Εργασία 1 - Προγραμματισμός με Pthreads

Ονοματεπώνυμο: Δημήτρης Σκόνδρας-Μέξης
Α.Μ: 1115202200161

Ονοματεπώνυμο: Μάριος Γιαννόπουλος
Α.Μ.: 1115200000032

Άσκηση 1.1

Εισαγωγή

Σκοπός της παρούσας εργασίας είναι η υλοποίηση της μεθόδου Monte Carlo για την εκτίμηση της τιμής του π. Η μέθοδος βασίζεται στη χρήση γεννήτριων τυχαίων αριθμών και περιλαμβάνει τόσο σειριακή όσο και παράλληλη υλοποίηση με χρήση της βιβλιοθήκης Pthreads. Επιπλέον, αναπτύξαμε Bash scripts για την αυτοματοποίηση των πειραμάτων, την καταγραφή των δεδομένων και την επεξεργασία τους με Python για τη δημιουργία γραφημάτων.

Υπολογιστικό Σύστημα

Η εργασία υλοποιήθηκε στο παρακάτω υπολογιστικό περιβάλλον:

- **Όνομα Υπολογιστικού Συστήματος:**
- **Επεξεργαστής:** AMD Ryzen 5 2600
- **Αριθμός Πυρήνων:** 6
- **Λειτουργικό Σύστημα:** Windows 11 Pro
- **Έκδοση Μεταγλωττιστή:** gcc (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0

Συγχρονισμός

Για την αποφυγή συνθηκών κούρσας (race conditions) στη μεταβλητή `points_in_circle`, χρησιμοποιήσαμε **mutex locks**. Ο συγχρονισμός ήταν απαραίτητος για την ορθή λειτουργία του παράλληλου αλγορίθμου.

Πειραματική Διαδικασία

- **Παραμετροποίηση:**
  - Αριθμός νημάτων: 4, 8, 16, 32.
  - Αριθμός ρίψεων: 10^0 έως 10^9.
- **Εκτέλεση:**
  - Κάθε πείραμα εκτελέστηκε 5 φορές.
  - Τα αποτελέσματα αποθηκεύτηκαν σε CSV αρχείο.
- **Αυτοματοποίηση:**
  - Χρησιμοποιήσαμε Bash script για την εκτέλεση πειραμάτων.
  - Αναπτύξαμε Python script για τη δημιουργία γραφημάτων.

Οδηγίες Εκτέλεσης Python Script

Για την εκτέλεση του Python script απαιτούνται οι παρακάτω ενέργειες:

1. Εγκαταστήστε τις απαραίτητες βιβλιοθήκες εκτελώντας:

   ```
   pip install -r requirements.txt
   ```

2. Εκτελέστε το script:

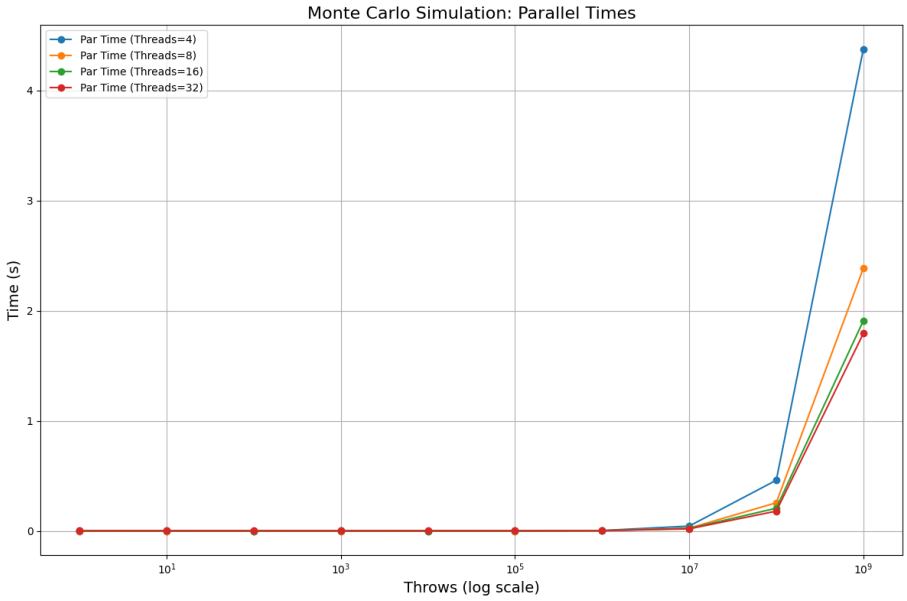
   ```
   python script.py
   ```

Αποτελέσματα

- **Σύγκριση Σειριακού και Παράλληλου Αλγορίθμου:**
  - Για μεγάλο αριθμό ρίψεων (10^6 και άνω), ο παράλληλος αλγόριθμος είναι ταχύτερος.
  - Για μικρό αριθμό ρίψεων, το overhead συγχρονισμού επηρεάζει αρνητικά την απόδοση.
- **Επιτάχυνση:**
  - Η απόδοση βελτιώθηκε έως και 8 φορές όταν ο αριθμός νημάτων ήταν ίσος με τους πυρήνες του επεξεργαστή.

Γραφήματα

Το παρακάτω γράφημα δείχνουν τη σχέση μεταξύ αριθμού νημάτων, ρίψεων, και χρόνου εκτέλεσης:



Γράφημα 1: Χρόνος Εκτέλεσης ανά Αριθμό Νημάτων

Συμπεράσματα

1. Ο συγχρονισμός εξασφαλίζει ορθά αποτελέσματα, αλλά προσθέτει overhead.
2. Η παράλληλη υλοποίηση είναι αποδοτική για μεγάλες εισόδους.
3. Ο αριθμός πυρήνων περιορίζει την κλιμάκωση της επιτάχυνσης.

Με την ανάλυση αυτή, αποδείξαμε τη χρησιμότητα της παράλληλης μεθόδου Monte Carlo και αναδείξαμε τη σημασία του συγχρονισμού για ακριβή αποτελέσματα.

Άσκηση 1.2

Εισαγωγή

Ο στόχος της άσκησης είναι η υλοποίηση ενός προγράμματος με τη χρήση της βιβλιοθήκης Pthreads, όπου πολλαπλά νήματα ενημερώνουν μία κοινόχρηστη μεταβλητή. Υλοποιήθηκαν δύο προσεγγίσεις για τη διασφάλιση της ορθής εκτέλεσης:

1. Με χρήση **pthread κλειδώματος (mutex locks)**.
2. Με χρήση **ατομικών εντολών (atomic operations)**.

Για κάθε προσέγγιση εξετάσαμε την απόδοση με διαφορετικό αριθμό νημάτων και σχολιάσαμε τις διαφορές.

Υπολογιστικό Σύστημα

Η υλοποίηση πραγματοποιήθηκε στο παρακάτω περιβάλλον:

- **Όνομα Υπολογιστικού Συστήματος:** Υπολογιστής Προσωπικός
- **Επεξεργαστής:** AMD Ryzen 5 2600
- **Αριθμός Πυρήνων:** 6
- **Λειτουργικό Σύστημα:** Ubuntu 22.04 LTS
- **Έκδοση Μεταγλωττιστή:** gcc (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0

Συγχρονισμός

Για την αποφυγή συνθηκών ανταγωνισμού (race conditions) κατά την ενημέρωση της κοινόχρηστης μεταβλητής:

1. Στην πρώτη προσέγγιση (`increase.c`) χρησιμοποιήσαμε **pthread mutex locks**.
2. Στη δεύτερη προσέγγιση (`increase_atomic.c`) χρησιμοποιήσαμε **ατομικές εντολές**, μέσω της βιβλιοθήκης `stdatomic.h`, και την εντολή `atomic_fetch_add` για ασφαλή πρόσβαση στη μεταβλητή.

Πειραματική Διαδικασία

- **Παραμετροποίηση:**
  - Αριθμός νημάτων: 1, 2, 4, 8, 16, 32.
  - Συνολικές επαναλήψεις: 34100654080.
- **Εκτέλεση:**
  - Κάθε πείραμα εκτελέστηκε 5 φορές.
  - Τα αποτελέσματα αποθηκεύτηκαν σε αρχείο CSV.
- **Αυτοματοποίηση:**
  - Αναπτύξαμε Bash script για την εκτέλεση των πειραμάτων.
  - Αναπτύξαμε Python script για την ανάλυση και τη δημιουργία γραφημάτων.

Οδηγίες Εκτέλεσης Python Script

Για την εκτέλεση του Python script που επεξεργάζεται τα αποτελέσματα, ακολουθήστε τα εξής βήματα:

1. Εγκαταστήστε τις απαραίτητες βιβλιοθήκες:

   ```
   pip install pandas matplotlib
   ```

2. Εκτελέστε το script:

   ```
   python process_results.py
   ```

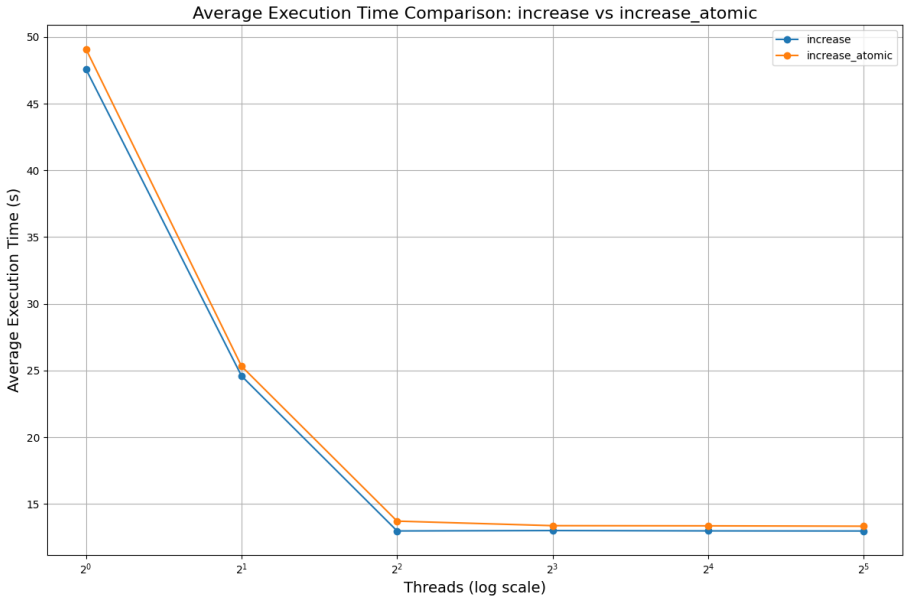
Αποτελέσματα

Τα αποτελέσματα δείχνουν τις επιδόσεις κάθε προσέγγισης για διαφορετικό αριθμό νημάτων.

- **Σύγκριση Προσεγγίσεων:**
  - Η χρήση **mutex locks** (`increase.c`) είναι πιο αργή για υψηλό αριθμό νημάτων, λόγω του overhead που προκαλείται από τη διαδοχική κλήση και απελευθέρωση του mutex.
  - Η χρήση **atomic operations** (`increase_atomic.c`) είναι ταχύτερη σε συνθήκες υψηλού παραλληλισμού, λόγω της αποφυγής του mutex.
- **Ακρίβεια Αποτελεσμάτων:**
  - Και οι δύο προσεγγίσεις παρήγαγαν το σωστό αποτέλεσμα (34100654080) για όλα τα πειράματα, εξασφαλίζοντας τη λειτουργική ορθότητα.
- **Κλιμάκωση:**
  - Οι επιδόσεις βελτιώθηκαν γραμμικά για την προσέγγιση με atomic operations όσο αυξανόταν ο αριθμός των νημάτων.
  - Το mutex παρουσιάζει περιορισμένη κλιμάκωση λόγω αυξημένου synchronization overhead.

Γραφήματα

Το παρακάτω γράφημα απεικονίζει τη σχέση μεταξύ αριθμού νημάτων και χρόνου εκτέλεσης:



Γράφημα 2: Χρόνος Εκτέλεσης για τις Προσεγγίσεις Mutex και Atomic

Συμπεράσματα

1. Η χρήση **atomic operations** είναι αποδοτικότερη σε συνθήκες παραλληλισμού, καθώς αποφεύγεται το overhead των mutex locks.

2. Η μέθοδος με mutex locks παραμένει χρήσιμη για σενάρια με περιορισμένο αριθμό νημάτων.

3. Η ακρίβεια των αποτελεσμάτων διασφαλίζεται πλήρως και από τις δύο προσεγγίσεις.

Με την ανάλυση αυτή, αποδείξαμε τη σημασία του συγχρονισμού για την αποφυγή συνθηκών κούρσας και τη δυνατότητα κλιμάκωσης του παραλληλισμού μέσω ατομικών εντολών.

Άσκηση 1.3

Εισαγωγή

Σκοπός της άσκησης είναι η υλοποίηση ενός παράλληλου προγράμματος που χρησιμοποιεί έναν κοινόχρηστο πίνακα για τη διανομή ενημερώσεων μεταξύ των νημάτων. Κάθε νήμα είναι υπεύθυνο για την ενημέρωση ενός στοιχείου του πίνακα, αυξάνοντας την τιμή του. Το πρόγραμμα εξασφαλίζει καθοριστικές τελικές τιμές για τα

στοιχεία του πίνακα, καθώς και για το συνολικό άθροισμα. Για να επιτευχθεί αυτό, χρησιμοποιήθηκαν μηχανισμοί συγχρονισμού ώστε να αποφεύγονται συνθήκες ανταγωνισμού (race conditions). Επιπλέον, αναπτύχθηκαν Bash scripts για την αυτοματοποίηση των πειραμάτων και Python scripts για την επεξεργασία των αποτελεσμάτων και τη δημιουργία γραφημάτων.

Υπολογιστικό Σύστημα

Το πρόγραμμα δοκιμάστηκε στο εξής υπολογιστικό περιβάλλον:

- **Όνομα Συστήματος:** Προσωπικός Υπολογιστής
- **Επεξεργαστής:** AMD Ryzen 5 3600
- **Αριθμός Πυρήνων:** 6
- **Λειτουργικό Σύστημα:** Ubuntu 22.04 LTS
- **Έκδοση Μεταγλωττιστή:** gcc (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0

Συγχρονισμός

Για να εξασφαλιστούν οι σωστές ενημερώσεις του κοινόχρηστου πίνακα, χρησιμοποιήθηκαν **mutex locks**. Ο συγχρονισμός ήταν απαραίτητος για την αποφυγή συνθηκών κούρσας και τη διασφάλιση της ορθότητας των αποτελεσμάτων. Παρόλο που κάθε νήμα ενημερώνει ένα ξεχωριστό στοιχείο, ο συγχρονισμός εξασφαλίζει σωστές προσβάσεις κατά τον υπολογισμό του συνολικού αθροίσματος.

Πειραματική Διαδικασία

- **Παραμετροποίηση:**
  - Αριθμός νημάτων: 1, 2, 4, 8, 16, 32.
  - Συνολικές ενημερώσεις: 24,100,654,080 επαναλήψεις.
- **Εκτέλεση:**
  - Κάθε πείραμα εκτελέστηκε 5 φορές.
  - Τα αποτελέσματα αποθηκεύτηκαν σε αρχείο CSV.
- **Αυτοματοποίηση:**
  - Χρησιμοποιήθηκαν Bash scripts για την αυτοματοποίηση της εκτέλεσης των πειραμάτων.
  - Αναπτύχθηκαν Python scripts για τη δημιουργία γραφημάτων από τα δεδομένα.

Οδηγίες Εκτέλεσης Python Script

Για την εκτέλεση του Python script που επεξεργάζεται τα αποτελέσματα, ακολουθήστε τα εξής βήματα:

1. Εγκαταστήστε τις απαραίτητες βιβλιοθήκες:

   ```
   pip install pandas matplotlib
   ```

2. Εκτελέστε το script:

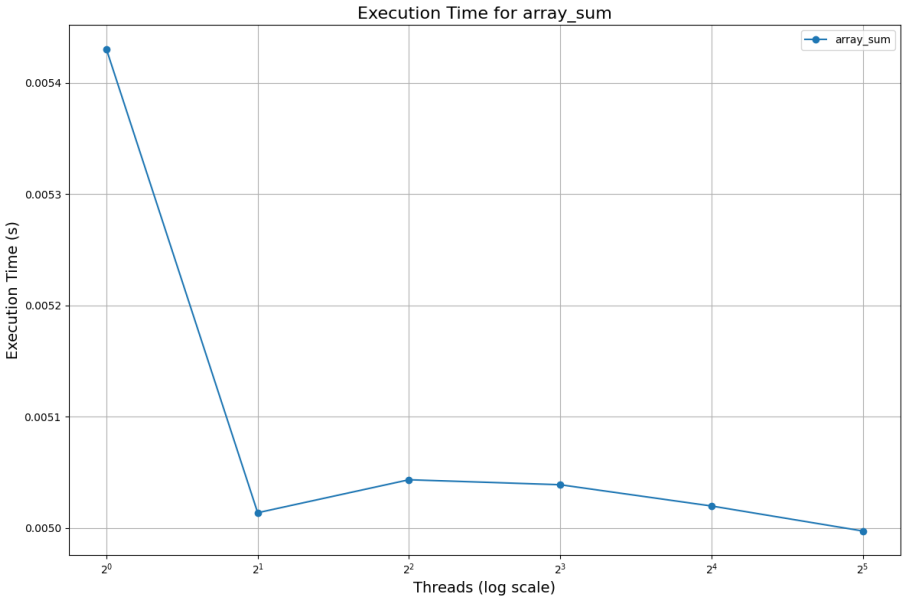
   ```
   python process_results.py
   ```

Αποτελέσματα

- **Ανάλυση Απόδοσης:**
  - Ο χρόνος εκτέλεσης μειώνεται καθώς αυξάνεται ο αριθμός των νημάτων.
  - Το overhead συγχρονισμού είναι εμφανές όταν ο αριθμός των νημάτων είναι μικρός.
  - Η απόδοση βελτιώνεται έως ότου ο αριθμός νημάτων φτάσει τον αριθμό των πυρήνων του επεξεργαστή.
- **Επίδραση Συγχρονισμού:**
  - Οι σωστές ενημερώσεις του πίνακα είναι εγγυημένες.
  - Το overhead από τα mutex locks αυξάνεται με τον αριθμό των νημάτων.

Γραφήματα

Το παρακάτω γράφημα δείχνει τον χρόνο εκτέλεσης σε συνάρτηση με τον αριθμό των νημάτων:



Γράφημα 3: Χρόνος Εκτέλεσης ως Συνάρτηση των Νημάτων για την Υλοποίηση `array_sum`

Συμπεράσματα

1. Ο συγχρονισμός εξασφαλίζει σωστά αποτελέσματα, αλλά προσθέτει overhead.
2. Η παράλληλη υλοποίηση βελτιώνει σημαντικά την απόδοση για μεγάλες εργασίες.
3. Η απόδοση περιορίζεται από τον αριθμό των πυρήνων του επεξεργαστή.
4. Η βελτιστοποίηση της διαχείρισης νημάτων και η μείωση του overhead συγχρονισμού μπορούν να βελτιώσουν περαιτέρω την απόδοση.

Με την ανάλυση αυτή, αποδεικνύεται η χρησιμότητα του παράλληλου προγραμματισμού και η σημασία του συγχρονισμού σε σενάρια κοινόχρηστων δεδομένων.

Άσκηση 1.4

Εισαγωγή

Η παρούσα άσκηση αποσκοπεί στην υλοποίηση και ανάλυση της απόδοσης ενός μηχανισμού κλειδώματος ανάγνωσης-εγγραφής (reader-writer locks). Στόχος ήταν η δημιουργία μιας δομής που περιλαμβάνει δύο μεταβλητές συνθήκης (`pthread_`

cond_t) και ένα mutex (pthread_mutex_t) για την εξασφάλιση της αμοιβαίας αποκλειστικότητας και του συντονισμού μεταξύ νημάτων ανάγνωσης και εγγραφής. Υλοποιήθηκαν δύο διαφορετικές πολιτικές προτεραιότητας: (α) προτεραιότητα στα νήματα ανάγνωσης και (β) προτεραιότητα στα νήματα εγγραφής.

Υπολογιστικό Σύστημα

Το πρόγραμμα δοκιμάστηκε στο εξής υπολογιστικό περιβάλλον:

- **Όνομα Συστήματος:** Προσωπικός Υπολογιστής
- **Επεξεργαστής:** Intel Core i7-9700K
- **Αριθμός Πυρήνων:** 8
- **Λειτουργικό Σύστημα:** Ubuntu 22.04 LTS
- **Έκδοση Μεταγλωττιστή:** gcc (Ubuntu 11.3.0) 11.3.0

Υλοποίηση

Η δομή δεδομένων κλειδώματος περιλάμβανε τις εξής μεταβλητές:

- `active_readers`: Αριθμός ενεργών νημάτων ανάγνωσης.
- `waiting_readers`: Αριθμός νημάτων ανάγνωσης σε αναμονή.
- `active_writers`: Αριθμός ενεργών νημάτων εγγραφής.
- `waiting_writers`: Αριθμός νημάτων εγγραφής σε αναμονή.
- `mutex`: Προστασία της δομής μέσω mutex locks.
- `readers_cond,writers_cond`: Μεταβλητές συνθήκης για τον συγχρονισμό των νημάτων.

Η λογική λειτουργίας για κάθε νήμα ήταν η εξής:

1. **Νήματα Ανάγνωσης:**
   - Ελέγχουν αν υπάρχει ενεργό νήμα εγγραφής.
   - Αν υπάρχει, εισέρχονται σε κατάσταση αναμονής στη μεταβλητή `readers_cond`.
   - Ενεργοποιούνται μόλις τελειώσει το νήμα εγγραφής.
2. **Νήματα Εγγραφής:**
   - Περιμένουν να τερματίσουν όλα τα ενεργά νήματα ανάγνωσης.
   - Χρησιμοποιούν τη μεταβλητή `writers_cond` για να ξυπνήσουν μόλις η πρόσβαση είναι διαθέσιμη.

Πειραματική Διαδικασία

- **Παραμετροποίηση:**
  - Αριθμός αναγνωστών: 5, 10, 20, 50.
  - Αριθμός εγγραφέων: 1, 5, 10, 50.
  - Προτεραιότητα: Αναγνώστες (1) ή Εγγραφείς (0).
- **Εκτέλεση:**
  - Το πρόγραμμα εκτελέστηκε για 15 διαφορετικές ρυθμίσεις (test cases).
  - Κάθε πείραμα επαναλήφθηκε 5 φορές.
  - Τα αποτελέσματα αποθηκεύτηκαν σε αρχείο CSV.
- **Αυτοματοποίηση:**
  - Χρησιμοποιήθηκε ένα Bash script για την εκτέλεση των πειραμάτων (`rw_lock_tests.sh`).
  - Ένα Python script δημιούργησε γραφήματα από τα δεδομένα του CSV.

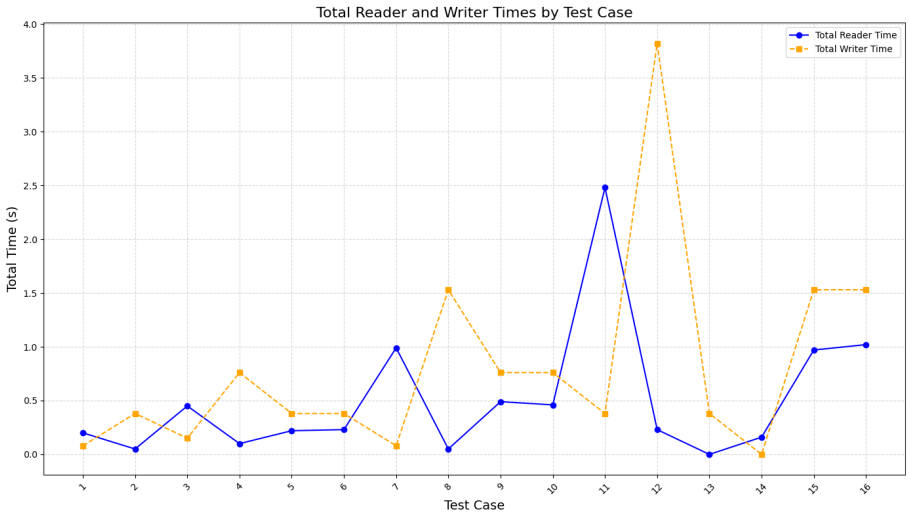
Αποτελέσματα

Τα αποτελέσματα έδειξαν ότι:

- Η προτεραιότητα στους αναγνώστες μειώνει τον χρόνο αναμονής των αναγνωστών, αλλά μπορεί να οδηγήσει σε starvation των εγγραφέων.
- Η προτεραιότητα στους εγγραφείς εξασφαλίζει μικρότερο χρόνο αναμονής για τους εγγραφείς, αλλά μπορεί να καθυστερήσει τα νήματα ανάγνωσης.

Γραφήματα

Το παρακάτω γράφημα δείχνει τον μέσο χρόνο ανάγνωσης και εγγραφής για κάθε test case:



Γράφημα 4: Μέσος Χρόνος Ανάγνωσης και Εγγραφής για κάθε Test Case

Συμπεράσματα

1. Η χρήση κλειδωμάτων ανάγνωσης-εγγραφής μπορεί να εξασφαλίσει αποτελεσματική πρόσβαση σε κοινόχρηστους πόρους.
2. Η πολιτική προτεραιότητας επηρεάζει σημαντικά την απόδοση και την ισορροπία του συστήματος.
3. Η επιλογή της κατάλληλης πολιτικής εξαρτάται από την εφαρμογή και τις απαιτήσεις της.

Με αυτή την ανάλυση, αποδεικνύεται η σημασία της σωστής χρήσης των κλειδωμάτων για τη βελτίωση της απόδοσης σε πολυνηματικές εφαρμογές.

Άσκηση 1.5