

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής



Εργασία Μαθήματος **Λειτουργικά Συστήματα**

Αριθμός εργασίας – Τίτλος εργασίας	1η Άσκηση - Διαδικεργασιακή επικοινωνία και το δείπνο των φιλοσόφων
Όνομα φοιτητή	Κατέβας Χρήστος
Αρ. Μητρώου	P18068
Ημερομηνία παράδοσης	07/01/2020



Εκφώνηση εργασίας

Στην ενότητα αυτή περιλάβετε την εκφώνηση της εργασίας.

Γράψτε ένα πρόγραμμα σε γλώσσα της επιλογής σας, το οποίο θα υλοποιεί το πρόβλημα του δείπνου των φιλοσόφων ως εξής:

- 1) Κάθε φιλόσοφος θα υλοποιείται ως μία διεργασία ή ένα νήμα.
- 2) Θα εμφανίζει αρχικά στο χρήστη τη δυνατότητα να ορίσει το πλήθος των φιλοσόφων (από $N = 3$ μέχρι 10). Στη συνέχεια θα δημιουργείται το αντίστοιχο πλήθος φιλοσόφων (νημάτων ή διεργασιών).
- 3) Κάθε φιλόσοφος θα παραμένει για ένα τυχαίο αριθμό δευτερολέπτων στην κατάσταση THINKING (δηλαδή blocked).
- 4) Ο κάθε φιλόσοφος θα έχει μία προτεραιότητα στην εκτέλεσή του, σε κάποια κλίμακα (π.χ. από 1 η χαμηλότερη έως 3 η υψηλότερη):
 - Η προτεραιότητα θα επιλέγεται τυχαία, συνεπώς μπορεί περισσότεροι από ένας φιλόσοφοι να έχουν την ίδια προτεραιότητα
 - Έστω ότι το κβάντο χρόνου είναι $Q=1\text{sec}$. Ο αλγόριθμος χρονοπρογραμματισμού θα πρέπει να δίνει τόσα κβάντα χρόνου σε κάθε φιλόσοφο, όση είναι η προτεραιότητά του. Π.χ. αν ένας φιλόσοφος έχει προτεραιότητα n , θα πέρνει n χρόνο εκτέλεση ίσος με $n*Q$.
 - Θα πρέπει να υπάρχει πρόβλεψη ώστε να προτιμώνται οι φιλόσοφοι με υψηλή προτεραιότητα. Αν δύο φιλόσοφοι έχουν ίδια προτεραιότητα, θα επιλέγεται εκείνος που έχει πάρει συνολικά λιγότερο χρόνο εκτέλεσης.
 - Θα πρέπει να υπάρχει πρόβλεψη ώστε να μην περιμένουν διαρκώς οι διεργασίες με χαμηλή προτεραιότητα (δηλαδή θέλουμε να πέρνουν περισσότερο χρόνο και πιο συχνά οι διεργασίες με υψηλή προτεραιότητα, αλλά να λαμβάνουν χρόνο και αυτές με χαμηλότερη προτεραιότητα).
- 5) Η διάρκεια της κατάστασης HUNGRY (ready) εξαρτάται από την πορεία εκτέλεσης του προγράμματος και δεν θα προσδιοριστεί από τον προγραμματιστή.
- 6) Κάθε φιλόσοφος θα εμφανίζει μηνύματα στην οθόνη, ώστε να δηλώνει την κατάσταση που βρίσκεται τη συγκεκριμένη ώρα του συστήματος. Όταν ο φιλόσοφος αλλάζει κατάσταση, τότε θα εμφανίζει και πάλι το αντίστοιχο μήνυμα (ένα μήνυμα για κάθε αλλαγή κατάστασης). π.χ.
 - Philosopher 1 is THINKING at time 14:50:10
 - Philosopher 2 is HUNGRY at time 14:50:12



- Philosopher 2 is EATING at time 14:50:13

7) Όταν ένας φιλόσοφος προσπαθεί να πάρει και τα δύο πιρούνια, μπορεί να πετύχει ή να αποτύχει. Να εμφανίζονται μηνύματα τα οποία να δείχνουν την επιτυχία ή αποτυχία να πάρει τα δύο πιρούνια, τη χρονική στιγμή κάθε τέτοιου γεγονότος, καθώς και τις τιμές των σημαφόρων που χρησιμοποιεί. Σε περίπτωση αποτυχίας να εμφανίζεται μήνυμα το οποίο να δηλώνει το λόγο της αποτυχίας (π.χ. Philosopher 2 failed to take fork 1, because Philosopher 1 was eating).

8) Να υπολογίσετε για κάθε φιλόσοφο το μέσο χρόνο αναμονής του για φαγητό (δηλαδή πόσο κατά μέσο όρο χρονικό διάστημα χρειάστηκε να περιμένει για να επιτύχει να πάρει τα δύο πιρούνια για να μεταβεί σε κατάσταση EATING. Τέλος να υπολογίστε το συνολικό μέσο χρόνο αναμονής για όλους τους φιλοσόφους.

9) Κάθε φιλόσοφος θα πρέπει να βρεθεί στην κατάσταση EATING συνολικά για 20 sec. Μόλις ολοκληρώσει αυτό το χρόνο, θα δηλώνει ότι τερμάτισε το δείπνο του, και θα παραμένει σε κατάσταση THINKING μέχρι να τερματίσουν όλοι οι φιλόσοφοι το δείπνο τους.

10) Το πρόγραμμα τερματίζει όταν όλοι οι φιλόσοφοι έχουν ολοκληρώσει το δείπνο τους (ο κάθε φιλόσοφος έφαγε για 20 sec).



ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1	Εισαγωγή	5
2	Τρόπος εκτέλεσης.....	6
3	Περιγραφή του προγράμματος.....	7
3.1	Struct	7
3.2	Time.....	7
3.3	*Paranoid	7
3.4	OneToRuleThemAll	9
3.5	Main	9
4	Επίδειξη της λύσης	10
5	Κώδικας	11
6	Βιβλιογραφικές Πηγές.....	21



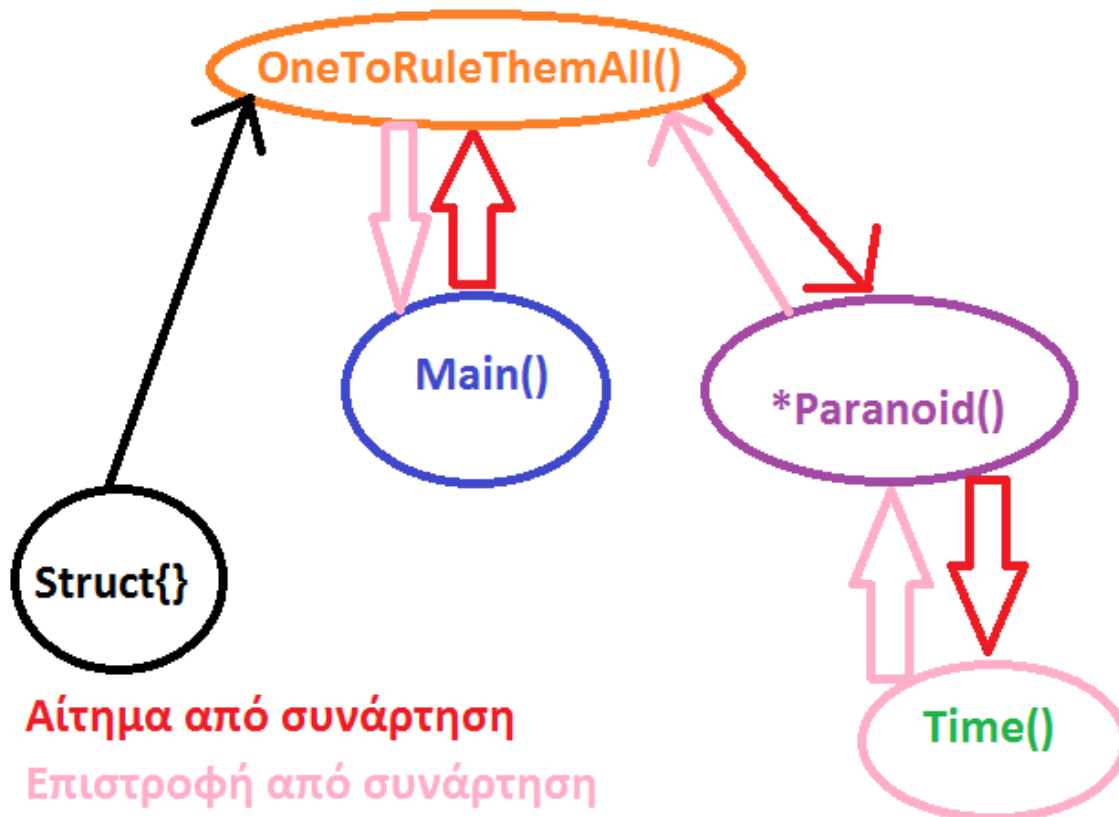
1 Εισαγωγή

Η εργασία που ακολουθεί είναι μία προσέγγιση του προβλήματος στο δείπνο των φιλοσόφων. Στόχος είναι ο συγχρονισμός των thread στην κρίσιμη περιοχή. Η υλοποίηση της «λύσης» έγινε με τη γλώσσα προγραμματισμού C. Χρησιμοποιήθηκε η βιβλιοθήκη της C για τα νήματα, pthread (POSIX threads και mutex). Το πρόγραμμα γράφτηκε και δοκιμάστηκε σε:

```
c@c:~$ cat /etc/*-release
PRETTY_NAME="Debian GNU/Linux 10 (buster)"
NAME="Debian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"

c@c:~$ uname -a
Linux c 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u2 (2019-11-
11) x86_64 GNU/Linux
```

Στην εικόνα 1.1 ακολουθεί ένα «χάρτης» του προγράμματος, με σκοπό να γίνει η κατανόηση της διαδρομής που ακολουθεί η εκτέλεση:



Εικόνα 1.1

2 Τρόπος εκτέλεσης

~Για την εκτέλεση πρέπει να κατεβάσετε την γλώσσα C και τη βιβλιοθήκη pthread

~Ανοίξτε την εφαρμογή Terminal

```
sudo apt-get install gcc
```

```
sudo apt-get install libpthread-stubs0-dev
```

~Μέσα στο φάκελο ~/ (root) υπάρχει ο φάκελος OS. Μπείτε στο φάκελο με την εξής εντολή



cd Os

~Διακρίνεται το αρχείο os.c το κάνετε compile με:

gcc os.c -lpthread

~Δημιουργείται το εκτελέσιμο αρχείο a.out τρέξτε το χρησιμοποιώντας το παρακάτω:

./a.out

~Μπορεί να χρησιμοποιηθεί το « time ./a.out » για να δείτε ακριβώς το χρόνο εκτέλεσης.

3 Περιγραφή του προγράμματος

Ακολουθεί μία περιγραφή του κώδικα. Πρώτα το struct και μετά οι συναρτήσεις. (Για την αναλυτική εξήγηση του κώδικα Βλέπε ενότητα 5 ή το αρχείο os.c. Περιέχονται λεπτομερής σχόλια για κάθε κομμάτι του κώδικα.)

3.1 Struct

Το Struct χρησιμοποιείται για την δόμηση ενός (ή πολλών στοιχείων) Στη συγκεκριμένη περίπτωση τα στοιχεία που δημιουργούνται αντιπροσωπεύουν ένα φιλόσοφο/νήμα/διεργασία. Μέσα στο struct ορίζεται ένα μπλοκ μεταβλητών, τύπου thread, mutex, ακεραίων, δεκαδικών.

3.2 Time

Συνάρτηση που χρησιμοποιείται για την εύρεση της τοπικής ώρας. Η συνάρτηση επιστρέφει ένα πίνακα χαρακτήρων (string) με την τοπική ώρα/ημερομηνία.

3.3 *Paranoid

Η συνάρτηση «εκτελεί» τον εκάστοτε φιλόσοφο/νήμα. Δέχεται σαν όρισμα ένα δείκτη, που αντιπροσωπεύει τον φιλόσοφο, μαζί με τα στοιχεία που τον διέπουν.

Τέλος επιστρέφει NULL. Η διαδικασία εκτέλεσης ενός νήματος έχει ως εξής όσο ο φιλόσοφος δεν έχει συμπληρώσει 20 δευτερόλεπτα στην κατάσταση EATING επαναλαμβάνει τα παρακάτω: Ο φιλόσοφος «κοιμάται» για τυχαία δευτερόλεπτα (κοιμάται από 1-3 δευτερόλεπτα για συντομία του συνολικού χρόνου εκτέλεσης του προγράμματος. Γίνεται προσπάθεια να «κλειδωθεί» το δεξί πιρούνι. Γίνεται προσπάθεια να «κλειδωθεί» το αριστερό πιρούνι. Αν δεν τα χρησιμοποιεί άλλος φιλόσοφος, τότε κλειδώνονται και ο φιλόσοφος τρώει. Αλλιώς περιμένει, μέχρι να είναι διαθέσιμο/α τα πιρούνι/α που του λείπουν για να φάει. (Η διαδικασία περιγράφεται καλύτερα στην Εικόνα 1.1 που ακολουθεί)



Εικόνα 2.1

Αφού ο φιλόσοφος «γευματίσει» όση ώρα του αναλογεί (όσο δηλαδή η προτεραιότητά του) «ξεκλειδώνει» τα πιρούνια, για να τα χρησιμοποιήσει ο επόμενος.

Η επιλογή για το ποιος φιλόσοφος θα φάει εξαρτάται από το πόση ώρα θα κοιμάται ένας φιλόσοφος μέχρι να πάρει τα πιρούνια. Παρατηρούμε έτσι ότι όλοι οι φιλόσοφοι γευματίζουν με «τυχαία» σειρά. Έτσι η προτεραιότητα της εκάστοτε διεργασίας/νήματος εξαρτάται ΜΟΝΟ από την προτεραιότητα της. Άρα τα νήματα με μεγάλη προτεραιότητα θα έχουν την ίδια «τύχη» στο να εκτελεστούν όσο και τα νήματα με μικρή προτεραιότητα. Η διαφορά είναι ότι τα μεγάλης προτεραιότητας



νήματα θα έχουν περισσότερο χρόνο εκτέλεσης, τη φορά. (Πχ νήμα προτεραιότητας 3, θα εκτελείται για 3 δευτερόλεπτα τη φορά. Ενώ νήμα με προτεραιότητα 1 θα εκτελείται 1 δευτερόλεπτο τη φορά. Μόλις συμπληρωθούν τα 20 δευτερόλεπτα, ο εκάστοτε φιλόσοφος δηλώνει ότι τέλειωσε το γεύμα του και αποχωρεί.

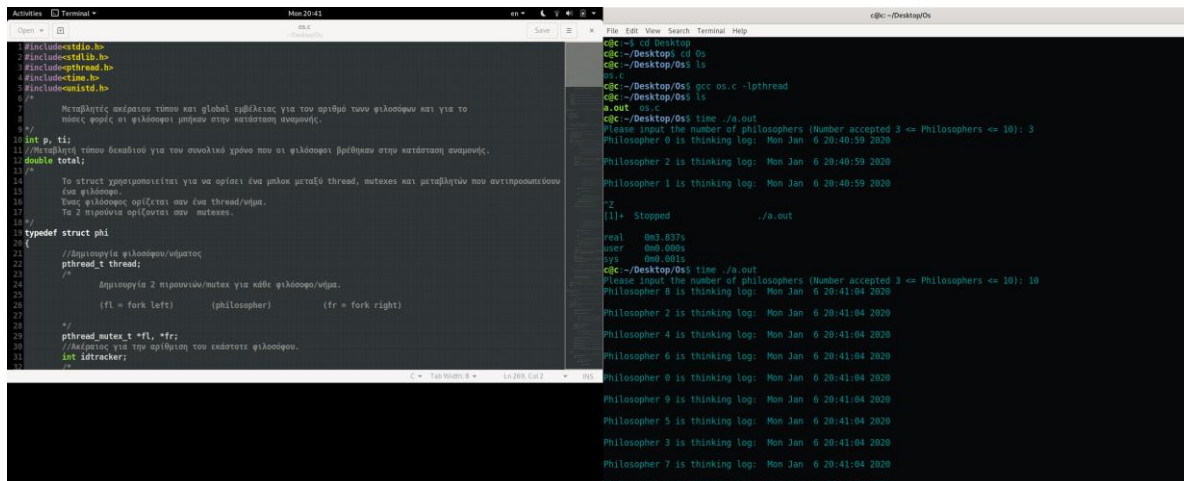
3.4 OneToRuleThemAll

Η συνάρτηση δέχεται σαν όρισμα τον αριθμό των φιλοσόφων/νημάτων που θα δειπνήσουν/εκτελεστούν. Τέλος επιστρέφει NULL. Αρχικοποιούνται τα στοιχεία του κάθε φιλοσόφου, με τη βοήθεια μιας for_loop, όπως (πχ ο αύξοντα αριθμός του φιλοσόφου θα ισούται με την μεταβλητή που χρησιμοποιεί η for_loop, η προτεραιότητά του, το νήμα που θα εκτελεστεί, ο αριθμός των πιρουνιών που θα χρησιμοποιήσει.) Ακολουθεί while_loop που εκτελείται κάθε φορά που ένας φιλόσοφος τελειώνει το γεύμα του. Γίνεται μέτρηση των φιλοσόφων που έχουν τερματίσει το δείπνο τους. Αν ο αριθμός τους είναι ίσος με τον αριθμό που έδωσε ο χρήστης (Βλέπε συνάρτηση main) τότε τερματίζει το γεύμα. Όλοι οι φιλόσοφοι/νήματα «σκοτώνονται» και βγαίνουν τα αντίστοιχα μνήματα. Υπολογίζεται ο χρόνος αναμονής κάθε φιλοσόφου (Χρόνος που παρέμεινε hungry, πόση ώρα δηλαδή πήρε μέχρι να αποκτήσει και τα 2 πιρούνια για να φάει.) ως εξής: Χρόνος Αναμονής του Φιλοσόφου / Φορές Αναμονής του φιλοσόφου. Τέλος με την ίδια πράξη εκτυπώνεται χρόνος αναμονής, σαν μέσος όρος, για όλους τους φιλοσόφους: Συνολικός Χρόνος Αναμονής όλων των Φιλοσόφων / Συνολικές Φορές Αναμονής όλων των Φιλοσόφων.

3.5 Main

Ο χρήστης καλείται να δώσει ένα αριθμό φιλοσόφων (μεταξύ 3^{ων} και 10 φιλοσόφων). Όταν ο χρήστης δώσει «σωστό» όρισμα, καλείται η συνάρτηση OneToRuleThemAll(Με όρισμα τον αριθμό των φιλοσόφων).

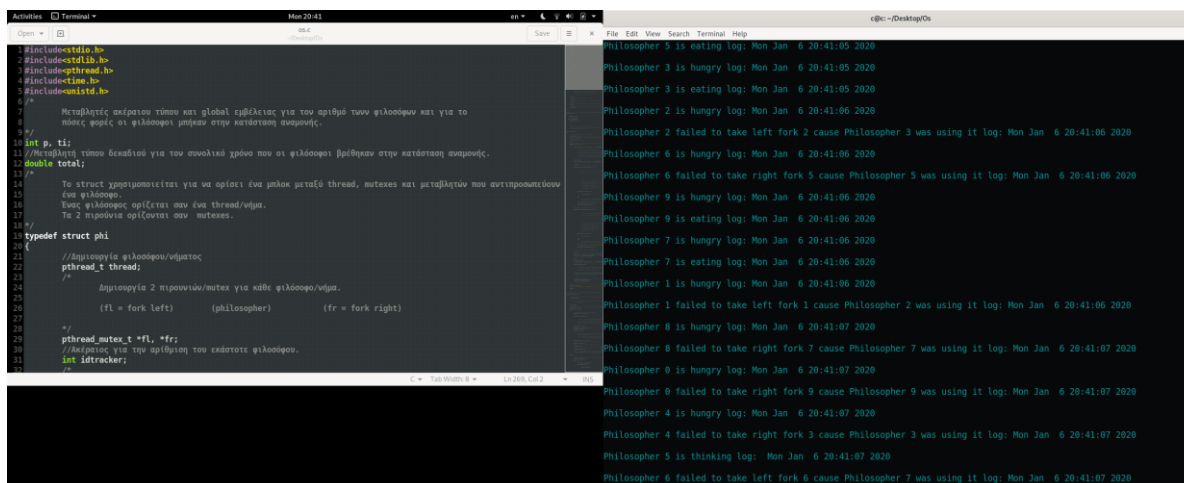
4 Επίδειξη της λύσης



```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<pthread.h>
4 #include<time.h>
5 #include<unistd.h>
6 /*
7  * Μεταβλητές ακέραιου τύπου και global εμβέλειες για τον αριθμό των φιλοσόφων και για το
8  * πόσες φορές οι φιλόσοφοι μπήκαν στην κατάσταση αναμονής.
9  */
10 int p, ti;
11 /*Μεταβλητή τύπου δεκαδικού για τον συνολικό χρόνο που οι φιλόσοφοι βρέθηκαν στην κατάσταση αναμονής.
12 double total;
13 */
14 /* To struct χρησιμοποιείται για να ορίσει ένα μικρό μεταξὺ thread, mutexes και μεταβλητών που αντιπροσωπεύουν
15  * ένα φιλόσοφο.
16  * Ένας φιλόσοφος ορίζεται σαν ένα thread/νήμα.
17  * Τα 2 πηρούνια ορίζονται σαν mutexes.
18  */
19 typedef struct phi
20 {
21     //Δημιουργία φιλοσόφου/νήματος
22     pthread_t thread;
23     /*
24     * Δημιουργία 2 πηρούνιων/mutex για κάθε φιλόσοφο/νήμα.
25     */
26     (fl = fork left) (philosopher) (fr = fork right)
27 }
28 /*
29 * pthread_mutex_t *fl, *fr;
30 * //Αίματος για την απόκτηση του εκδότου φιλοσόφου.
31 int idtracker;
32 */
```

```
c@C:~/Desktop/OS
c@C:~/Desktop/OS$ gcc os.c -lpthread
c@C:~/Desktop/OS$ ls
a.out os.c
c@C:~/Desktop/OS$ time ./a.out
Please input the number of philosophers (Number accepted 3 <= Philosophers <= 10): 3
Philosopher 0 is thinking log: Mon Jan 6 20:40:59 2020
Philosopher 2 is thinking log: Mon Jan 6 20:40:59 2020
Philosopher 1 is thinking log: Mon Jan 6 20:40:59 2020
^Z
[1]+  Stopped                  ./a.out
real 0m3.837s
user 0m0.000s
sys 0m0.001s
c@C:~/Desktop/OS$ time ./a.out
Please input the number of philosophers (Number accepted 3 <= Philosophers <= 10): 10
Philosopher 0 is thinking log: Mon Jan 6 20:41:04 2020
Philosopher 2 is thinking log: Mon Jan 6 20:41:04 2020
Philosopher 4 is thinking log: Mon Jan 6 20:41:04 2020
Philosopher 6 is thinking log: Mon Jan 6 20:41:04 2020
Philosopher 8 is thinking log: Mon Jan 6 20:41:04 2020
Philosopher 9 is thinking log: Mon Jan 6 20:41:04 2020
Philosopher 5 is thinking log: Mon Jan 6 20:41:04 2020
Philosopher 3 is thinking log: Mon Jan 6 20:41:04 2020
Philosopher 7 is thinking log: Mon Jan 6 20:41:04 2020
Philosopher 1 is thinking log: Mon Jan 6 20:41:04 2020
```

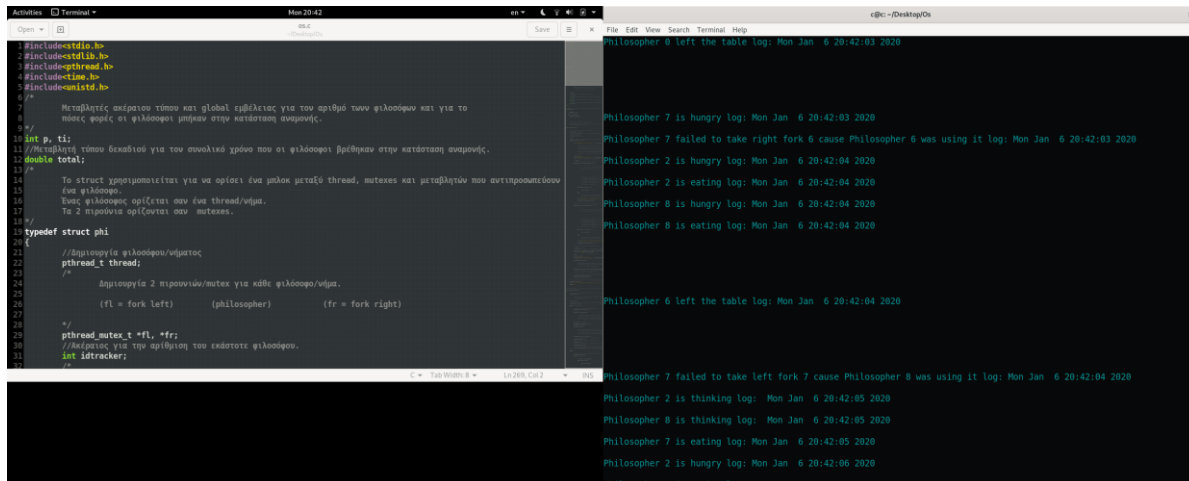
Εικόνα 3.1



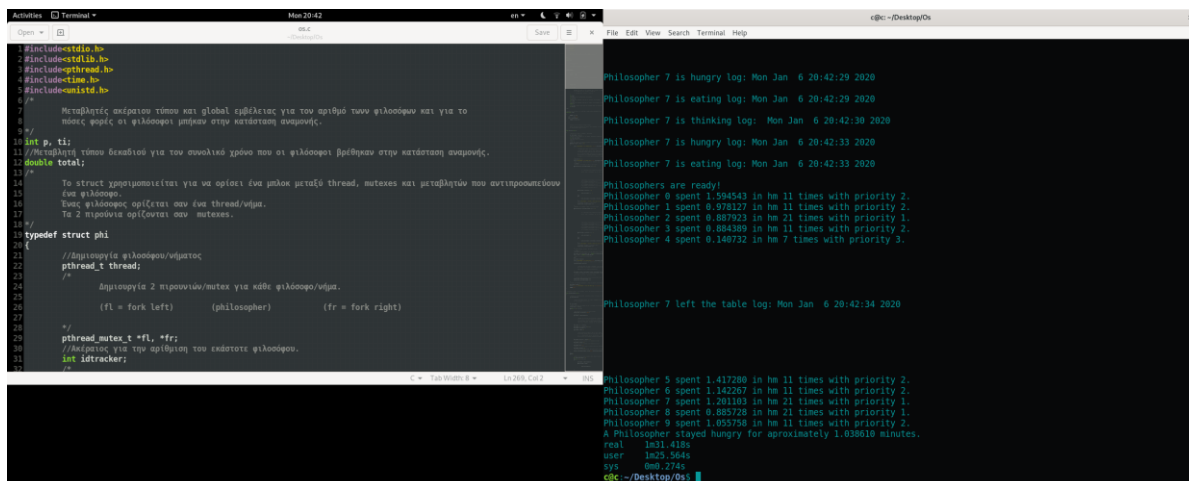
```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<pthread.h>
4 #include<time.h>
5 #include<unistd.h>
6 /*
7  * Μεταβλητές ακέραιου τύπου και global εμβέλειες για τον αριθμό των φιλοσόφων και για το
8  * πόσες φορές οι φιλόσοφοι μπήκαν στην κατάσταση αναμονής.
9  */
10 int p, ti;
11 /*Μεταβλητή τύπου δεκαδικού για τον συνολικό χρόνο που οι φιλόσοφοι βρέθηκαν στην κατάσταση αναμονής.
12 double total;
13 */
14 /* To struct χρησιμοποιείται για να ορίσει ένα μικρό μεταξὺ thread, mutexes και μεταβλητών που αντιπροσωπεύουν
15  * ένα φιλόσοφο.
16  * Ένας φιλόσοφος ορίζεται σαν ένα thread/νήμα.
17  * Τα 2 πηρούνια ορίζονται σαν mutexes.
18  */
19 typedef struct phi
20 {
21     //Δημιουργία φιλοσόφου/νήματος
22     pthread_t thread;
23     /*
24     * Δημιουργία 2 πηρούνιων/mutex για κάθε φιλόσοφο/νήμα.
25     */
26     (fl = fork left) (philosopher) (fr = fork right)
27 }
28 /*
29 * pthread_mutex_t *fl, *fr;
30 * //Αίματος για την απόκτηση του εκδότου φιλοσόφου.
31 int idtracker;
32 */
```

```
Philosopher 5 is eating log: Mon Jan 6 20:41:05 2020
Philosopher 3 is hungry log: Mon Jan 6 20:41:05 2020
Philosopher 3 is eating log: Mon Jan 6 20:41:05 2020
Philosopher 2 is hungry log: Mon Jan 6 20:41:06 2020
Philosopher 2 failed to take left fork 2 cause Philosopher 3 was using it log: Mon Jan 6 20:41:06 2020
Philosopher 6 is hungry log: Mon Jan 6 20:41:06 2020
Philosopher 6 failed to take right fork 5 cause Philosopher 5 was using it log: Mon Jan 6 20:41:06 2020
Philosopher 9 is hungry log: Mon Jan 6 20:41:06 2020
Philosopher 9 is eating log: Mon Jan 6 20:41:06 2020
Philosopher 7 is hungry log: Mon Jan 6 20:41:06 2020
Philosopher 7 is eating log: Mon Jan 6 20:41:06 2020
Philosopher 1 is hungry log: Mon Jan 6 20:41:06 2020
Philosopher 1 failed to take left fork 1 cause Philosopher 2 was using it log: Mon Jan 6 20:41:06 2020
Philosopher 8 is hungry log: Mon Jan 6 20:41:07 2020
Philosopher 8 failed to take right fork 7 cause Philosopher 7 was using it log: Mon Jan 6 20:41:07 2020
Philosopher 0 is hungry log: Mon Jan 6 20:41:07 2020
Philosopher 0 failed to take right fork 9 cause Philosopher 9 was using it log: Mon Jan 6 20:41:07 2020
Philosopher 4 is hungry log: Mon Jan 6 20:41:07 2020
Philosopher 4 failed to take right fork 3 cause Philosopher 3 was using it log: Mon Jan 6 20:41:07 2020
Philosopher 5 is thinking log: Mon Jan 6 20:41:07 2020
Philosopher 6 failed to take left fork 6 cause Philosopher 2 was using it log: Mon Jan 6 20:41:07 2020
```

Εικόνα 3.2



Εικόνα 3.3



Εικόνα 3.4

5 Κώδικας

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<time.h>
#include<unistd.h>

/*
```



Μεταβλητές ακέραιου τύπου και global εμβέλειας για τον αριθμό των φιλοσόφων και για το

πόσες φορές οι φιλόσοφοι μπήκαν στην κατάσταση αναμονής.

*/

int p, ti;

//Μεταβλητή τύπου δεκαδικού για τον συνολικό χρόνο που οι φιλόσοφοι βρέθηκαν στην κατάσταση αναμονής.

double total;

/*

Το struct χρησιμοποιείται για να ορίσει ένα μπλοκ μεταξύ thread, mutexes και μεταβλητών που αντιπροσωπεύουν

ένα φιλόσοφο.

Ένας φιλόσοφος ορίζεται σαν ένα thread/νήμα.

Τα 2 πιρούνια ορίζονται σαν mutexes.

*/

typedef struct phi

{

//Δημιουργία φιλοσόφου/νήματος

pthread_t thread;

/*

Δημιουργία 2 πιρουνιών/mutex για κάθε φιλόσοφο/νήμα.

(fl = fork left) (philosopher) (fr = fork right)

*/

pthread_mutex_t *fl, *fr;

//Ακέραιος για την αρίθμηση του εκάστοτε φιλοσόφου.

int idtracker;

/*

Μεταβλητή που ορίζει αν ένας φιλόσοφος έχει τελειώσει το γεύμα του.

0: ο φιλόσοφος δεν έχει τελειώσει το γεύμα του.

1: ο φιλόσοφος έχει τελειώσει το γεύμα του.

*/



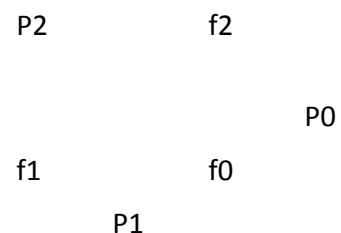
```
int ready;
//Μεταβλητή για την προτεραιότητα του φιλοσόφου.
int priority;
//Μεταβλητή που καταγράφει πόση ώρα έχει δειπνήσει ένας φιλόσοφος.
int tiktok;
//Μεταβλητή που καταγράφει πόση ώρα έχει περάσει ένας φιλόσοφος στο Hungry
mode.
double hm;
//Η μεταβλητή μετράει πόσες φορές ένας φιλόσοφος μπήκε στην κατάσταση
αναμονής.
int hmtimes;
} ph;

//Συνάρτηση που επιστρέφει την τοπική ώρα σε μορφή πίνακα χαρακτήρων/string.
const char* Time()
{
    struct tm* local;
    time_t t = time(NULL);
    local = localtime(&t);
    return asctime(local);
}

/*
    Συνάρτηση για την εκτέλεση του κάθε φιλοσόφου.
    Δέχεται σαν όρισμα των pointer που δείχνει ποιος φιλόσοφος είναι από αυτούς
    που κατασκευάστηκες βάση του struct.
*/
void *Paranoid(void *x)
{
    //Ορίζεται σαν "πρώτος" φιλόσοφος ο: philosopher.
    ph *philosopher = (ph *)x;
    //Ακεραίου τύπου μεταβλητές για τον επόμενο, προηγούμενο φιλόσοφο και το bonus.
    int other_philosopher;
```



```
//Ορισμούς ρολογιού με όνομα t.  
clock_t t;  
//Loop: Όσο ο χρόνος του εκάστοτε φιλοσόφου είναι μικρότερος των 20sec συνέχισε.  
while(philosopher->tiktok <= 20)  
{  
    printf("Philosopher %d is thinking log: %s \n", philosopher->idtracker, Time());  
    /*  
        Ο φιλόσοφος/νήμα τίθεται σε κατάσταση ύπνου για "τυχαίο χρόνο"  
        Ο τυχαίος χρόνος βγαίνει από τον τύπο: (Τυχαίος ακέραιος modulo 3)  
        που επιστρέφει αριθμός μεταξύ του 0 και του 2 συν 1(για την περίπτωση  
του 0 ή πολλαπλασιών του 3).  
        Έτσι ο το νήμα θα κοιμηθεί μεταξύ των 1 και 3 δευτερολέπτων.  
        Αντίστοιχα αντί για 3 μπορούμε να βάλουμε οποιοδήποτε αριθμό.  
        Επιλέχθηκε το 3 για συντομία χρόνου εκτέλεσης του προγράμματος  
συνολικά.  
    */  
    sleep(rand()%3 + 1);  
    printf("Philosopher %d is hungry log: %s \n", philosopher->idtracker, Time());  
    //Αρχή του ρολογιού (Μέτρηση χρόνου στην κατάσταση hungry.  
    t = clock();  
    //Προσπάθεια του φιλοσόφου να πάρει το δεξί πιρούνι.  
    if(pthread_mutex_trylock(philosopher->fr) != 0)  
    {  
        /*  
            Για το αριστερό πιρούνι διακρίνου την εξής πείρρωση.  
            Έστω ότι έχουμε 3 φιλοσόφους, P0, P1, P2 και τα αντίστοιχα  
πιρούνια f0, f1, f2.
```





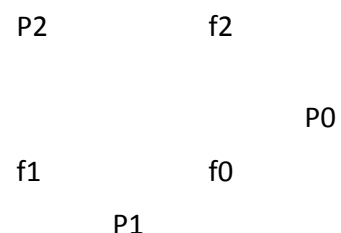
Αν βρισκόμαστε στον Φιλόσοφο 0 τότε το δεξί πιρούνι είναι το 2
και ο φιλόσοφος στα δεξιά είναι ο 2.

Αν βρισκόμαστε στον Φιλόσοφο 1 τότε το δεξί πιρούνι είναι το 0
και ο φιλόσοφος στα δεξιά είναι ο 0.

Αν βρισκόμαστε στον Φιλόσοφο 2 τότε το δεξί πιρούνι είναι 1
και ο φιλόσοφος στα δεξιά είναι ο 1.

```
*/  
if(philosopher->idtracker == 0)  
{  
    other_philosopher = p - 1;  
}  
else  
{  
    other_philosopher = philosopher->idtracker - 1;  
}  
printf("Philosopher %d failed to take right fork %d cause Philosopher %d  
was using it log: %s \n",philosopher->idtracker, other_philosopher, other_philosopher, Time());  
//Ο φιλόσοφος "κλειδώνει" το δεξί πιρούνι.  
pthread_mutex_lock(philosopher->fr);  
}  
//Προπάθεια του φιλοσόφου να πάρει το αριστερό πιρούνι.  
if(pthread_mutex_trylock(philosopher->fl) != 0)  
{  
    /*
```

Για το αριστερό πιρούνι διακρίνου την εξής πείπρωση.
Έστω ότι έχουμε 3 φιλοσόφους, P0, P1, P2 και τα αντίστοιχα
πιρούνια f0, f1, f2.





Αν βρισκόμαστε στον Φιλόσοφο 0 τότε το αριστερό πιρούνι είναι το 0

και ο φιλόσοφος στα αριστερά είναι ο 1.

Αν βρισκόμαστε στον Φιλόσοφο 1 τότε το αριστερό πιρούνι είναι το 1

και ο φιλόσοφος στα αριστερά είναι ο 2.

Αν βρισκόμαστε στον Φιλόσοφο 2 τότε το αριστερό πιρούνι είναι το 2

και ο φιλόσοφος στα αριστερά είναι ο 0.

 $\ast/$

```
if(philosopher->idtracker == p - 1)
```

 $\{$

```
other_philosopher = 0;
```

}

else

{

```
other_philosopher = philosopher->idtracker + 1;
```

}

```
printf("Philosopher %d failed to take left fork %d cause Philosopher %d  
was using it log: %s \n",philosopher->idtracker, philosopher->idtracker, other_philosopher,  
Time());
```

//Ο φιλόσοφος "κλειδώνει" το αριστερό πιρούνι.

```
pthread_mutex_lock(&philosopher->fl);
```

}

//Ο χρόνος αναμονής τελειώνει. Άρα σταματά το ρολόι.

```
t = clock() -t;
```

//Καταχώρηση της ώρας αναμονής, πόση ώρα πέρασε μέχρι ο φιλόσοφος να πάρει και τα δύο πιρούνια.

```
philosopher->hm = philosopher->hm + ((double)t)/CLOCKS_PER_SEC;
```

```
total = total + ((double)t)/CLOCKS_PER_SEC;
```

```
//Αύξηση του μετρητή για το Hungry mode.
```

ti++;

```
philosopher->hmtimes++;
```

```
printf("Philosopher %d is eating log: %s \n", philosopher->idtracker, Time());
```




```
//Ο φιλόσοφος τρώει. Κοιμάται όσα seconds είναι και η προτεραιότητά του.
sleep(philosopher->priority);
/*
    Ο συνολικός χρόνος που έφαγε ο φιλόσοφος, είναι όσο και η
    προτεραιότητά του.
    Οπότε μπορούμε να μετρήσουμε το συνολικό χρόνο μόνο με ένα
    ακέραιο,
    αυξάνοντάς τον όσο η εκάστοτε προτεραιότητα.
*/
philosopher->tiktok = philosopher->tiktok + philosopher->priority;
/* Παρομοίως θα συνέβαινε και στον πίνακα που κρατούσε το χρόνο.
ti[philosopher->idtracker] = philosopher->tiktok;
*/
//Ο φιλόσοφος πέρασε priority seconds στο Eating mode. Οπότε "ξεκλειδώνει"
τα πιρούνια.
pthread_mutex_unlock(philosopher->fr);
pthread_mutex_unlock(philosopher->fl);
}
//Η λήξη της επανάληψης σηματοδοτεί τη λήξη του γεύματος του φιλοσόφου.
philosopher->ready = 1;
printf("\n\n\n\nPhilosopher %d left the table log: %s \n\n\n\n\n", philosopher-
>idtracker, Time());
return NULL;
}

//Η συνάρτηση "ελέγχει" τους φιλοσόφους. Δέχεται σαν όρισμα των αριθμό των φιλοσόφων.
void OneToRuleThemAll(int p)
{
    //Πίνακας των threads/φιλοσόφων. Δημιουργία p θέσεων σε πίνακα όσoι και οι
    φιλόσοφοι.
    ph philosophers[p];
    /*
        Ο pointer Philosopher ορίζεται ως ο πρώτος φιλόσοφος και δείχνει την αρχή
        της "αλυσίδας" στοιχείων
    */
}
```



που θα δημιουργηθούν από το struct

```
*/  
ph *philosopher;  
//Πίνακας τύπου mutex και μεγέθους όσο και οι φιλόσοφοι, που αντιπροσωπεύει τα  
"πιδούνια".  
pthread_mutex_t forks[p];  
for(int i = 0; i < p; i++)  
{  
    //Αρχικοποίηση των στοιχείων του κάθε φιλοσόφου.  
    pthread_mutex_init(&forks[i], NULL);  
    //ti[i] = 0;  
    philosopher = &philosophers[i];  
    /*  
        Ο αριθμός των πιδουνιών που αντιστοιχούν σε κάθε φιλόσοφο.  
        Το δεξί πιδούνι είναι απλά ο αριθμός του φιλοσόφου.  
        Το αριτερό πιδούνι είναι το modulo μεταξύ του μετρητή (αυξημένου  
κατά 1 για την περίπτωση που  
είμαστε στον "τελευταίο" φιλόσοφο) και του συνολικού αριθμού  
φιλοσόφων.  
    */  
    philosopher->fr = &forks[i];  
    if(i == 0)  
    {  
        philosopher->fl = &forks[p - 1];  
    }  
    else  
    {  
        philosopher->fl = &forks[i - 1]  
    }  
    philosopher->idtracker = i;  
    philosopher->ready = 0;  
    /*
```



Η τυχαία προτεραιότητα βγαίνει από τον τύπο: (Τυχαίος ακέραιος modulo 3)

που επιστρέφει αριθμός μεταξύ του 0 και του 2 συν 1(για την περίπτωση του 0 ή πολλαπλασίων του 3).

Έτσι ο το νήμα/φιλόσοφος θα έχει προτεραιότητα, ισοπίθανα, μεταξύ των 1 και 3.

Αντίστοιχα αντί για 3 μπορούμε να βάλουμε οποιοδήποτε αριθμό-προτεραιότητα.

```
*/
philosopher->priority = rand() % 3 + 1;
//Χρόνος που περνά ο φιλόσοφος στην κατάσταση που τρώει.
philosopher->tiktok = 0;
//Χρόνος που περνά ο φιλόσοφος στην κατάσταση που πεινά.
philosopher->hm = 0.0;
//Μετρητής για το πόσες φορές ο φιλόσοφος μπήκε στην κατάσταση που
πεινά.
philosopher->hmtimes = 0;
//Δημιουργία του φιλοσόφου/thread. Με κρίσημο σημείο την συνάρτηση
*Paranoid.
pthread_create(&philosopher->thread,NULL,Paranoid,philosopher);
}
while(1)
{
    //Μετράει κάθε φορά πόσες διεργασίες έχουν πλήρως εκτελεστεί,
    "διατρέχοντας" το struct.
    int ready_philosophers = 0;
    for(int i = 0; i < p; i++)
    {
        philosopher = &philosophers[i];
        if(philosopher->ready )
        {
            ready_philosophers++;
        }
    }
}
```



```
/*
    Εάν ο μετρητής ready_philosophers ισούται με τον αριθμό των
    φιλοσόφων, τότε διατρέχουμε ένα προς ένα τα thread "σκοτώνοντάς"
    τα.

    Μαζί και τα πιρούνια αντίστοιχα.
*/
if(ready_philosophers == p)
{
    printf("Philosophers are ready! \n");
    for(int i = 0; i < p; i++)
    {
        philosopher = &philosophers[i];
        pthread_join(philosopher->thread, NULL);
        pthread_mutex_destroy(&forks[i]);
        printf("Philosopher %d spent %f in hm %d times with priority
%d.\n", philosopher->idtracker, philosopher->hm/philosopher->hmtimes, philosopher-
>hmtimes, philosopher->priority);
    }
    printf("A Philosopher stayed hungry for aproximately %f minutes.",
total/ti);
    break;
}
}

//Κύρια συνάρτηση.
void main(int argc, char *argv[])
{
    while(1)
    {
        printf("Please input the number of philosophers (Number accepted 3 <=
Philosophers <= 10): ");
        scanf("%d", &p);
        if( (p>=3) && (p<=10) )
```



```
        {  
            break;  
        }  
    }  
    ti = 0;  
    total = 0.0;  
    //Συνάρτηση για τη δημιουργία και τον έλεγχο των φιλοσόφων.  
    OneToRuleThemAll(p);  
}
```

6 Βιβλιογραφικές Πηγές

1. https://rosettacode.org/wiki/Dining_philosophers?fbclid=IwAR35BnQp0sIrl1RIOgzGQecL5WmYUMMp_UOXIEjkNWpPhQyq12K5Tn50w7k
2. <https://www.geeksforgeeks.org/dining-philosopher-problem-using-semaphores/?fbclid=IwAR0Gj0BNkCSMP7JQgefViqhF217WtzFrFWZNkEMu7p3dFo3fNbdSSfVSD5A>
3. <https://www.geeksforgeeks.org/time-h-localtime-function-in-c-with-examples/?fbclid=IwAR1L00sITWTRB8q1P8dzpWghlvuZhEYFpyKdVCdLcSgHTpESeWEvPeM0-k>
4. <https://www.geeksforgeeks.org/multithreading-c-2/>
5. <https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/>
6. <https://stackoverflow.com/questions/20276010/c-creating-n-threads?fbclid=IwAR1p-DwSTm295X-SLFTzBXT0zWITWhd1pSZatibNLtZYLWaX4i4hf4laWg>
7. https://el.wikipedia.org/wiki/%CE%9D%CE%AE%CE%BC%CE%B1%CF%84%CE%B1_P_OSIX
8. The C Programming Language. Book by Brian Kernighan and Dennis Ritchie (https://upload.wikimedia.org/wikipedia/commons/0/07/C_Programming.pdf)
9. Modern Operating Systems. Book by Andrew S. Tanenbaum