ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ

# Εξάρχου Αθανάσιος

2η Εργασία στο μάθημα **Λειτουργικά Συστήματα**

Ταύρος, 5 Φεβρουαρίου 2024

Περιεχόμενα

# Άσκηση 2

## Κώδικας

Ο κώδικας της 2ης εργασίας  που δημιουργήθηκε μαζί με τα σχόλια είναι:

```
/* includes */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <signal.h>
#include <unistd.h>
#include <sys/wait.h>
#include <pthread.h>
#include <semaphore.h>
#include <fcntl.h>

/* defines */
#define LINES 100
#define NUMS_PER_LINE 50
#define THREADS 4
#define LINES_PER_THREAD (LINES / THREADS)
```

```
/* global variable declaration/initialization */
sem_t *sem;
pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;
int linesthreadread[THREADS];
int sum = 0; /* global variable to contain the total sum */

/* function declaration */
void sighandler(int signum);
void *thread_func(void *args);
void fileReader();

/* handles the threads */
void* thread_func(void *args) {

    /* variable declaration */
    int *argPointer = args;
    char buffer[256];
    long int partialSum = 0;
    int threadlines = 0;
    int fd;

    fd = open("data.txt", O_RDONLY, 0400); /* opens "data.txt" with read-only permission */
    /* in the case that the file failed to open, prints out an appropriate message and exits */
    if (fd == -1) {
```

```
    perror("An error occurred while attempting to open the file: \"data.txt\"! Exiting...");
    exit(1);
}

/* the threads are given access to 25 lines each */
for (long int i = 0; i < LINES_PER_THREAD; i++) {

/* in the case that the reading of the file failed, prints out an appropriate message and exits */
    if (read(fd, buffer, sizeof(buffer)) == -1) {
        perror("A problem occurred while reading the file! Exiting...");
        exit(1);
    }
    /* makes each number have a space (" ") between itself and its neighbour */
    char *token = strtok(buffer, " ");
    while (token != NULL) {

        /* checks whether the value is a number or a space and acts accordingly */
        if (strcmp(token, " ") != 0) {

            /* converts the token to a number */
            int value = atoi(token);
            /* adds the number to partialSum */
            partialSum += value;
```

```
            }
                token = strtok(NULL, " ");
            }
        threadlines++; //increments the lines read by the current thread
    }

    close(fd); /* closes the file */

    //https://docs.oracle.com/cd/E19455-01/806-5257/sync-12/index.html
    pthread_mutex_lock(&mymutex);
    sum += partialSum; /* puts the current thread's sum to the total sum-counter variable */
    pthread_mutex_unlock(&mymutex);

    linesthreadread[*argPointer] = threadlines;

    //https://man7.org/linux/man-pages/man3/sem_post.3.html
    sem_post(sem); /* notifies Process 2 that the current thread has finished reading */

    //https://stackoverflow.com/questions/20824229/when-to-use-pthread-exit-and-when-to-use-pthread-join-in-linux
    pthread_exit(NULL);
}

/* handles the signals */
void sighandler(int signum) {
```

```
    if (signum == SIGINT || signum == SIGTERM) {

        char input; //user's choice variable
        /* informs the user on the following action via appropriate message */
        printf("You are about to exit the program. Do you wish to proceed with this action? (y/n): ");
        scanf(" %c", &input); /* reads the user's input */

        while (input != 'y' && input != 'Y' && input != 'n' && input != 'N') {
            printf("Invalid input! Enter either 'y' to exit the program or 'n' to continue: ");
            scanf(" %c", &input); /* reads the user's input */
        }
        /* case where the user agreed to exit the program */
        if (input == 'y' || input == 'Y') {



            sem_post(sem); /* wakes up the next thread (in the case that one exists) */
            sem_close(sem); /* closes the semaphore */
            sem_unlink("/it2022134"); /* unlinks the semaphore */
            exit(0); //successfully exits the program
        }
    }
}
```

```
/* handles the reading of the file */
void fileReader() {

    sem_wait(sem); /* waits for the previous thread to finish operating (if it exists) */

    pthread_t threads[THREADS];
    int count[THREADS];

    /* initializes the threads */
    for (int i = 0; i < THREADS; i++) {

        count[i] = i; /* counter variable */
        /* in the case that the creation of the thread fails, prints out an appropriate message and exits */
        if (pthread_create(&threads[i], NULL, thread_func, &count[i]) != 0) {
            perror("An error occurred in the process of creating a thread. Exiting...");
            exit(1);
        }
    }

    /* notifies all 4 threads to begin reading */
    sem_post(sem);

    /* joins the threads */
    for (int i = 0; i < THREADS; i++) {
```

```
        pthread_join(threads[i], NULL);
    }

    /* displays the number of lines read by each thread */
    for (int i = 0; i < THREADS; i++) {
        printf("Thread no.%d read %d lines.\n", i, linesthreadread[i]);
    }

    printf("Total sum of numbers contained in \"data.txt\": %d\n", sum);
}

int main (int argc, char *argv[]) {

    srand(time(NULL)); /* randomization */

    /* calls the signal handler function with the signal provided*/
    signal(SIGINT, sighandler);
    signal(SIGTERM, sighandler);

    const char *semName = "/it2022134";
    /* creates and initializes my semaphore */
    sem = sem_open(semName, O_CREAT, 0600, 1);
    /* in the case that the creation of the semaphore fails, prints out an appropriate message and exits */
    if (sem == SEM_FAILED) {
```

```
    perror("An error occurred in the process of creating a semaphore! Exiting...");
    exit(1);
}

/* process id variable */
pid_t pid;

/* creates parent and child processes */
pid = fork();

/* child case */
if (pid == 0) {
    /* child process which takes on the role of writing to the file */
    int fd;
    fd = open("data.txt", O_CREAT | O_WRONLY | O_TRUNC, 0600);
    /* in the case that the opening of the file fails, prints out an appropriate message and exits */
    if (fd == -1) {

        perror("An error occurred while trying to open the file! Exiting");
        exit(1);
    }

    /* variable declaration */
```

```
char buffer[256];


for (int i = 0; i < LINES; i++) {
    for (int j = 0; j < NUMS_PER_LINE; j++) {


        /* variable declaration */
        int rng = rand() % 100;
        //https://www.geeksforgeeks.org/snprintf-c-library/
        int len = snprintf(buffer, sizeof(buffer), "%d ", rng);


        /* in the case that the writing to the file fails, prints out an appropriate message and exits */
        if (write(fd, buffer, len) == -1) {
            perror("Error writing to the file");
            close(fd); /* closes the file */
            exit(1);
        }
    }
    if (i < 99) {
        if (write(fd, "\n", 1) == -1) {
            /* in the case that the writing to the file fails, prints out an appropriate message and exits */
            perror("Error writing newline to the file");
            close(fd);
            exit(1);
        }
```

```
        }
    }

    close(fd); /* closes the file */

    sem_post(sem); /* notifies the parent process to proceed */

    fileReader(); /* calls the fileReader function */

} else if (pid > 0) { /* parent case */

    waitpid(pid, NULL, 0);  /* waits for the child process to finish */

    sem_unlink("/it2022134"); /* unlinks my semaphore */

    //https://www.ibm.com/docs/en/zos/2.2.0?topic=functions-pthread-mutex-destroy-delete-mutex-object
    pthread_mutex_destroy(&mymutex); /* deletes the mutex */

//https://stackoverflow.com/questions/35926364/how-important-is-it-to-call-sleep-in-a-infinite-while-loop
    while (1)
        sleep(1);

} else {
    /* in the case that the creation of the child process fails, prints out an appropriate message and exits */
```

```
        perror("An error occurred while creating Process no.1! Exiting...");
        exit(1);
    }

    return 0;
}
```

## Τρόπος Εκτέλεσης

Στην αρχή του κώδικα, δημιουργούνται με τη χρήση της fork() δύο διεργασίες, οι διεργασίες πατέρας και παιδί. Ο πατέρας έχει λάβει το ρόλο εγγραφής 100 γραμμών με 50 αριθμούς(0-99), όπου η κάθε μία γραμμή εγγράφεται στο αρχείο “data.txt”, ενώ το παιδί παίζει το ρόλο διαβάσματος αυτών των αριθμών μέσω τεσσάρων νημάτων τα οποία διαβάζουν 25 γραμμές το καθένα. Οι αριθμοί αυτοί είναι τυχαίοι και η διεργασία παιδί εκτελείται μόνο εφόσον έχει τελειώσει ο πατέρας.

Επιπρόσθετα, ελέγχονται και τα σήματα SIGINT & SIGTERM για την έξοδο από το πρόγραμμα.

Με χρήση σημαφόρων υλοποιείται ο συγχρονισμός των διεργασιών και επιβεβαιώνεται ότι δεν θα υπάρξει λάθος.

## Ενδεικτικές εκτελέσεις (screenshots):

*Βασική διεργασία η οποία ελέγχει τη δημιουργία των κατάλληλων διεργασιών και τον έλεγχο του προγράμματος*

**Screenshots**

Η δημιουργία του parent-child pid γίνεται στη main, όπως και ο έλεγχος του προγράμματος

main:

```c
int main (int argc, char *argv[]) {

    srand(time(NULL)); /* randomization */

    /* calls the signal handler function with the signal provided*/
    signal(SIGINT, sighandler);
    signal(SIGTERM, sighandler);

    const char *semName = "/it2022134";
    /* creates and initializes my semaphore */
    sem = sem_open(semName, O_CREAT, 0600, 1);
    /* in the case that the creation of the semaphore fails, prints out an appropriate message and exits */
    if (sem == SEM_FAILED) {

        perror("An error occurred in the process of creating a semaphore! Exiting...");
        exit(1);
    }

    /* process id variable */
    pid_t pid;

    /* creates parent and child processes */
    pid = fork();

    /* child case */
    if (pid == 0) {
        /* child process which takes on the role of writing to the file */
        int fd;
        fd = open("data.txt", O_CREAT | O_WRONLY | O_TRUNC, 0600);
        /* in the case that the opening of the file fails, prints out an appropriate message and exits */
        if (fd == -1) {

            perror("An error occurred while trying to open the file! Exiting");
            exit(1);
        }

        /* variable declaration */
        char buffer[256];
```

```c

        for (int i = 0; i < LINES; i++) {
            for (int j = 0; j < NUMS_PER_LINE; j++) {

                /* variable declaration */
                int rng = rand() % 100;
                //https://www.geeksforgeeks.org/snprintf-c-library/
                int len = snprintf(buffer, sizeof(buffer), "%d ", rng);

                /* in the case that the writing to the file fails, prints out an appropriate message and exits */
                if (write(fd, buffer, len) == -1) {
                    perror("Error writing to the file");
                    close(fd); /* closes the file */
                    exit(1);
                }
            }
            if (i < 99) {
                if (write(fd, "\n", 1) == -1) {
                    /* in the case that the writing to the file fails, prints out an appropriate message and exits */
                    perror("Error writing newline to the file");
                    close(fd);
                    exit(1);
                }
            }
        }

        close(fd); /* closes the file */

        sem_post(sem); /* notifies the parent process to proceed */

        fileReader(); /* calls the fileReader function */

    } else if (pid > 0) { /* parent case */

        waitpid(pid, NULL, 0);  /* waits for the child process to finish */

        sem_unlink("/it2022134"); /* unlinks my semaphore */
```

```c

        sem_unlink("/it2022134"); /* unlinks my semaphore */

        //https://www.ibm.com/docs/en/zos/2.2.0?topic=functions-pthread-mutex-destroy-delete-mutex-object
        pthread_mutex_destroy(&mymutex); /* deletes the mutex */

    //https://stackoverflow.com/questions/35926364/how-important-is-it-to-call-sleep-in-a-infinite-while-loop
        while (1)
            sleep(1);

    } else {
        /* in the case that the creation of the child process fails, prints out an appropriate message and exits */
        perror("An error occurred while creating Process no.1! Exiting...");
        exit(1);
    }

    return 0;
}
```

Συγχρονισμός των 2 διεργασιών εγγραφής ανάγνωσης

**Screenshots**

Συγχρονισμός των 2 διεργασιών εγγραφής και ανάγνωσης γίνεται στη συνάρτηση *“sighandler”*

```c
/* case where the user agreed to exit the program */
if (input == 'y' || input == 'Y') {


    sem_post(sem); /* wakes up the next thread (in the case that one exists) */
    sem_close(sem); /* closes the semaphore */
    sem_unlink("/it2022134"); /* unlinks the semaphore */
    exit(0); //successfully exits the program
}
```

Επίσης, συγχρονισμός των 2 διεργασιών εγγραφής και ανάγνωσης γίνεται στη συνάρτηση *“fileReader”*

```c
/* handles the reading of the file */
void fileReader() {

    sem_wait(sem); /* waits for the previous thread to finish operating (if it exists) */

    pthread_t threads[THREADS];
    int count[THREADS];

    /* initializes the threads */
    for (int i = 0; i < THREADS; i++) {

        count[i] = i; /* counter variable */
        /* in the case that the creation of the thread fails, prints out an appropriate message and exits */
        if (pthread_create(&threads[i], NULL, thread_func, &count[i]) != 0) {
            perror("An error occurred in the process of creating a thread. Exiting...");
            exit(1);
        }
    }

    /* notifies all 4 threads to begin reading */
    sem_post(sem);
```

Τέλος, συγχρονισμός των 2 διεργασιών εγγραφής και ανάγνωσης γίνεται στη *“main”*

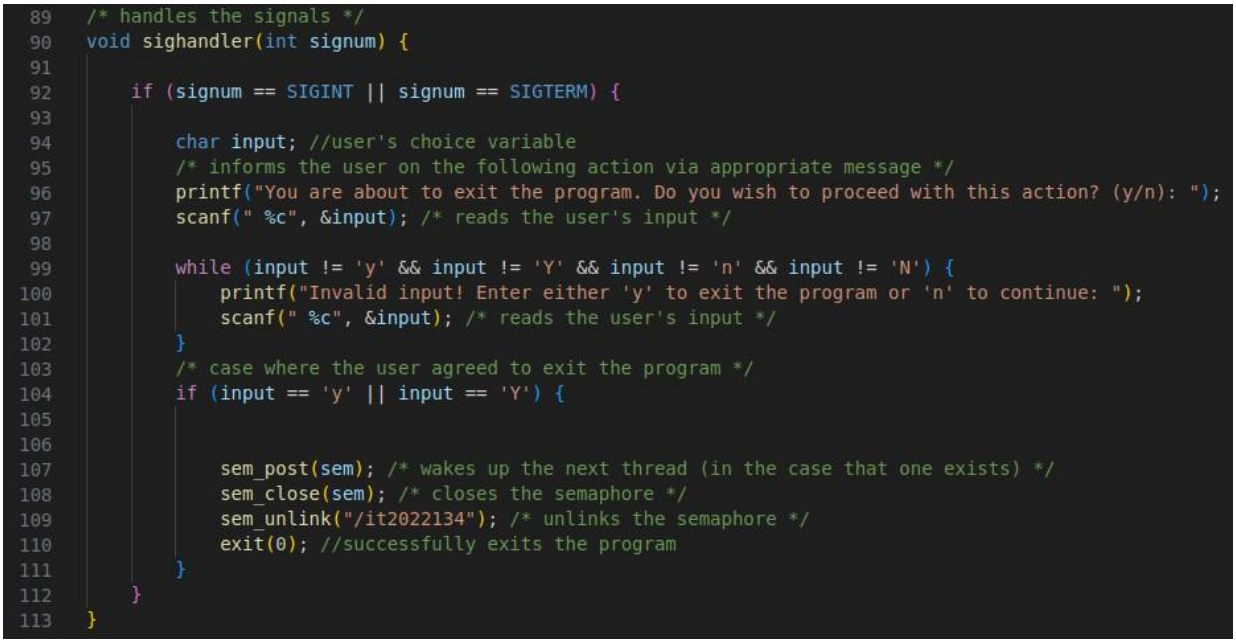
```
158     const char *semName = "/it2022134";
159     /* creates and initializes my semaphore */
160     sem = sem_open(semName, O_CREAT, 0600, 1);
161     /* in the case that the creation of the semaphore fails, prints out an appropriate message and exits */
162     if (sem == SEM_FAILED) {
163 
164         perror("An error occurred in the process of creating a semaphore! Exiting...");
165         exit(1);
166     }
```

```
216         sem_post(sem); /* notifies the parent process to proceed */
217 
218         fileReader(); /* calls the fileReader function */
219 
220     } else if (pid > 0) { /* parent case */
221 
222         waitpid(pid, NULL, 0);  /* waits for the child process to finish */
223 
224         sem_unlink("/it2022134"); /* unlinks my semaphore */
225 
226         //https://www.ibm.com/docs/en/zos/2.2.0?topic=functions-pthread-mutex-destroy-delete-mutex-object
227         pthread_mutex_destroy(&mymutex); /* deletes the mutex */
228 
229     //https://stackoverflow.com/questions/35926364/how-important-is-it-to-call-sleep-in-a-infinite-while-loop
230         while (1)
231             sleep(1);
```

*Διαχείριση σημάτων*

**Screenshots**

Η διαχείριση των σημάτων γίνεται στη συνάρτηση “sighandler”



```c
/* handles the signals */
void sighandler(int signum) {

    if (signum == SIGINT || signum == SIGTERM) {

        char input; //user's choice variable
        /* informs the user on the following action via appropriate message */
        printf("You are about to exit the program. Do you wish to proceed with this action? (y/n): ");
        scanf(" %c", &input); /* reads the user's input */

        while (input != 'y' && input != 'Y' && input != 'n' && input != 'N') {
            printf("Invalid input! Enter either 'y' to exit the program or 'n' to continue: ");
            scanf(" %c", &input); /* reads the user's input */
        }
        /* case where the user agreed to exit the program */
        if (input == 'y' || input == 'Y') {


            sem_post(sem); /* wakes up the next thread (in the case that one exists) */
            sem_close(sem); /* closes the semaphore */
            sem_unlink("/it2022134"); /* unlinks the semaphore */
            exit(0); //successfully exits the program
        }
    }
}
```

*Δημιουργία νημάτων και πέρασμα παραμέτρων*

**Screenshots**

Η δημιουργία νημάτων και το πέρασμα παραμέτρων στα νήματα γίνεται στη συνάρτηση “fileReader”

```c
    pthread_t threads[THREADS];
    int count[THREADS];

    /* initializes the threads */
    for (int i = 0; i < THREADS; i++) {

        count[i] = i; /* counter variable */
        /* in the case that the creation of the thread fails, prints out an appropriate message and exits */
        if (pthread_create(&threads[i], NULL, thread_func, &count[i]) != 0) {
            perror("An error occurred in the process of creating a thread. Exiting...");
            exit(1);
        }
    }

```

*Συγχρονισμός νημάτων και σωστή διαδικασία μέτρησης αποτελέσματος*

**Screenshots**

*Συγχρονισμός νημάτων:*

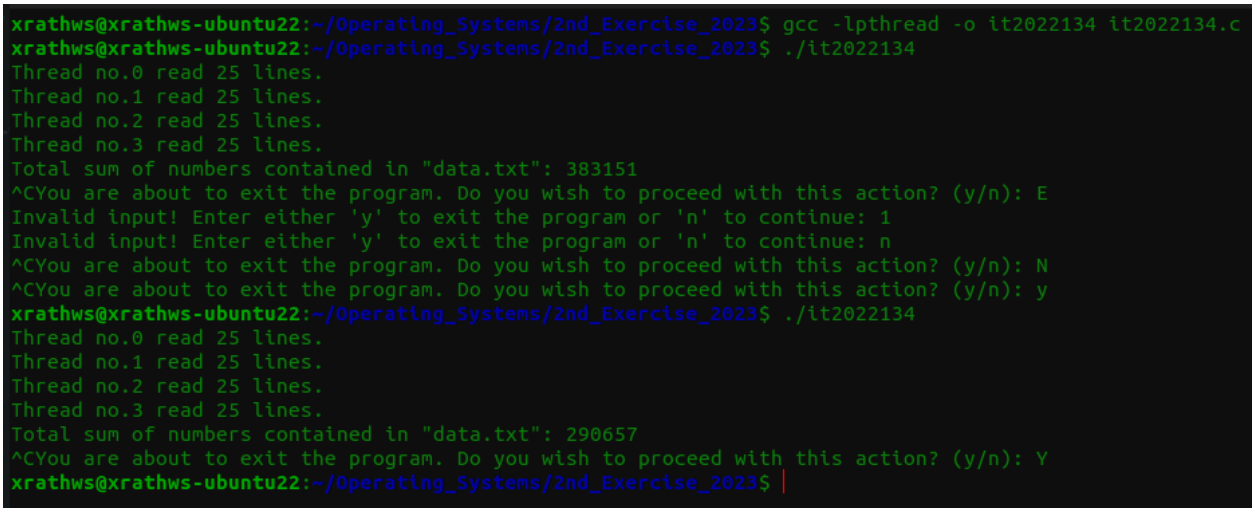
```c
/* joins the threads */
for (int i = 0; i < THREADS; i++) {
    pthread_join(threads[i], NULL);
}
```

*Διαδικασία μέτρησης αποτελέσματος:*

```c
/* the threads are given access to 25 lines each */
for (long int i = 0; i < LINES_PER_THREAD; i++) {

/* in the case that the reading of the file failed, prints out an appropriate message and exits */
    if (read(fd, buffer, sizeof(buffer)) == -1) {
        perror("A problem occurred while reading the file! Exiting...");
        exit(1);
    }
    /* makes each number have a space (" ") between itself and its neighbour */
    char *token = strtok(buffer, " ");
    while (token != NULL) {

        /* checks whether the value is a number or a space and acts accordingly */
        if (strcmp(token, " ") != 0) {

            /* converts the token to a number */
            int value = atoi(token);
            /* adds the number to partialSum */
            partialSum += value;
        }
            token = strtok(NULL, " ");
        }
    threadlines++; //increments the lines read by the current thread
}

close(fd); /* closes the file */

//https://docs.oracle.com/cd/E19455-01/806-5257/sync-12/index.html
pthread_mutex_lock(&mymutex);
sum += partialSum; /* puts the current thread's sum to the total sum counter-variable */
pthread_mutex_unlock(&mymutex);
```

*Ομαλή εκτέλεση προγράμματος, error handling, τεκμηρίωση*

**Screenshots**




Στο παραπάνω screenshot, φαίνεται η ομαλή εκτέλεση του προγράμματος.

Error handling:

Ο κώδικας υλοποιήθηκε με τέτοιο τρόπο, έτσι ώστε να δέχεται μόνο τα ‘y’, ‘Y’, ‘n’ και ‘N’ ως σωστή είσοδο από το χρήστη, για την έξοδό του ή όχι από το πρόγραμμα.

## Γενικά Σχόλια/Παρατηρήσεις

Γενικά, η όλη κατανόηση της απόλυτα σωστής υλοποίησης του κώδικα ήταν μια πολύ χρονοβόρα διαδικασία, η οποία τελικά δεν ολοκληρώθηκε τελείως και με απόλυτη επιτυχία.

## Με δυσκόλεψε / δεν υλοποίησα

Η διαδικασία υπολογισμού του συνολικού αθροίσματος από τα νήματα αποδείχθηκε αρκετά περίπλοκη και δεν κατόρθωσα να την υλοποιήσω πλήρως. Ως αποτέλεσμα, η τιμή που εμφανίζεται στο μήνυμα *“Total sum of numbers contained in data.txt”* είναι εσφαλμένη και δεν αντικατοπτρίζει τον πραγματικό αριθμητικό υπολογισμό.

# Συνοπτικός Πίνακας

| 2η Εργασία | | |
|---|---|---|
| **Λειτουργία** | Υλοποιήθηκε **(ΝΑΙ/ΟΧΙ/ΜΕΡΙΚΩΣ)** | **Συνοπτικές Παρατηρήσεις** |
| Βασική διεργασία η οποία ελέγχει τη δημιουργία των κατάλληλων διεργασιών και τον έλεγχο του προγράμματος | **ΝΑΙ** | |
| Συγχρονισμός των 2 διεργασιών εγγραφής ανάγνωσης | **ΝΑΙ** | |
| Διαχείριση σημάτων | **ΜΕΡΙΚΩΣ** | Δεν έχω καταφέρει να επιβεβαιώσω ότι διαχειρίζομαι σωστά το *SIGTERM.* |
| Δημιουργία νημάτων και πέρασμα παραμέτρων | **ΝΑΙ** | |
| Συγχρονισμός νημάτων και σωστή διαδικασία μέτρησης αποτελέσματος | **ΜΕΡΙΚΩΣ** | Η τιμή του *total sum* δεν είναι αντικειμενική στη πραγματικότητα. |
| Ομαλή εκτέλεση προγράμματος, error handling, τεκμηρίωση | **ΝΑΙ** | |