- **<u>Άσκηση 1</u>**
    1. Ο χρόνος εκτέλεσης των εκτελέσιμων που εκτελούν συγχρονισμό είναι μεγαλύτερος σε σχέση με το χρόνο εκτέλεσης του αρχικού προγράμματος χωρίς συγχρονισμό. Αυτό συμβαίνει γιατί για να υλοποιηθεί ο συγχρονισμός χρησιμοποιούνται κλειδώματα και ατομικές λειτουργίες που καθυστερούν το πρόγραμμα, μιας και μερικά νήματα περιμένουν να ξεκλειδωθεί ένα τμήμα για να συνεχίσουν την εκτέλεσή τους.
    (+χρόνος εκτέλεσης με time(1)).
    2. Η χρήση ατομικών λειτουργιών είναι πιο γρήγορη σε σχέση με τη χρήση των POSIX mutexes. Αυτό συμβαίνει γιατί η ατομική λειτουργία υλοποιείται σε assembly με μία μόνο εντολή σε αντίθεση με τα POSIX mutexes υλοποιούνται με παραπάνω.
    3. Χρησιμοποιείται μία εντολή που προϋπάρχει υλοποιημένη στο hardware του επεξεργαστή.

    Atomic
    L2:
    .loc 1 58 0
    lock addl $1, 12(%esp) (1 εντολή για πρόσθεση)

    .L7:
    .loc 1 81 0
    lock subl $1, 12(%esp) (1 εντολή για αφαίρεση)

    4. Χρησιμοποιούνται πολλές εντολές assembly σε αντίθεση με πριν, γεγονός που εξηγεί και την αύξηση του χρόνου εκτέλεσης.

    .L2:
    .loc 1 62 0
    subl $12, %esp
    .cfi_def_cfa_offset 28
    pushl $mutex
    .cfi_def_cfa_offset 32
    call pthread_mutex_lock (5 εντολές για κλείδωμα στην πρόσθεση)

    “

.loc 1 64 0
movl $mutex, (%esp)
call pthread_mutex_unlock (2 εντολες για ξεκλείδωμα στην πρόσθεση)

.L7:
.loc 1 85 0
subl $12, %esp
.cfi_def_cfa_offset 28
pushl $mutex
.cfi_def_cfa_offset 32
call pthread_mutex_lock(5 εντολές για κλείδωμα στην αφαίρεση)

.loc 1 87 0
movl $mutex, (%esp)
call pthread_mutex_unlock(2 εντολές για ξεκλείδωμα για αφαίρεση)

- **<u>Άσκηση 2</u>**
    1. Χρειάζονται τόσοι σημαφόροι, όσα και τα παιδιά που δημιουργούνται. Στην ενδεικτική περίπτωση των τριών παιδιών, χρησιμοποιούμε τρεις σημαφόρους.

Time(1)

1 νήμα

real 0m1.262s

user 0m1.184s

sys 0m0.040s

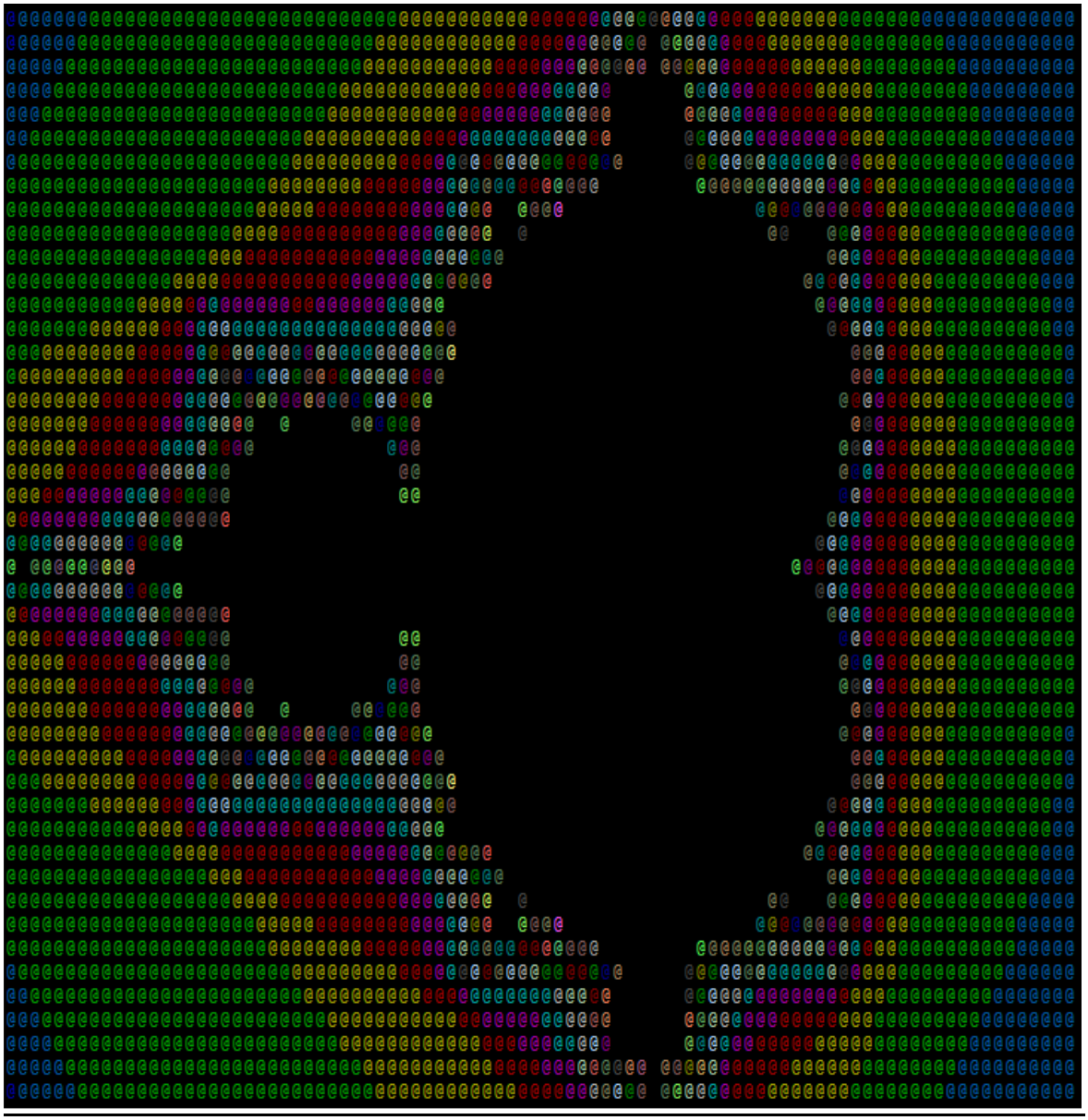
2 νήματα

real 0m0.765s
user 0m1.188s
sys 0m0.052s

    2.
    3. Στο σχήμα που υλοποιήσαμε, δεν συμπεριλάβαμε στο κρίσιμο τμήμα την φάση υπολογισμού της γραμμής αλλά μονάχα τη φάση εξόδου κάθε γραμμής που παράγεται. Επομένως, η κάθε διεργασία δεν χρειάζεται να περιμένει για να υπολογίσει την κάθε γραμμή, αλλά την υπολογίζει ταυτόχρονα με τις υπόλοιπες και συνεπώς είναι έτοιμη

μόλις έρθει η σειρά της μόνο να την τυπώσει. Συνεπώς, είναι πιο γρήγορο το πρόγραμμα.

4. Πατώντας Ctrl-C ενώ εκτελείται το πρόγραμμα, στέλνουμε σήμα –KILL στη διεργασία και καθώς δεν διαχειρίζεται αυτό το σήμα, αναγκάζεται να τερματιστεί. Επομένως, η αλλαγή των χρωμάτων των χαρακτήρων δεν επαναφέρεται και παραμένει στο χρώμα που τέθηκε τελευταίο, και συνεπώς παραμένει χρωματισμένο το terminal. Για να αντιμετωπίσουμε αυτό το πρόβλημα, θα μπορούσαμε να διαχειριζόμαστε το σήμα Ctrl-C (signal(SIGINT, sighandler)) και στη συνάρτηση sighandler() θα εκτελείται η εντολή reset_xterm_color(1), η οποία επαναφέρει το χρωματισμό στο κανονικό, και μετά η εντολή exit(1), ώστε να σηματοδοτείται ο μη κανονικός τερματισμός του προγράμματος.



- **<u>Άσκηση 3</u>**
  1. Στο σχήμα συγχρονισμού μας, επειδή χρησιμοποιούμε 2 διαφορετικά conditional variables, δίνουμε προτεραιότητα στον δάσκαλο να φύγει, αν ο αριθμός των παιδιών καλύπτεται από τους υπόλοιπους δασκάλους. Οπότε στην συγκεκριμένη περίπτωση, μόλις φύγουν αρκετά παιδιά και μπορεί να αποχωρήσει ο δάσκαλος, θα φύγει. Αντίθετα, αν φύγει ένα

παιδί, αλλά δεν επαρκούν οι δάσκαλοι για να φύγει κάποιος, τότε ένα νέο παιδί μπορεί να εισέλθει.

2. Ναι, υπάρχουν καταστάσεις συναγωνισμού. Για παράδειγμα, όταν ένα παιδί θέλει να μπει στο νηπιαγωγείο και ταυτόχρονα ένας δάσκαλος θέλει να αποχωρήσει. Αυτές τις περιπτώσεις συναγωνισμού, τις χειριζόμαστε με conditional variables και mutexes.