ΗΥ-486 – Αρχές Κατανεμημένου Υπολογισμού
Εαρινό Εξάμηνο 2018-2019

1η Προγραμματιστική Εργασία

Προθεσμία παράδοσης: 3/5 στις 23:59.

1. Γενική Περιγραφή

Στην πρώτη προγραμματιστική εργασία καλείστε να υλοποιήσετε ένα διαμοιραζόμενο σύστημα παραγωγής και διαχείρισης αεροπορικών κρατήσεων. Η προγραμματιστική εργασία θα πρέπει να υλοποιηθεί στην γλώσσα C με τη χρήση της βιβλιοθήκης pthreads.

Προσοχή: Η εργασία είναι, εν μέρει, διαφορετική για τους προπτυχιακούς και τους μεταπτυχιακούς φοιτητές. Κάθε μέρος αναφέρεται αναλυτικά παρακάτω.

2. Υλοποίηση

Στην εργασία αυτή θα πρέπει να υλοποιήσετε ένα διαμοιραζόμενο σύστημα παραγωγής και διαχείρισης αεροπορικών κρατήσεων. Οι λειτουργίες που θα πραγματοποιούνται στο σύστημα είναι η παραγωγή και η διαχείριση αεροπορικών κρατήσεων. Θα υπάρχουν δύο οικογένειες νημάτων (threads), τα **πρακτορεία**, τα οποία θα εκτελούν την παραγωγή των αεροπορικών κρατήσεων, και οι **αεροπορικές εταιρίες**, οι οποίες θα εκτελούν τη διαχείριση των αεροπορικών κρατήσεων. Ακόμη, θα υπάρχει ένα επιπλέον νήμα, ο **ελεγκτής**, ο οποίος θα εκτελεί κάποιους ελέγχους για την εξασφάλιση της ορθότητας του συστήματος.

Πιο συγκεκριμένα, θα δίνετε έναν ακέραιο Α ως όρισμα της συνάρτησης main κατά την εκκίνηση του προγράμματος. Το πλήθος των αεροπορικών εταιριών θα είναι Α και το πλήθος των πρακτορείων θα είναι $\Pi = A^2$. Κάθε αεροπορική εταιρία έχει ένα μοναδικό αναγνωριστικό, το οποίο παίρνει μια σταθερή τιμή από 1 έως Α. Θεωρείστε ότι κάθε εταιρία πραγματοποιεί μία μόνο πτήση και άρα ο αριθμός των διαθέσιμων πτήσεων είναι Α. Θα υπάρχει ένας πίνακας με όνομα *flights*, μεγέθους Α, ο οποίος αναπαριστά τις διαθέσιμες πτήσεις για τις οποίες μπορούν να πραγματοποιηθούν κρατήσεις. Κάθε θέση του πίνακα *flights* περιέχει έναν δείκτη σε ένα struct, το *flight_reservations*, το οποίο περιέχει δείκτες στις δομές που αποθηκεύουν τις κρατήσεις της πτήσης. Θεωρείστε ότι, για κάθε θέση i του πίνακα flights, οι κρατήσεις που βρίσκονται στη θέση i ανήκουν στην αεροπορική εταιρία με αναγνωριστικό i+1.

```
struct flight_reservations {
        struct stack *completed_reservations;
        struct queue *pending_reservations;
}
```

Στο παραπάνω struct, το πεδίο *completed_reservations* είναι ένας δείκτης σε μια διαμοιραζόμενη στοίβα η οποία περιέχει τις κρατήσεις της πτήσης που έχουν ολοκληρωθεί, και το πεδίο *pending_reservations* είναι ένας δείκτης σε μια διαμοιραζόμενη ουρά η οποία περιέχει τις κρατήσεις της πτήσης που βρίσκονται σε αναμονή. Κάθε στοίβα έχει περιορισμένη χωρητικότητα. Παρ' όλα αυτά, για παιδαγωγικούς λόγους θα την υλοποιήσετε ως δυναμική στοίβα όπως περιγράφεται στη συνέχεια. Όταν μια πτήση γεμίσει, οι επιπρόσθετες κρατήσεις που την αφορούν, τοποθετούνται στην ουρά αναμονής της πτήσης.

Η διαμοιραζόμενη στοίβα θα πρέπει να υλοποιηθεί ως εξής:

- Διαμοιραζόμενη στοίβα που υλοποιείται χρησιμοποιώντας **coarse-grained synchronization** (δηλαδή ο συγχρονισμός κατά την πρόσβαση στην στοίβα επιτυγχάνεται χρησιμοποιώντας ένα **coarse-grained lock**). Η διαμοιραζόμενη στοίβα αναπαρίσταται από το struct:

  ```
  struct stack {
      struct stack_reservation *top;
      pthread_mutex_t top_lock;
      int size;
      int capacity;
  }
  ```

  Κάθε κόμβος της στοίβας αναπαρίσταται από το struct:

  ```
  struct stack_reservation {
      struct Reservation reservation;
      struct stack_reservation *next;
  }
  ```

Το *capacity* είναι η χωρητικότητα της στοίβας, δηλαδή το μέγιστο πλήθος κρατήσεων που μπορούν να αποθηκευτούν σ' αυτή τη στοίβα. Το *size* είναι το μέγεθος της στοίβας, δηλαδή το πλήθος των κρατήσεων που βρίσκονται αποθηκευμένες στη στοίβα. Η χωρητικότητα κάθε στοίβας εξαρτάται από τη θέση της στον πίνακα flights. Συγκεκριμένα, η χωρητικότητα της στοίβας της τελευταίας θέσης του πίνακα (δηλ. της θέσης A-1) θα πρέπει να ισούται με $(3/2)*A^2$, και η στοίβα κάθε προηγούμενης θέσης του πίνακα θα πρέπει να έχει χωρητικότητα κατά A στοιχεία λιγότερα από τη στοίβα της επόμενης θέσης. Επομένως:

- Η στοίβα της προτελευταίας θέσης του πίνακα (δηλ. της θέσης A-2) θα έχει χωρητικότητα $(3/2)*A^2 – A$
- Η στοίβα της προηγούμενης θέσης (δηλ. της θέσης A-3) θα έχει χωρητικότητα $(3/2)*A^2 – 2*A$
- ...
- Η στοίβα της πρώτης θέσης (δηλ. της θέσης 0) θα έχει χωρητικότητα $(3/2)*A^2 – (A–1)*A$.

Τα διαφορετικά μεγέθη των στοιβών αντιστοιχούν σε αεροπλάνα διαφορετικού πλήθους θέσεων.

Κάθε κράτηση, ανεξαρτήτως από το αν θα αποθηκευτεί στη στοίβα ή στην ουρά, αναπαρίσταται από το struct:

```
struct Reservation {
        int agency_id;
        int reservation_number;
}
```

Στο παραπάνω struct, το *agency_id* είναι το αναγνωριστικό του πρακτορείου που παρήγαγε την κράτηση. Κάθε πρακτορείο έχει ένα μοναδικό αναγνωριστικό, το οποίο παίρνει μια σταθερή τιμή από 1 έως Π. Το *reservation_number* είναι ο αριθμός της κράτησης.

Όταν μια πτήση γεμίσει, οι επιπρόσθετες κρατήσεις που την αφορούν, τοποθετούνται στην ουρά αναμονής της πτήσης. Η διαμοιραζόμενη αυτή ουρά θα πρέπει να υλοποιηθεί ως εξής:

Προπτυχιακοί φοιτητές:

- **Unbounded Total Queue** με χρήση **locks,** όπως έχει διδαχθεί στο μάθημα (κεφάλαιο 5). Η διαμοιραζόμενη ουρά αναπαρίσταται από το struct:

```
struct queue {
    struct queue_reservation *head;
    struct queue_reservation *tail;
    pthread_mutex_t head_lock;
    pthread_mutex_t tail_lock;
}
```

Μεταπτυχιακοί φοιτητές:

- **Unbounded Lock-Free Queue** χωρίς τη χρήση locks, όπως έχει διδαχθεί στο μάθημα (κεφάλαιο 5). Η διαμοιραζόμενη ουρά αναπαρίσταται από το struct:

```
struct queue {
    struct queue_reservation *head;
    struct queue_reservation *tail;
}
```

Κάθε κόμβος της ουράς αναπαρίσταται από το struct:

```
struct queue_reservation {
        struct Reservation reservation;
        struct queue_reservation *next;
}
```

όπου το *struct Reservation* έχει περιγραφεί παραπάνω.

Το πρακτορείο με αναγνωριστικό *agency_id* παράγει συνολικά Α κρατήσεις με τους εξής αριθμούς κράτησης:

i*Π + agency_id

όπου Π είναι το πλήθος των πρακτορείων, ενώ το i παίρνει τις τιμές $0 \le i \le A-1$. Επομένως:

- Το πρακτορείο με **agency_id=1** παράγει κρατήσεις (struct Reservation) με αριθμούς: **1, Π+1, 2*Π+1, 3*Π+1, ..., (A-1)*Π + 1.**
- Το πρακτορείο με **agency_id=2** παράγει κρατήσεις με αριθμούς: **2, Π+2, 2*Π+2, 3*Π+2, ..., (A-1)*Π + 2.**
- ...
- Το πρακτορείο με **agency_id=Π** παράγει κρατήσεις με αριθμούς: **Π, 2*Π, 3*Π, 4*Π, ..., A*Π.**

Επομένως, ο συνολικός αριθμός κρατήσεων που θα πραγματοποιηθούν σε ολόκληρο το σύστημα είναι A^3. Η παραγωγή κρατήσεων και οι χωρητικότητες των στοιβών έχουν επιλεγεί έτσι ώστε:

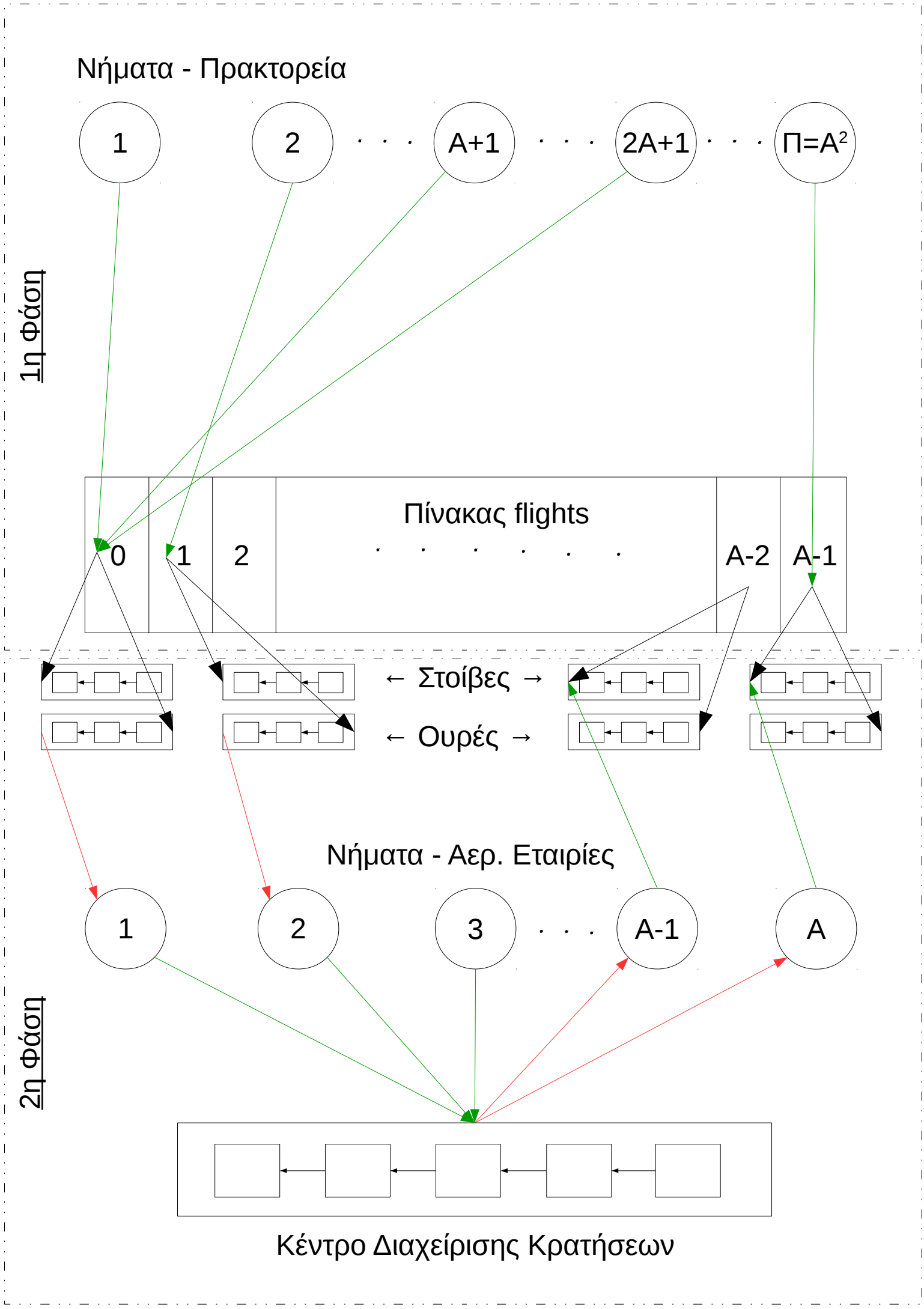
1) Η συνολική χωρητικότητα όλων των στοιβών επαρκεί για όλες τις κρατήσεις.
2) Κάποιες στοίβες (περίπου οι μισές) γεμίζουν και οι υπόλοιπες κρατήσεις για την ίδια πτήση εισάγονται στην ουρά της πτήσης, ενώ στις υπόλοιπες ο αριθμός των στοιχείων που εισάγονται δεν επαρκεί για να γεμίσουν (άρα υπάρχουν ουρές που θα παραμείνουν άδειες). Αυτές οι στοίβες θα εξυπηρετήσουν εν τέλει τις κρατήσεις που έχουν αποθηκευτεί στις ουρές.

3. Ροή Εκτέλεσης

Η εκτέλεση του διαμοιραζόμενου συστήματος αεροπορικών κρατήσεων θα χωριστεί σε δύο φάσεις. Η πρώτη φάση είναι η φάση δημιουργίας και εισαγωγής των κρατήσεων στις στοίβες και τις ουρές του πίνακα flights. Η δεύτερη φάση είναι η φάση διαχείρισης των κρατήσεων από τις αεροπορικές εταιρίες. Η δεύτερη φάση πρέπει να αρχίσει να εκτελείται αφού τελειώσει η πρώτη φάση (δηλαδή οι δύο φάσεις **δεν** θα πρέπει να εκτελεστούν παράλληλα).

3.1 Α’ Φάση

Στην πρώτη φάση, που είναι η φάση δημιουργίας και εισαγωγής των κρατήσεων, τα πρακτορεία θα παράγουν τις κρατήσεις και θα τις εισάγουν στις στοίβες και τις ουρές των πτήσεων του πίνακα flights. Πιο συγκεκριμένα, το πρακτορείο με agency_id=X θα εισάγει τις κρατήσεις του στην ουρά ή τη στοίβα της πτήσης που βρίσκεται στη θέση ((X-1) mod A) του πίνακα flights. Εφ’ όσον υπάρχουν A^2 πρακτορεία στο σύστημα, Α πρακτορεία τοποθετούν κρατήσεις ταυτόχρονα στη στοίβα και στην ουρά της κάθε θέσης, λόγω του τρόπου που έχουν αποδοθεί χωρητικότητες στις στοίβες. Άρα, συνολικά, σε κάθε θέση του πίνακα *flights* (ουρά και στοίβα) θα γίνουν A^2 κρατήσεις. Αυτό θα πραγματοποιείται με τον εξής τρόπο: Αρχικά, κάθε πρακτορείο θα εισάγει τις κρατήσεις του στη στοίβα της πτήσης. Όταν η στοίβα της πτήσης γεμίσει, το πρακτορείο συνεχίζει να εισάγει τις κρατήσεις του στην ουρά αυτής της πτήσης. Πρέπει να σημειωθεί ότι η δημιουργία και η εισαγωγή των κρατήσεων στις στοίβες και τις ουρές του πίνακα flights **θα πρέπει να πραγματοποιείται παράλληλα από όλα τα πρακτορεία**.


Νήματα - Πρακτορεία
1
2
A+1
2A+1
Π=A²
1η Φάση
Πίνακας flights
0
1
2
A-2
A-1
← Στοίβες →
← Ουρές →
Νήματα - Αερ. Εταιρίες
1
2
3
A-1
A
2η Φάση
Κέντρο Διαχείρισης Κρατήσεων

Αφού ολοκληρωθεί η δημιουργία και η εισαγωγή όλων των κρατήσεων στις στοίβες και τις ουρές των πτήσεων, η πρώτη φάση θα ολοκληρωθεί με έναν έλεγχο από τον ελεγκτή. Ο ελεγκτής είναι ένα ειδικό νήμα που έχει ως αναγνωριστικό τον αριθμό 0. Στο τέλος της πρώτης φάσης, εκτελεί τους παρακάτω ελέγχους:

1. **Stack overflow check:** Το πλήθος των κρατήσεων που περιέχει η στοίβα κάθε πτήσης πρέπει να είναι μικρότερο ή ίσο από τη χωρητικότητα της εκάστοτε στοίβας. Αφού ολοκληρωθεί αυτός ο έλεγχος για κάθε θέση i του πίνακα flights, θα πρέπει να τυπώνεται στην οθόνη το μήνυμα:

   Flight i : stack overflow check passed (capacity: X, found: Y)

   όπου X είναι η χωρητικότητα της στοίβας της θέσης i και Y το πλήθος των στοιχείων της στοίβας.
2. **Total size check:** Το συνολικό πλήθος των κρατήσεων από όλες τις πτήσεις πρέπει να ισούται με την προβλεπόμενη τιμή (A^3). Αφού ολοκληρωθεί αυτός ο έλεγχος, θα πρέπει να τυπώνεται στην οθόνη το μήνυμα:

   total size check passed (expected: X, found: Y)

   όπου X είναι η προβλεπόμενη τιμή του συνολικού πλήθους των κρατήσεων από όλες τις πτήσεις και Y είναι ο συνολικός αριθμός των κρατήσεων που βρέθηκαν σε όλες τις πτήσεις, μετά από μια διάσχιση των στοιβών και ουρών του πίνακα flights από τον ελεγκτή.
3. **Total keysum check:** Το άθροισμα των reservation_number των κρατήσεων από όλες τις πτήσεις πρέπει να ισούται με την προβλεπόμενη τιμή ((A^6 + A^3)/2). Αφού ολοκληρωθεί αυτός ο έλεγχος, θα πρέπει να τυπώνεται στην οθόνη το μήνυμα:

   total keysum check passed (expected: X, found: Y)

   όπου X είναι η προβλεπόμενη τιμή του αθροίσματος των reservation_number των κρατήσεων από όλες τις πτήσεις και Y είναι το άθροισμα των reservation_number των κρατήσεων που βρέθηκαν σε όλες τις πτήσεις, μετά από μια διάσχιση των στοιβών και ουρών του πίνακα flights από τον ελεγκτή.

Αν οποιοσδήποτε από τους παραπάνω ελέγχους αποτύχει, το πρόγραμμα πρέπει να εμφανίζει κατάλληλο μήνυμα σφάλματος (όπου θα φαίνεται ποιός έλεγχος απέτυχε και γιατί) και η εκτέλεση θα τερματίζει.

Για να βεβαιωθείτε ότι ο ελεγκτής θα αρχίσει τους ελέγχους μετά το πέρας όλων των εισαγωγών των κρατήσεων στις στοίβες και τις ουρές, θα πρέπει να χρησιμοποιήσετε ένα barrier, που θα ονομάζεται *barrier_start_1st_phase_checks*. Τα πρακτορεία θα τερματίζουν την εκτέλεσή τους περιμένοντας σε αυτό το barrier ενώ ο ελεγκτής θα αρχίζει την εκτέλεσή του περιμένοντας σε αυτό το barrier.

3.2 Β’ Φάση

Στη δεύτερη φάση, που είναι η φάση διαχείρισης των κρατήσεων, κάθε αεροπορική εταιρία διαχειρίζεται τις κρατήσεις της πτήσης που της ανήκει. Η διαχείριση των κρατήσεων πραγματοποιείται με τον ακόλουθο τρόπο: Όποια αεροπορική εταιρία έχει κρατήσεις στην ουρά της πτήσης της, τότε διαγράφει αυτές τις κρατήσεις μία προς μία από την ουρά αυτή και τις εισάγει σε

μια άλλη δομή που ονομάζεται κέντρο διαχείρισης των κρατήσεων. Παράλληλα, όποια αεροπορική εταιρία δεν έχει κρατήσεις στην ουρά της πτήσης της και έχει ακόμα χώρο στη στοίβα της πτήσης της, διαγράφει κρατήσεις μία προς μία από το κέντρο διαχείρισης των κρατήσεων (εφ' όσον υπάρχουν κρατήσεις στο κέντρο διαχείρισης) και τις εισάγει στη στοίβα της πτήσης της, μέχρις ότου η στοίβα αυτή να γεμίσει (είτε να αδειάσει το κέντρο διαχείρισης).

Το κέντρο διαχείρισης των κρατήσεων θα πρέπει να υλοποιηθεί ως εξής:

Προπτυχιακοί φοιτητές:

- **Linked List with Lazy Synchronization**, όπως έχει διδαχθεί στο μάθημα (κεφάλαιο 5). Η διαμοιραζόμενη λίστα αναπαρίσταται από το struct:

  ```
  struct list {
      struct list_reservation *head;
      struct list_reservation *tail;
  }
  ```

  Κάθε κόμβος της λίστας αναπαρίσταται από το struct:

  ```
  struct list_reservation {
      struct Reservation reservation;
      int marked;
      pthread_mutex_t lock;
      struct list_reservation *next;
  }
  ```

Μεταπτυχιακοί φοιτητές:

- **Leaf-oriented BST with Lazy Synchronization** (όπως αυτό που σας ζητήθηκε να υλοποιήσετε στην 1η σειρά θεωρητικών ασκήσεων, Άσκηση 3).

Η ταξινόμηση των κρατήσεων στο κέντρο διαχείρισης θα γίνει με βάση το **reservation_number** της κάθε κράτησης.

Αφού ολοκληρωθεί η διαχείριση όλων των κρατήσεων, η δεύτερη φάση θα ολοκληρωθεί με έναν έλεγχο από τον ελεγκτή. Στο τέλος της δεύτερης φάσης, ο ελεγκτής εκτελεί τους παρακάτω ελέγχους:

1. **Stack overflow check:** Όμοια με την 1η φάση.
2. **Total size check:** Όμοια με την 1η φάση.
3. **Total keysum check:** Όμοια με την 1η φάση.
4. **Reservations completion check:** Ελέγχει αν το κέντρο διαχείρισης των κρατήσεων, καθώς και όλες οι ουρές, είναι άδειες. Αφού ολοκληρωθεί αυτός ο έλεγχος, θα πρέπει να τυπώνεται στην οθόνη το μήνυμα:

   reservations completion check passed

Αν οποιοσδήποτε από τους παραπάνω ελέγχους αποτύχει, το πρόγραμμα πρέπει να εμφανίζει κατάλληλο μήνυμα λάθους (όπου θα φαίνεται ποιός έλεγχος απέτυχε και γιατί) και η εκτέλεση θα τερματίζει.

Για να βεβαιωθείτε ότι η διαχείριση των κρατήσεων θα αρχίσει μετά το πέρας του ελέγχου της πρώτης φάσης από τον ελεγκτή, θα πρέπει να χρησιμοποιήσετε ένα barrier, που θα ονομάζεται *barrier_start_2nd_phase*. Οι αεροπορικές εταιρίες θα αρχίζουν την εκτέλεσή τους περιμένοντας σε αυτό το barrier ενώ ο ελεγκτής θα συνεχίζει την εκτέλεσή του (μετά το πέρας του ελέγχου της πρώτης φάσης) περιμένοντας σε αυτό το barrier.

Για να βεβαιωθείτε ότι ο ελεγκτής θα αρχίσει τον έλεγχο της δεύτερης φάσης μετά το πέρας της διαχείρισης όλων των κρατήσεων, θα πρέπει να χρησιμοποιήσετε ένα barrier, που θα ονομάζεται *barrier_start_2nd_phase_checks*. Οι αεροπορικές εταιρίες θα τερματίζουν την εκτέλεσή τους περιμένοντας σε αυτό το barrier ενώ ο ελεγκτής θα συνεχίζει την εκτέλεσή του (μετά την αναμονή του στο *barrier_start_2nd_phase*) περιμένοντας σε αυτό το barrier.

4. Αρχικοποίηση κι Εκτέλεση Προγράμματος

Το πρόγραμμα θα πρέπει να δέχεται ως είσοδο έναν φυσικό αριθμό Α, ο οποίος αναπαριστά το πλήθος των αεροπορικών εταιριών καθώς και το μέγεθος του πίνακα flights.

5. Παράδοση Εργασίας

Για την παράδοση της εργασίας θα πρέπει να χρησιμοποιήσετε το πρόγραμμα **turnin**, που υπάρχει εγκατεστημένο στα μηχανήματα του τμήματος. Συγκεκριμένα, η εντολή παράδοσης είναι:

turnin project1@hy486

Για να επιβεβαιώσετε ότι η υποβολή της εργασίας σας ήταν επιτυχής μπορείτε να χρησιμοποιήσετε την εντολή:

verify-turnin project1@hy486

Προσοχή: Τα παραδοτέα σας θα πρέπει να περιέχουν ό,τι χρειάζεται (τα αρχεία πηγαίου κώδικα και ένα README όπου θα εξηγείτε πώς κάνατε compile και πως τρέξατε τον κώδικά σας, εναλλακτικά μπορείτε να φτιάξετε Makefile) και να είναι σωστά δομημένα ώστε να κάνουν compile και να εκτελούνται **στα μηχανήματα της σχολής**, όπου και θα γίνει η εξέταση της εργασίας. Ο κώδικάς σας θα πρέπει να περιέχει σχόλια.