
# ΔΕΥΤΕΡΗ ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

2023-2024

ΣΥΝΤΑΚΤΕΣ

ΘΕΟΔΩΡΟΣ ΚΑΤΣΑΝΤΑΣ
AM1097459

up1097459@ac.upatras.gr

ΑΓΓΕΛΙΚΗ ΔΟΥΒΡΗ AM1097441
up1097441@ac.upatras.gr

ΑΓΑΠΗ ΑΥΓΟΥΣΤΙΝΟΥ AM1093327

up1093327@ac.upatras.gr

# Γενικά Σχόλια

Η άσκηση ζητά την υλοποίηση ενός scheduler διεργασιών σε ένα λειτουργικό σύστημα Unix (Ubuntu στην περίπτωση μας). Ο δρομολογητής μας έπρεπε να λαμβάνει ως είσοδο μια λίστα εφαρμογών προς εκτέλεση,να τις διαβάζει από ένα αρχείο,να τις αποθηκεύει σε μια κατάλληλη δομή δεδομένων και στη συνέχεια να δρομολογεί την εκτέλεσή τους βάσει μιας συγκεκριμένης πολιτικής(FCFS, RR).

Στην FCFS οι εφαρμογές εκτελούνται με τη σειρά άφιξής τους, δηλαδή με τη σειρά που αναφέρονται στο αρχείο εισόδου.

Στην Round Robin (RR) οι εφαρμογές εκτελούνται κυκλικά για ένα προκαθορισμένο διάστημα.

Στην διαδικασία υλοποίησης του δρομολογητή εμφανίστηκαν συχνά προβλήματα όσον αφορά τον χρόνο εκτέλεσης των προγραμμάτων. Πολλές φορές η FCFS δεν τερμάτιζε ποτέ, το ίδιο και η RR. Για να αντιμετωπιστεί αυτό το πρόβλημα τροποποιήσαμε λίγο τον κώδικα μας και χρησιμοποιήσαμε το ‘alarm’ το οποίο όπως φάνηκε εν τέλει έλυσε το πρόβλημα.

Ένα ακόμη πρόβλημα που αντιμετωπίσαμε ήταν κατά την εκτέλεση του homogeneous.txt το οποίο δεν εμφάνιζε τα επιθυμητά αποτελέσματα, λύση σε αυτό ήταν μια τροποποίηση στη main.

Τέλος, εφαρμόσαμε την εμφάνιση ενός μηνύματος για τον χρόνο εκτέλεσης σε ορισμένες περιπτώσεις για να μελετήσουμε την διαφορά χρόνου ανάμεσα στις διεργασίες.

Όσον αφορά τη 2η φάση, ήταν ελαφρώς πιο δύσκολη η υλοποίηση των signals και της perform_io ώστε να έχουμε τα επιθυμητά αποτελέσματα. Πολλές φορές κατά την εκτέλεση της scheduler_io το τερματικό έκλεινε και έπρεπε να ακολουθήσουμε όλη την διαδικασία από την αρχή. Ακόμη, ο RR δεν εκτελούνταν σωστά και έμπαινε σε infinite loop. Με τη χρήση της sleep και του alarm διορθώθηκε.

Συνοψίζοντας, όλα τα ζητούμενα έχουν υλοποιηθεί και λειτουργούν πλήρως, καθώς επίσης και η έξτρα μονάδα της 2ης φάσης, μιας και υλοποιήθηκε το I/O τόσο για τον FCFS αλλά και τον RR.

Ακολουθεί ανάλυση κάθε ζητούμενου με screenshots κώδικα όπου απαιτείται και επεξήγηση του αντίστοιχου κώδικα.

# Ζητούμενα

## 1ο ΖΗΤΟΥΜΕΝΟ

Σε αυτό το σημείο καλούμαστε να υλοποιήσουμε τις βασικές δομές και λειτουργίες του δρομολογητή.

Αρχικά κατασκευάζουμε τις δομές Process, Node, Queue όπως φαίνεται παρακάτω:

```
// Definition of the process structure
struct Process {
    char name[20];
    int pid;
    enum ProcessState state;
    time_t entry_time;
    time_t start_time;  // Track process start time
    time_t end_time;    // Track process end time
};
```

name: Το όνομα της διεργασίας.

pid: Ο αριθμητικός αναγνωριστικός της διεργασίας.

state: Η κατάσταση της διεργασίας (NEW, RUNNING, STOPPED, EXITED).

entry_time: Ο χρόνος που η διεργασία μπαίνει στο σύστημα.

start_time: Ο χρόνος έναρξης εκτέλεσης της διεργασίας.

end_time: Ο χρόνος λήξης εκτέλεσης της διεργασίας.

```
struct Node {
    struct Process data;
    struct Node* next;
    struct Node* prev; // Add a previous pointer
};
```

data: Δεδομένα του κόμβου (διεργασία).

next: Δείκτης στον επόμενο κόμβο.

prev: Δείκτης στον προηγούμενο κόμβο (χρησιμοποιείται στην ουρά).

```c
// Definition of the queue structure
struct Queue {
    struct Node* front;
    struct Node* rear;
};
```

front: Δείκτης στον πρώτο κόμβο της ουράς.

rear: Δείκτης στον τελευταίο κόμβο της ουράς.

Έπειτα ορίζουμε τις συναρτήσεις που θα χρησιμοποιήσουμε:

```c
// Addition of an element to the queue
void enqueue(struct Queue* queue, struct Process process) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        perror("Error creating new node");
        exit(EXIT_FAILURE);
    }
    newNode->data = process;
    newNode->next = NULL;

    if (queue->rear == NULL) {
        newNode->prev = NULL;
        queue->front = queue->rear = newNode;
    } else {
        newNode->prev = queue->rear;
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}

// Removal of an element from the queue
struct Process dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        fprintf(stderr, "Queue is empty\n");
        exit(EXIT_FAILURE);
    }

    struct Process process = queue->front->data;
    struct Node* temp = queue->front;

    queue->front = queue->front->next;

    if (queue->front == NULL) {
        queue->rear = NULL;
    } else {
        queue->front->prev = NULL;
    }

    free(temp);
    return process;
```

Με τη χρήση των enqueue, dequeue προστίθενται και αφαιρούνται οι διεργασίες στην ουρά.

Τέλος, ορίζουμε και 2 συναρτήσεις για την αντιμετώπιση των σημάτων ‘SIGSTOP’, ‘SIGCONT’, ‘SIGCHLD’

```
void handle_stop_continue(int signum) {
    if (signum == SIGSTOP) {
        printf("Received SIGSTOP signal.\n");
        process_stopped = 1;
    } else if (signum == SIGCONT) {
        printf("Received SIGCONT signal.\n");
        process_stopped = 0;
    }
}

void handle_child_termination(int signum) {
    if (signum == SIGCHLD) {
        int status;
        pid_t child_pid;
        while ((child_pid = waitpid(-1, &status, WNOHANG)) > 0) {
            printf("Child process with PID %d has terminated.\n", child_pid);
            // Update process information as needed, e.g., set EXITED state
        }
    }
}
```

## 2ο ΖΗΤΟΥΜΕΝΟ

Στον κώδικά μας, η υλοποίηση της πολιτικής FCFS βρίσκεται στη συνάρτηση run_fcfs. Ακολουθεί η επεξήγηση του τρόπου λειτουργίας της:

1) Αφαιρείται η πρώτη διεργασία από την ουρά, χρησιμοποιώντας τη συνάρτηση dequeue.
2) Η διεργασία που αφαιρέθηκε τίθεται σε κατάσταση RUNNING, και ο χρόνος έναρξης καταγράφεται.
3) Στη συνέχεια, δημιουργείται ένα νέο child process με το fork και το child εκτελεί το πρόγραμμα της διεργασίας χρησιμοποιώντας το execlp.
4) Το parent process περιμένει την ολοκλήρωση της εκτέλεσης του child process χρησιμοποιώντας την waitpid.
5) Όταν η διεργασία ολοκληρωθεί, η κατάσταση της διεργασίας τίθεται σε EXITED, και ο χρόνος λήξης καταγράφεται.
6) Τέλος, εκτυπώνονται πληροφορίες για την ολοκλήρωση της διεργασίας, π.χ., το PID, το όνομα και ο συνολικός χρόνος εκτέλεσης.

Ακολουθεί παράδειγμα εκτέλεσης της FCFS:

```
theokatsa@LaptopKatsa:/mnt/c/Users/theok_iykl84u/OneDrive/Desktop/project2_scheduler/scheduler$ ./scheduler FCFS reverse.txt
Running Scheduler with Policy: FCFS
Executing process: ../work/work7 (PID: 1)
process 1610 begins
process 1610 ends
Process ../work/work7 (PID: 1) has exited.
Total execution time: 5 seconds
Executing process: ../work/work6 (PID: 2)
process 1611 begins
process 1611 ends
Process ../work/work6 (PID: 2) has exited.
Total execution time: 5 seconds
Executing process: ../work/work5 (PID: 3)
process 1612 begins
process 1612 ends
Process ../work/work5 (PID: 3) has exited.
Total execution time: 4 seconds
Executing process: ../work/work4 (PID: 4)
process 1613 begins
process 1613 ends
Process ../work/work4 (PID: 4) has exited.
Total execution time: 3 seconds
Executing process: ../work/work3 (PID: 5)
process 1614 begins
process 1614 ends
Process ../work/work3 (PID: 5) has exited.
Total execution time: 3 seconds
Executing process: ../work/work2 (PID: 6)
process 1615 begins
process 1615 ends
Process ../work/work2 (PID: 6) has exited.
Total execution time: 1 seconds
Executing process: ../work/work1 (PID: 7)
process 1616 begins
process 1616 ends
Process ../work/work1 (PID: 7) has exited.
```

## **3ο ΖΗΤΟΥΜΕΝΟ**

Στον κώδικά μας, η υλοποίηση της πολιτικής RR βρίσκεται στη συνάρτηση run_rr. Ακολουθεί η επεξήγηση του τρόπου λειτουργίας της:

1) Όπως και στην FCFS, αφαιρείται η πρώτη διεργασία από την ουρά.
2) Η διεργασία που αφαιρέθηκε τίθεται σε κατάσταση RUNNING και καταγράφεται ο χρόνος έναρξης.
3) Δημιουργείται ένα νέο child process με το fork και το παιδί εκτελεί το πρόγραμμα της διεργασίας χρησιμοποιώντας το execlp.
4) Η parent process κοιμάται για ένα χρονικό διάστημα (μέσω της sleep).
5) Στέλνεται το σήμα SIGSTOP στο child process προσομοιώνοντας το πέρασμα του time lapse.
6) Αν η child process εξακολουθεί να εκτελείται, τοποθετείται πίσω στην ουρά ως STOPPED.

Ακολουθεί παράδειγμα εκτέλεσης της RR:

```
theokatsa@LaptopKatsa:/mnt/c/Users/theok_iykl84u/OneDrive/Desktop/project2_scheduler/scheduler$ ./scheduler RR 1000 reverse.txt
Running Scheduler with Policy: RR
Executing process: ../work/work7 (PID: 1)
process 808 begins
Process ../work/work7 (PID: 1) paused. Elapsed time: 1 seconds
Executing process: ../work/work6 (PID: 2)
process 809 begins
process 809 ends
Process ../work/work6 (PID: 2) has exited.
Total execution time: 5 seconds
Executing process: ../work/work5 (PID: 3)
process 810 begins
Process ../work/work5 (PID: 3) paused. Elapsed time: 1 seconds
Executing process: ../work/work4 (PID: 4)
process 811 begins
process 811 ends
Process ../work/work4 (PID: 4) has exited.
Total execution time: 3 seconds
Executing process: ../work/work3 (PID: 5)
process 812 begins
process 808 ends
Child process with PID 808 has terminated.
Process ../work/work3 (PID: 5) paused. Elapsed time: 0 seconds
Executing process: ../work/work2 (PID: 6)
process 813 begins
process 813 ends
Process ../work/work2 (PID: 6) has exited.
Total execution time: 2 seconds
Executing process: ../work/work1 (PID: 7)
process 814 begins
process 814 ends
Child process with PID 814 has terminated.
Process ../work/work1 (PID: 7) paused. Elapsed time: 1 seconds
```

## 4ο ΖΗΤΟΥΜΕΝΟ

Αρχικά εκτελέσαμε το ./scheduler FCFS homogeneous.txt και το output ήταν το εξής:

```
theokatsa@LaptopKatsa:/mnt/c/Users/theok_iykl84u/OneDrive/Desktop/project2_s
cheduler/scheduler$ ./scheduler FCFS homogeneous.txt
Running Scheduler with Policy: FCFS
Executing process: ../work/work7 (PID: 1)
process 1630 begins
process 1630 ends
Process ../work/work7 (PID: 1) has exited.
Total execution time: 6 seconds
Executing process: ../work/work7 (PID: 2)
process 1631 begins
process 1631 ends
Process ../work/work7 (PID: 2) has exited.
Total execution time: 6 seconds
Executing process: ../work/work7 (PID: 3)
process 1632 begins
process 1632 ends
Process ../work/work7 (PID: 3) has exited.
Total execution time: 6 seconds
Executing process: ../work/work7 (PID: 4)
process 1633 begins
process 1633 ends
Process ../work/work7 (PID: 4) has exited.
Total execution time: 5 seconds
Executing process: ../work/work7 (PID: 5)
process 1634 begins
process 1634 ends
Process ../work/work7 (PID: 5) has exited.
Total execution time: 5 seconds
```

Είναι εμφανής ο τρόπος λειτουργίας της FCFS. Εκτελεί τις διεργασίες με τη σειρά που φτάνουν στην ουρά, αφού μια διεργασία ολοκληρωθεί, παρουσιάζονται οι σχετικές πληροφορίες. Είναι λογικό ο χρονός εκτέλεσης να είναι σχεδόν ίδιος καθώς στο συγκεκριμένο αρχείο ζητείται να εκτελεστεί η ίδια διεργασία πολλές φορές.

Όταν εκτελεστεί το ίδιο αρχείο αλλά με το RR τα αποτελέσματα είναι τα εξής:

```
theokatsa@LaptopKatsa:/mnt/c/Users/theok_iykl84u/OneDrive/Desktop/project2_scheduler/scheduler$ ./scheduler RR 1000 homogeneous.txt
Running Scheduler with Policy: RR
Executing process: ../work/work7 (PID: 1)
process 824 begins
Process ../work/work7 (PID: 1) paused. Elapsed time: 1 seconds
Executing process: ../work/work7 (PID: 2)
process 825 begins
process 825 ends
Process ../work/work7 (PID: 2) has exited.
Total execution time: 6 seconds
Executing process: ../work/work7 (PID: 3)
process 826 begins
Process ../work/work7 (PID: 3) paused. Elapsed time: 1 seconds
Executing process: ../work/work7 (PID: 4)
process 827 begins
process 824 ends
Child process with PID 824 has terminated.
process 827 ends
Process ../work/work7 (PID: 4) has exited.
Total execution time: 5 seconds
Executing process: ../work/work7 (PID: 5)
process 828 begins
Process ../work/work7 (PID: 5) paused. Elapsed time: 1 seconds
```

Η αισθητή διαφορά είναι ότι η RR εκτελεί κάθε διεργασία για ένα συγκεκριμένο χρονικό διάστημα και έπειτα παύει δίνοντας το ελεύθερο στην επόμενη διεργασία να ξεκινήσει.

Ας προχωρήσουμε τώρα στην επόμενη εκτέλεση που αφορά το reverse.txt

Με τη χρήση του FCFS είχαμε το παρακάτω:

```
theokatsa@LaptopKatsa:/mnt/c/Users/theok_iykl84u/OneDrive/Desktop/project2_scheduler/scheduler$ ./scheduler FCFS reverse.txt
Running Scheduler with Policy: FCFS
Executing process: ../work/work7 (PID: 1)
process 1652 begins
process 1652 ends
Process ../work/work7 (PID: 1) has exited.
Total execution time: 5 seconds
Executing process: ../work/work6 (PID: 2)
process 1653 begins
process 1653 ends
Process ../work/work6 (PID: 2) has exited.
Total execution time: 5 seconds
Executing process: ../work/work5 (PID: 3)
process 1654 begins
process 1654 ends
Process ../work/work5 (PID: 3) has exited.
Total execution time: 4 seconds
Executing process: ../work/work4 (PID: 4)
process 1655 begins
process 1655 ends
Process ../work/work4 (PID: 4) has exited.
Total execution time: 3 seconds
Executing process: ../work/work3 (PID: 5)
process 1656 begins
process 1656 ends
Process ../work/work3 (PID: 5) has exited.
Total execution time: 3 seconds
Executing process: ../work/work2 (PID: 6)
process 1657 begins
process 1657 ends
Process ../work/work2 (PID: 6) has exited.
Total execution time: 1 seconds
Executing process: ../work/work1 (PID: 7)
process 1658 begins
process 1658 ends
Process ../work/work1 (PID: 7) has exited.
```

Σε αυτή την περίπτωση παρατηρούμε ότι ο χρόνος εκτέλεσης των διεργασιών αντιστρέφεται σε σχέση με τη σειρά εισόδου τους. Αυτό είναι απολύτως λογικό μιας και ο FCFS εκτελεί τις διεργασίες ανάποδα από αυτό που βλέπουμε εδώ, δηλ. εκτελεί πρώτα την work1 μετάτ την work2 κ.ο.κ.

Με τη χρήση του RR έχουμε το εξής:

```
theokatsa@LaptopKatsa:/mnt/c/Users/theok_iykl84u/OneDrive/Desktop/project2_scheduler/scheduler$ ./scheduler RR 1000 reverse.txt
Running Scheduler with Policy: RR
Executing process: ../work/work7 (PID: 1)
process 830 begins
Process ../work/work7 (PID: 1) paused. Elapsed time: 1 seconds
Executing process: ../work/work6 (PID: 2)
process 831 begins
process 831 ends
Process ../work/work6 (PID: 2) has exited.
Total execution time: 5 seconds
Executing process: ../work/work5 (PID: 3)
process 832 begins
Process ../work/work5 (PID: 3) paused. Elapsed time: 1 seconds
Executing process: ../work/work4 (PID: 4)
process 833 begins
process 833 ends
Process ../work/work4 (PID: 4) has exited.
Total execution time: 3 seconds
Executing process: ../work/work3 (PID: 5)
process 834 begins
process 830 ends
Child process with PID 830 has terminated.
Process ../work/work3 (PID: 5) paused. Elapsed time: 0 seconds
Executing process: ../work/work2 (PID: 6)
process 835 begins
process 835 ends
Process ../work/work2 (PID: 6) has exited.
Total execution time: 2 seconds
Executing process: ../work/work1 (PID: 7)
process 836 begins
process 836 ends
Child process with PID 836 has terminated.
Process ../work/work1 (PID: 7) paused. Elapsed time: 1 seconds
```

Τα ζητούμενα στο run.sh που απαιτούν την scheduler.io θα τα δούμε παρακάτω

## 5ο ΖΗΤΟΥΜΕΝΟ

Εδώ καλούμαστε να φτιάξουμε έναν δρομολογητή με βάση τον προηγούμενο, ο οποίος πάλι θα χειρίζεται τους αλγορίθμους FCFS και RR, ωστόσο αυτή τη φορά θα μπορεί να διαχειρίζεται διεργασίες που έχουν είσοδο-έξοδο.

Ακολουθεί η επεξήγηση του κώδικα:

1) Ο χρήστης περνάει στο πρόγραμμα τον τύπο του χρονοδρομολογητή (FCFS ή RR) και το όνομα του αρχείου εισόδου που περιέχει τα ονόματα των διεργασιών. Οι διεργασίες αναπαρίστανται από τη δομή struct Process και αποθηκεύονται σε μια ουρά (struct Queue).
2) Η συνάρτηση run_fcfs εκτελεί τις διεργασίες με βάση την πολιτική FCFS.
3) Για κάθε διεργασία, δημιουργεί ένα παιδί, το οποίο εκτελεί συνάρτηση perform_io για προσομοίωση εισόδου/εξόδου για 5 δευτερόλεπτα.
4) Ο γονέας περιμένει το παιδί να ολοκληρώσει και ενημερώνει την κατάσταση της διεργασίας.
5) Η συνάρτηση run_rr εκτελεί τις διεργασίες με βάση την πολιτική RR.
6) Για κάθε διεργασία, δημιουργεί ένα παιδί, το οποίο εκτελεί συνάρτηση perform_io για προσομοίωση εισόδου/εξόδου για 5 δευτερόλεπτα.
7) Καθορίζεται ένας συναγερμός (alarm) για το καθορισμένο χρονικό διάστημα (time_quantum).
8) Ο γονέας περιμένει για την ολοκλήρωση της διεργασίας ή την ενεργοποίηση του συναγερμού και ενημερώνει την κατάσταση της διεργασίας.

9) Εάν η διεργασία δεν έχει ολοκληρωθεί, ξεκινάει η εκτέλεση της επόμενης διεργασίας.
10) Η συνάρτηση perform_io χρησιμοποιείται για την προσομοίωση εισόδου/εξόδου, χειρίζεται τα σήματα SIGUSR1 και SIGUSR2, και εξάγει το παιδί με επιτυχία.
11) Οι συναρτήσεις χειρισμού σημάτων (signal handlers) χρησιμοποιούνται για την αγνόηση των σημάτων SIGUSR1 και SIGUSR2 κατά τη διάρκεια της εισόδου/εξόδου.

Γενικά, ο κώδικας εκτελεί διαδοχικά τις διεργασίες, εισάγοντάς τες στην ουρά και εκτελώντας τον χρονοδρομολογητή με βάση τον επιλεγμένο τύπο (FCFS ή RR). Οι διεργασίες προσομοιώνονται να εκτελούν εισόδους/εξόδους και ενημερώνεται ο χρήστης για την πρόοδο της εκτέλεσης.

## 6ο ΖΗΤΟΥΜΕΝΟ

Σε αυτό το σημείο θα δούμε πως εκτελείται το πρόγραμμα μας άμα τρέξουμε το run.sh (σ.σ. τις αντίστοιχες περιπτώσεις για το scheduler_io).

Παράδειγμα ./scheduler_io FCFS mixed.txt

```
theokatsa@LaptopKatsa:/mnt/c/Users/theok_iykl84u/OneDrive/Desktop/project2_scheduler/scheduler$ ./scheduler_io FCFS mixe
d.txt
Running Scheduler with Policy: FCFS
Executing process: ../work/work5x2_io (PID: 1)
Process ../work/work5x2_io (PID: 1) has exited.
Executing process: ../work/work7 (PID: 2)
Process ../work/work7 (PID: 2) has exited.
Executing process: ../work/work6 (PID: 3)
Process ../work/work6 (PID: 3) has exited.
```

Παράδειγμα ./scheduler_io RR 1000 mixed.txt

```
theokatsa@LaptopKatsa:/mnt/c/Users/theok_iykl84u/OneDrive/Desktop/project2_scheduler/scheduler$ ./scheduler_io RR 1000 m
ixed.txt
Running Scheduler with Policy: RR
Executing process: ../work/work5x2_io (PID: 1)
Process ../work/work5x2_io (PID: 1) has exited.
Total execution time: 5000 milliseconds
Executing process: ../work/work7 (PID: 2)
Process ../work/work7 (PID: 2) has exited.
Total execution time: 5000 milliseconds
Executing process: ../work/work6 (PID: 3)
Process ../work/work6 (PID: 3) has exited.
Total execution time: 5000 milliseconds
```

**Σχόλια**

Όλα τα ζητούμενα έχουν υλοποιηθεί, η εκτέλεση τους γίνεται με το ./scheduler(_io) FCFS(RR 1000) ονομα.txt

Έχει γίνει απόπειρα υλοποίησης και του μπόνους ερωτήματος.

Θα παρακαλούσαμε να σημειωθεί πως η παράδοση της εργασίας ήταν προγραμματισμένη να γίνει προ 2 ημερών μαζί με το 1ο Πρότζεκτ, αλλά κατά την μεταφορά των αρχείων έγινε λάθος με τα scheduler.c και scheduler_io.c με αποτέλεσμα ο αρχικός κώδικας να διαγραφεί και ο νέος-τελικός κώδικας να μην είναι στο ίδιο επίπεδο που είχε φτάσει ο αρχικός, ωστόσο είναι επαρκής για τα ζητούμενα. Κατά την αρχική υλοποίηση όλα λειτουργούσαν ομαλά και τα προβλήματα που αντιμετωπίστηκαν ήταν τα ίδια που συναντήσαμε και στην 2η υλοποίηση του κώδικα(με την πίεση του χρόνου αυτή τη φορά).