Ενσωματωμένα Συστήματα Παργματικού Χρόνου
Εργασία #2

Ονοματεπώνυμο: Τσιράκης Ορέστης
AEM: 9995
mail: otsirakv@ece.auth.gr
github link: https://github.com/Orestistsira/auth-rtes-ex2

Εισαγωγή

Η παρακάτω εργασία αφορά την υλοποίηση ενός Timer, με δύο συναρτήσεις που τρέχουν νήματα της μορφής producer και consumer. Μία συνάρτηση που θέλουμε να εκτελέσουμε μπάινει κάθε περίοδο στην ουρά από έναν producer, ενώ οι consumers βγάζουν τις συναρτήσεις από την ουρά και τις εκτελούν. Σκοπός της εργασίας είναι να ελέγξουμε πόσο ακριβείς είναι οι Timers, καθώς λειτουργεί το σύστημα cosnumer-producer.

Υλοποίηση

Timer: Η δομή Timer υλοποιεί τον Timer που περιγράφηκε παραπάνω, και αποθηκεύει παραμέτρους όπως period, tasksToExecute, startDelay, timerFnc() και errorFnc(). Η συνάρτηση timerInit() αρχικοποιεί έναν Timer, ενώ οι συναρτήσεις start() και startat() τον ξεκινούν. Συγκεκριμένα, η συνάρτηση startat() καθορίζει και την χρονκή στιγμή που θα ξεκινήσει να εκτελείται η συνάρτηση.

```
typedef struct {
        int period;
        int tasksToExecute;
        int startDelay;
        void *(*startFcn)(void *arg);
        void *(*stopFcn)(void *arg);
        void *(*timerFcn)(void *arg);
        void *(*errorFcn)();
        void *userData;
        Queue *queue;
        pthread_t tid;
        void *(*producer)(void *arg);
        int *tJobIn;
        int *tDrift;
        pthread_mutex_t *tMut;
} Timer;
```

Queue: Η δομή Queue υλοποιεί την FIFO ουρά και κρατάει αντικείμενα τύπου WorkFunction, όπως ακριβώς και στην πρώτη εργασία.

```
typedef struct {
        WorkFunction *buf;
        long size, head, tail;
        int full, empty;
        pthread_mutex_t *mut;
        pthread_cond_t *notFull, *notEmpty;
} Queue;
```

Producer: Η συνάρτηση producer() υλοποιεί όλη τη λογική του Timer που την κάλεσε, ξεκινώντας ένα WorkFunction κάθε περίοδο. Δημιουργεί τα ορίσματα της συνάρτησης και εφόσον η ουρά δεν είναι γεμάτη, την προσθέτει. Σε περίπτωση που η ουρά είναι γεμάτη, τρέχει η συνάρτηση errFcn και αυξάνεται ο jobsLostCounter. Τέλος, αποθηκεύει τον χρόνο tJobIn, δηλάδή τον χρόνο που έκανε ο producer να προσθέσει μια δουλεία στην ουρά, καθώς και τον χρόνο tDrift. Ο χρόνος tDrift αφορά τη χρονική μετατόπιση που παρατηρείται μεταξύ διαδοχικών εκτελέσεων του timer πέραν της περιόδου του, κάτι που συμβαίνει λόγω του χρόνου που παίρνει ο ίδιος ο producer για να εκτελεστεί, αλλά και λόγω λανθανουσών διακοπών που επιβάλλονται από το λειτουργικό.

Cosnumer: Η συνάρτηση consumer() υλοποιεί την λειτουργία των νημάτων καταναλωτών. Αρχικά εκτελείται η συνάρτηση που βγαίνει από την ουρά και στην συνέχεια υπολογίζονται οι χρόνοι tJobWait και tJobDur, οι οποίοι αναπαρηστούν τον χρόνο αναμονής μιας εργασίας στην ουρά και τη διάρκεια εκτέλεσης της δουλειάς από τον consumer αντίστοιχα.

Main: Η συνάρτηση main() είναι υπεύθυνη για την εκτέλεση του προγράμματος μας. Αρχικά για να μπορούμε εύκολα τα εκτελούμε διάφορα πειράματα, ζητείται από τον χρήστη το μέγεθος της ουράς, ο χρόνος σε δευτερόλεπτα που θα τρέξουν οι Timers και ποιοί Timers θα τρέξουν. Στη συνέχεια, δημιουργούνται τα consumer threads και εκκινούνται οι timers. Η main() περιμένει μέχρι να τελειώσουν οι Timers, τερματίζει με ομαλό και ασφαλή τρόπο τους consumers οι οποίοι περιμένουν σε μια άδεια ουρά, θέτοντας το flag quit, αποθηκεύει τα στατιστικά που επιλέξαμε σε αρχεία και απελευθερώνει τη μνήμη που δεσμεύτηκε για το πείραμα.

Πείραμα

Εκτελέσαμε τα παρακάτω πειράματα σε Raspberry Pi Zero, με περιόδους των Timer:

1. T = 1s
2. T = 0.1s
3. T = 0.01s
4. T = 1s, T = 0.1s, T = 0.01s

Το κάθε πείραμα διήρκησε 1 ώρα, ενώ στο τελευταίο πείραμα εκτελούνταν τα 3 πρώτα tasks ταυτόχρονα. Στους πίνακες που ακολουθούν, το πείραμα που ανήκει ο κάθε Timer αναγράφεται σε παρένθεση.

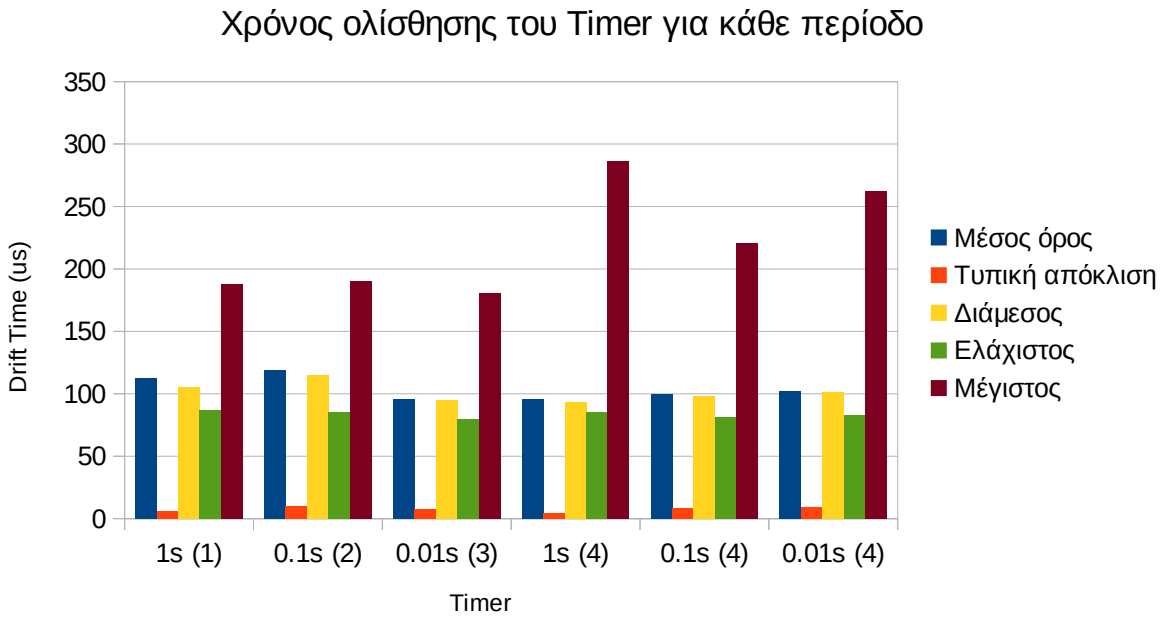
Αποτελέσματα

Χρόνος ολίσθησης του Timer για κάθε περίοδο (us):

Timer	Μέσος όρος	Τυπική απόκλιση	Διάμεσος	Ελάχιστος	Μέγιστος
1s (1)	112,59	6,19	105	87	188
0.1s (2)	118.72	9,89	115	85	190
0.01s (3)	95,8	7,37	95	80	181
1s (4)	95,41	4,63	93	85	286
0.1s (4)	99,69	8,15	98	81	221
0.01s (4)	102,43	9,56	101	83	262

Παρατηρούμε πως ο μέσος όρος, η τυπική απόκλιση και η διάμεσος του χρόνου ολίσθησης είναι μικρότερα και για τους 2 πρώτους Timers, όταν αυτοί έτρεχαν ταυτόχρονα. Αντίθετα, τα στατιστικά αυτά ήταν μεγαλύτερα για τον Timer με περιόδο T = 0.01s, όταν έτρεχε παράλληλα με τους άλλους

δύο. Ο μέγιστος χρόνος ολίσθησης φαίνεται να αυξάνεται όταν εκτελούνται και οι 3 Timers ταυτόχρονα, σε σύγκριση με την απομωνομένη εκτέλεση. Αυτό συμβαίνει, καθώς κατά την ταυτόχρονη εκτέλεση υπάρχει πιθανότητα 2 ή παραπάνω Timers να πρέπει να προσθέσουν στην ουρά ταυτόχρονα, οπότε μόνο ο ένας θα συνεχίσει την εκτέλεση και οι υπόλοιποι θα βρίσκονται σε αναμονή.



Χρόνος που έκανε ο producer να βάλει μια κλήση στην ουρά (us):

Timer	Μέσος όρος	Τυπική απόκλιση	Διάμεσος	Ελάχιστος	Μέγιστος
1s (1)	5,21	1,73	4	3	82
0.1s (2)	4,03	0,83	4	2	71
0.01s (3)	3,9	0,54	4	3	75
1s (4)	4,12	1,86	4	3	205

Χρόνος που έκανε ο consumer να βγάλει μια κλήση από την ουρά (us):

Timer	Μέσος όρος	Τυπική απόκλιση	Διάμεσος	Ελάχιστος	Μέγιστος
1s (1)	26,41	2,81	26	24	53
0.1s (2)	24,82	3,01	24	13	62
0.01s (3)	23,89	1,43	23	22	63
1s (4)	25,85	4,88	25	2	297

Ποσοστό χρήσης CPU:

Timer	Χρήση CPU
1s (1)	0.0%
0.1s (2)	0.1%
0.01s (3)	0.9%
1s (4)	1.0%

Στον παραπάνω πίνακα φαίνεται η χρήση CPU ανάλογα με τον αριθμό πειράματος. Τα αποτελέσματα είναι τα αναμενόμενα, καθώς ένας Timer με 10 φορές μικρότερη περίοδο από έναν άλλο, θα έπρεπε να χρησιμοποιεί τη CPU 10 φορές περισσότερο. Αυτό μπορεί να παρατηρηθεί αν συγκρίνουμε τη σχεδόν μηδενική χρήση του CPU από τον 0.1s Timer σε σχέση με τον 0.01s Timer, καθώς και το ότι η κύρια χρήση CPU όταν τρέχουν και οι τρεις Timers οφείλεται στον Timer με τη μικρότερη περίοδο.

Λειτουργία πραγματικού χρόνου:

Για λειτουργία πραγματικού χρόνου, το μέγεθος της ουράς εξαρτάται αρχικά από τον αριθμό των Timers που τρέχουν. Θεωρόντας ότι κάθε Timer προσθέτει μόνο μια δουλειά στην ουρά, το μέγεθος της ουράς πρέπει να είναι ίσο ή μεγαλύτερο από τον αριθμό των Timers. Ο χρόνος εκτέλεσης της timerFcn() πρέπει να είναι σίγουρα μικρότερος της περιόδου του Timer που την προσθέτει στην ουρά, αλλιώς θα είχαμε συνεχώς αυξανόεμενες συναρτήσεις προς εκτέλεση, χωρίς να προλαβαίνουν οι consumers να τις εκτελέσουν. Στη δική μας περίπτωση, ο μέσος χρόνος εκτέλεσης της συνάρτησης timerFcn() ήταν 52us, επομένως, ακόμη και ένας consumer ήταν αρκετός για την εκτέλεση των κλήσεων που προθέτονταν και από τους τρεις Timers.