**2η εργαστηριακή:Συγχρονισμός**

Ειρήνη Στρουμπάκου 03121183

Χαράλαμπος Παπαδόπουλος 03120199

Κάνοντας make με το υπάρχον makefile παρατηρούμε ότι δημιουργούνται δύο διαφορετικά εκτελέσιμα αρχεία: simplesync-mutex και simplesync-atomic

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

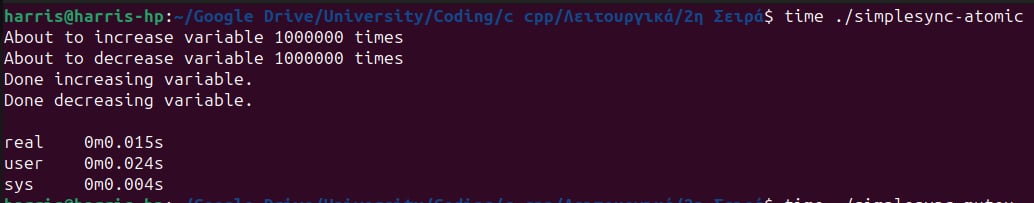
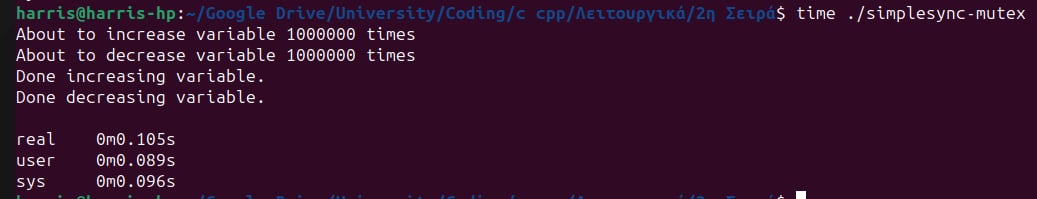
Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

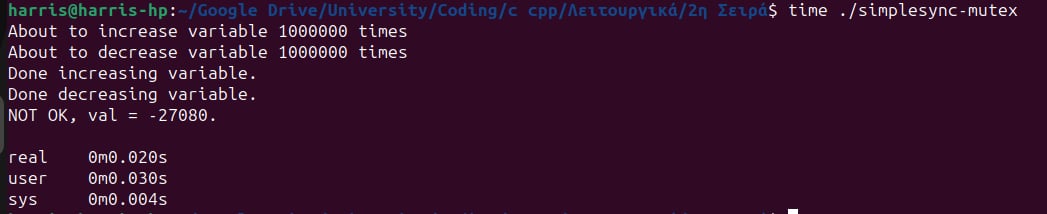
Περιγραφή που δημιουργήθηκε αυτόματα

Παρατηρούμε ότι στο makefile για την δημιουργία του εκτελέσιμου δίνονται διαφορετικές παράμετροι, αφού στο simplesync-mutex χρησιμοποιώ DSYNC\_MUTEX, ενώ στο simplesync-atomic χρησιμοποιώ DSYNC\_ATOMIC.

**1:Συγχρονισμός σε υπάρχοντα κώδικα**

1)





Παρατηρούμε ότι ο χρόνος εκτέλεσης του προγράμματος χωρίς συγχρονισμό είναι πιο γρήγορο από το πρόγραμμα με τα mutex, το οποίο είναι το αναμενόμενο, εφόσον στα mutex χρησιμοποιούμε περισσότερες εντολές.

2) Παρατηρούμε ότι ο χρόνος εκτέλεσης χρησιμοποιώντας ατομικές λειτουργίες είναι πολύ μικρότερος σε σχέση με τον χρόνο εκτέλεσης με χρήση mutex. Αυτό οφείλεται στο γεγονός ότι οι ατομικές λειτουργίες σε αντίθεση με τα mutex δεν περιλαμβάνουν διαδικασίες κλειδώματος(locking).

3)

Για την συνάρτηση increase, οι εντολές που αντιστοιχούν στις ατομικές λειτουργίες είναι οι εξής:

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, γραφικά

Περιγραφή που δημιουργήθηκε αυτόματα

Για την συνάρτηση decrease, οι εντολές που αντιστοιχούν στις ατομικές λειτουργίες είναι οι εξής:

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, γραφικά

Περιγραφή που δημιουργήθηκε αυτόματα

4) Για την συνάρτηση increase, οι εντολές που αντιστοιχούν στα mutex είναι οι εξής:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

Για την συνάρτηση decrease, οι εντολές που αντιστοιχούν στα mutex είναι οι εξής:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Περιγραφή που δημιουργήθηκε αυτόματα

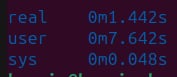
Στην 1η άσκηση δε αντιμετωπίσαμε κάποιο πρόβλημα, καθώς τα ζητούμενα της άσκησης ήταν ξεκάθαρα. Επομένως, χρησιμοποιήσαμε απλά ατομικές λειτουργίες και mutex στα αντίστοιχα κομμάτια της άσκησης.

**2: Παράλληλος υπολογισμός του συνόλου Mandelbrot**

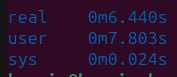
1. Επιλέγουμε το πλήθος των σημαφόρων να είναι ίσο με το πλήθος των Nthreads, εφόσον κάθε thread έχει τον δικό του σημαφόρο.

2)

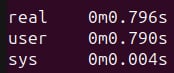
mandel με σημαφόρους



mandel με condition variables



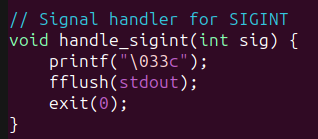
mandel με σειριακή εκτέλεση



Χρησιμοποιώντας την εντολή cat /proc/cpuinfo βλέπουμε ότι έχουμε 6 πυρήνες, οπότε τρέχουμε το πρόγραμμα για 6 threads. Η βέλτιστη επίδοση επιτυγχάνεται για 6 threads αφού κάθε thread καταλαμβάνει από έναν πυρήνα, διαφορετικά έχουμε απώλεια επίδοσης λόγω των context switches που συμβαίνουν σε κάθε πυρήνα.

3) Επιλέγουμε να έχουμε μόνο μία μεταβλητή συνθήκης το counter, το οποίο ξεκινώντας από το 0 αυξάνεται κάθε φορά κατά ένα δίνοντας την άδεια σε νέο thread με τον αντίστοιχο αριθμό να εκτελέσει τον επιθυμητό κώδικα. Μόλις φτάσει στο Nthreads το counter ξαναξεκινάει από το μηδέν. Εφόσον χρησιμοποιούμε μόνο μία μεταβλητή, σε κάθε εκτέλεση του προγράμματος “ξυπνάνε” ‘ολα τα threads και ελέγχουν εάν έχει έρθει η σειρά τους να τρέξουν. Επομένως, έχουμε πρόβλημα επίδοσης, αφού για να ολοκληρωθεί η εκτέλεση του προγράμματος μας λόγω των πολλαπλών ελέγχων χρειάζεται αρκετός χρόνος.

4) Παρατηρούμε ότι το παράλληλο πρόγραμμα δεν εμφανίζει επιτάχυνση σε σχέση με το αρχικό. Αυτό πιθανώς, για τα condition variables, οφείλεται στο πρόγραμμα επίδοσης που αναφέρθηκε και στο ερώτημα 3. Όσον αφορά τους σημαφόρους, κλειδώνουμε μόνο το μέρος του output, δηλαδή την έξοδο και όχι το compute(είσοδος). Ωστόσο, και εδώ φαίνεται ότι υπάρχουν περιθώρια βελτιστοποίησης δεδομένου ότι το πρόγραμμα μας με σημαφόρους τρέχει πιο αργά από το αρχικό.

5) Εάν πατήσουμε CTRL-C κατά την εκτέλεση του προγράμματος, τότε το τερματικό μας μένει με χρώμα και χρειάζεται την εντολή reset για να επανέλθει στην αρχική του μορφή. Οπότε, για να αποφύγουμε αυτό το πρόβλημα, χρησιμοποιώντας τον ακόλουθο κώδικα, καταφέραμε τώρα κάθε φορά που πατάμε CTRL-C να κάνει από μόνο του reset το πρόγραμμα μέσω του χειρισμού του σήματος CTRL-C.

Στην 2η άσκηση στο 1ο μέρος επιλέξαμε να δημιουργήσουμε έναν πίνακα από σημαφόρους, του οποίου το μέγεθος είναι ίσο με το πλήθος των NTHREADS. Με αυτόν τον τρόπο, κάθε thread έχει τον δικό του σημαφόρο και ενημερώνεται όταν έχει έρθει η σειρά του να δράσει. Επιπλέον, με το που τελειώσει την λειτουργία του, δηλαδή το output της γραμμής που είχε αναλάβει, ενημερώνει τον επόμενο σημαφόρο ότι πρέπει να δράσει. Έτσι, επομένως, εξασφαλίζεται ότι θα τυπωθούν τα outputs με την σωστή σειρά, αφού για να ενεργοποιηθεί ένας σημαφόρος πρέπει να λάβει σήμα από τον προηγούμενό του.

Στο 2ο μέρος, όπως αναφέρθηκε και στο ερώτημα 3, έχουμε μία condition variable, την counter, η οποία συμβολίζει τον αριθμό του thread, του οποίου είναι η σειρά να δράσει. Με το που ολοκληρωθεί το output αυξάνεται το counter , γίνεται broadcast και “ξυπνάνε” όλες οι διεργασίες και ελέγχουν αν έχει έρθει η σειρά τους, δηλαδή αν ο αριθμός τους είναι ίσος με τον counter. Αν όχι ξαναμπαίνουν σε διαδικασία sleep μέχρι το επόμενο broadcast. Το thread του οποίου ο αριθμός ισούται με το counter ξεκινάει το output.

Στο 2ο μέρος της άσκησης 2 αντιμετωπίσαμε το εξής πρόβλημα: αρχικά θεωρούσαμε ότι με το που δημιουργηθούν τα threads ξεκινούν με την προκαθορισμένη τους σειρά το πρώτο τους output και ότι έπειτα από αυτό είναι αναγκαίος ο έλεγχος με τα condition variables. Πιο συγκεκριμένα σε 4 πχ threads, τα πρώτα outputs θα ήταν 1,2,3,4. Ωστόσο, καθώς δεν υπάρχει προκαθορισμένος χρόνος εκτέλεσης του output και της δημιουργίας ενός thread, ακόμη και τα πρώτα 4 outputs βγαίνανε τυχαία. Διορθώσαμε επομένως αυτό το πρόβλημα θέτοντας εξαρχής τον έλεγχο για το counter και εξασφαλίσαμε την σωστή σειρά εκτέλεσης των threads.