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 «περιμένει» μέχρι να διαβαστούν δεδομένα και να δημιουργηθεί χώρος. Αν ο αγωγός είναι άδειος, η pipe_read πρέπει να «περιμένει» μέχρι να γραφτούν δεδομένα ή να κληθεί η pipe_close (μόνο τότε η pipe_read επιστρέφει 0 χωρίς να έχει διαβάσει δεδομένα).

Βασιστείτε στην τεχνική της «κυκλικής» αποθήκης, έτσι ώστε να μην προκύπτουν ανεπιθύμητες συνθήκες ανταγωνισμού υπό ταυτόχρονη εκτέλεση των pipe_write/close και pipe_read. Δοκιμάστε την υλοποίηση σας μέσω ενός προγράμματος που χρησιμοποιεί έναν αγωγό μεγέθους K και δύο νήματα, με το ένα γράφει στον αγωγό N bytes και το άλλο να τα διαβάζει (K και N είναι παράμετροι του προγράμματος).

1.2 Παράλληλος υπολογισμός fractals

Στην ιστοσελίδα του μαθήματος δίνεται ένα πρόγραμμα που υπολογίζει/σχεδιάζει το Mandelbrot set. Το πρόγραμμα υποδιαιρεί την περιοχή σε N τμήματα, υπολογίζει κάθε τμήμα ξεχωριστά και παρουσιάζει το αποτέλεσμα με γραφικό τρόπο σε ένα παράθυρο. Ο υπολογισμός μπορεί να επαναληφθεί πολλές φορές.

Αλλάξετε το πρόγραμμα έτσι ώστε κάθε τμήμα να υπολογίζεται από ένα ξεχωριστό νήμα «εργάτη», με το κυρίως νήμα να σχεδιάζει τα επιμέρους αποτελέσματα άμεσα, με το που επιστρέφονται από τους εργάτες:

```
main thread:

while (next problem region defined) {
  create N jobs and assign to workers
  notify workers
  while (not all workers done) {
    wait for some worker to finish job
    retrieve & draw the result
  }
}
```

```
worker thread:

while (1) {
  wait for main to assign job
  retrieve job parameters
  perform the Mandelbrot computation
  store results
  notify main
}
```

Τα νήματα εργάτες δεν πρέπει να καταστρέφονται. Αν ο χρήστης ζητήσει επανάληψη του υπολογισμού, το πρόγραμμα πρέπει να χρησιμοποιεί τους ίδιους εργάτες (όχι να δημιουργεί νέα νήματα κάθε φορά).

1.3 Παράλληλο quicksort

Υλοποιήστε μια παράλληλη αναδρομική έκδοση του quicksort, έτσι ώστε η ταξινόμηση των στοιχείων του πίνακα να γίνεται από $N = 2^k - 1$ νήματα (μαζί με το κυρίως νήμα), όπου $k>0$ είναι ο επιθυμητός βαθμός παραλληλισμού. Κάθε νήμα διαχωρίζει το τμήμα του πίνακα που του έχει ανατεθεί σε δύο τμήματα, και στην συνέχεια (α) αν δεν έχει επιτευχθεί ο επιθυμητός βαθμός παραλληλισμού, αναθέτει την ταξινόμηση των δύο τμημάτων σε δύο νέα νήματα, (β) διαφορετικά, πραγματοποιεί την ταξινόμηση μόνο του. Ανάλογα με το μέγεθος του πίνακα, η ταξινόμηση μπορεί να ολοκληρωθεί χωρίς να επιτευχθεί ο επιθυμητός βαθμός παραλληλισμού.

Δοκιμάστε την υλοποίηση σας μέσω ενός προγράμματος που διαβάζει M στοιχεία τα οποία αποθηκεύει σε έναν πίνακα, καθώς και τον επιθυμητό βαθμό παραλληλισμού, στην συνέχεια ταξινομεί τον πίνακα χρησιμοποιώντας την παράλληλη έκδοση του quicksort, και τέλος εκτυπώνει το αποτέλεσμα.

Σημείωση για όλες τις παραπάνω 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).