Λειτουργικά Συστήματα Υπολογιστών (τμήμα 1)

3η εργαστηριακή αναφορά

Ομάδα oslaba24	
Αναστάσιος Παπαζαφειρόπουλος	el18079
Νικόλαος Παγώνας	el18175

Πηγαίος Κώδικας

Ο πηγαίος κώδικας της πρώτης και της δεύτερης άσκησης βρίσκεται στα αρχεία simplesync.c και mandel.c αντίστοιχα.

Απαντήσεις στις ερωτήσεις

Άσκηση 1

Μελετώντας προσεκτικά το παρεχόμενο Makefile, βλέπουμε ότι από το ίδιο αρχείο πηγαίου κώδικα (simplesync.c) δημιουργούνται δύο εκτελέσιμα, simplesync-atomic και simplesync-mutex. Αυτό επιτυγχάνεται με το πέρασμα των DSYNC_ATOMIC και DSYNC_MUTEX αντίστοιχα. Το πρόγραμμα ελέγχει κάθε φορά ποια από τις δύο παραμέτρους έχει περαστεί (μόνο μία από τις δύο μπορεί να περαστεί ανά εκτελέσιμο), και στην συνέχεια μέσω των if-statements εκτελούνται διαφορετικά κομμάτια κώδικα κάθε φορά.

Ερώτημα 1

Χρησιμοποιώντας την εντολή time, μπορούμε να μετρήσουμε τον χρόνο των εκτελέσιμων. Δημιουργούμε τρία εκτελέσιμα:

- Το simplesync, που προκύπτει από τον αρχικό κώδικα χωρίς καμία τροποποίηση, και άρα δεν χρησιμοποιεί κανενός είδους συγχρονισμό,
- το simplesync-mutex, που χρησιμοποιεί mutexes, και
- το simplesync-atomic, που χρησιμοποιεί GCC atomic operations

Έτσι προκύπτουν:

Για το simplesync:

```
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
NOT OK, val = 215278.

real    0m0.035s
user    0m0.063s
sys     0m0.004s
```

Για το simplesync-mutex:

```
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done decreasing variable.
Done increasing variable.
OK, val = 0.

real    0m1.060s
user    0m1.601s
sys     0m0.512s
```

Για το simplesync-atomic:

```
About to increase variable 10000000 times
About to decrease variable 10000000 times
Done increasing variable.
Done decreasing variable.
OK, val = 0.

real    0m0.340s
user    0m0.672s
sys     0m0.000s
```

Παρατηρούμε ότι ο χρόνος εκτέλεσης του αρχικού προγράμματος χωρίς συγχρονισμό είναι τουλάχιστον μία τάξη μεγέθους μικρότερος, σε σχέση με τα προγράμματα που έχουν συγχρονισμό. Αυτό είναι λογικό, καθώς δεν υπάρχει κάποιος περιορισμός στα threads, δηλαδή δεν χρειάζεται κανένα thread να περιμένει.

Ερώτημα 2

Όπως φαίνεται από τις παραπάνω εικόνες, η χρήση ατομικών λειτουργιών για τον συγχρονισμό είναι γρηγορότερη μέθοδος από τη χρήση POSIX mutexes. Αυτό συμβαίνει επειδή οι ατομικές λειτουργίες μεταφράζονται σε μία κατά προσέγγιση εντολή assembly. Ενώ, αντίθετα η έννοια των POSIX mutexes είναι μια πιο high level υλοποίηση, της οποίας η assembly εκτελεί και κλήση ρουτίνας (αυτό παρουσιάζεται εκτενέστερα στα επόμενα ερωτήματα).

Ερώτημα 3

Χρησιμοποιώντας τις παραμέτρους -S και -g, παράγουμε τα αρχεία simplesync-atomic.s και simplesync-mutex.s, τα οποία περιέχουν ενδιάμεσο κώδικα Assembly, μαζί με πληροφορίες γραμμών πηγαίου κώδικα. Έτσι, μπορούμε να δούμε σε ποιες εντολές assembly αντιστοιχεί

ο κώδικας που γράψαμε. Αγνοούμε τις γραμμές που ξεκινούν με “.”, αφού απλά βοηθούν στην περιήγηση του αρχείου.

Για την __sync_add_and_fetch(ip,1);

```
.loc 1 49 3 is_stmt 1 view .LVU16
.loc 1 54 4 view .LVU17
lock addl   $1, (%rbx)
.loc 1 48 21 view .LVU18
```

Παρατηρούμε ότι η __synd_add_and_fetch() αντιστοιχεί σε μία εντολή assembly (addl), στην οποία μάλιστα χρησιμοποιείται lock.

Για την __sync_sub_and_fetch(ip,1);

```
.loc 1 79 3 is_stmt 1 view .LVU46
.loc 1 84 4 view .LVU47
lock subl   $1, (%rbx)
.loc 1 78 21 view .LVU48
```

Και εδώ η __sync_sub_and_fetch() αντιστοιχεί σε μία εντολή assembly, subl αυτή τη φορά, αφού έχουμε αφαίρεση. Φυσικά, κι εδώ έχουμε χρήση lock.

Ερώτημα 4

Αντίστοιχα, για την εντολή: pthread_mutex_lock(&mutex);

```
.loc 1 59 0
leaq  mutex(%rip), %rbp

.loc 1 59 0
movq  %rbp, %rdi
call  pthread_mutex_lock@PLT
```

Παρατηρούμε ότι η pthread_mutex_lock(&mutex); αντιστοιχεί στην κλήση ρουτίνας (pthread_mutex_lock), η οποία υλοποιεί το lock.

Και για την εντολή: pthread_mutex_unlock(&mutex);

```
.loc 1 64 0
movq  %rbp, %rdi
.loc 1 61 0
addl  $1, %eax
movl  %eax, (%r12)
.loc 1 64 0
call  pthread_mutex_unlock@PLT
```

Παρατηρούμε ότι και εδώ καλείται η ρουτίνα pthread_mutex_unlock, στην οποία υλοποιείται το unlock.

Όπως φαίνεται και από τα παραπάνω, είναι λογικό η υλοποίηση με mutexes να απαιτεί παραπάνω χρόνο σε σχέση με την υλοποίηση με atomic operations, αφού στην πρώτη περίπτωση γίνεται κλήση ρουτίνας, ενώ στην δεύτερη περίπτωση έχουμε απλά μια εντολή assembly.

Άσκηση 2

Ερώτημα 1

Για το σχήμα συγχρονισμού που υλοποιήσαμε χρειάζονται N σημαφόροι, όπου N είναι ο αριθμός των threads. Κάθε σημαφόρος αντιστοιχεί σε ένα thread. Η λογική είναι η εξής:

Κάθε thread περιμένει να ενεργοποιηθεί ο σημαφόρος που του αντιστοιχεί προκειμένου να τυπώσει. Αφού το thread ολοκληρώσει την εκτύπωση, ενεργοποιεί τον σημαφόρο του επόμενου thread, οπότε τώρα μπορεί το επόμενο thread να τυπώσει κλπ. Όταν φτάσουμε στο τέλος, επιστρέφουμε στην αρχή, κυκλικά. Έτσι εξασφαλίζουμε ότι οι γραμμές θα εκτυπωθούν με την σωστή σειρά (ακόμα κι αν ο υπολογισμός των γραμμών γίνεται παράλληλα/με διαφορετική σειρά).

Ερώτημα 2
Παρακάτω παρουσιάζονται τα αποτελέσματα της εκτέλεσης της εντολής time(1) για:

Σειριακό πρόγραμμα:

```
real    0m0,537s
user    0m0,513s
sys     0m0,024s
```

Παράλληλο πρόγραμμα με δύο νήματα υπολογισμού:

```
real    0m0,269s
user    0m0,495s
sys     0m0,016s
```

Ερώτημα 3

Το παράλληλο πρόγραμμα που φτιάξαμε εμφανίζει επιτάχυνση. Ο χρόνος για 2 νήματα είναι σχεδόν ο μισός σε σχέση με τον χρόνο για 1 νήμα (σειριακός υπολογισμός). Αυτό συμβαίνει διότι ο υπολογισμός των γραμμών του συνόλου Mandelbrot δεν αποτελεί κομμάτι του κρίσιμου τμήματος και άρα γίνεται παράλληλα. Η εκτύπωση των γραμμών (και μόνο αυτή) είναι αναγκαστικά κομμάτι του κρίσιμου τμήματος, αλλιώς το σύνολο Mandelbrot θα τυπωνόταν ανακατεμένο. Ο υπολογισμός των γραμμών όχι μόνο δεν μας ενδιαφέρει να γίνεται σειριακά, αλλά αντίθετα ο σειριακός υπολογισμός καθυστερεί χωρίς λόγο το πρόγραμμά μας.

Ερώτημα 4

Σε περίπτωση που πατήσουμε Ctrl-C τερματίζει το πρόγραμμά μας (μέσω του σήματος SIGINT). Επιπλέον, το χρώμα των γραμμών του τερματικού θα αλλάξει και θα μετατραπεί σε αυτό που είχε επιλεχθεί για τύπωμα αμέσως πριν τον τερματισμό της εκτέλεσης του προγράμματος. Για να αποφευχθεί η αλλαγή χρώματος, μπορούμε να υλοποιήσουμε έναν δικό μας signal handler που θα πιάνει το SIGINT, ώστε αν ο χρήστης πατήσει Ctrl-C, πρώτα θα επαναφέρεται το επιλεγμένο χρώμα στη default ρύθμιση και έπειτα θα τερματίζει το πρόγραμμα.