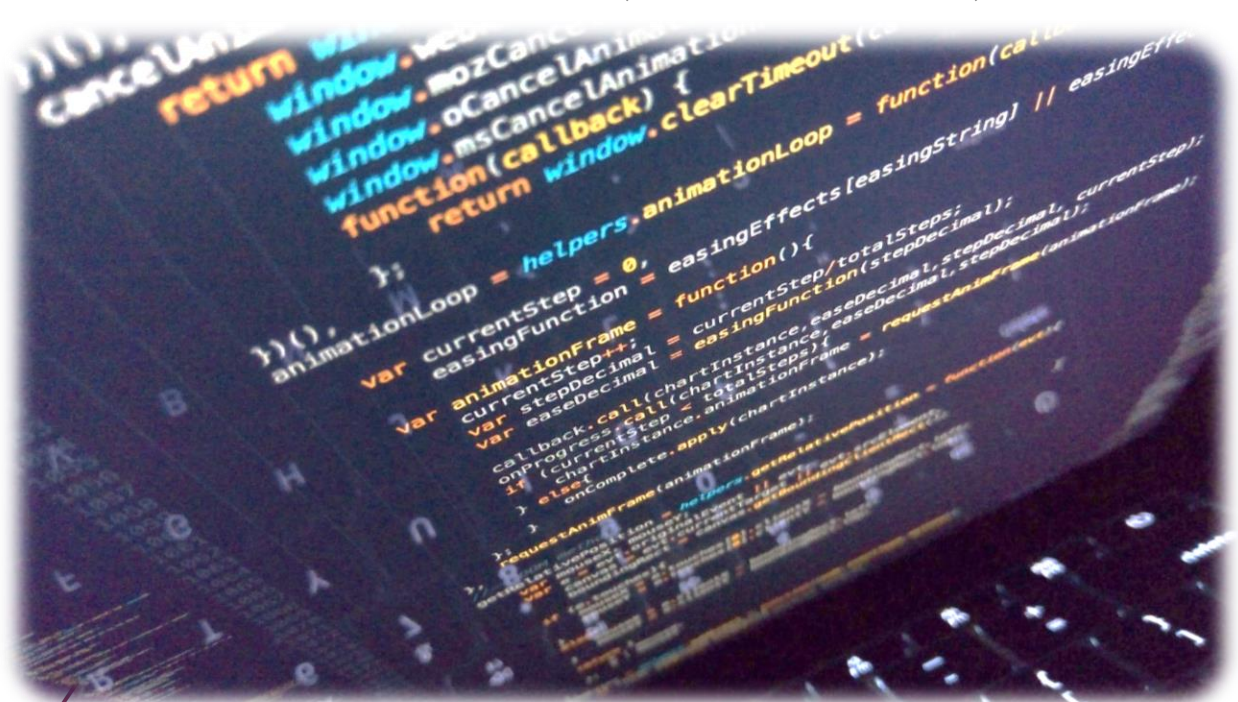


7/6/2021

# ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ II

## ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ #2

THREADS, SEMAPHORES, SOCKETS



**ΒΕΛΑΣΚΟ ΠΑΟΛΑ**

A.M: cs161020

ΕΞΑΜΗΝΟ: 10

Εργαστήριο 2020-2021, Τμήμα Δ9 Παρασκευή 10:00-12:00  
Κος ΜΙΧ. ΙΟΡΔΑΝΑΚΗΣ

## Πίνακας περιεχομένων

<b>ΣΚΟΠΟΣ ΕΡΓΑΣΙΑΣ .....</b>	<b>3</b>
<b>ΕΚΦΩΝΗΣΗ .....</b>	<b>4</b>
<b>ΥΛΟΠΟΙΗΣΗ ΕΡΓΑΣΙΑΣ .....</b>	<b>5</b>
<b>ΑΣΚΗΣΗ 1 .....</b>	<b>5</b>
ΚΩΔΙΚΑΣ .....	5
ΤΕΚΜΗΡΙΩΣΗ.....	7
Σχόλια.....	7
Εκτέλεση κώδικα.....	7
Built-in συναρτήσεις .....	7
Ροή .....	7
Αποτελέσματα .....	8
Εκτέλεση 1.....	8
<b>ΑΣΚΗΣΗ 2 .....</b>	<b>9</b>
ΚΩΔΙΚΑΣ .....	9
ΤΕΚΜΗΡΙΩΣΗ.....	14
Σχόλια.....	14
Εκτέλεση κώδικα.....	14
Built-in συναρτήσεις .....	14
Ροή .....	14
Αποτελέσματα .....	15
Εκτέλεση 1.....	15
Εκτέλεση 2.....	17
<b>ΑΣΚΗΣΗ 3 .....</b>	<b>19</b>
ΚΩΔΙΚΑΣ .....	19
Server .....	19
Client .....	22
ΤΕΚΜΗΡΙΩΣΗ.....	26
Σχόλια.....	26
Εκτέλεση κώδικα.....	26
Built-in συναρτήσεις .....	26

Ροή .....	26
Αποτελέσματα .....	28
Ένας συνδεδεμένος client.....	28
Δύο συνδεδεμένοι clients.....	29
Τέσσερις συνδεδεμένοι clients.....	30
<b>ΣΥΜΠΕΡΑΣΜΑ</b> .....	31
<b>ΒΙΒΛΙΟΓΡΑΦΙΑ</b> .....	31

## ΣΚΟΠΟΣ ΕΡΓΑΣΙΑΣ

Η εργασία 2 αποτελεί τη συνέχεια εμπέδωσης των γνώσεών μας πάνω στα νήματα. Συγκεκριμένα, μας απαιτεί να χρησιμοποιήσουμε semaphores και pthread\_barrier για να συγχρονίσουμε τα νήματα. Επίσης, εισάγει το φοιτητή στις έννοιες client – server. Για την επικοινωνία μεταξύ της, γίνεται η χρήση των named sockets, ενώ για την αποστολή δεδομένων και λήψη γίνεται η χρήση των συναρτήσεων recv και send.

## ΕΚΦΩΝΗΣΗ

### ΕΡΓΑΣΤΗΡΙΟ ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ II

#### Εαρινό Εξάμηνο 2020-21 - Άσκηση #2

Ημ. Παράδοσης: 30/5/2021

##### A.

Γράψτε ένα πρόγραμμα σε C και με χρήση της βιβλιοθήκης των Pthreads, η εκτέλεση του οποίου θα έχει ως αποτέλεσμα να τυπώνεται επαναληπτικά η ακολουθία:

`<one><two><three><one><two><three><one><two><three>.....`

Θα πρέπει (για να πετύχετε το παραπάνω) να εκκινήσετε στο πρόγραμμά σας τρία (3) διαφορετικά threads (το ένα να τυπώνει `<one>`, το άλλο να τυπώνει `<two>` και το τρίτο να τυπώνει `<three>`), και στη συνέχεια να τα συγχρονίσετε κατάλληλα μεταξύ τους. Για τον απαιτούμενο συγχρονισμό θα πρέπει να χρησιμοποιήσετε **σημαφόρους**.

*/\* στη συνέχεια προσπαθήστε να δώσετε (δεν ζητείται υποχρεωτικά) μια άλλη έκδοση του παραπάνω προγράμματος, στο οποίο για τον απαιτούμενο συγχρονισμό να χρησιμοποιήσετε **condition variables** (η προσπάθειά σας για την απαίτηση αυτή θα συνυπολογιστεί προσθετικά στον τελικό βαθμό της άσκησης – θα μετρήσει δηλαδή ως 'bonus') \*/*

##### B.

Έστω ένας δυσδιάστατος πίνακας ακεραίων  $A$  ( $N \times N$ ). Γράψτε ένα πρόγραμμα το οποίο θα βρίσκει το 'μέγιστο' στοιχείο ( $m$ ) του πίνακα  $A$  [με τη βοήθεια 'p' threads όπου το κάθε thread θα υπολογίζει το επιμέρους μέγιστο  $N/p$  γραμμών του πίνακα – τα 'p', 'N', 'A' θα πρέπει να τα δίνει ο χρήστης ή να τα διαβάζετε από αρχείο – επίσης θεωρείστε ότι το 'N' είναι ακέραιο πολλαπλάσιο του 'p']. Στη συνέχεια το πρόγραμμα θα πρέπει να φτιάχνει επίσης παράλληλα (με τη βοήθεια των 'p' threads που έχουν εξαρχής δημιουργηθεί) ένα νέο πίνακα  $D$  ( $N \times N$ ) (τον οποίον θα τυπώνει στο τέλος στην οθόνη), στον οποίον θα αποτυπώνεται η 'απόσταση' του κάθε στοιχείου ( $A_{ij}$ ) του πίνακα  $A$  από το  $m$ :

$$D_{ij} = m - A_{ij}$$

[χρησιμοποιήστε μεταξύ των άλλων για τον απαιτούμενο συγχρονισμό το μηχανισμό 'barrier synchronization' των Pthreads - συμβουλευτείτε σχετικά το παράδειγμα 'bar-ex.c' από το αρχείο THREADS-EXAMPLES.zip (αναρτημένο στο Eclass), καθώς και το ακόλουθο URL: [http://pubs.opengroup.org/onlinepubs/009695399/functions/pthread\\_barrier\\_wait.html](http://pubs.opengroup.org/onlinepubs/009695399/functions/pthread_barrier_wait.html)]

##### Γ.

Γράψτε δύο προγράμματα, ένα πρόγραμμα **server** και έναν πρόγραμμα **client** (το οποίο θα μπορούν να το εκτελούν εν δυνάμει πολλοί clients - και να μπορούν να εξυπηρετούνται ταυτόχρονα από το server), τα οποία θα μπορούν να επικοινωνούν μεταξύ τους (με χρήση UNIX-domain stream sockets) επαναληπτικά ως εξής:

- Ο client θα στέλνει στο server μία ακολουθία ακεραίων μήκους  $N$ , την οποία θα διαβάζει από το χρήστη.
- Ο server αφού παραλάβει την ακολουθία θα υπολογίζει το μέσο όρο της και αν αυτός είναι πάνω από 10 θα στέλνει πίσω στον client τόσο (α) τον μέσο όρο που βρήκε όσο και (β) ένα κατάλληλο μήνυμα αποδοχής (π.χ. 'Sequence Ok'). Ειδιάλλως θα στέλνει μόνο ένα μήνυμα αποτυχίας (π.χ. 'Check Failed').
- Ο client θα πρέπει απλά μετά από κάθε επικοινωνία να τυπώνει ότι απάντηση του έστειλε ο server στην οθόνη, και να ζητά την επόμενη ακολουθία από το χρήστη. Η επικοινωνία θα τελειώνει (από πλευράς του client) όταν ο χρήστης δηλώσει ότι δεν επιθυμεί να δώσει άλλη ακολουθία προς έλεγχο.
- Θα πρέπει τέλος επίσης ο server να 'διατηρεί' κάπου (πάντα ενημερωμένο) το συνολικό αριθμό των αιτήσεων/ακολουθιών που έχουν περάσει επιτυχώς τον έλεγχο (πόσες δηλαδή μέχρι αυτή τη στιγμή - από όλους τους clients), καθώς και πόσες έχουν ελεγχθεί συνολικά.

##### Παραδοτέα:

Κώδικας, σχολιασμός/τεκμηρίωση, ενδεικτικά τρεξίματα και χρόνοι εκτέλεσης (ερώτημα Β).

Σχετικά με τη δομή και το format του/ων παραδοτέου/ων σας καλείστε να ακολουθήσετε τις οδηγίες που θα σας δώσει πιο συγκεκριμένα ο Καθηγητής του τμήματός σας.



## ΥΛΟΠΟΙΗΣΗ ΕΡΓΑΣΙΑΣ

## ΑΣΚΗΣΗ 1

## ΚΩΔΙΚΑΣ

```
/* ;; -----  
;;                      ΒΕΛΑΣΚΟ ΠΑΟΛΑ          cs161020  
;; -----  
;;                      Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών  
;;                      Εργαστήριο ΛΣΙΙ 2020/21 - Εργασία 2.1:  
;;                      THREADS, SEMAPHORES, SOCKETS  
;; -----  
;; Υπεύθυνος μαθήματος: ΜΑΜΑΛΗΣ ΒΑΣΙΛΕΙΟΣ  
;; Καθηγητές: ΙΟΡΔΑΝΑΚΗΣ ΜΙΧΑΛΗΣ  
;; -----  
*/  
  
#include <pthread.h>  
#include <semaphore.h>  
#include <stdio.h>  
#include <unistd.h>  
  
void *printThread01();  
void *printThread02();  
void *printThread03();  
  
#define NUM_THREADS 3  
pthread_t tid[NUM_THREADS]; /* array of thread IDs */  
sem_t semA, semB, semC;    // declare 3 semaphores  
  
#define LOOP 15  
  
int main(int argc, char *argv[])  
{  
    int i, ret;  
  
    // initialize semaphores  
    sem_init(&semA, 0, 1); // Has value 1 so it can start from here  
    sem_init(&semB, 0, 0);  
    sem_init(&semC, 0, 0);  
  
    // δημιουργία 3 νημάτων  
    pthread_create(&tid[0], NULL, printThread01, NULL);  
    pthread_create(&tid[1], NULL, printThread02, NULL);  
    pthread_create(&tid[2], NULL, printThread03, NULL);  
  
    for (i = 0; i < NUM_THREADS; i++)
```

```
pthread_join(tid[i], NULL);

printf("\n\n End of main: reporting that all %d threads have
terminated\n", i);

return 0;
} /* main */

// Συνάρτηση που θα εκτελέσει το νήμα 1
void *printThread01()
{
    int i;

    for (i = 0; i < LOOP; i++)
    {
        sem_wait(&semA);
        printf("<one>");
        sem_post(&semB);
    }
}

// Συνάρτηση που θα εκτελέσει το νήμα 2
void *printThread02()
{
    int i;

    for (i = 0; i < LOOP; i++)
    {
        sem_wait(&semB);
        printf("<two>");
        sem_post(&semC);
    }
}

// Συνάρτηση που θα εκτελέσει το νήμα 3
void *printThread03()
{
    int i;

    for (i = 0; i < LOOP; i++)
    {
        sem_wait(&semC);
        printf("<three>");
        sem_post(&semA);
    }
}
```

## ΤΕΚΜΗΡΙΩΣΗ

*Σχόλια*

- Ο κώδικας κάνει επιτυχής compile
- Όλα τα προαπαιτούμενα της άσκησης έχουν υλοποιηθεί .

*Εκτέλεση κώδικα*

- ```
• gcc -o a ask01.c -lpthread
• ./a
```

*Built-in συναρτήσεις*

Στο πρόγραμμα έχουν γίνει η χρήση των παρακάτω συναρτήσεων:

- pthread\_create() : για τη δημιουργία ενός νέου νήματος
- pthread\_join(): για το συγχρονισμό των διεργασιών
- sem\_init(): για την αρχικοποίηση των semaphores
- sem\_wait(): για κλείδωμα
- sem\_post(): για ξεκλείδωμα

*Ροή*

Η ροή του προγράμματος ορίζεται ως εξής:

- Το κύριο νήμα/διεργασία που είναι η main()
  1. Αρχικοποιεί τα semaphores
  2. Δημιουργεί 3 νήματα
- Τα νήματα εκτελούν τον αντίστοιχο κώδικα που τους έχει ανατεθεί με τη σειρά που ορίζει η άσκηση

**Έγινε η χρήση των sem\_wait() και sem\_post() για το συγχρονισμό εκτέλεσης των νημάτων.**



## Αποτελέσματα

Παρακάτω ακολουθεί ενδεικτική εκτέλεση, Παρατηρούμε ότι εκτελούνται όντως με τη σειρά που αναλύθηκε παραπάνω και δεν γίνονται τυχαίες εκτελέσεις-σειρά. Βλέπουμε ότι εκτυπώθηκαν με τη σειρά <one><two><three> 15 φορές.

### Εκτέλεση 1

```
velasco@DESKTOP-  
A3QHN88:/mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά  
Συστήματα II/Ergasia2$ gcc -o a ask01.c -lpthread  
velasco@DESKTOP-  
A3QHN88:/mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά  
Συστήματα II/Ergasia2$ ./a  
<one><two><three><one><two><three><one><two><three><one><two><three><one><two>  
<three><one><two><three><one><two><three><one><two><three><one><two><three><  
one><two><three><one><two><three><one><two><three><one><two><three><one><two>  
<three><one><two><three>  
  
End of main: reporting that all 3 threads have terminated  
velasco@DESKTOP-  
A3QHN88:/mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά  
Συστήματα II/Ergasia2$
```

## ΑΣΚΗΣΗ 2

### ΚΩΔΙΚΑΣ

```
/*  
;; -----  
;;                                ΒΕΛΑΣΚΟ ΠΑΟΛΑ          cs161020  
;; -----  
;;                                Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών  
;;                                Εργαστήριο ΛΣΙΙ 2020/21 - Εργασία 2.2:  
;;                                THREADS, SEMAPHORES, SOCKETS  
;; -----  
;;                                Υπεύθυνος μαθήματος: ΜΑΜΑΛΗΣ ΒΑΣΙΛΕΙΟΣ  
;;                                Καθηγητές: ΙΟΡΔΑΝΑΚΗΣ ΜΙΧΑΛΗΣ  
;; -----  
*/  
  
#include <pthread.h>  
#include <semaphore.h>  
#include <stdio.h>  
#include <unistd.h>  
#include <stdlib.h>  
  
// Δήλωση mutex για συγχρονισμό  
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;  
  
// Δήλωση barrier για συγχρονισμό  
pthread_barrier_t our_barrier;  
  
int maxElement; // μέγιστη τιμή  
  
// struct για τα δεδομένα που θα σταλούν/λάβει  
struct dataToFunction  
{  
    int lines;  
    int arraySize;  
    int **arrayOriginal;  
    int **arrayOriginalD;  
    int thread_number;  
};  
  
// αρχικοποίηση πίνακα[][]  
void initArrayA(int **mainArrayA, int arrayA_size);  
  
// συνάρτηση που θα εκτελέσει κάθε νήμα  
void *findMax(void *arg);  
  
int main()  
{  
    int **mainArrayA; // Αρχικός πίνακας A
```

```
int **arrayD;          // Αρχικός πίνακας D
int arrayA_size;       // Μέγεθος πίνακα A: n
int number_threads;    // Αριθμός νημάτων: p
int i, j;

// Ζητάει το μέγεθος του πίνακα: n
printf("Give length of array A: ");
scanf("%d", &arrayA_size);

// allocate memory for mainArrayA
mainArrayA = (int **)malloc(arrayA_size * sizeof(int *));
for (i = 0; i < arrayA_size; i++)
    mainArrayA[i] = (int *)malloc(arrayA_size * sizeof(int));

// Check if malloc was successfully completed
if (mainArrayA == NULL)
{
    printf("Error: Not available memory\n");
    exit(EXIT_FAILURE);
}

// allocate memory for arrayD
arrayD = (int **)malloc(arrayA_size * sizeof(int *));
for (i = 0; i < arrayA_size; i++)
    arrayD[i] = (int *)malloc(arrayA_size * sizeof(int));

// Check if malloc was successfully completed
if (arrayD == NULL)
{
    printf("Error: Not available memory\n");
    exit(EXIT_FAILURE);
}

// Ζητάει τον αριθμό των νημάτων: p
printf("Give number of threads: ");
scanf("%d", &number_threads);
pthread_barrier_init(&our_barrier, NULL, number_threads); // Αρχικοποίηση
, το 3ο όρισμα ορίζει πόσα νήματα θα συμμετάσχουν

// Αρχικοποίηση διανύσματος A με τη χρήση συνάρτησης initArrayA
initArrayA(mainArrayA, arrayA_size);

// αρχικοποίηση μέγιστης τιμής με το πρώτο στοιχείο του πίνακα
maxElement = mainArrayA[0][0];

/* Εκτύπωση διανύσματος A
printf("\nContents of array A\n");

for (i = 0; i < arrayA_size; i++)
{
    for (j = 0; j < arrayA_size; j++)
        printf("mainArrayA[%d][%d] = %d\n", i, j, mainArrayA[i][j]);
```

```

    */

    // πόσες γραμμές θα λάβει κάθε νήμα
    int linesOfArray;
    linesOfArray = arrayA_size / number_threads;

    /***** Δημιουργία νημάτων *****/

    pthread_t *array_threads;

    // allocate memory for mainArrayA
    array_threads = (pthread_t *)malloc(sizeof(pthread_t) * number_threads);
    // Check if malloc was successfully completed
    if (array_threads == NULL)
    {
        printf("Error: Not available memory\n");
        exit(EXIT_FAILURE);
    }

    // Δεδομένα που θα σταλθούν στα νήματα - Αρχικοποίηση
    struct dataToFunction data_thread;
    data_thread.lines = linesOfArray;
    data_thread.arrayOriginal = mainArrayA;
    data_thread.arraySize = arrayA_size;
    data_thread.thread_number = 0;
    data_thread.arrayOriginalD = arrayD;

    for (i = 0; i < number_threads; i++)
    {
        // Έτοιμη συνάρτηση για τη δημιουργία νημάτων. Θα εκτελέσουν τη
        // συνάρτηση findMax και θα πάρουν δεδομένα τη δομή data_thread
        pthread_create(&array_threads[i], NULL, findMax, (void
        *)&data_thread);
    }

    // Εγγύηση ότι το αρχικό νήμα θα εκτυπώσει το αποτέλεσμα αφού
    // ολοκληρωθούν όλες οι διεργασίες
    for (i = 0; i < number_threads; i++)
        pthread_join(array_threads[i], NULL);

    // Απελευθέρωση πόρων
    pthread_mutex_destroy(&mutex);
    pthread_barrier_destroy(&our_barrier);

    /* Εκτύπωση διανύσματος μέγιστης τιμής */
    printf("\n\nBack to main thread\nMaximum value in array = %d\n",
    maxElement);

    /* Εκτύπωση διανύσματος D */
    printf("\nContents of array D\n");
    for (i = 0; i < arrayA_size; i++)
    {

```

```
        for (j = 0; j < arrayA_size; j++)
            printf("arrayD[%d][%d] = %d\n", i, j, arrayD[i][j]);
    }

    printf("In main: Finish and Exit\n");

    return 0;
}

// Συνάρτηση: αρχικοποίηση πίνακα A
void initArrayA(int **mainArrayA, int arrayA_size)
{
    int i, j;
    printf("\nGive numbers for mainArrayA\n");
    for (i = 0; i < arrayA_size; i++)
    {
        for (j = 0; j < arrayA_size; j++)
        {
            printf("mainArrayA[%d][%d]: ", i, j);
            scanf("%d", &mainArrayA[i][j]);
        }
    }
}

//Υλοποίηση συνάρτησης που θα εκτελούν τα βήματα για τον υπολογισμό
void *findMax(void *arg)
{
    // Λήψη δεδομένων
    struct dataToFunction *str = (struct dataToFunction *)arg;

    int i, j;
    int local_max = 0;

    // Έλεγχος και δημιουργία συγχρονισμού με mutex
    if (pthread_mutex_lock(&mutex))
    {
        perror("mutex_lock");
        exit(3);
    }

    // print which thread it is
    str->thread_number = str->thread_number + 1;
    printf("\n\nHello I am thread %d (not real thread id) \n", str->thread_number);

    // Αρχικοποίηση από ποιο σημείο του πίνακα θα ξεκινήσουν τον υπολογισμό
    //κάθε νήμα
    int linesLocal = (str->thread_number - 1) * str->lines;
    //printf("Value of lines assigned to me = %d\nMy job to look for the max
    number will start at A[%d][0]\n\n", str->lines, linesLocal);
}
```

```
// Αρχικοποίηση για το τοπικό μέγιστο με το πρώτο στοιχείο
local_max = str->arrayOriginal[linesLocal][0];
// printf("Local max before = %d\n\n", local_max);

// Εκτύπωση σημείων του πίνακα που θα ψάξει
// Ψάχνοντας στον τοπικό πίνακα το μέγιστο αριθμό
for (i = linesLocal; i < linesLocal + str->lines; i++)
{
    for (j = 0; j < str->arraySize; j++)
    {
        // printf("mainArrayA[%d][%d] = %d\n", i, j, str-
        >arrayOriginal[i][j]);
        if (str->arrayOriginal[i][j] > local_max)
        {
            local_max = str->arrayOriginal[i][j];
        }
    }
}

// Εκτύπωση μέγιστης τιμής ανάμεσα στα στοιχεία που έψαξε.
printf("Local max = %d\n\n", local_max);

// Ενημέρωση καθολικής τιμής για μέγιστο αριθμό
if (local_max > maxElement)
{
    maxElement = local_max;
}
// Έλεγχος επιτυχούς τερματισμού συγχρονισμού των mutex
if (pthread_mutex_unlock(&mutex))
{
    perror("pthread_mutex_unlock() error");
    exit(4);
}

// reassure that every thread has found its maximum value
// απαραίτητο για να είμαστε σίγουροι ότι θα γίνει ταυτόχρονα ο παρακάτω
υπολογισμός μεταξύ των άλλων νημάτων
pthread_barrier_wait(&our_barrier);

// Αρχικοποίηση πίνακα με την απόσταση κάθε στοιχείου από το μέγιστο
αριθμό
for (i = linesLocal; i < linesLocal + str->lines; i++)
{
    for (j = 0; j < str->arraySize; j++)
    {
        str->arrayOriginalD[i][j] = maxElement - str-
        >arrayOriginal[i][j];
    }
}
```

```
pthread_exit(NULL);  
}
```

## ΤΕΚΜΗΡΙΩΣΗ

### Σχόλια

- Ο κώδικας κάνει επιτυχής compile
- Όλα τα προαπαιτούμενα της άσκησης έχουν υλοποιηθεί .
- Έχουν γίνει οι κατάλληλοι έλεγχοι για κάθε κλήση συνάρτησης.

### Εκτέλεση κώδικα

- `gcc -o a ask02.c -lpthread`
- `./a`

### Built-in συναρτήσεις

Στο πρόγραμμα έχουν γίνει η χρήση των παρακάτω συναρτήσεων:

- `pthread_create()` : για τη δημιουργία ενός νέου νήματος
- `pthread_join()`: για το συγχρονισμό των διεργασιών
- `pthread_mutex_lock()`: για το συγχρονισμό των διεργασιών – αμοιβαίος αποκλεισμός για κρίσιμα σημεία.
- `pthread_barrier_init()`: για την αρχικοποίηση του barrier
- Για την απελευθέρωση των πόρων

```
pthread_mutex_destroy()  
pthread_barrier_destroy()
```

- `pthread_barrier_wait()`: για το συγχρονισμό των νημάτων

### Ροή

Η ροή του προγράμματος ορίζεται ως εξής:

- Το κύριο νήμα/διεργασία που είναι η `main()` ζητάει από το χρήστη να εισάγει
  1. το μέγεθος του πίνακα
  2. τον αριθμό των νημάτων και



### 3. την εισαγωγή δεδομένων στον πίνακα/διάνυσμα A

- Γίνεται ο υπολογισμός για τη διαμοίραση των κατάλληλων στοιχείων του κάθε νήματος.
- Υστερα, δημιουργούμε μια δομή τύπου data στην οποία αρχικοποιούμε τα μέλη της με τιμές που θέλουμε περαστούν στα νήματα. να
- Στη συνέχεια, δημιουργούνται τα νήματα ενώ το κύριο πρόγραμμα μπλοκάρεται μέχρι να ολοκληρώσουν όλες τις εργασίες που έχει λάβει το κάθε νήμα.
- Τα νήματα εκτελούν τη συνάρτηση findMax (), στην οποία γίνεται ο κατάλληλος υπολογισμός για την εύρεση της μέγιστης τιμής στα στοιχεία που τους έχουν ανατεθεί.
- Έπειτα ενημερώνεται η καθολική μεταβλητή maxElement – χρήση lock/unlock
- Στη συνέχεια, αφού όλα τα νήματα έχουν ολοκληρώσει τη συνεισφορά τους για την εύρεση της μέγιστης τιμής, γίνεται ο υπολογισμός και αρχικοποίηση του πίνακα D
- Τέλος, αφού έχουν τερματίσει όλα τα νήματα, η ροή επιστρέφεται στο κύριο πρόγραμμα, στο οποίο γίνεται και η εκτύπωση της μεταβλητής maxElement και του πίνακα D.

**Κατά την υλοποίηση της άσκησης, παρατηρήθηκε ότι ήταν η απαραίτητη χρήση των mutex καθώς υπήρχαν μεταβλητές που αποτελούσαν κρίσιμα σημεία.**

- `str->thread_number`
- `maxElement`

**Οι μεταβλητές αυτές είναι κοινές στα νήματα, οπότε κάθε τροποποίηση τους χωρίς συγχρονισμό μπορούν να φέρουν λάθος αποτελέσματα.**

#### Αποτελέσματα

Παρακάτω ακολουθούν στιγμιότυπα του τερματικού από διαφορετικές εκτελέσεις του κώδικα. Στην εκτέλεση 1, γίνεται εκτύπωση για λόγο debugging όλοι οι υπολογισμοί. Στη δεύτερη εκτέλεση, αφαιρέθηκαν αυτές ώστε να γίνουν πιο κατανοητά τα αποτελέσματα.

#### Εκτέλεση 1

```
velasco@DESKTOP-
A3QHN88:/mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά
Συστήματα II/Ergasia2$ gcc -o a ask02.c -lpthread
velasco@DESKTOP-
A3QHN88:/mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά
Συστήματα II/Ergasia2$ ./a
Give length of array A: 4
Give number of threads: 2

Give numbers for mainArrayA
mainArrayA[0][0]: 1
mainArrayA[0][1]: 2
mainArrayA[0][2]: 3
```

```
mainArrayA[0][3]: 4
mainArrayA[1][0]: 5
mainArrayA[1][1]: 6
mainArrayA[1][2]: 7
mainArrayA[1][3]: 8
mainArrayA[2][0]: 9
mainArrayA[2][1]: 10
mainArrayA[2][2]: 11
mainArrayA[2][3]: 12
mainArrayA[3][0]: 13
mainArrayA[3][1]: 14
mainArrayA[3][2]: 15
mainArrayA[3][3]: 16
```

Contents of array A

```
mainArrayA[0][0] = 1
mainArrayA[0][1] = 2
mainArrayA[0][2] = 3
mainArrayA[0][3] = 4
mainArrayA[1][0] = 5
mainArrayA[1][1] = 6
mainArrayA[1][2] = 7
mainArrayA[1][3] = 8
mainArrayA[2][0] = 9
mainArrayA[2][1] = 10
mainArrayA[2][2] = 11
mainArrayA[2][3] = 12
mainArrayA[3][0] = 13
mainArrayA[3][1] = 14
mainArrayA[3][2] = 15
mainArrayA[3][3] = 16
```

Hello I am thread 1 (not real thread id)

Value of lines assigned to me = 2

My job to look for the max number will start at A[0][0]

```
mainArrayA[0][0] = 1
mainArrayA[0][1] = 2
mainArrayA[0][2] = 3
mainArrayA[0][3] = 4
mainArrayA[1][0] = 5
mainArrayA[1][1] = 6
mainArrayA[1][2] = 7
mainArrayA[1][3] = 8
Local max = 8
```

Hello I am thread 2 (not real thread id)

Value of lines assigned to me = 2

My job to look for the max number will start at A[2][0]

```
mainArrayA[2][0] = 9
mainArrayA[2][1] = 10
mainArrayA[2][2] = 11
mainArrayA[2][3] = 12
mainArrayA[3][0] = 13
mainArrayA[3][1] = 14
mainArrayA[3][2] = 15
mainArrayA[3][3] = 16
Local max = 16
```

Back to main thread

**Maximum value in array = 16**

Contents of array D

```
arrayD[0][0] = 15
arrayD[0][1] = 14
arrayD[0][2] = 13
arrayD[0][3] = 12
arrayD[1][0] = 11
arrayD[1][1] = 10
arrayD[1][2] = 9
arrayD[1][3] = 8
arrayD[2][0] = 7
arrayD[2][1] = 6
arrayD[2][2] = 5
arrayD[2][3] = 4
arrayD[3][0] = 3
arrayD[3][1] = 2
arrayD[3][2] = 1
arrayD[3][3] = 0
In main: Finish and Exit
```

### Εκτέλεση 2

```
velasco@DESKTOP-
A3QHN88:/mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά
Συστήματα II/Ergasia2$ ./a
Give length of array A: 4
Give number of threads: 2

Give numbers for mainArrayA
mainArrayA[0][0]: 34
mainArrayA[0][1]: 56
mainArrayA[0][2]: 324
mainArrayA[0][3]: 234
mainArrayA[1][0]: 234
mainArrayA[1][1]: 656453
```

```
mainArrayA[1][2]: 43  
mainArrayA[1][3]: 243  
mainArrayA[2][0]: 234  
mainArrayA[2][1]: 523  
mainArrayA[2][2]: 465  
mainArrayA[2][3]: 42  
mainArrayA[3][0]: 341  
mainArrayA[3][1]: 2345  
mainArrayA[3][2]: 246  
mainArrayA[3][3]: 426
```

```
Hello I am thread 1 (not real thread id)  
Local max = 656453
```

```
Hello I am thread 2 (not real thread id)  
Local max = 2345
```

```
Back to main thread  
Maximum value in array = 656453
```

```
Contents of array D  
arrayD[0][0] = 656419  
arrayD[0][1] = 656397  
arrayD[0][2] = 656129  
arrayD[0][3] = 656219  
arrayD[1][0] = 656219  
arrayD[1][1] = 0  
arrayD[1][2] = 656410  
arrayD[1][3] = 656210  
arrayD[2][0] = 656219  
arrayD[2][1] = 655930  
arrayD[2][2] = 655988  
arrayD[2][3] = 656411  
arrayD[3][0] = 656112  
arrayD[3][1] = 654108  
arrayD[3][2] = 656207  
arrayD[3][3] = 656027  
In main: Finish and Exit
```

## ΑΣΚΗΣΗ 3

## ΚΩΔΙΚΑΣ

## Server

```
/* ;; -----  
;;                      ΒΕΛΑΣΚΟ ΠΑΟΛΑ          cs161020  
;; -----  
;;                      Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών  
;;                      Εργαστήριο ΛΣΙΙ 2020/21 - Εργασία 3.1:  
;;                      THREADS, SEMAPHORES, SOCKETS  
;; -----  
;;                      Υπεύθυνος μαθήματος: ΜΑΜΑΛΗΣ ΒΑΣΙΛΕΙΟΣ  
;;                      Καθηγητές: ΙΟΡΔΑΝΑΚΗΣ ΜΙΧΑΛΗΣ  
;; -----  
*/  
  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <sys/un.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <pthread.h>  
#include <semaphore.h>  
  
#define SOCK_PATH "socket"  
  
int correct_sequence; // Number of correct checks that the avg > 10  
int numberOfRequests; // Number of total requests (Whether sequence is ok or  
fail)  
  
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;  
  
void *checkAverage(void *arg); // Συνάρτηση που θα εκτελέσει κάθε νήμα  
  
int main()  
{  
    int sockfd, newsockfd, t, i, len;  
    struct sockaddr_un local, remote;  
    pthread_t thread[50]; // max 50 threads  
  
    // Create socket  
    if ((sockfd = socket(AF_UNIX, SOCK_STREAM, 0)) == -1)  
    {  
        perror("socket");  
        exit(1);  
    }
```

```
}

local.sun_family = AF_UNIX;
strcpy(local.sun_path, SOCK_PATH);
unlink(local.sun_path);
len = strlen(local.sun_path) + sizeof(local.sun_family);

// bind
if (bind(socketfd, (struct sockaddr *)&local, len) == -1)
{
    perror("bind");
    exit(1);
}

// listen / waiting for client
if (listen(socketfd, 5) == -1)
{
    perror("listen");
    exit(1);
}

// Initializations
i = 0;
correct_sequence = 0;

for (;;)
{
    printf("Waiting for connection...\n");
    t = sizeof(remote);
    // Accept client connection on socket.
    if ((newsocketfd = accept(socketfd, (struct sockaddr *)&remote, &t))
== -1)
    {
        perror("accept");
        exit(1);
    }
    printf("Client %d has connected.\n", i + 1);
    //create thread
    pthread_create(&thread[i++], NULL, (void *)&checkAverage, (void
*) &newsocketfd);
}

// Απελευθέρωση πόρου
pthread_mutex_destroy(&mutex);

return 0;
}

void *checkAverage(void *arg)
{
    int sock = *(int *)arg;
    int choice, i, flag;
```

```
char server_response[100];

int array_size, *array_received;
float avg;

choice = 0;

do
{
    int sum = 0;

    // receive array size from client
    recv(sock, &array_size, sizeof(int), 0);

    // allocate memory for array_received
    array_received = (int *)malloc(array_size * sizeof(int));

    // Check if malloc was successfully completed
    if (array_received == NULL)
    {
        printf("Error: Not available memory\n");
        exit(EXIT_FAILURE);
    }

    // receive array data from client
    recv(sock, array_received, sizeof(int) * array_size, 0);

    // sum
    for (i = 0; i < array_size; i++)
    {
        sum += array_received[i];
    }

    // average
    avg = sum / array_size;
    //printf("Average = %f", avg);

    // flag for correct sequences
    if (avg > 10)
        flag = 0;
    else
        flag = 1;

    if (flag == 0) // If sequence was correct
    {
        strcpy(server_response, "Sequence Correct");
        // Έλεγχος και δημιουργία συγχρονισμού με mutex
        if (pthread_mutex_lock(&mutex))
        {
            perror("mutex_lock");
            exit(3);
        }
    }
}
```



```
    }
    correct_sequence++;
    numberOfRequests++;
    // Έλεγχος επιτυχούς τερματισμού συγχρονισμού των mutex
    if (pthread_mutex_unlock(&mutex))
    {
        perror("pthread_mutex_unlock() error");
        exit(4);
    }
}
else if (flag == 1) // If sequence is not correct
{
    strcpy(server_response, "Check Failed");
    // Έλεγχος και δημιουργία συγχρονισμού με mutex
    if (pthread_mutex_lock(&mutex))
    {
        perror("mutex_lock");
        exit(3);
    }
    numberOfRequests++;
    // Έλεγχος επιτυχούς τερματισμού συγχρονισμού των mutex
    if (pthread_mutex_unlock(&mutex))
    {
        perror("pthread_mutex_unlock() error");
        exit(4);
    }
}
else
    strcpy(server_response, "Failed to check (other)");

send(sock, server_response, 50, 0);
send(sock, &avg, sizeof(float), 0);

// Response of client if he wants to continue
recv(sock, &choice, sizeof(int), 0);

printf("Correct Sequences checks : %d\n", correct_sequence);
printf("Number of total requests : %d\n", numberOfRequests);
} while (!choice);

close(sock); // Close socket connection
pthread_exit(NULL);
}
```

### Client

```
/* ;; -----
;;                                ΒΕΛΑΣΚΟ ΠΑΟΛΑ      cs161020
;; -----
```

```
;; Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών
;; Εργαστήριο ΛΣΙΙ 2020/21 - Εργασία 3.2:
;; THREADS, SEMAPHORES, SOCKETS
;; -----
;; Υπεύθυνος μαθήματος: ΜΑΜΑΛΗΣ ΒΑΣΙΛΕΙΟΣ
;; Καθηγητές: ΙΟΡΔΑΝΑΚΗΣ ΜΙΧΑΛΗΣ
;; -----
*/

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define SOCK_PATH "socket"

void initializeArray(int arrayClient[], int arrayA_size); // function to
initialize array by client
int userChoice(int choice); // function for
client to choose

int main()
{
    int sockfd, t;
    struct sockaddr_un server;

    int choice, array_size, i;
    int *arrayClient;

    char server_response_received[50];
    float average;

    // Create socket
    if ((sockfd = socket(AF_UNIX, SOCK_STREAM, 0)) == -1)
    {
        perror("Error creating socket");
        exit(1);
    }

    // Server
    server.sun_family = AF_UNIX;
    strcpy(server.sun_path, SOCK_PATH);

    // Connect client to server - connect()
    if (connect(sockfd, (struct sockaddr *)&server, sizeof(struct
sockaddr_un)) == -1)
    {
        perror("Error connection to server");
        exit(1);
    }
}
```

```
// Program start print
printf("Client connected to server\n");

choice = 0;
do
{
    // Μέγεθος πίνακα
    printf("Give size of array: ");
    do
    {
        scanf("%d", &array_size);
        if (array_size < 0)
        {
            printf("Array size cannot be negative!\n");
            printf("Give array size again: ");
        }
    } while (array_size < 0);

    // send the size of array to server
    send(socketfd, &array_size, sizeof(int), 0);

    // allocate memory for arrayClient
    arrayClient = (int *)malloc(sizeof(int) * array_size);

    // Check if malloc was successfully completed
    if (arrayClient == NULL)
    {
        printf("Error: Not available memory\n");
        exit(EXIT_FAILURE);
    }

    // Αρχικοποίηση του πίνακα
    initializeArray(arrayClient, array_size);

    // Send the data of arrayClient to server
    send(socketfd, arrayClient, sizeof(int) * array_size, 0);

    // Receive server's response
    t = recv(socketfd, server_response_received, 50, 0);
    server_response_received[t] = '\0';
    recv(socketfd, &average, sizeof(float), 0);

    printf("\nAverage calculated by server: %f\n", average);
    printf("Server Response: %s\n", server_response_received);

    // Exit or continue?
    choice = userChoice(choice);

    // send to server client's response
    send(socketfd, &choice, sizeof(choice), 0);
} while (choice != 2);
```

```
}

// Συνάρτηση: αρχικοποίηση πίνακα
void initializeArray(int arrayClient[], int arrayA_size)
{
    int i;
    printf("\nGive numbers for array\n");
    for (i = 0; i < arrayA_size; i++)
    {
        printf("arrayA[%d]: ", i);
        scanf("%d", &arrayClient[i]);
    }
}

// Συνάρτηση: επιλογή χρήστη-πελάτη
int userChoice(int choice)
{
    while (1)
    {
        printf("\nWould you like to give new data?\n");
        printf("1.) Enter new data \n");
        printf("2.) Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                printf("\nChoice 1\n\n");
                return 1;
                break;

            case 2:
                printf("\nExit!\n");
                return 2;
                break;

            default:
                printf("\nThis is not a valid option. Please Select\n\n");
                break;
        }
    }
}
```

## ΤΕΚΜΗΡΙΩΣΗ

### Σχόλια

- Οι κώδικες κάνουν επιτυχής compile
- Όλα τα προαπαιτούμενα της άσκησης έχουν υλοποιηθεί .
- Έχουν γίνει οι κατάλληλοι έλεγχοι για κάθε κλήση συνάρτησης.

### Εκτέλεση κώδικα

Πρώτα τρέχουμε το server και μετά από διαφορετικά τερματικά τρέχουμε τα clients.

```
• gcc -o a ask03_server.c -lpthread
• ./a
• gcc -o b ask03_client.c
• ./b
```

### Built-in συναρτήσεις

Στο πρόγραμμα έχουν γίνει η χρήση των παρακάτω συναρτήσεων:

- pthread\_create() : για τη δημιουργία ενός νέου νήματος
- pthread\_join(): για το συγχρονισμό των διεργασιών
- pthread\_mutex\_lock(): για το συγχρονισμό των διεργασιών – αμοιβαίος αποκλεισμός για κρίσιμα σημεία.
- send(): αποστολή δεδομένων στο socket
- recv(): λήψη δεδομένων από το socket
- bind(): για τη σύνδεση της πόρτας με το server
- listen(): για να ακούει σε αυτή την πόρτα
- accept(): για να λάβει κάποιο πελάτη
- connect(): για να συνδεθεί ο πελάτης στον εξυπηρετητή
- close(): για να κλείσουμε το socket
- socket(): για να δημιουργήσουμε socket

### Ροή

Η ροή του προγράμματος ορίζεται ως εξής:

- Εκτελούμε πρώτα το server

- Στη συνέχεια ο server βρίσκεται σε αναμονή μέχρι να του συνδεθεί κάποιος πελάτης
- Εκτελούμε μετά τον πελάτη
- Η main() ζητάει από το χρήστη να εισάγει
  1. το μέγεθος του πίνακα
  2. την εισαγωγή δεδομένων στον πίνακα/διάνυσμα A
- Ο client περιμένει απάντηση από το server, ενώ ο server κάνει τους κατάλληλους υπολογισμούς για να βρει το Μ.Ο, αύξηση μετρητών και να στείλει τα κατάλληλα μηνύματα.
- Ο client μόλις λάβει τα μηνύματα εκτυπώνει στην οθόνη το Μ.Ο και ύστερα γίνεται επανάληψη για εισαγωγή νέων δεδομένων.

**Κατά την υλοποίηση της άσκησης, παρατηρήθηκε ότι ήταν η απαραίτητη χρήση των mutex καθώς υπήρχαν μεταβλητές που αποτελούσαν κρίσιμα σημεία.**

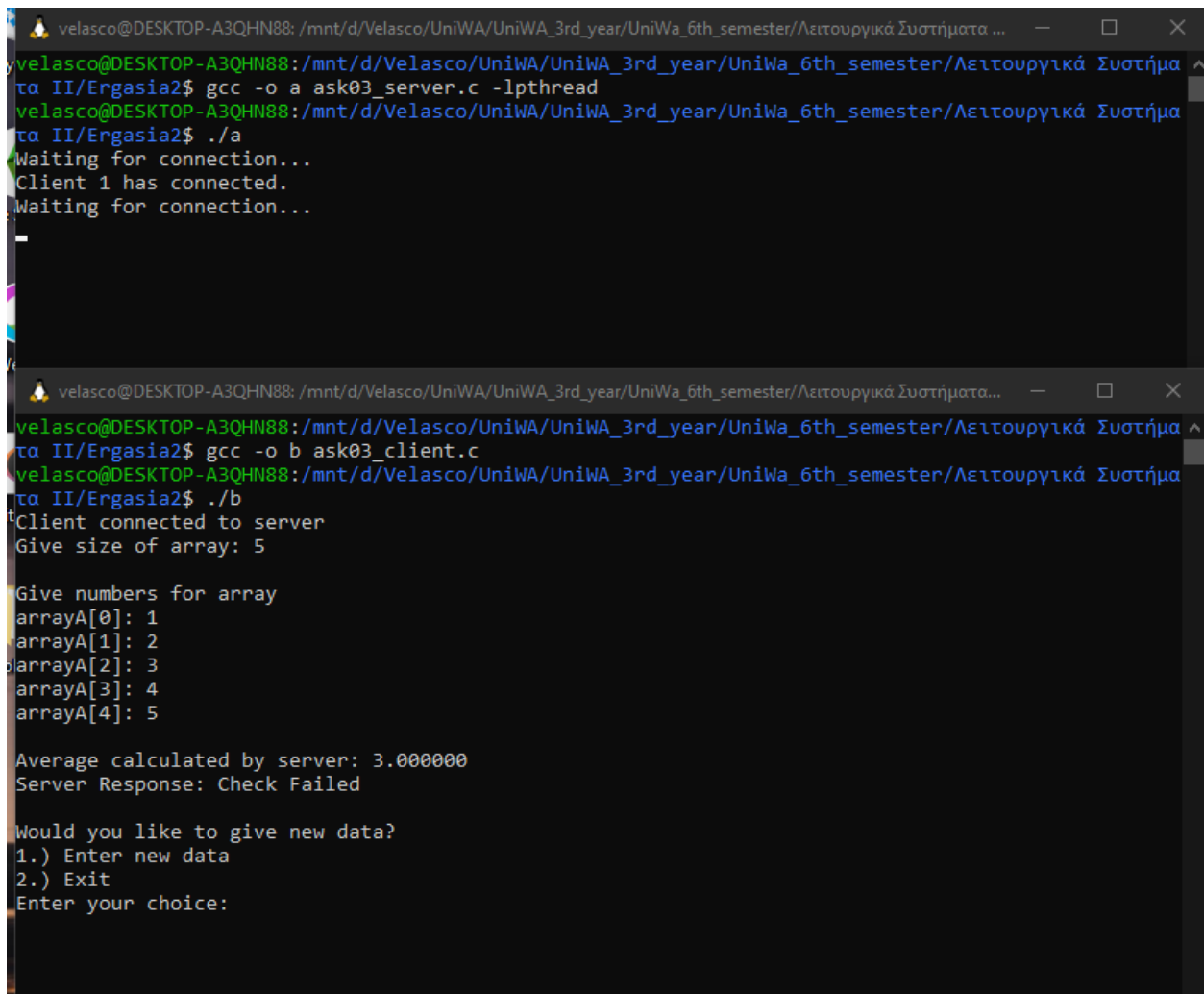
- `correct_sequence`
- `numberOfRequests`

**Οι μεταβλητές αυτές είναι κοινές στα νήματα, οπότε κάθε τροποποίηση τους χωρίς συγχρονισμό μπορούν να φέρουν λάθος αποτελέσματα.**

## Αποτελέσματα

Παρακάτω ακολουθούν στιγμιότυπα του τερματικού από διαφορετικές εκτελέσεις του κώδικα. Παρατηρούμε ότι γίνονται σωστά οι υπολογισμοί, ενώ εκτυπώνονται τα επιθυμητά αποτελέσματα. Βλέπουμε επίσης ότι στον εξυπηρετητή μπορούν να συνδεθούν ταυτόχρονα πελάτες.

### Ένας συνδεδεμένος client



```
velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα ...  
velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2$ gcc -o a ask03_server.c -lpthread  
velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2$ ./a  
Waiting for connection...  
Client 1 has connected.  
Waiting for connection...  
-  
velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα...  
velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2$ gcc -o b ask03_client.c  
velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2$ ./b  
Client connected to server  
Give size of array: 5  
  
Give numbers for array  
arrayA[0]: 1  
arrayA[1]: 2  
arrayA[2]: 3  
arrayA[3]: 4  
arrayA[4]: 5  
  
Average calculated by server: 3.000000  
Server Response: Check Failed  
  
Would you like to give new data?  
1.) Enter new data  
2.) Exit  
Enter your choice:
```



Δύο συνδεδεμένοι clients

```
velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2
Evelasco@DESKTOP-A3QHN88:/mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2$ gcc -o a ask03_server.c -lpthread
Lvelasco@DESKTOP-A3QHN88:/mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2$ ./a
Waiting for connection...
Client 1 has connected.
Waiting for connection...
Client 2 has connected.
Waiting for connection...
Correct Sequences checks : 1
Number of total requests : 2

velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2
velasco@DESKTOP-A3QHN88:/mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2$ gcc -o b ask03_client.c
velasco@DESKTOP-A3QHN88:/mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2$ ./b
Client connected to server
Give size of array: 5

Give numbers for array
arrayA[0]: 1
arrayA[1]: 2
arrayA[2]: 3
arrayA[3]: 4
arrayA[4]: 5

Average calculated by server: 3.000000
Server Response: Check Failed

Would you like to give new data?
1.) Enter new data
2.) Exit
Enter your choice: 1

Choice 1

Give size of array:

velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2
velasco@DESKTOP-A3QHN88:/mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Λειτουργικά Συστήματα II/Ergasia2$ ./b
Client connected to server
Give size of array: 3

Give numbers for array
arrayA[0]: 23
arrayA[1]: 214
arrayA[2]: 314

Average calculated by server: 183.000000
Server Response: Sequence Correct

Would you like to give new data?
1.) Enter new data
2.) Exit
Enter your choice: 1

Choice 1

Give size of array:
```

## Τέσσερις συνδεδεμένοι clients

```
velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Αστυνομία Συστήματα II/Ergasia2
velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Αστυνομία Συστήματα II/Ergasia2
a ask01_server.c -lpthread
velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Αστυνομία Συστήματα II/Ergasia2
Client 1 has connected.
Client 2 has connected.
Waiting for connection...
Correct Sequences checks : 1
Number of total requests : 2
Correct Sequences checks : 1
Number of total requests : 2
Client 3 has connected.
Waiting for connection...
Client 4 has connected.
Waiting for connection...

Select velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Αστυνομία Συστήματα II
arrayA[4]: 5
Average calculated by server: 3.000000
Server Response: Check Failed

Would you like to give new data?
1.) Enter new data
2.) Exit
Enter your choice: 1
Choice 1
Give size of array: -9
Array size cannot be negative!
Give array size again: 5

Select velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Αστυνομία Συστήματα II
velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Αστυνομία Συστήματα II
Client connected to server
Give size of array: 3
Give numbers for array
arrayA[0]: 23
arrayA[1]: 214
arrayA[2]: 314
Average calculated by server: 183.000000
Server Response: Sequence Correct

Would you like to give new data?
1.) Enter new data
2.) Exit
Enter your choice: 1
Choice 1
Give size of array:

velasco@DESKTOP-A3QHN88: /mnt/d/Velasco/UniWA/UniWA_3rd_year/UniWa_6th_semester/Αστυνομία Συστήματα II/Ergasia2
Client connected to server
Give size of array: 10
Give numbers for array
arrayA[0]: 23
arrayA[1]: 413
arrayA[2]: 34
arrayA[3]: 133145
arrayA[4]: 315
arrayA[5]: 31413
arrayA[6]: 124
arrayA[7]: 213
arrayA[8]: 1234
arrayA[9]: 124
Average calculated by server: 16703.000000
Server Response: Sequence Correct

Would you like to give new data?
1.) Enter new data
2.) Exit
Enter your choice: 9
This is not a valid option. Please Select Another

Would you like to give new data?
1.) Enter new data
2.) Exit
Enter your choice:

sias2$ gcc -o a ask01.c -lpthread
sias2$ ./a
two<three>one<two><three>one<two><three>on
sias2$ gcc -o a ask01.c -lpthread
sias2$ ./a
two<three>one<two><three>one<two><three>on
sias2$
```

## ΣΥΜΠΕΡΑΣΜΑ

Έχοντας πλέον εκπονήσει την εργαστηριακή άσκηση 2, έχουμε καταλάβει την πλήρη λειτουργία των threads και πως μπορούμε να τα συγχρονίσουμε με barriers ή με τη χρήση των semaphores. Επίσης, μάθαμε πώς μπορούμε να αναστέλλουμε ή να μπλοκάρουμε μια διεργασία μέχρι να ολοκληρωθεί μια άλλη.

Κατά την υλοποίηση των προγραμμάτων client – server, είδαμε πως μπορούμε να επικοινωνήσουμε προγράμματα που τρέχουν σε άλλη μηχανήματα με τη χρήση των socket. Συγκεκριμένα, σε αυτή την άσκηση υλοποιήθηκε παράδειγμα για UNIX domain sockets. Τέλος, είδαμε πως μπορεί με τη χρήση των νημάτων, ένας εξυπηρετητής να εξυπηρετεί ταυτόχρονα πελάτες.

## ΒΙΒΛΙΟΓΡΑΦΙΑ

*Eclass UniWA*. (χ.χ.). Ανάκτηση από <https://eclass.uniwa.gr>:

<https://eclass.uniwa.gr/modules/document/index.php?course=CS122&openDir=/5c812783j8cF>