_3 Περιγραφή σε Ψευτοκώδικα**  Ας υποθέσουμε ότι το mutex mtxRight είναι αρχικοποιημένο ως unlocked και το mtxLeft αντίθετα.  #define capacity N  char CurrentDir;  int CarsPassed=capacity;  int CarsLeft=0;  void \*Car(){  //”gather” cars that want to cross the bridge  if (direction=='r'){ lock (mtxRight);}  else {lock (mtxLeft);}    enterBridge(direction);  exitBridge (direction);  }  void enterBridge(char direction){  CarsPassed--;  //if N cars pass the bridge wait till bridge is empty  //and the direction is changed  //else unlock the appropriate mutex  if (CarsPassed!=0){  unlock (mtxRight/mutexLeft);  // mtxLeft for direction='l',mtxRight for direction='r'  }  }  void exitBridge(char direction){  carsLeft++;  //when N cars have left the bridge change to the opposite direction  if (carsLeft==capacity){  carsleft=0;  carsPassed=0;  changeCurrDirection(direction);  unlock (mtxRight/mutexLeft);  //unlock opposite direction mutex  }else { unlock (mtxRight/mutexLeft);} //unlock same direction mutex  }  Σημ.: Στο πρόγραμμα, λόγω της μη-συνεχής εισόδου της λύσης μας, προσθέσαμε και τη λειτουργία τερματισμού του προγράμματος όταν δεν υπάρχουν άλλα αμάξια να περάσουν τη γέφυρα. |
| **1\_4 Επίλυση με την εισαγωγή του “φαναριού”**  Όσο αφορά τα κριτήρια εκτέλεσης που έθετε η άσκηση, η λύση μας βασίστηκε στην μετάφραση των υποπροβλημάτων σε ένα ενιαίο πρόβλημα, εισάγοντας στο πρόβλημα το φανάρι, και η επιδίωξη γενικής λύσης. Μεταφράσαμε δηλαδή το:  (1)Αμοιβαίος αποκλεισμός: Δεν επιτρέπεται να υπάρχουν πάνω στη γέφυρα οχήματα που κινούνται προς αντίθετες κατευθύνσεις.  (2)Χωρητικότητα: Δεν επιτρέπεται να υπάρχουν πάνω στη γέφυρα παραπάνω από Ν οχήματα.  (3)Πρόοδος: Από τη στιγμή που ένα όχημα περιμένει να διασχίσει τη γέφυρα, κάποια στιγμή θα το καταφέρει, ακόμα και αν η κίνηση από το αντίθετο ρεύμα είναι συνεχής.  Από την ένωση του κριτηρίου (1) με (2) και την χρήση του φαναριού καταλήξαμε στο:  (4)Επιτρέπεται να υπάρχουν πάνω στη γέφυρα μέχρι Ν οχήματα ίδιας κατεύθυνσης και όχι διαφορετικής, ανά αλλαγή της κατεύθυνσης του φαναριού.  Και του (4) με το (3) στο :  (5)Επιτρέπεται να υπάρχουν πάνω στη γέφυρα μέχρι Ν οχήματα ίδιας κατεύθυνσης και όχι διαφορετικής, ανά αλλαγή της κατεύθυνσης του φαναριού, η οποία πρέπει να γίνεται τόσο συχνά ώστε να εξυπηρετεί όλες τις κατευθύνσεις.  Είναι φανερό πλέον ότι η εύρεση ενός δίκαιου κριτηρίου αλλαγής της κατεύθυνσης, αποτελεί το κλειδί στην λύση του προβλήματος. Εφόσον η χρήση του χρόνου ως κριτήριο θα μπορούσε να οδηγεί σε “αδικίες” (πχ για τον ίδιο χρόνο να περάσουν τρία δεξιά και τέσσερα αριστερά αμάξια) κρίναμε ότι το πλήθος των αμαξιών που περνούν τη γέφυρα είναι το απαραίτητο μέτρο εναλλαγής καθώς:  1) Πάντα ξέρουμε ότι θα περάσουν Ν αμάξια και ότι θα υπάρχουν το πολύ N αμάξια πάνω στη γέφυρα (χωρητικότητα).  2)Μετά από Ν αμάξια μίας κατεύθυνσης ακολουθούν πάντα Ν αμάξια της αντίθετης (No starvation).  Έτσι κάθε φορά που περνάει τη γέφυρα αριθμός αμαξιών ίσος με τη χωρητικότητα της, αλλάζουμε την επιτρεπόμενη κατεύθυνση. |
| **1\_5 Γιατί δουλεύει**  **Αμοιβαίος αποκλεισμός:** Τα “κρίσιμα” σημεία στον κώδικα, είναι εκείνα που γίνεται κοινή διαχείριση global μεταβλητών. Με τη χρήση κατάλληλων mutex όμως γνωρίζουμε ότι ένα νήμα τη φορά επεξεργάζεται εκείνο το κομμάτι του προγράμματος. Επιπλέον με την χρήση των mtxRight και mtxLeft γνωρίζουμε ότι στη γέφυρα θα προχωρήσουν μόνο δεξιά ή μόνο αριστερά αμάξια, ποτέ και τα δύο μαζί.  **Απουσία λιμοκτονίας:** Εφόσον λύσαμε το ζήτημα (3) λύσαμε και το κίνδυνο λιμοκτονίας. Επίσης, στο κανονικό πρόγραμμα, οι μεταβλητές countl και countr διασφαλίζουν ότι ακόμα και αν τα αμάξια από μία κατεύθυνση σταματήσουν να έρχονται, η αντίθετη κατεύθυνση παίρνει την προτεραιότητα. (πχ χωρίς τις μεταβλητές αυτές αν η χωρητικότητα είναι 5 και έχουμε 30 αριστερά και 23 δεξιά αμάξια, κάποια στιγμή το πρόγραμμα θα περίμενε για άλλα δύο δεξιά αμάξια για να παραχωρήσει τη προτεραιότητα αριστερά)  **1\_6 Λύση μόνο για πεπερασμένο αριθμό αμαξιών;**  Το ψεγάδι της λύσης μας είναι σίγουρα η πεπερασμένη, και όχι συνεχής, είσοδος. Ωστόσο με μικρές αλλαγές στο πρόγραμμα (ένα while loop με εμφωλευμένο το Loop που δημιουργεί τα νήματα, ωστέ να διαβάζει πεπερασμένες συχνότητες Ν αμαξιών όσες φορές το απαιτήσει ο χρήστης) θα μπορούσαμε να πετύχουμε μια γενική λύση για αρκετά μεγάλο αριθμό αμαξιών. Στη προκειμένη περίπτωση, μας ενδιέφερε να δημιουργήσουμε έναν αλγόριθμο που λύνει το πρόβλημα. Εφόσων ο αλγόριθμος μας μπορεί να λειτουργήσει για ένα αμάξι (Φ(1)) και μπορεί να λειτουργήσει για έναν αριθμό n αμαξιών, όπως και n+1 αμάξια (Φ(n), Φ(n+1)) τότε καταλαβαίνουμε ότι, από την απόδειξη μέσω επαγωγής, μπορεί να λειτουργήσει για κάθε χ που ανήκει στο πεδίο ορισμού του. (Φ(χ) για κάθε χ εDΦ). |