ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

Εργασία Μαθήματος ***Λειτουργικά Συστήματα***

Αριθμός εργασίας – Τίτλος εργασίας	1η Άσκηση - Διαδιεργασιακή επικοινωνία και το δείπνο των φιλοσόφων
Όνομα φοιτητή	Φώτης-Άγγελος Σταυρόπουλος
Αρ. Μητρώου	Π18145
Ημερομηνία παράδοσης	7/1/2020

Εκφώνηση 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=1sec. 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).

Οδηγίες:

1. Η εργασία είναι **προαιρετική και ατομική.** Σε περίπτωση που την υλοποιήσετε θα λάβετε bonus βαθμολογία επί του τελικού βαθμού σας.
2. Το παραδοτέο θα περιλαμβάνει: (α) το έγγραφο το οποίο θα περιλαμβάνει αναλυτική περιγραφή του κώδικα και ενδεικτικά screenshots από την εκτέλεση του προγράμματος. (β) τον πηγαίο κώδικα και (γ) το εκτελέσιμο αρχείο.
   Η παράδοση θα γίνει **αποκλειστικά μέσω του eclass σε ένα .zip ή .rar αρχείο.**
3. Για τη συγγραφή της άσκησης θα χρησιμοποιήσετε το template που θα βρείτε στα έγγραφα του μαθήματος.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1 Εισαγωγή

Το πρόβλημα που κλήθηκα να επιλύσω αφορούσε μια παραλλαγή του δείπνου των φιλοσόφων(Dining philosophers problem) του Edsger Dijkstra. Η παραλλαγή του αφορούσε άγνωστο αριθμό φιλοσόφων που δίνεται απ'τον χρήστη κατά την έναρξη του προγράμματος(αριθμό μεταξύ του 3 και του 10 καθώς έτσι ζητήθηκε).Η δική μου λύση υλοποιήθηκε χρησιμοποιώντας την γλώσσα προγραμματισμού C. Για την εκτέλεση του προγράμματος δίδονται οδηγίες εδώ:
Σε **Ubuntu** λειτουργικό σύστημα, **αποθηκεύστε** το αρχείο στο **Desktop**.Ανοίξτε το **Terminal** ύστερα **μεταφερθείτε** στο **Desktop** με την εντολή***«cd Desktop»*** χρησιμοποιώντας την βιβλιοθήκη **lpthread** για να γίνει "***compile***" με την εντολή ***«gcc όνομα αρχείου.c -lpthread»*** και εκτελέστε το αρχείο που δημιουργήθηκε γράφοντας στον τερματικό ***«./a.out»***.

2 Περιγραφή του προγράμματος

Οι βιβλιοθήκες που χρησιμοποίησα ήταν:

<pthread.h>

<time.h>

<stdio.h>

<stdlib.h>

<unistd.h>

2.1 Ανάλυση Πηγαίου Κώδικα βήμα προς βήμα

Αρχικά δημιούργησα την main function με σκοπό να δεχτεί στην global μεταβλητή philosophers από τον χρήστη τον αριθμό των φιλοσόφων ζητώντας του να εισάγει έναν αριθμό μεταξύ του 3 και του 10.Αφότου δοθεί σωστός αριθμός απ'τον χρήστη το while loop τελειώνει και καλείται μέσω της main η συνάρτηση Selector όπου δημιουργώ τα threads για την εκτέλεση του προγράμματος και τα καταστρέφω με την λήξη των εργασιών που έφεραν εις πέρας. Ύστερα το πρόγραμμα τερματίζεται αφού δεν εκτελείται κάτι άλλο.

```
int main()
{
printf("THE DINING PHILOSOPHERS\n");
printf("Enter the number of philosophers you'd like to test my code on(Number|n| has to be n>=3 and n<=10 : ");
scanf ("%d",&philosophers);
while(philosophers<3 || philosophers >10)
{
        printf("Please enter a number between 3 and 10 as asked above: ");
        scanf ("%d",&philosophers);
                if (philosophers >=3 && philosophers <=10){break;}
}

Selector(philosophers);

        return 0;
}
```

2.1.1 Ανάλυση συναρτήσεων 1 προς μία 1

Struct Phil

Το συγκεκριμένο struct περιέχει τις μεταβλητές που είναι αναγκαίες για την εκτέλεση και την λειτουργία των threads και των mutexes.

Η ακέραιου τύπου μεταβλητή Philoindex αντιπροσωπεύει κάθε φιλόσοφο ξεχωριστά και χρησιμοποιείται σαν «ταυτότητα».

Το pthread_t thread είναι ένα POSIX thread το οποίο κάθε ένα κάνει «host» κάθε φιλόσοφο ξεχωριστά.

Τα pthread_mutex_t *fork_left,*fork_right είναι τα «πιρούνια» κάθε φιλόσοφου όπου ο καθένας έχει ένα αριστερό κι ένα δεξί. Η ιδέα είναι να υπάρξει ένας μηχανισμός ώστε όταν δύο παράλληλες διεργασίες θέλουν να χρησιμοποιήσουν ένα κοινό πόρο να μην υπάρχει σύγκρουση επεξεργασίας αλλά να διασφαλίζεται ένας αμοιβαίος αποκλεισμός.

Η ακέραιου τύπου μεταβλητή MealProg αντιπροσωπεύει το γεύμα κάθε φιλοσόφου κατά την εκτέλεση του προγράμματος.Με απλά λόγια δηλώνει κάθε φορά αν ένας φιλόσοφος έχει τελειώσει το γεύμα του ή όχι(Υπενθύμιση κάθε φιλόσοφος τελειώνει το γεύμα του εφόσον έχει βρεθεί σε κατάσταση EATING για 20 δευτερόλεπτα).

Η δεκαδικού τύπου μεταβλητή hungryM0de_total αντιπροσωπεύει τον χρόνο που βρίσκεται που ο φιλόσοφος (thread) βρίσκεται σε κατάσταση HUNGRY(Ready).

Η ακέραιου τύπου μεταβλητή hungryMode_times αντιπροσωπεύει τις φορές που ο φιλόσοφος(thread) βρίσκεται σε κατάσταση HUNGRY(Ready).

Η ακέραιου τύπου μεταβλητή Threpriority αντιπροσωπεύει μία άγνωστη ακέραια τιμή «προτεραιότητας» (από 1 ως 3) εκτέλεσης που δίνεται σε κάθε φιλόσοφο(thread) κατά την δημιουργία του.

Η μεταβλητή philosopher είναι αντιπροσωπευτική του struct που έχουμε ορίσει.

(Ανοίξτε τον πηγαίο κώδικα που βρίσκεται στον ίδιο φάκελο με το αρχείο παρουσίασης και κοιτάξτε μεταξύ των γραμμών 8 και 18)

void Selector

Η συνάρτηση **Selector** όπου δημιουργούνται τα threads και τα mutexes για την εκτέλεση του προγράμματος και τα καταστρέφονται με την λήξη των εργασιών που έφεραν εις πέρας.

pthread_mutex_t forks[N] Δήλωση πίνακα μεγέθους ίδιου με του δοσμένου αριθμού φιλοσόφων όπου δημιουργούνται ισάριθμα πιρούνια(mutexes).

philosopher philosophers[N]; Δήλωση πίνακα μεγέθους ίδιου με του δοσμένου αριθμού φιλοσόφων όπου δημιουργούνται ισάριθμες μεταβλητές τύπου struct.

philosopher *philo; Pointer που χρησιμοποιείται σαν «εκπρόσωπος» μεταξύ του περιεχομένου του struct και της συνάρτησης.

int i; Μία ακέραιου τύπου μεταβλητή που χρησιμοποιείται στους επαναληπτικούς βρόχους.

Ακολουθεί ένας επαναληπτικός for βρόχος που αρχικοποιούνται τα πιρούνια(mutexes).

Κι ύστερα άλλος ένας βρόχος όπου αρχικοποιούνται όλα τα στοιχεία του φιλοσόφου(thread) απ'το struct.

pthread_create(&philo->thread,NULL,Philo,philo); Δημιουργία του φιλοσόφου(thread) που δέχεται ως όρισμα την pointer συνάρτηση Philo όπου χρησιμοποιείται σαν την «ρουτίνα» του thread και τον pointer-«εκπρόσωπο» philo.

int dining=1; Μία ακέραιου τύπου μεταβλητή που σηματοδοτεί την δημιουργία και εκτέλεση των threads και των mutexes.

Ακολουθεί ένας while βρόχος όπου επαναλαμβάνεται όσο τα threads και τα mutexes εκτελούνται.

int c=0; Μια ακεραίου τύπου μεταβλητή που λειτουργεί ως μετρητής και ελεγκτής για το αν ένας φιλόσοφος έχει τελειώσει το γεύμα του.

Ακολουθεί ακόμη ένας επαναληπτικός for βρόχος που τους φιλοσόφους.

philo=&philosophers[i]; όπου ορίζουμε ποιον φιλόσοφο(thread) εκτελούμε μέσω pointer

if(philo->MealProg){c +=1;} Ένα απλό if statement όπου ελέγχουμε αν κάποιος φιλόσοφος(thread) τελείωσε το γεύμα του.

if (c == N) Ακόμη ένα απλό if statement όπου ελέγχουμε αν όλοι οι φιλόσοφοι(threads) έχουν τελειώσει το γεύμα τους.

```
for (i=0; i<N; i++)
                {
                        philo = &philosophers[i];
                        pthread_join(philo->thread,NULL);}
```

Ακολουθεί ο παραπάνω βρόχος όπου εκτελείται με σκοπό να «ελευθερώσουμε» τα threads που χρειαστήκαμε και χρησιμοποιήσαμε.

printf("All philosophers have finished! \n"); Ένα απλό μήνυμα στον χρήστη ενημερώνοντας τον ότι όλοι οι φιλόσοφοι τελείωσαν το γεύμα τους.

double AllPhilosophersHungryTime; Δήλωση μίας δεκαδικού τύπου μεταβλητής όπου θα υπολογίσουμε τον συνολικό χρόνο που βρέθηκαν οι φιλόσοφοι(threads) σε κατάσταση HUNGRY(Ready).

Ακόμη ένας επαναληπτικός for βρόχος που θα υπολογίσουμε τον συνολικό μέσο χρόνο μέσω της συνάρτησης hungry() και θα «ελευθερώσουμε» τα mutexes.

pthread_mutex_destroy(&forks[i]); «Ελευθέρωση» των πιρουνιών(mutexes).

AllPhilosophersHungryTime+=hungry(philo->Philoindex,philo->hungryMode_times,

philo->hungryM0de_total); Υπολογισμός του συνολικού χρόνου μέσω της συνάρτησης hungry()

printf("The average time spent on READY(HUNGRY) mode for all philosophers is %f seconds! \n",(AllPhilosophersHungryTime/N)); Ένα απλό μήνυμα στον χρήστη που φανερώνουμε τον συνολικό μέσο χρόνο.

dining = 0; Στην μεταβλητή dining δίνουμε την τιμή 0 με σκοπό να δηλώσουμε την επιτυχία όλων των φιλοσόφων να ολοκληρώσουν το γεύμα τους (πετυχημένη εκτέλεση και «ελευθέρωση» των threads και των mutexes).

(Ανοίξτε τον πηγαίο κώδικα που βρίσκεται στον ίδιο φάκελο με το αρχείο παρουσίασης και κοιτάξτε μεταξύ των γραμμών 108 και 161)

void *Philo

Η pointer συνάρτηση *Philo όπου θα εκτελεστούν οι εργασίες των threads.

philosopher *philo = (philosopher *)p; Pointer ο οποίος «εκπροσωπεί» τα στοιχεία κάθε ενός φιλοσόφου απ'το struct στην συνάρτηση.

pthread_mutex_t *leftFork, *rightFork; Τα 2 mutexes που αντιπροσωπεύουν το αριστερό και το δεξί πιρούνι.

int get_leftfork, get_rightfork; Δήλωση δύο ακέραιου τύπου μεταβλητών που θα δηλώνουν αν η προσπάθεια των φιλοσόφων(threads) να χρησιμοποιήσουν τα πιρούνια(mutexes) ήταν επιτυχημένη ή όχι.

int exTime =0; Αρχικοποίηση μιας ακεραίου τύπου μεταβλητής όπου θα αποθηκεύεται ο χρόνος εκτέλεσης της κατάστασης EATING.

time_t hungry_start,hungry_end; Δήλωση δύο μεταβλητών όπου μετριέται ο χρόνος που ξεκίνησε και σταμάτησε να είναι σε κατάσταση HUNGRY(Ready) ο φιλόσοφος με σκοπό να υπολογιστεί ο πραγματικός χρόνος αργότερα.

while (exTime< 20) Ακολουθεί ένας βρόχος επανάληψης while όπου επαναλαμβάνεται κάθε φορά που ο συνολικός χρόνος εκτέλεσης exTime είναι μικρότερος από 20 δευτερόλεπτα(ο φιλόσοφος δεν έχει τελειώσει το γεύμα του).Ο βρόχος περιέχει τα παρακάτω:

printf("Philosopher %d is thinking at: %s \n", philo->Philoindex,GetTime());

Ένα απλό μήνυμα στον χρήστη όπου τον ενημερώνει ότι ο φιλόσοφος βρίσκεται σε κατάσταση THINKING(Blocked)

sleep(1+ rand()%6); Ο φιλόσοφος βρίσκεται σε κατάσταση THINKING(Blocked) για τυχαίο χρόνο καθώς ζητείται από την άσκηση.

Ακολουθεί αρχικοποίηση των δύο πιρουνιών(mutexes) βάσει των στοιχείων του struct

leftFork = philo->fork_left;

rightFork = philo->fork_right;

printf("Philosopher %d is hungry at %s \n", philo->Philoindex,GetTime());

Ένα απλό μήνυμα στον χρήστη όπου τον ενημερώνει ότι ο φιλόσοφος άλλαξε κατάσταση από THINKING(Blocked) σε HUNGRY(Ready).Άρα και την προσπάθεια του να απολαύσει το γεύμα του.

hungry_start = time(NULL); Αποθήκευση του χρόνου όπου βρέθηκε ο φιλόσοφος(thread) σε κατάσταση HUNGRY(Ready) για πρώτη φορά.

get_leftfork = pthread_mutex_trylock(leftFork); Πρώτη προσπάθεια να «σηκώσει»(lock mutex) το αριστερό πιρούνι ο φιλόσοφος.

Ακολουθεί ένα if (get_leftfork != 0) statement που μας ενημερώνει αν η προσπάθεια του φιλοσόφου ήταν επιτυχής ή όχι.(Κάθε τιμή πέρα του 0 θεωρείται ως αποτυχημένη προσπάθεια)

printf("Philosopher %d failed to take fork %d ,because philosopher %d was using it at: %s \n",philo->Philoindex,philo->Philoindex,previous(philo->Philoindex), GetTime()); Ένα απλό μήνυμα που ενημερώνει τον χρήστη γιατί η προσπάθεια του φιλοσόφου ήταν ανεπιτυχής.

pthread_mutex_lock(leftFork); Ακόμη μία προσπάθεια να «σηκώσει»(lock mutex) το αριστερό πιρούνι,με τον φιλόσοφο(thread) να περιμένει(thread blocks) για το πιρούνι ώστε να προσπαθήσει να το ξανασηκώσει.

Ακολουθεί η ίδια τεχνική και για το δεξί πιρούνι:

```
get_rightfork= pthread_mutex_trylock(rightFork);
if (get_rightfork != 0){
printf("Philosopher %d failed to take fork %d , cause philosopher %d was using it at: %s \n",philo->Philoindex,next(philo->Philoindex),next(philo->Philoindex),GetTime());
pthread_mutex_lock(rightFork);
}
```

hungry_end = time(NULL); Ο φιλόσοφος βγαίνει απ'την κατάσταση HUNGRY(Ready) μόλις καταφέρει να πάρει και τα δύο πιρούνια(mutexes) και μεταβαίνει σε κατάσταση EATING.

philo->hungryM0de_total += difftime(hungry_end,hungry_start); Ο χρόνος όπου ο φιλόσοφος βρισκόταν σε κατάσταση HUNGRY(Ready) προστίθεται στον αντίστοιχο μετρητή του στο struct.

philo->hungryMode_times++; Προστίθεται κάθε φορά η τιμή 1 όπου ο φιλόσοφος βρέθηκε σε κατάσταση HUNGRY(Ready).

printf("Philosopher %d is eating at: %s \n",philo->Philoindex,GetTime()); Ένα απλό μήνυμα ενημερώνοντας τον χρήστη ότι ο φιλόσοφος βρίσκεται σε κατάσταση EATING.

sleep(philo->Threpriority); Ο χρόνος όπου ο φιλόσοφος βρίσκεται σε κατάσταση EATING εξαρτάται απ'το κβάντο χρόνο εκτέλεσης του όπου είναι Q=1 και το Threpriority(προταιρεότητα) που του είχε δοθεί τυχαία στην αρχή εκτέλεσης του προγράμματος.

exTime += philo->Threpriority; Ο μετρητής χρόνου εκτέλεσης αυξάνεται κατά το κβάντο χρόνου εκτέλεσης όπου είναι Q(1s)*Threpriority.

Ο φιλόσοφος ύστερα αφήνει τα δύο πιρούνια(δηλ. τα mutexes από lock γίνονται unlock) και γίνεται unlock πρώτο το δεξιά αφού ήταν το τελευταίο που χρησιμοποιήθηκε(έγινε lock)

pthread_mutex_unlock(rightFork);

pthread_mutex_unlock(leftFork);

Ως εδώ τα στοιχεία περιέχονται εντός του βρόχου while.

printf("Philosopher %d finished eating and is leaving the table at: %s \n",philo->Philoindex,GetTime()); Ένα απλό μήνυμα που εμφανίζεται στον χρήστη σε περίπτωση που κάποιος φιλόσοφος ολοκλήρωσε το γεύμα του (δηλ. exTime=>20).

philo->MealProg = 1; Ενημερώνουμε την μεταβλητή MealProg ότι ο συγκεκριμένος φιλόσοφος ολοκλήρωσε το γεύμα του έτσι ώστε να μην χρειαστεί να ξαναπασχοληθεί ο φιλόσοφος(thread).

(Ανοίξτε τον πηγαίο κώδικα που βρίσκεται στον ίδιο φάκελο με το αρχείο παρουσίασης και κοιτάξτε μεταξύ των γραμμών 56 και 105).

double hungry()

Η συνάρτηση **double hungry()** δέχεται τρία ορίσματα ως εξής:

int Philosopher,int hungryModeTimes,double hungryModeTotal

Χρησιμοποιείται για τον υπολογισμό του μέσου χρόνου που βρέθηκε ο φιλόσοφος σε κατάσταση HUNGRY().

printf("Philosopher %d has been on READY(HUNGRY) mode for %f seconds on average! \n",Philosopher,(hungryModeTotal/hungryModeTimes));

Τυπώνει ένα απλό μήνυμα στον χρήστη κι απλά διαιρεί τον συνολικό χρόνο του φιλοσόφου διά τις φορές που βρέθηκε σε κατάσταση HUNGRY(Ready).

Και την επιστρέφει πίσω(την διαίρεση) ώστε να υπολογισθεί και ο συνολικός μέσος χρόνος για όλους τους φιλοσόφους.

(Ανοίξτε τον πηγαίο κώδικα που βρίσκεται στον ίδιο φάκελο με το αρχείο παρουσίασης και κοιτάξτε μεταξύ των γραμμών 46 και 50).

int previous()

Η **previous()** είναι μια βοηθητική συνάρτηση που υποδεικνύει είτε τον «αριστερά» φιλόσοφο είτε το «αριστερά» πιρούνι αυτού που την καλεί.

(Ανοίξτε τον πηγαίο κώδικα που βρίσκεται στον ίδιο φάκελο με το αρχείο παρουσίασης και κοιτάξτε μεταξύ των γραμμών 39 και 44).

int next()

Η **next** είναι μία βοηθητική συνάρτηση που υποδεικνύει είτε τον «δεξιά» φιλόσοφο είτε το «δεξιά» πιρούνι αυτού που την καλεί.

(Ανοίξτε τον πηγαίο κώδικα που βρίσκεται στον ίδιο φάκελο με το αρχείο παρουσίασης και κοιτάξτε μεταξύ των γραμμών 32 και 37).

const char *GetTime()

Η ***GetTime** είναι μία βοηθητική συνάρτηση που υποδεικνύει την ώρα και την ημερομηνία σε string έτσι ώστε να μπορεί να χρησιμοποιηθεί προς όφελος του χρήστη σε μηνύματα που εμφανίζονται στην οθόνη.

(Ανοίξτε τον πηγαίο κώδικα που βρίσκεται στον ίδιο φάκελο με το αρχείο παρουσίασης και κοιτάξτε μεταξύ των γραμμών 24 και 30).

3 Επίδειξη της λύσης

```
THE DINING PHILOSOPHERS
Enter the number of philosophers you'd like to test my code on(Number|n| has to be n>=3 and n<=10) : 5
Philosopher 1 is thinking at: Tue Jan  7 21:44:30 2020

Philosopher 5 is thinking at: Tue Jan  7 21:44:30 2020

Philosopher 2 is thinking at: Tue Jan  7 21:44:30 2020

Philosopher 3 is thinking at: Tue Jan  7 21:44:30 2020

Philosopher 4 is thinking at: Tue Jan  7 21:44:30 2020
```

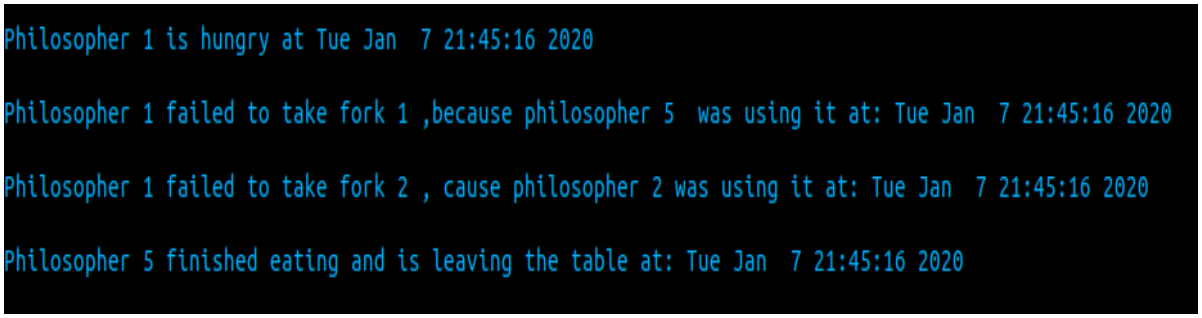
Εικόνα 1. Αρχή του προγράμματος με σωστό input απ'τον χρήστη.
(Σημείωση ότι σε περίπτωση λάθος input ζητείται εκ νέου εισαγωγή αριθμού εώς ότου δοθεί σωστό)

Γραμμή 1	**Γραμμή 2**	**Γραμμή 3**	**Γραμμή 4**	**Γραμμή 5**	**Γραμμή 6**	**Γραμμή 7**
Τίτλος Εργασίας	Μήνυμα εισαγωγής αριθμού	Κατάσταση Φιλοσόφου1	Κατάσταση Φιλοσόφου5	Κατάσταση Φιλοσόφου2	Κατάσταση Φιλοσόφου3	Κατάσταση Φιλοσόφου4

```
Philosopher 2 is hungry at Tue Jan  7 21:44:31 2020

Philosopher 2 is eating at: Tue Jan  7 21:44:31 2020

Philosopher 1 is hungry at Tue Jan  7 21:44:32 2020

Philosopher 1 failed to take fork 2 , cause philosopher 2 was using it at: Tue Jan  7 21:44:32 2020
```

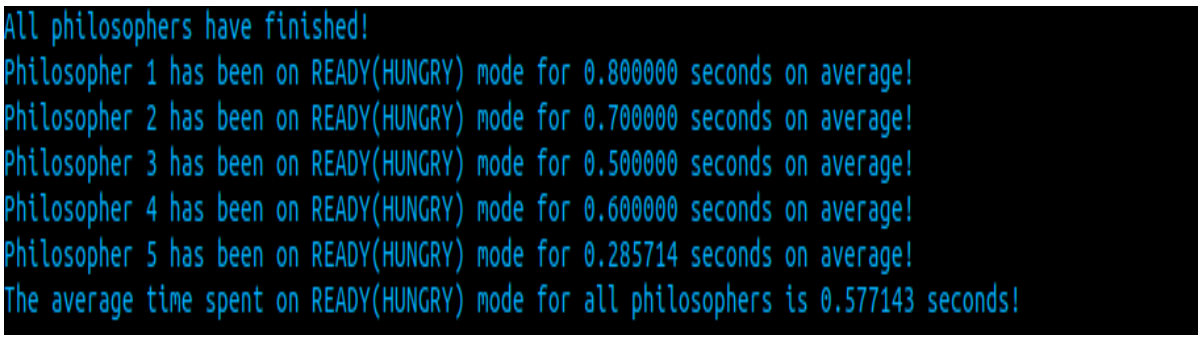
Εικόνα 2.Κατάσταση φιλοσόφων 1 και 2,ανεπιτυχής προσπάθεια φιλοσόφου1

Γραμμή 1	**Γραμμή 2**	**Γραμμή 3**	**Γραμμή 4**
HUNGRY(Ready)Κατάσταση Φιλοσόφου2	EATING Κατάσταση Φιλοσόφου2	HUNGRY(Ready)Κατάσταση Φιλοσόφου1	Αποτυχία Φιλοσόφου1 λόγω του 2



Εικόνα 3.Διπλή ανεπιτυχής προσπάθεια φιλοσόφου1 και τερματισμός φιλοσόφου5

Γραμμή 1	**Γραμμή 2**	**Γραμμή 3**	**Γραμμή 4**
HUNGRY(Ready)Κατάσταση Φιλοσόφου1	Ανεπιτυχής προσπάθεια φιλοσόφου1 λόγω του φιλοσόφου5	Ανεπιτυχής προσπάθεια φιλοσόφου1 λόγω του φιλοσόφου2	Τέλος γεύματος αφού το ολοκλήρωσε απ'τον Φιλόσοφο5



Εικόνα 4.Τέλος δείπνου για όλους του φιλοσόφους και χρόνοι κατάστασης HUNGRY(Ready)

Γραμμή 1	**Γραμμή 2**	**Γραμμή 3**	**Γραμμή 4**	**Γραμμή 5**	**Γραμμή 6**	**Γραμμή 7**
Ενημέρωση χρήστη ότι όλοι οι φιλόσοφοι τελείωσαν το γεύμα τους.	Μέσος χρόνος για τον φιλόσοφο1 σε κατάσταση HUNGRY(Ready).	Μέσος χρόνος για τον φιλόσοφο2 σε κατάσταση HUNGRY(Ready).	Μέσος χρόνος για τον φιλόσοφο3 σε κατάσταση HUNGRY(Ready).	Μέσος χρόνος για τον φιλόσοφο4 σε κατάσταση HUNGRY(Ready).	Μέσος χρόνος για τον φιλόσοφο5 σε κατάσταση HUNGRY(Ready).	Μέσος χρόνος για όλους τους φιλοσόφους σε κατάσταση HUNGRY(Ready).

4 Βιβλιογραφικές Πηγές

Ιστοσελίδες :

1) https://rosettacode.org/wiki/Dining_philosophers?fbclid=IwAR35BnQp0sIrl1RIOgzGQecL5WmYUMMp_UOXIEjkNWpPhQyq12K5Tn50w7k

2) https://www.geeksforgeeks.org/dining-philosopher-problem-using-semaphores/?fbclid=IwAR0Gj0BNkCSMP7JQqefViqhF217WtzFrFWZNkEMu7p3dFo3fNbdSSfVSD5A

3) https://www.geeksforgeeks.org/multithreading-c-2/

4) https://stackoverflow.com/questions/20276010/c-creating-n-threads?fbclid=IwAR1p-DwSTm295X-SLFTzBXTOzWlTWhd1pSZatibNLltZYLWaX4i4hf4laWg

5) https://el.wikipedia.org/wiki/%CE%9D%CE%AE%CE%BC%CE%B1%CF%84%CE%B1_POSIX

6) https://www.youtube.com/channel/UChsuwkz_OWhz6pUTGK2vXZw

7) https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/

Βιβλία :

Andrew S. Tanenbaum,*Σύγχρονα Λειτουργικά Συστήματα.*

Δημήτρης Αποστόλου && Ιωάννης-Χρήστος Παναγιωτόπουλος,*Αρχές Προγραμματισμού με C++.*

Sartaj Sahni,*Δομές Δεδομένων,Αλγόριθμοι και Εφαρμογές στη C++.*