ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

Εργασία Μαθήματος ***Λειτουργικά Συστήματα***

Αριθμός εργασίας – Τίτλος εργασίας	1η Προαιρετική Εργασία: Δια-διεργασιακή επικοινωνία με το δείπνο των φιλοσόφων
Όνομα φοιτητή	Νικόλαος Σέγκος
Αρ. Μητρώου	Π16125
Ημερομηνία παράδοσης	09/01/2018

Εκφώνηση εργασίας

Στην ενότητα αυτή περιλάβετε την εκφώνηση της εργασίας.

Γράψτε ένα πρόγραμμα σε C ή σε Java 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 ως εξής:
   - Έστω ότι το κβάντο χρόνου είναι Q=1sec. Στον n-οστό φιλόσοφο θα δίδεται χρόνος εκτέλεσης ίσος με T(n) = n*Q (π.χ. στον φιλόσοφο 1, δίδεται χρόνος 1*Q, στον φιλόσοφο 8 δίδεται χρόνος 8*Q).

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

Contents

1 Εισαγωγή

Ο σκοπός της υλοποίησης είναι να παρέχει συγχρονισμό σε ένα αριθμό απο threads με σκοπό τη σωστή είσοδο και έξοδο απο τη κρίσιμη περιοχή ενώ ταυτόχρονα, η παραμονή των thread στη κρίσιμη περιοχή να διέπεται χρονικά απο τον τύπο **T(n) = n*Q** με κβάντο χρόνου Q=1 sec και n,τον αριθμό του φιλοσόφου, αλλά και να παρέχονται στατιστικά στοιχεία για το χρόνο αναμονής του κάθε thread και όλων των thread μαζί.

Η υλοποίηση έγινε με χρήση **POSIX threads & mutex** στη γλώσσα **C** με τις βιβλιοθήκες **pthread, time, stdio, stdlib, unistd.** Το compiling έγινε μέσω του **GNU C Compiler**(**GCC**) σε περιβάλλον **GNU/Linux Debian 8 "Jessie".**
Σε περίπτωση που θέλετε να κάνετε compile τον πηγαίο κώδικα, χρησιμοποιήστε παρόμοιο περιβάλλον με μορφή εντολής **gcc [file_name] –lpthread –[other_options**] .

2 Περιγραφή του προγράμματος

Ακολουθεί αναλυτικά η περιγραφή για gκάθε function και struct του κώδικα αλλά και όλος ο πηγαίος κώδικας για να μπορεί να δωθεί μια καλύτερη ιδέα της σειράς εκτέλεσης.

Μια πολύ σημαντική υποσημείωση είναι, πως εδώ ο πηγαίος κώδικας παραδίδεται απλά για να κατανοηθεί η δομή και οι κλήσεις των συναρτήσεων συνολικά. Αν επιθυμείτε να δείτε αναλυτικά τη λογική του κώδικα, είτε διαβάστε τη παρακάτω ανάλυση, είτε διαβάστε το αρχείο του πηγαίου κώδικα, ο οποίος είναι πλήρως σχολιασμένος(στην αγγλική γλώσσα) αλλά για λόγους αναγνωσιμότητας αφαιρέθηκαν τα σχόλια εδώ.

2.1 Struct phil

typedef struct phil //Ένα άκρως σημαντικό μέρος της υλοποίησης. Αυτό το struct περιέχει μεταβλητές, τις οποίες θα έχει κάθε φιλόσοφος ως thread και είναι απαραίτητες για την ορθή λειτουργεία της ρουτίνας του thread.

{

int indexer; // Ο αριθμός του φιλοσόφου.

pthread_t thread; // Το thread στο οποίο θα υλοποιείται ο φιλόσοφος.

pthread_mutex_t *f_l, *f_r; //Τα 2 πιρούνια-mutex που αντιστοιχούν στο thread του φιλοσόφου,με βάση της θεωρητική τους θέση ως αριστερά και δεξιά του.

int ful; // Η μεταβλητή αυτή δηλώνει αν ο φιλόσοφος έχει τελειώσει το γεύμα του ή όχι.

double time_total; // Η μεταβλητή αυτή μετράει πόσο χρόνο έχει περάσει ο φιλόσοφος σε κατάσταση αναμονής(WAITING) για να «φάει»(να μπεί στη κρίσιμη περιοχή).

int w8; // Η μεταβλητή αυτή μετράει πόσες φορές ο φιλόσοφος μπήκε σε κατάσταση αναμονής.

} philly;

2.2 Function GetTime()

const char * GetTime() // Το type signature “const char *” της συνάρτησης, της επιτρέπει να επιστρέφει αποτέλεσμα τύπου string.

{

time_t rawtime; // Μεταβλητή τύπου time_t

rawtime = time(NULL); //Κάνοντας κλήση τη συνάρτηση time με όρισμα NULL λαμβάνουμε τη τρέχουσα ώρα.

return ctime(&rawtime); //Με τη χρήση της ctime μετατρέπουμε τον τύπο time_t σε string.

}

2.3 Function Next_One(int i)

int next_one(int i) // Η μέθοδος αυτή λαμβάνει ως είσοδο τον αριθμό του φιλοσόφου και επιστρέφει τον αριθμό του φιλοσόφου στα δεξιά του.

{

int x = (i+1)%phillies; // Το modulo υπάρχει ώστε αν γίνει κλήση από τον ν-οστό φιλόσοφο να μην επιστραφεί αριθμός 0.

return x;

}

2.4 Function Prev_One(int i)

int prev_one(int i) // Η μέθοδος αυτή λαμβάνει ως είσοδο τον αριθμό του φιλοσόφου και επιστρέφει τον αριθμό του φιλοσόφου στα αριστερά του.

```
{
        int x = (i-1)%phillies;
        if (x==0){x=phillies;}      // Ο έλεγχος γίνεται ώστε να αποφευγθεί επιστροφή
του 0  αν γίνει κλήση από τον 1ο φιλόσοφο
        return x;
}
```

2.5 Function Rdy_Avg(int i,int w,double t)

double rdy_avg(int i,int w,double t) // Αυτή η συνάρτηση παίρνει ως είσοδο τον αριθμό του φιλοσόφου,τον συνολικό χρόνο και τις φορές που αυτός πέρασε σε κατάσταση αναμονής και τυπώνει και επιστρέφει τον μέσο χρόνο αναμονής.

```
{
        printf("Philosopher %d has been on READY(HUNGRY) mode for %f seconds on average.
\n",i,(t/w) );      // Εκτύπωση του μηνύματος
        return t/w;                     // Επιστροφή του μέσου χρόνου αναμονής.
}
```

2.6 Function *PhilFunc(void *p)

void *PhilFunc(void *p) // Ένα άλλο άκρως σημαντικό μέρος του προγράμματος είναι αυτή η pointer συνάρτηση η οποία λειτουργεί ως η συνάρτηση ρουτίνας εκτέλεσης του κάθε thread.

```
{
        philly *philo = (philly *)p;        //  Ένα pointer του struct το οποίο παίρνει ως
typecasted είσοδο ένα pointer για το struct με τις μεταβλητές του φιλοσόφου.
        pthread_mutex_t *fork_left, *fork_right;// Τα 2 mutex: αριστερό και δεξί.
        int lfork_grab_succ, rfork_grab_succ; // 2 μεταβλητές οι οποίες δείχνουν αν η
ενέργεια απόκτησης του mutex είναι επιτυχής ή όχι.
        int fullness =0;                //  Η μεταβλητή μετράει το χρόνο εκτέλεσης(παραμονής
στη κρίσιμη περιοχή).
        time_t start_rdy,end_rdy;       // 2 μεταβλητές που μετράνε την έναρξη και τη
λήξη του χρόνου αναμονής.
        while (fullness <= 20)          //  Η επανάληψη πραγματοποιείται όσο ο
συνολικός χρ΄νος εκτέλεσης είναι κάτω του 20.
```

```
        {
                printf("Philosopher %d is thinking at %s \n", philo->indexer,GetTime());

                sleep(1+ rand()%6);             // Ο φιλόσοφος περνάει ένα τυχαίο ποσό χρόνου σε μπλοκαρισμένη κατάσταση.
                fork_left = philo->f_l;         //  Το αριστερό mutex τίθεται με βάση το περιεχόμενο του struct.
                fork_right = philo->f_r;        //  Το δεξί mutex τίθεται με βάση το περιεχόμενο του struct.

                printf("Philosopher %d is hungry at %s  \n", philo->indexer,GetTime());
                start_rdy = time(NULL);                 // Έναρξη της κατάστασης αναμονής και της χρονομέτρησης της.
                lfork_grab_succ = pthread_mutex_trylock(fork_left); // Προσπάθεια να κλειδωθεί  το αριστερό mutex από το thread, αν είναι επιτυχής, η μεταβλητή επιτυχίας θα γίνει 0.
            if (lfork_grab_succ != 0)                   // Αν η προσπάθεια είναι ανεπιτυχής.
   {
        printf("Philosopher %d failed to take fork %d , cause philosopher %d  was using it at %s \n",philo->indexer,philo->indexer,prev_one(philo->indexer), GetTime());    //Εκτύπωση μυνήματος αποτυχίας.
        pthread_mutex_lock(fork_left);              // Το thread μπλοκάρει μέχρι το mutex να ελευθερωθεί και τότε το thread ξαναπροσπαθεί να το κλειδώσει.
   }
   rfork_grab_succ = pthread_mutex_trylock(fork_right);     // Προσπάθεια να κλειδωθεί  το δεξί mutex από το thread, αν είναι επιτυχής, η μεταβλητή επιτυχίας θα γίνει 0.
  if (rfork_grab_succ != 0)                         // Αν η προσπάθεια είναι ανεπιτυχής.
   {
      printf("Philosopher %d failed to take fork %d , cause philosopher %d was using it at %s \n",philo->indexer,next_one(philo->indexer),next_one(philo->indexer),GetTime());  //Εκτύπωση μυνήματος αποτυχίας.
     pthread_mutex_lock(fork_right);            // Το thread μπλοκάρει μέχρι το mutex να ελευθερωθεί και τότε το thread ξαναπροσπαθεί να το κλειδώσει.
   }
                                               // Το thread μπαίνει στη κρίσιμη περιοχή.
```

end_rdy = time(NULL); // Αφού τα 2 mutex κλειδωθούν η κατασταση αναμονής τελειώνει.

philo->time_total += difftime(end_rdy,start_rdy); // Ο χρόνος αναμονής προσθέτεται στο συνολικό χρόνο αναμονής.

philo->w8++; // Οι φορές αναμονής μεγαλώνουν κατά 1.

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

sleep(philo->indexer); // Ο χρόνος εκτέλεσης του κάθε φιλόσοφου διέπεται χρονικά απο τον τύπο **T(n) = n*Q** με κβάντο χρόνου Q=1 sec και n,τον αριθμό του φιλοσόφου.

fullness += philo->indexer; // Ο συνολικό χρόνος εκτέλεσης μεγαλώνει ανάλογα.

pthread_mutex_unlock(fork_right); // Ξεκλειδώνει το δεξί mutex διότι αυτό κλειδώθηκε τελευταίο.

pthread_mutex_unlock(fork_left); // Ξεκλειδώνει το αριστερό mutex διότι αυτό κλειδώθηκε πρώτο.

// Το thread βγαίνει από τη κρίσιμη περιοχή.

}

printf("Philosopher %d is full and is leaving the table at %s \n",philo->indexer,GetTime());

philo->ful = 1; // Όταν η εκτέλεση ξεπεράσει τα 20 δευτερόλεπτα, η μεταβλητή ολοκλήρωσης του γεύματος γίνεται 1.

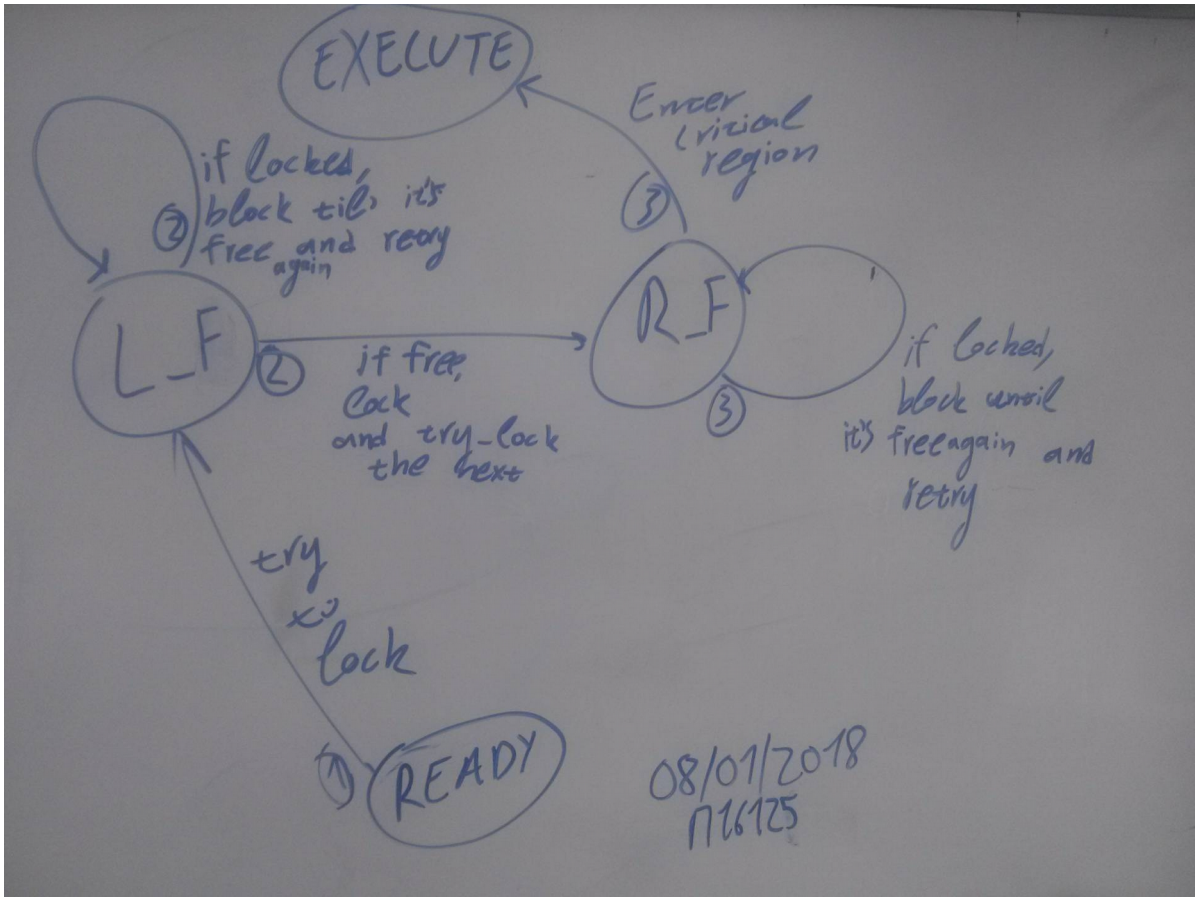
return NULL;

}

2.6.1 Εξήγηση της διαδικασίας απόκτησης των mutex.

Για πιο εύκολη εξήγηση θα λέμε τα mutex πιρούνια και τα thread φιλοσόφους και θα δούμε την διαδικασία ως μια απλή ιστορία.

Ο φιλόσοφος αφού τελειωσει να σκέφτεται, αρχίζει να πεινάει. Αρχικά προσπαθεί να πιάσει το αριστερό πιρούνι. Αν το χρησιμοποιεί άλλος, περιμένει το πιρούνι να ξαναγίνει ελεύθερο και ξαναπροσπαθεί. Αφού πιάσει το αριστερό πιρούνι, επαναλαμβάνει τη διαδικασία για το δεξί πιρούνι ενώ ταυτόχρονα κρατάει το αριστερό πιρούνι. Αφού τα πιάσει και τα 2, ξεκινάει να τρώει.



Σχεδιαγραμμα της διαδικασίας.

2.7 Function Master_Controller(int N)

void master_controller(int N) // Η συνάρτηση αυτή αναλαμβάνει να συντονίσει τη δημιουργία των threads και των mutex αλλά και τη καταστροφή τους όταν αυτό θα χρειαστεί.

{

pthread_mutex_t forks[N]; // Πίνακας από mutex που αντιπροσωπεύουν τα πιρούνια.

philly philosophers[N]; // Πίνακας από thread αντιπροσωπεύουν τους φιλοσόφους.

philly *philo; // Pointer ίδιου τύπου με τους φιλοσόφους το οποίο θα λειτουργεί σαν αντιπρόσωπος του κάθε φιλοσόφου στο struct.

int i; // Απλή μεταβλητή για επαναλήψεις.

```
for (i =0; i< N ; i++)                    // Δημιουργία mutex.
{
        pthread_mutex_init(&forks[i],NULL);
}
for (i =0; i< N; i++)                     // Γέμισμα του struct για κάθε φιλόσοφο και δημιουργία των thread.
{
        philo = &philosophers[i];         // Ο αντιπρόσωπος λαμβάνει το φιλόσοφο τον οποίο θα αντιπροσωπέυσει.
        philo->time_total=0.0;            //Εισχώρηση τιμής μεταβλητής
        philo->w8=0;                      //Εισχώρηση τιμής μεταβλητής
        philo->ful = 0;                   //Εισχώρηση τιμής μεταβλητής
        philo->indexer = i+1;             // Ορισμός αριθμού φιλοσόφους.
        philo->f_l = &forks[i];           // Εισχώρηση δεξιού mutex.
        philo->f_r = &forks[(i+1)%N];     // Εισχώρηση αριστερού mutex
        pthread_create(&philo->thread,NULL,PhilFunc,philo);   // Δημιουργία του thread με όρισμα το thread id του φιλοσόφου, τη ρουτίνα εκτέλεσης και ως όρισμα της τον αντιπρόσωπο.
}
int running = 1;                  //  Η μεταβλητή δηλώνει ότι τα mutex και τα thread λειτουργούν.
while (running)                   //Όσο τα thread και τα mutex λειτουργούν.
{
        int g=0;                  // Μετρητής των «χορτάτων» φιλοσόφων.
        for (i=0; i<N; i++)       //Επανάληψη μέχρι να έχουν τελειώσει όλοι οι φιλόσοφοι το γεύμα τους.
        {
                philo = &philosophers[i];
                if(philo->ful){g +=1;}
        }
        if (g == N)               // Αν όλοι οι φιλόσοφοι τελείωσαν το γεύμα τους.
```

```
                {
                        for (i=0; i<N; i++)                                     // Διαγραφή των
thread.
                        {
                                philo = &philosophers[i];
                                pthread_join(philo->thread,NULL);
                        }
                        printf("All philosophers are full now \n");
                        double all_phil_avg;
                        for (i=0; i<N; i++)                     //Υπολογιμσμός μέσωου χρόνου
αναμονής,συνολικού μέσου χρόνου αναμονής και διαγραφή των mutex.
                        {
                                pthread_mutex_destroy(&forks[i]);
                                philo = &philosophers[i];
                                all_phil_avg            +=rdy_avg(philo->indexer,philo->w8,philo-
>time_total);            // Each philosopher's average time is computed and added to the total
average.
                        }
                        printf("The average time spent on READY(HUNGRY) mode for all
philosophers is %f seconds. \n",(all_phil_avg/N));


                        running = 0;
                }
        }




}
```

2.8 Main

```
int main()              // Η κύρια συνάρτηση του προγράμματος.
```

```
{
        printf("___THE DINING PHILOSOPHERS___  \n");
        for(;;)                                          // Ατέρμονη επανάληψη.
        {
                printf("Enter the number of the philosophers(Warning! The number must be between 3 and 10 : ");
                scanf("%d",&phillies);          // Εισαγωγή αριθμού φιλοσόφων απο το ρήστη.

                if ((phillies <=10) && (phillies >=3)){break;}          //Εφόσον η είσοδος είανι σωστή, η επανάληψη σταματάει.
        }

        master_controller(phillies);          //Κλήση της συντονιστικής συνάρτησης

        return 0;
}
```

2.9 Ολόκληρος ο πηγαίος κώδικας

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


typedef struct phil
{
 int indexer;
        pthread_t thread
        pthread_mutex_t *f_l, *f_r;
        int ful;
        double time_total;
```

```
        int w8;
} philly;

int phillies;

const char * GetTime()
{
time_t rawtime;
rawtime = time(NULL);

return ctime(&rawtime);
}

int next_one(int i)
{
        int x = (i+1)%phillies;
        if (x==0){x=phillies;}
        return x;
}

int prev_one(int i)
{
        int x = (i-1)%phillies;
        if (x==0){x=phillies;}
        return x;
}

double rdy_avg(int i,int w,double t)   // A custom function to calculate the average time spent on READY mode.
{
        printf("Philosopher %d has been on READY(HUNGRY) mode for %f seconds on average. \n",i,(t/w) );
```

```
        return t/w;
}




void *PhilFunc(void *p)
{
        philly *philo = (philly *)p;
        pthread_mutex_t *fork_left, *fork_right
        int lfork_grab_succ, rfork_grab_succ;
        int fullness =0;
        time_t start_rdy,end_rdy;
        while (fullness <= 20)
        {
                printf("Philosopher %d is thinking at %s \n", philo->indexer,GetTime());

                sleep(1+ rand()%6);
                fork_left = philo->f_l;
                fork_right = philo->f_r;
                printf("Philosopher %d is hungry at %s  \n", philo->indexer,GetTime());
                start_rdy = time(NULL);
                lfork_grab_succ = pthread_mutex_trylock(fork_left);
        // A "soft" attempt to acquire the left mutex is made.
   if (lfork_grab_succ != 0)
   {
        printf("Philosopher %d failed to take fork %d , cause philosopher %d  was using it at
%s \n",philo->indexer,philo->indexer,prev_one(philo->indexer), GetTime());
        pthread_mutex_lock(fork_left);
   }
   rfork_grab_succ = pthread_mutex_trylock(fork_right);
```

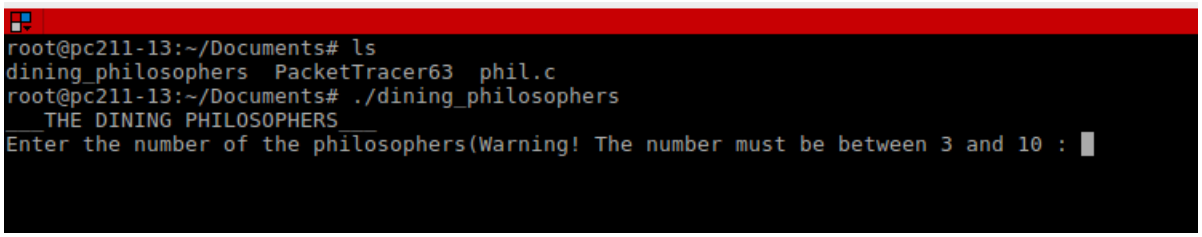
```
   if (rfork_grab_succ != 0)
   {
      printf("Philosopher %d failed to take fork %d , cause philosopher %d was using it at %s \n",philo->indexer,next_one(philo->indexer),next_one(philo->indexer),GetTime());
     pthread_mutex_lock(fork_right);
   }


          end_rdy = time(NULL);
          philo->time_total += difftime(end_rdy,start_rdy);
          philo->w8++;
                printf("Philosopher %d is eating at %s  \n",philo->indexer,GetTime());
                sleep(philo->indexer);
                fullness += philo->indexer;
                pthread_mutex_unlock(fork_right);
                pthread_mutex_unlock(fork_left);

          }
          printf("Philosopher %d is full and is leaving the table  at %s \n",philo->indexer,GetTime());
          philo->ful = 1;
return NULL;
}


void master_controller(int N)
{
          pthread_mutex_t forks[N];
          philly philosophers[N];
          philly *philo;
          int i;

          for (i =0; i< N ; i++)
          {
```

```
            pthread_mutex_init(&forks[i],NULL);
        }
        for (i =0; i< N; i++)
        {
            philo = &philosophers[i];
            philo->time_total=0.0;
            philo->w8=0;
            philo->ful = 0;
            philo->indexer = i+1;
            philo->f_l = &forks[i];
            philo->f_r = &forks[(i+1)%N];
        pthread_create(&philo->thread,NULL,PhilFunc,philo);
        }
        int running = 1;                                    while (running)
            {
            int g=0;
            for (i=0; i<N; i++)
{
                philo = &philosophers[i];
        if(philo->ful){g +=1;}                                                  }
            if (g == N)                         // If all philosophers have ended their meals.
            {
                for (i=0; i<N; i++)                                             //          Thread
destruction loop
                {
                    philo = &philosophers[i];
                    pthread_join(philo->thread,NULL);
                }
                printf("All philosophers are full now \n");
                double all_phil_avg;
        for (i=0; i<N; i++)                                                             {
                    pthread_mutex_destroy(&forks[i]);
                    philo = &philosophers[i];
```

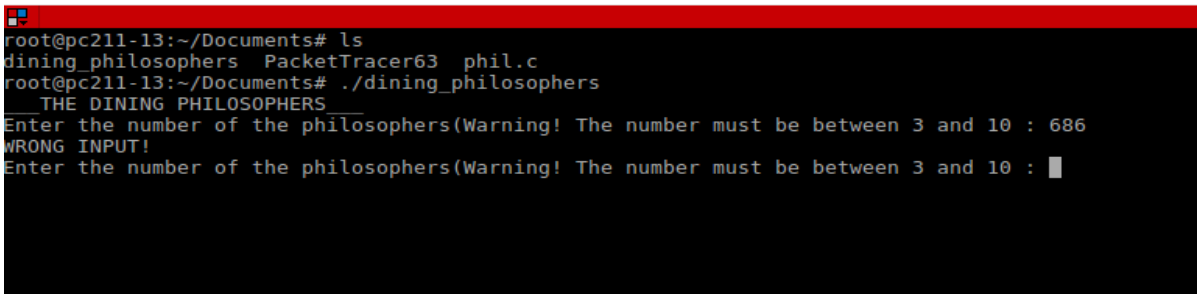
```
                                all_phil_avg+=rdy_avg(philo->indexer,philo->w8,philo-
>time_total);
}
                        printf("The average time spent on READY(HUNGRY) mode for all
philosophers is %f seconds. \n",(all_phil_avg/N));
                        running = 0;                                }
        }




}



int main()
{
        printf("___THE DINING PHILOSOPHERS___  \n");
        for(;;)
        {
                printf("Enter the number of the philosophers(Warning! The number must be
between 3 and 10 : ");
                scanf("%d",&phillies);                          if ((phillies <=10) &&
(phillies >=3)){break;}                 }


        master_controller(phillies);


        return 0;
}
```

3 Επίδειξη της λύσης

Κάποια ενδεικτικά screenshot απο την εκτέλεση του προγράμματος.



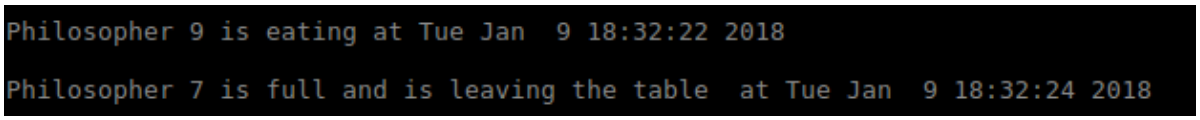
i) Το εναρκτήριο μήνυμα και η αναμονή για είσοδο απο το χρήστη.



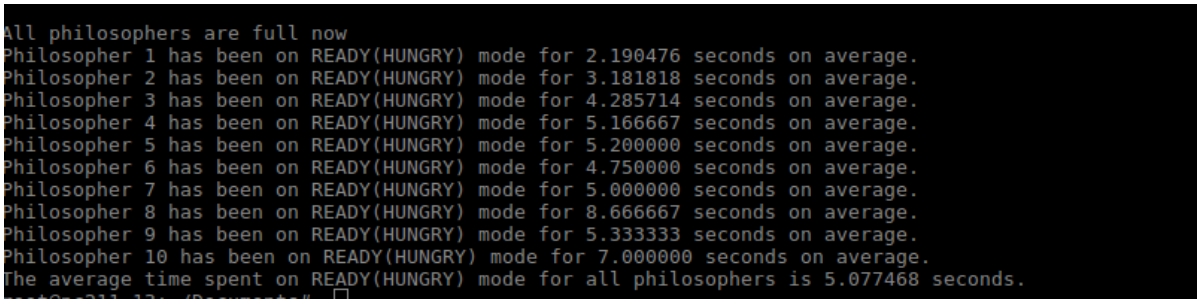
ii)Περίπτωση λανθασμένης εισόδου.

```
Philosopher 8 failed to take fork 8 , cause philosopher 7  was using it at Tue Jan  9 18:31:45 2018
^[[24~Philosopher 1 is hungry at Tue Jan  9 18:31:46 2018
Philosopher 1 failed to take fork 2 , cause philosopher 2 was using it at Tue Jan  9 18:31:46 2018
Philosopher 4 is hungry at Tue Jan  9 18:31:46 2018
Philosopher 4 failed to take fork 4 , cause philosopher 3  was using it at Tue Jan  9 18:31:46 2018
Philosopher 3 is thinking at Tue Jan  9 18:31:47 2018
Philosopher 4 failed to take fork 5 , cause philosopher 5 was using it at Tue Jan  9 18:31:47 2018
Philosopher 2 is eating at Tue Jan  9 18:31:47 2018
Philosopher 5 is thinking at Tue Jan  9 18:31:47 2018
Philosopher 6 failed to take fork 7 , cause philosopher 7 was using it at Tue Jan  9 18:31:47 2018
Philosopher 4 is eating at Tue Jan  9 18:31:47 2018
Philosopher 3 is hungry at Tue Jan  9 18:31:49 2018
Philosopher 2 is thinking at Tue Jan  9 18:31:49 2018
Philosopher 3 failed to take fork 4 , cause philosopher 4 was using it at Tue Jan  9 18:31:49 2018
Philosopher 1 is eating at Tue Jan  9 18:31:49 2018
Philosopher 1 is thinking at Tue Jan  9 18:31:50 2018
Philosopher 5 is hungry at Tue Jan  9 18:31:50 2018
Philosopher 5 failed to take fork 5 , cause philosopher 4  was using it at Tue Jan  9 18:31:50 2018
Philosopher 9 is thinking at Tue Jan  9 18:31:50 2018
Philosopher 10 is eating at Tue Jan  9 18:31:50 2018
Philosopher 2 is hungry at Tue Jan  9 18:31:51 2018
Philosopher 2 failed to take fork 3 , cause philosopher 3 was using it at Tue Jan  9 18:31:51 2018
Philosopher 4 is thinking at Tue Jan  9 18:31:51 2018
Philosopher 5 failed to take fork 6 , cause philosopher 6 was using it at Tue Jan  9 18:31:51 2018
Philosopher 3 is eating at Tue Jan  9 18:31:51 2018
Philosopher 7 is thinking at Tue Jan  9 18:31:51 2018
Philosopher 8 is eating at Tue Jan  9 18:31:51 2018
Philosopher 6 is eating at Tue Jan  9 18:31:51 2018
Philosopher 4 is hungry at Tue Jan  9 18:31:52 2018
Philosopher 4 failed to take fork 4 , cause philosopher 3  was using it at Tue Jan  9 18:31:52 2018
Philosopher 7 is hungry at Tue Jan  9 18:31:52 2018
Philosopher 7 failed to take fork 7 , cause philosopher 6  was using it at Tue Jan  9 18:31:52 2018
```

iii)Μήνυματα αποτυχίας απόκτησης πιρουνιών(αριστερών και δεξιών) .



iv) Μήνυμα ολοκλήρωσης γεύματος φιλοσόφου.



v) Στατιστικά χρόνου αναμονής στη λήξη του προγράμματος.

4 Βιβλιογραφικές Πηγές

1.- Αρχές Προγραμματισμού σε C/C++,Χ.Παναγιωτόπουλος & Δ.Αποστόλου : Μεταβλητές Δομής σελ.119

2.- Αρχές Προγραμματισμού σε C/C++,Χ.Παναγιωτόπουλος & Δ.Αποστόλου : Μεταβλητές Δεικτών σελ.180

3.-Επεξήγηση και πληροφορίες για τη βιβλιοθήκη pthread - The Open Group Library

4.-Επεξήγηση και πληροφορίες για τη βιβλιοθήκη time - The Open Group Library