ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ

ΠΑΝΑΓΙΩΤΗΣ ΠΕΤΕΙΝΑΡΗΣ

2η Εργασία στο μάθημα **Λειτουργικά Συστήματα**

Ταύρος, ΧΧ Ιανουαρίου 2023

Περιεχόμενα

Άσκηση 2

Κώδικας

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

```
#include <stdio.h>
#include <unistd.h>
```

```
#include <sys/wait.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <fcntl.h>
#include <signal.h>


//This is the shared array which I stored the number of appearance of every
letter.
int countLetterSharedArray[26] = {0};

//This is for protecting the countLetterSharedArray from concurrent access by
the four threads I create later.
//It locks the mutex and prevent other threads from accessing the shared array
until the first thread has unlocked the mutex
pthread_mutex_t lock;


//This is the function which I handle the signals SIGINT and SIGTERM.
void signalHandler() {
    char answer;

    //I ask the user if he wants to terminate the program and the scan his
answer.
    //If it is 'y' the program will finish with exit code 0, else if it is 'n'
it will continue.
    printf("Received a signal. Do you want to terminate the program?"
           "y or n\n");
    scanf("%c", &answer);
    if(answer == 'y') {
        printf("Program has been terminated\n");
        exit(0);
    }
}

//This function is for reading from the data.txt.
//Every thread will read 500 characters from the file using a file descriptor.
//I use a void pointer function and a void pointer argument because this
function is used later in the pthread_create.
```

```
void *countLettersInThread(void *arg) {

    //Casting the argument as long offset.
    long offset = (long) arg;

    int localCountArray[26] = {0};

    //The open system call open the path 'data.txt' in a read only mode.
    int file_descriptor;
    file_descriptor = open("data.txt", O_RDONLY);

    //If the open system call returns -1 then I print an error message and
terminate with exit code 1.
    if (file_descriptor == -1) {
        printf("Error Message : Can not open file!");
        exit(1);
    }

    //The Iseek function is used for moving the file_descriptor in a specific
position in the 'data.txt' through the offset variable.
    lseek(file_descriptor, offset, SEEK_SET);

    int i;

    //Using a for loop 500 times, the read function ,using the file_descriptor,
read from the file
    //a character, one at a time, and store it in the address of the c
variable.
    for (i = 0; i < 500; i++) {
        char c;
        read(file_descriptor, &c, 1);

        //I check if the character that the function read from the file is
among 'a'-'z'.
        //I decrease the c variable value by the ascii value of the letter 'a'
because
        //I want to simulate the positions of the table as the letters of the
alphabet.
        if (c >= 'a' && c <= 'z') {
            localCountArray[c - 'a']++;
        }
    }
```

```
    //This close system call close the file_descriptor that was opened with
open system call
    //and makes it available for reuse.
    close(file_descriptor);

    //With this function I acquire the lock and then can safely access the
shared array.
    pthread_mutex_lock(&lock);

    //Using the for loop I move the localCountArray values in the shared array.
    for (i = 0; i < 26; i++) {
        countLetterSharedArray[i] += localCountArray[i];
    }

    //With that functions the lock is released to be used again from another
thread.
    //Then exit the thread that has been called.
    pthread_mutex_unlock(&lock);
    pthread_exit(NULL);
}

int main() {

    int pid;
    srand(time(0));

    //With this function I handle the two specific signals with the function
signalHandler.
    signal(SIGINT, signalHandler);
    signal(SIGTERM, signalHandler);

    //I raise a signal for testing the signalHandler function.
    //raise(SIGTERM);

    //Creating the new process
    pid = fork();


    //Using the if to check the fork function response. If it is less than 0 an
error
    //has occurred and then I print an error message in stderr stream.
```

```
    if (pid < 0) {
        fprintf(stderr, "The Fork function failed!");
        return 1;

    } else if (pid == 0) {

        //Here I open the file 'data.txt' in write mode.
        FILE *file;
        file = fopen("data.txt", "w");

        //Checking if the file has opened, if not print error message.
        if (file == NULL) {
            printf("Error Message : Can not open file!\n");
            return 1;
        }

        //Here I create a string with 2001 characters. With the for loop I fill
the string
        //with random characters from 'a' to 'z'.
        char randomCharactersString[2001];
        for (int i = 0; i < 2000; i++) {
            randomCharactersString[i] = (rand() % 26) + 'a';
        }

        //Define the end of the string.
        randomCharactersString[2000] = '\0';

        //Print the string to the file 'data.txt' nad then close the stream to
the file.
        fprintf(file, "%s\n", randomCharactersString);
        fclose(file);

        printf("First process: Writing in the file complete\n");

    } else {

        // Wait for first process to finish writing to file and then proceed to
the second process.
        wait(NULL);

        //Here I store the unique ID that is created from pthread_create
        //and then initialize a mutex with the address of lock and without any
```

```
attribute.
        pthread_t threads_ID[4];
        pthread_mutex_init(&lock, NULL);


        //A variable for checking if the thread was successfully created.
        int threadCheck;

        //Using a for loop for every thread.
        for (int i = 0; i < 4; i++) {

            //Setting the offset so that every thread will read 500 characters.
            long offset = i * 500;

            //Here I create the new threads. First I store the threads ID in
the address of threads_ID, then setting NULL the attributes of the threads.
            //Using a pointer to the function countLetterInThread that the
threads will execute. In the end I cast the offset as void pointer so I can
            //use it in the function.
            threadCheck = pthread_create(&threads_ID[i], NULL,
countLettersInThread, (void *) offset);

            //Checking if the threads was created correctly, if not print an
error message.
            if(threadCheck !=0 ) {
                printf("Error Message : Thread did not created successfully!");
                return 1;
            }

            //This function block the execution of the calling thread until the
joined thread finish the execution.
            pthread_join(threads_ID[i], NULL);
        }

        //Using a for loop to print the string with characters.
        for (int i = 0; i < 26; i++) {
            printf("Letter %c has appeared : %d times\n", i + 'a',
countLetterSharedArray[i]);
        }

        printf("Second  process: Counting letters complete\n");
```

```
        //This function destroy the mutex.
        pthread_mutex_destroy(&lock);

    }

    return 0;
}
```

Τρόπος Εκτέλεσης

1) gcc -o it21978 it21978.c -lpthread για να κάνουμε compile.

2) ./it21978 για να το τρέξουμε.

Ενδεικτικές εκτελέσεις (screenshots):

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

Screenshots

```
//Creating the new process
pid = fork();



//Using the if to check the fork function response. If it is less than 0 an error
//has occurred and then I print an error message in stderr stream.
if (pid < 0) {
    fprintf( stream: stderr,  format: "The Fork function failed!");
    return 1;

} else if (pid == 0) {
```

```
    printf( format: "First process: Writing in the file complete\n");

} else {

    // Wait for first process to finish writing to file and then proceed to the second process.
    wait( stat_loc: NULL);
```

Η δημιουργία των δύο διεργασιών και ο έλεγχος. Στο τέλος κάθε διεργασίας εκτυπώνεται και μήνυμα, πως ολοκληρώθηκε επιτυχημένα. Φαινεται σε παρακατω screenshot.

Συγχρονισμός των 2 διεργασιών εγγραφής ανάγνωσης

Screenshots

```
// Wait for first process to finish writing to file and then proceed to the second process.
wait( stat_loc: NULL);
```

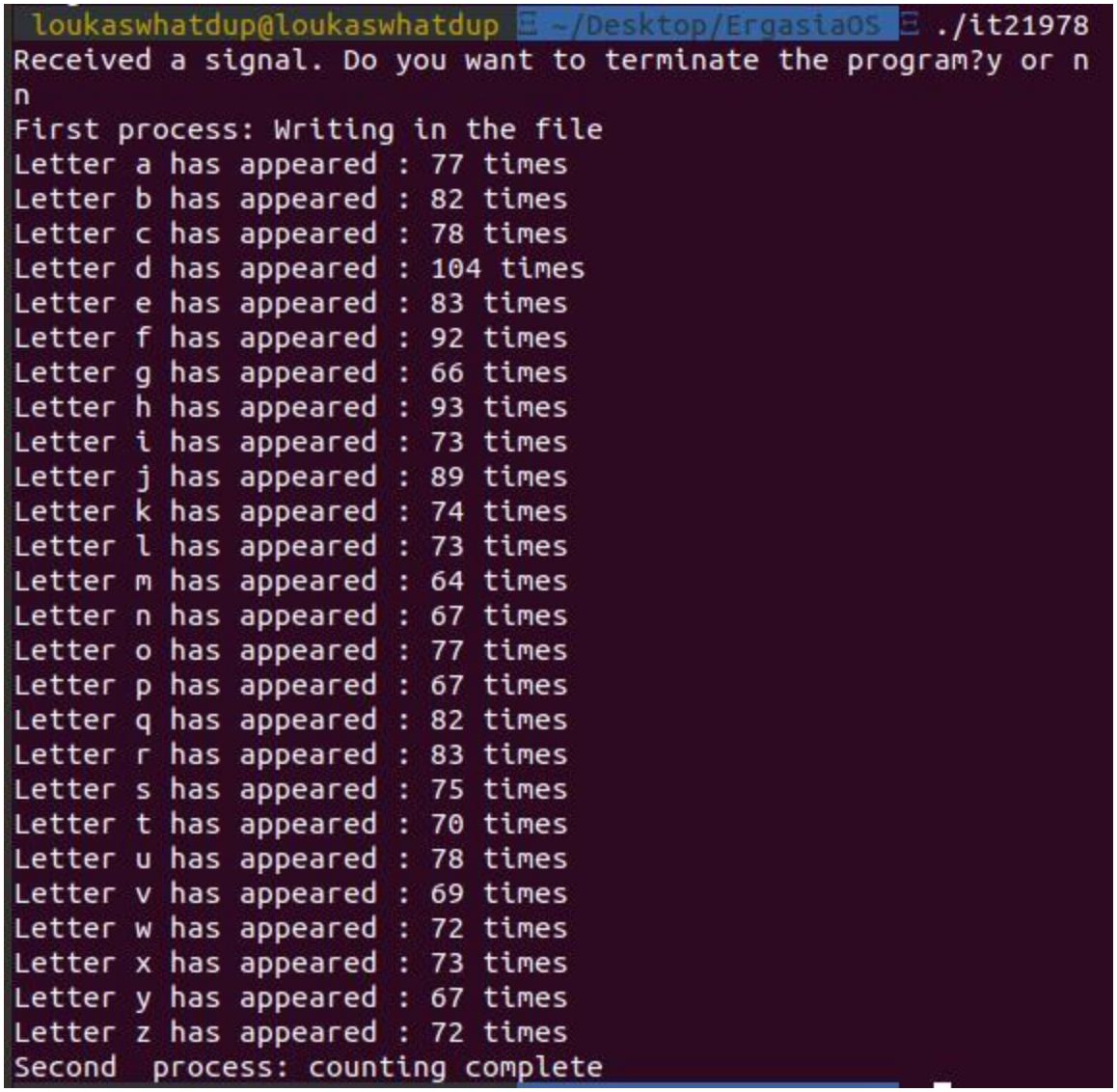
Πρέπει πρώτα να τελειώσει η πρώτη διεργασία για να αρχίσει η δεύτερη

Διαχείριση σημάτων

Screenshots

Στην συγκεκριμένη περίπτωση έχω κάνει raise ένα σήμα SIGTERM. Ανάλογα με την απάντηση εκτελείται και το πρόγραμμα.

```
loukaswhatdup@loukaswhatdup ~/Desktop/ErgasiaOS ./it21978
Received a signal. Do you want to terminate the program?y or n
y
Program has been terminated
```



Δημιουργία νημάτων και πέρασμα παραμέτρων

Screenshots

```
//Using a for loop for every thread.
for (int i = 0; i < 4; i++) {

    //Setting the offset so that every thread will read 500 characters.
    long offset = i * 500;

    //Here I create the new threads. First I store the threads ID in the address of threads_ID, then setting NULL the attributes of the threads.
    //Using a pointer to the function countLetterInThread that the threads will execute. In the end I cast the offset as void pointer so I can
    //use it in the function.
    threadCheck = pthread_create( newthread: &threads_ID[i],  attr: NULL,  start_routine: countLettersInThread,  arg: (void *) offset);

    //Checking if the threads was created correctly, if not print an error message.
    if(threadCheck !=0 ) {
        printf( format: "Error Message : Thread did not created successfully!");
        return 1;
    }

    //This function block the execution of the calling thread until the joined thread finish the execution.
    pthread_join( th: threads_ID[i],  thread_return: NULL);
}
```

Η δημιουργια των νημάτων με την pthread_create και οι παράμετροι που της περνάμε.

Συγχρονισμός νημάτων και σωστή διαδικασία μέτρησης αποτελέσματος

Screenshots

```
//With this function I acquire the lock and then can safely access the shared array.
pthread_mutex_lock( mutex: &lock);

//Using the for loop I move the localCountArray values in the shared array.
for (i = 0; i < 26; i++) {
    countLetterSharedArray[i] += localCountArray[i];
}

//With that functions the lock is released to be used again from another thread.
//Then exit the thread that has been called.
pthread_mutex_unlock( mutex: &lock);
pthread_exit( retval: NULL);
```

Το lock του mutex για την μη ταυτόχρονη χρήση του shared array.

```
pthread_mutex_init( mutex: &lock,  mutexattr: NULL);
```

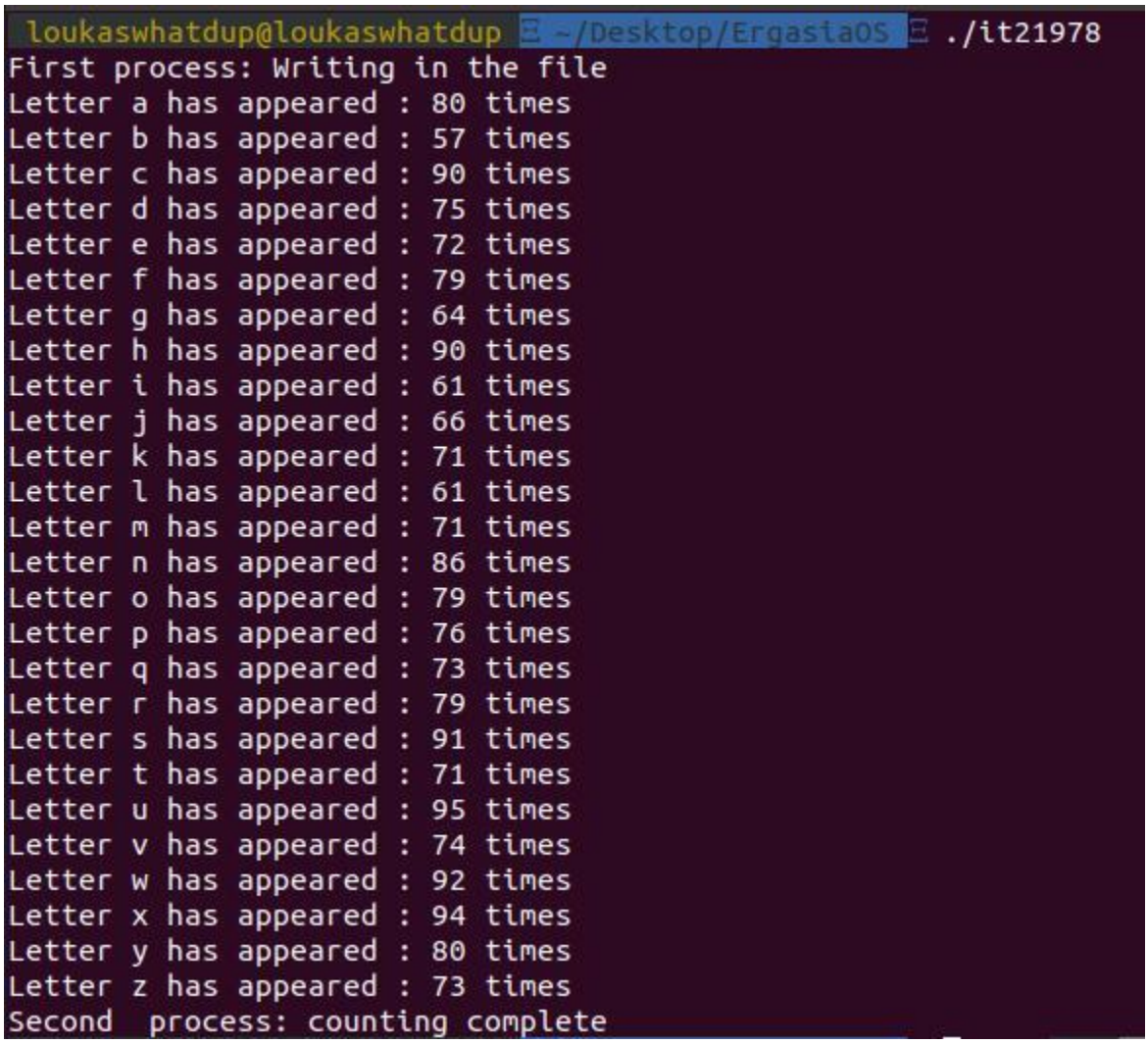
Η αρχικοποίηση του mutex.

```
//I check if the character that the function read from the file is among 'a'-'z'.
//I decrease the c variable value by the ascii value of the letter 'a' because
//I want to simulate the positions of the table as the letters of the alphabet.
if (c >= 'a' && c <= 'z') {
    localCountArray[c - 'a']++;
}
```

Η μέτρηση των γραμμάτων.

```
//This function block the execution of the calling thread until the joined thread finish the execution.
pthread_join( th: threads_ID[i], thread_return: NULL);
```

Συγχρονισμος των threads. Πρέπει να τελειώσει το ένα thread για να ξεκινήσει το επόμενο που μπαίνει στο pthread_join.



Τα αποτελέσματα.

Ομαλή εκτέλεση προγράμματος, error handling, τεκμηρίωση

Screenshots

```
if(threadCheck !=0 ) {
    printf( format: "Error Message : Thread did not created successfully!");
    return 1;
}
```

Έλεγχος στην δημιουργία των threads.

```c
//Checking if the file has opened, if not print error message.
if (file == NULL) {
    printf( format: "Error Message : Can not open file!\n");
    return 1;
}
```

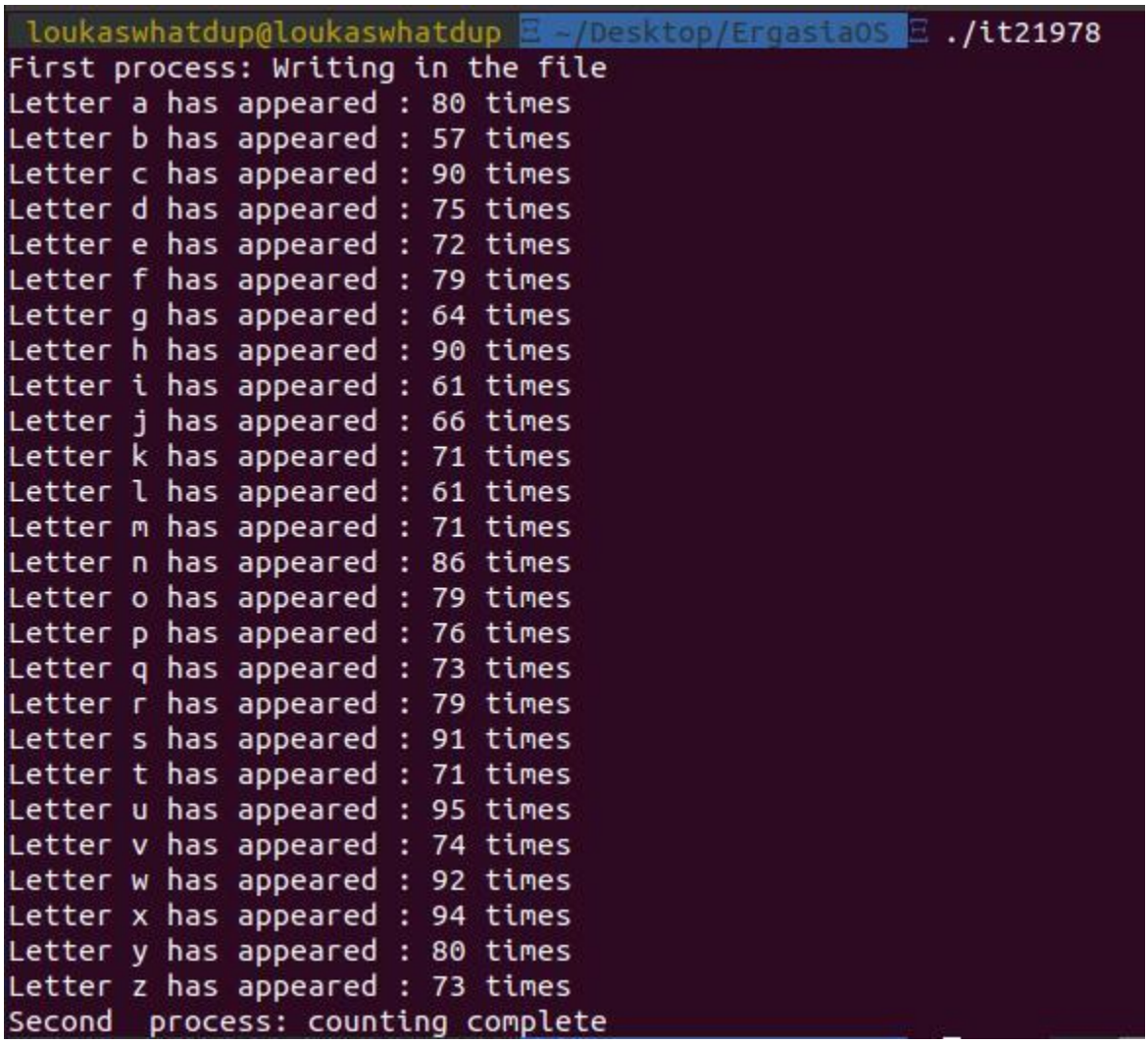
Έλεγχος για το σωστό άνοιγμα στο αρχείο.

```c
if (pid < 0) {
    fprintf( stream: stderr,  format: "The Fork function failed!");
    return 1;
```

Έλεγχος για το fork.

```c
//If the open system call returns -1 then I print an error message and terminate with exit code 1.
if (file_descriptor == -1) {
    printf( format: "Error Message : Can not open file!");
    exit( status: 1);
}
```

Έλεγχος για τον file descriptor.



Ομαλή εκτέλεση του προγράμματος.

Γενικά Σχόλια/Παρατηρήσεις

Με δυσκόλεψε / δεν υλοποίησα

Δεν κατάλαβα πλήρως τι ακριβώς χρειαζόταν να δείξουμε στα screenshots οπότε έβαλα το κομμάτι κώδικα για το κάθε τι που ζητείται.

Δεν υλοποίησα τον συγχρονισμο των διεργασιών χρησιμοποιώντας σημαφόρους.

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	**ΜΕΡΙΚΩΣ**	Δεν έκανα χρήση σημαφόρων για τον συγχρονισμό της σειράς εκτέλεσης.
Διαχείριση σημάτων	**ΝΑΙ**	
Δημιουργία νημάτων και πέρασμα παραμέτρων	**ΝΑΙ**	
Συγχρονισμός νημάτων και σωστή διαδικασία μέτρησης αποτελέσματος	**ΝΑΙ**	
Ομαλή εκτέλεση προγράμματος, error handling, τεκμηρίωση	**ΝΑΙ**	