Σειρά Εργασιών 1

1.1 FIFO pipe

Υλοποιήστε έναν αγωγό FIFO μιας κατεύθυνσης για την επικοινωνία ανάμεσα σε δύο νήματα, ως ένα ανεξάρτητο τμήμα λογισμικού με τις λειτουργίες void pipe_init(int size) για την αρχικοποίηση του αγωγού με χωρητικότητα size, void pipe_write(char c) για την τοποθέτηση ενός byte στον αγωγό, void pipe_close() για το κλείσιμο του αγωγού, και int pipe_read(char *c) για την ανάγνωση ενός byte από τον αγωγό. Αν ο αγωγός είναι γεμάτος, η pipe_write πρέπει να «μπλοκάρει» μέχρι να διαβαστούν δεδομένα και να δημιουργηθεί χώρος. Αν ο αγωγός είναι άδειος, η pipe_read πρέπει να «μπλοκάρει» μέχρι να γραφτούν δεδομένα ή να κληθεί η pipe_close οπότε η pipe_read επιστρέφει 0 χωρίς να έχουν διαβαστεί δεδομένα.

Βασιστείτε στην τεχνική της κυκλικής αποθήκης (circular/ring buffer). Σκεφτείτε κατά πόσο μπορεί να προκύψουν ανεπιθύμητες συνθήκες ανταγωνισμού για ταυτόχρονη εκτέλεση των pipe_write/close και pipe_read. Ελέγξτε την υλοποίηση σας μέσω ενός προγράμματος όπου δύο νήματα επικοινωνούν μεταξύ τους μέσω ενός αγωγού μεγέθους 64 bytes, για την μεταφορά (μεγάλων) αρχείων.

1.2 Αναγνώριση πρώτων αριθμών

Υλοποιήστε την συνάρτηση int primetest(int v) που επιστρέφει 1 αν το v είναι πρώτος αριθμός, διαφορετικά επιστρέφει 0. Στην συνέχεια υλοποιήστε ένα πρόγραμμα που διαβάζει από την είσοδο του μια ακολουθία ακεραίων, και για κάθε έναν από αυτούς ελέγχει κατά πόσο είναι πρώτος μέσω της primetest(). Για να επιταχυνθεί ο υπολογισμός, το πρόγραμμα δημιουργεί και στην συνέχεια χρησιμοποιεί n νήματα εργάτες (το n είναι όρισμα του προγράμματος), στο πνεύμα του παρακάτω σχήματος:

```
main thread:

create workers
while (job exists) {
  wait for a worker to become available
  assign next job to an available worker
  notify the worker to process the job
}
wait for all workers to become available
notify workers to terminate
wait for workers to terminate
```

```
worker thread:


while (1) {
  notify main that I am available
  wait for notification by main
  if notified to terminate, break
  process assigned job
}
notify main that I will terminate
```

Στο τέλος, το πρόγραμμα περιμένει να ολοκληρωθούν τυχόν εκκρεμείς υπολογισμοί, και καταστρέφει τα νήματα που δημιούργησε, προτού τερματίσει το ίδιο. Προσπαθήστε να αναλύσετε την απόδοση του προγράμματος ως συνάρτηση του αριθμού των νημάτων.

1.3 Quicksort

Υλοποιήστε μια παράλληλη αναδρομική έκδοση του quicksort όπου η ταξινόμηση των στοιχείων του πίνακα να γίνεται από πολλά νήματα. Σε κάθε επίπεδο αναδρομής, ένα ξεχωριστό νήμα (αρχικά το κυρίως νήμα) ελέγχει το τμήμα του πίνακα που του έχει ανατεθεί (για το κυρίως νήμα, ολόκληρος ο πίνακας). Αν αυτό έχει μήκος < 2 τότε το νήμα δεν κάνει τίποτα και τερματίζει, διαφορετικά πραγματοποιεί το βήμα του «διαχωρισμού» για το κομμάτι του πίνακα που του έχει ανατεθεί και αναθέτει αναδρομικά την ταξινόμηση των δύο υπο-τμημάτων σε δύο νέα νήματα που δημιουργεί για αυτό τον σκοπό, περιμένει να τερματίσουν και μετά επιστρέφει το ίδιο. Τα επιπλέον νήματα και οι μεταβλητές που χρησιμοποιούνται για τον συγχρονισμού τους, πρέπει να δημιουργούνται και να καταστρέφονται δυναμικά κατά την αναδρομή.

Δοκιμάστε την υλοποίηση σας μέσω ενός προγράμματος που διαβάζει από την είσοδο του μια ακολουθία ακεραίων που αποθηκεύει σε έναν καθολικό πίνακα, στην συνέχεια ταξινομεί τον πίνακα χρησιμοποιώντας την παράλληλη έκδοση του quicksort, και τέλος εκτυπώνει το αποτέλεσμα.

Σημείωση για όλες τις εργασίες: Η υλοποίηση πρέπει να γίνει σε C με χρήση της βιβλιοθήκης pthreads. Ο συγχρονισμός μεταξύ των νημάτων πρέπει να υλοποιηθεί με **απλές κοινές μεταβλητές και ενεργή αναμονή** χωρίς να χρησιμοποιηθεί κάποιος από τους μηχανισμούς συγχρονισμού των pthreads (μπορείτε όμως να χρησιμοποιήσετε την λειτουργία yield για εθελοντική παραχώρηση του επεξεργαστή).