ΑΝΑΦΟΡΑ ΑΠΑΛΛΑΚΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Σύντομη περιγραφή κώδικα και της λειτουργικότητας του

Σε αυτή την εργασία κληθήκαμε να υλοποιήσουμε τη λειτουργία ενός Timer, η λειτουργικότητα του οποίου βασίζεται στο παράδειγμα producer-consumer της πρώτη εργασίας.
Για την υλοποίηση του Timer, κατασκευάστηκε ένα struct που περιλαμβάνει όλες τις απαραίτητες μεταβλητές όπως αυτές δίνονται στην εκφώνηση της εργασίας(π.χ period, TasksToExecute κλπ) ενω για την αρχικοποίηση του timer object (των μεταβλητων που περιέχει δηλαδή) δημιουργήθηκε μια συνάρτηση timerInit(). Ο timer ξεκινά όταν κληθεί η συνάρτηση start(), η οποία δημιουργεί το νήμα (producer) που θα τοποθετεί ανα τακτά χρονικα διαστήματα - καθορισμένα απο το period - - ,δείκτες συναρτήσεων στην ουρά.
Σε περίπτωση που θέλουμε ο timer να ξεκινήσει σε μια συγκεκριμένη χρονική στιγμή στο μέλλον, τότε για την αρχικοποίηση του χρησιμοποιείται η startat(), η οποία δέχεται την ακριβή επιθυμητή ημερομηνία και ώρα εκκίνησης του timer, υπολογίζει τα us της ημερομηνίας και ώρας απο το ***unix epoch*** και αφαιρώντας τα αντίστοιχα us της τωρινής χρονικής στιγμής υπολογίζει πόσα us πρέπει το νήμα producer να κάνει sleep ώστε να ξεκινήσει τη σωστή χρονική στιγμή. Η υλοποίηση που περιγράφηκε παραπάνω έγινε με τη βοήθεια του struct tm ,μιας μεταβλητής τύπου time_t και της συνάρτησης mktime() που μετατρέπει μια ακριβή ημερομηνία και ώρα σε δευτερόλεπτα.

Μόλις το νήμα producer ξεκινήσει καλείται η startfcn() και έπειτα η usleep() για startDelay secs , όπως ορίζει η εκφώνηση. Έπειτα ελέγχεται έαν η δημιουργία αυτού του thread έγινε απο την startat(), οπότε και θα πρέπει η usleep() να ξανακληθεί για κατάλληλο χρονικό διάστημα όπως περιγράψαμε παραπάνω και αμέσως μετά εκτελείται ένας βρόχος for, για TasksToExecute φορές για την τοποθέτηση TasksToExecute αριθμό απο tasks στην ουρά.

Σε κάθε επανάληψη του βροχου, το thread του producer προσπαθεί να πάρει το mutex το οποίο διεκδικούν και τα consumer threads, εαν η ουρά δεν είναι άδεια, και τα νήματα producer των άλλων timers, εαν τρέχουν όλοι οι timers μαζί. Οι εργασίες που τοποθετούνται στην ουρά ουσιαστικά είναι objects του struct workfunction που δημιουργήθηκε στην προηγούμενη εργασία.

Σε περίπτωση που η ουρά είναι γεμάτη τότε εκτελείται η ErrorFcn() που μας ειδοποιεί για αυτό το γεγονός εκτυπώνοντας αντίστοιχο μήνυμα στην οθόνη ενω όταν προστεθούν όλα τα tasks στην ουρά εκτελείται η συνάρτηση StopFcn().

Αντίστοιχα με την προηγούμενη εργασία, τα νήματα consumers είναι υπεύθυνα να βγάζουν απο την ουρά και να εκτελούν τις συναρτήσεις. Έτσι, όταν η ουρά δεν είναι πια άδεια τα νήματα ξυπνούν και συμμετέχουν στο race για τη διεκδίκηση του mutex.

Αντιμετώπιση Drifting

Στην υλοποίηση που δημιουργήθηκε παρατηρείται μια μετατόπιση στο χρόνο που “χαλάει” την επιθυμητή περιοδικότητα που πρέπει να έχει ο timer για να είναι αξιόπιστος. Η χρονική αυτή μετατόπιση οφείλεται αφενός στην ανακρίβεια της usleep() αυτης καθαυτής, αφετέρου και στο γεγονός ότι ο χρόνος μεταξύ δύο προσθέσεων στην ουρά δεν είναι πάντα ο ίδιος, πράγμα που θα επέτρεπε τη ρύθμιση του χρόνου που θα χρησιμοποιηθεί στην usleep() μόνο μια φορά. Αντίθετα μεταξύ εκτελέσεων της queueadd() (ίδια συναρτηση,όπως δόθηκε στην πρώτη εργασία) ο χρόνος μπορεί να διαφέρει σημαντίκα για διάφορους λόγους με κυριότερο τον χρόνο που κάνει να πάρει το mutex το νήμα του producer για την επόμενη πρόσθεση στην ουρά.

Για την αντιμετώπιση αυτης της χρονικής μετατόπισης ορίστηκε μια μεταβλητή timeToSleep που θα μας βοηθήσει να αλλάζουμε δυναμικά τον χρόνο που θα κοιμάται το νήμα producer τοποθετώντας τη σαν όρισμα στην usleep(). Για τον υπολογισμό της τιμής της timeToSleep , αρχικά με κλήση της gettimeofday() πρίν από κάθε εκτέλεση της queueadd() μετράμε πόσος χρόνος πέρασε μεταξύ δύο προσθέσεων στην ουρά. Ακριβώς, επειδή η timeToSleep καθορίζεται απο αυτον τον χρόνο, και κατα την πρόσθεση του πρώτου task ο χρόνος αυτος δεν έχει υπολογιστεί ακόμα, αρχικά η usleep() καλείται με όρισμα την περίοδο που δίνεται.

Μετά τη δεύτερη επανάληψη του βρόχου, αφαιρώντας απο τον χρόνο μεταξυ δυο προσθέσεων την περίοδο βρίσκουμε πόσο περισσότερο ή λιγοτερο ξεφύγαμε απο την περίοδο. Αυτη τη διαφόρά ονομάσαμε drift και παρακάτω αλλάζουμε την timeToSleep αφαιρώντας απο την υπάρχουσα τιμή της, την τιμή του drift(σημειώνουμε ότι εάν η τιμή του drift είναι αρνητική, τότε ουσιαστικά το drift θα προστεθεί στην timeToSleep), προσπαθώντας με αυτόν τον τρόπο να διορθώσουμε το λάθος.

Στην περίπτωση που η τιμή του drift είναι μεγαλύτερη απο την timeToSleep, τότε η timeToSleep θα πάρει αρνητική τιμή, πράγμα αδύνατο, οπότε και θέτουμε την τιμή 0 για την timeToSleep, ώστε την επόμενη φορά η συνάρτηση να τοποθετηθεί αμέσως στην ουρά.

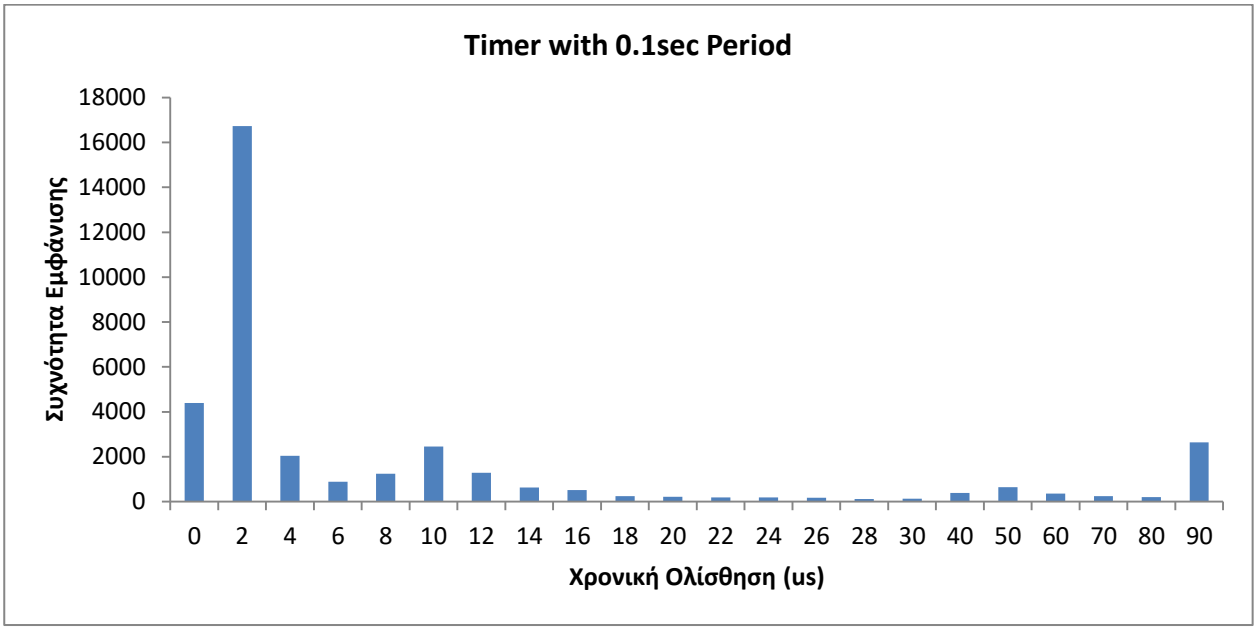
Πειράματα-Μετρήσεις

Τα συνολικά πειράματα που έγιναν ήταν 4, όπου σε ένα έτρεξαν ταυτόχρονα και οι τρεις timers με περιόδους 1sec, 0.1sec και 0.01 sec ενω στα υπόλοιπα 3 έτρεξε κάθε timer μόνος του.
Στα πειράματα αυτά μετρήθηκαν τα παρακάτω είτε ο κάθε timer έτρεχε ξεχωριστά είτε ταυτόχρονα με τους υπόλοιπους:

- Ηαπόλυτη τιμή του drift για κάθε timer- χρονική ολίσθηση απο την περίοδο
- Ο χρόνος που ξοδεύεται απο κάθε timer για να βάλει μια κλήση στην ουρά.Χρονική διαφορά,δηλαδη, μεταξύ έναρξης προσπάθειας ανάληψης του mutex απο το producer thread μέχρι και την επιστροφή της queueadd()
- Ο χρόνος παραμονής κάθε δείκτη σε συνάρτηση στην ουρά, απο τη στιγμή που εισήχθει στην ουρά μέχρι και να αφαιρεθεί απο αυτή αλλα προτού εκτελεστεί.

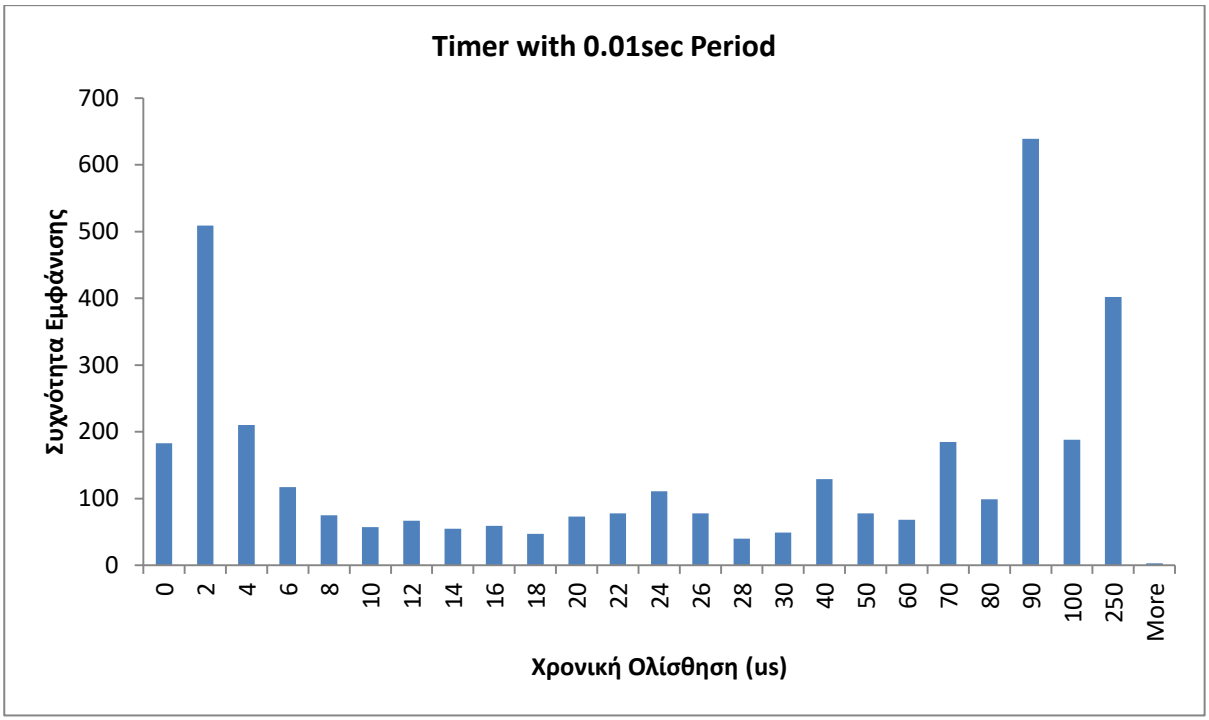
Αποτελέσματα απομονωμένης εκτέλεσης κάθε timer

Χρονική ολίσθηση- Drift



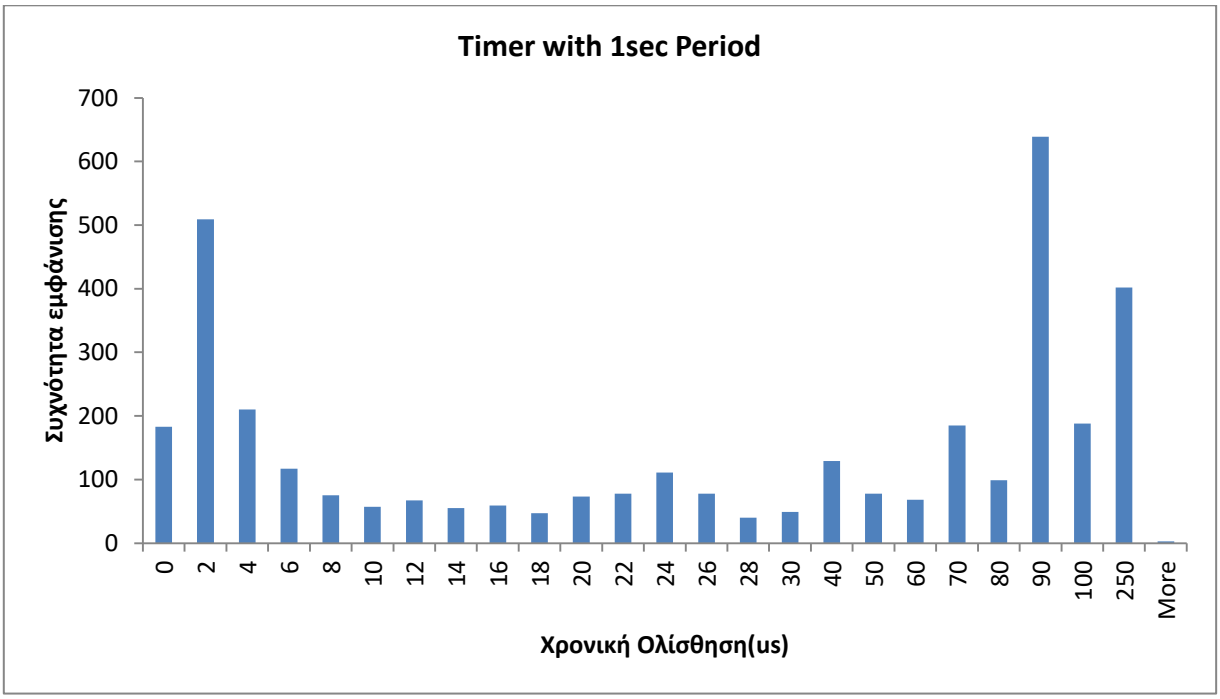
Μέσος όρος:23us **Διάμεσος**:3us **Τυπική απόκλιση**:81,16us **Μέγιστο**:8593us **Ελάχιστο**:0us

Παρατηρούμε ότι οι περισσοτερες τιμές βρίσκονται κοντα στα 2us ενω ενας μικρός αριθμός ξεπερνάει τα 90us. Ο μέσος όρος είναι 23 us που σε σύγκριση με την περίοδο που είναι 100ms είναι μια ικανοποιητικά μικρή απόκλιση.



Μέσος όρος:30us **Διάμεσος**:3us **Τυπική απόκλιση**:115.78us **Μέγιστο**:11802us **Ελάχιστο**:0us

Στο παραπάνω ιστόγραμμα βλέπουμε αρκετές τιμές να είναι πάνω απο 90us,και αντίστοιχα κάτω απο 6us γεγονός που οδηγεί σε ένα μέσο όρο στα 30 us.



Μέσος όρος:47us **Διάμεσος**:30us **Τυπική απόκλιση**:44.35us **Μέγιστο**:281us **Ελάχιστο**:0us

Παρόμοια με παραπάνω ο μέσος όρος είναι στα 47 us και υπάρχουν αρκετές τιμές πάνω απο 100us,όμως η περίοδος του timer είναι 1sec και οι απόκλισεις αυτές είναι απειροελάχιστες.

Χρόνος προσθήκης μιάς διεργασίας στην ουρά

Timer 1sec Period

Μέσος όρος:3.51us **Διάμεσος**:4us **Τυπική απόκλιση**:0.63us **Μέγιστο**:5us **Ελάχιστο**:1us

Timer 0.1sec Period

Μέσος όρος:3.46us **Διάμεσος**:4us **Τυπική απόκλιση**:0.62us **Μέγιστο**:6us **Ελάχιστο**:1us

Timer 0.01sec Period

Μέσος όρος:3.43us **Διάμεσος**:3us **Τυπική απόκλιση :** 0.71:us **Μέγιστο**:88us **Ελάχιστο**:1us

Οι μέσες τιμές των χρόνων προσθήκης είναι σχεδόν ίδιες και για τους 3 timers στις απομονωμένες εκτελέσεις τους, γεγονός που οφείλεται στις απλές συναρτήσεις που προστίθενται στην ουρά αλλα και στο ότι ο χρόνος προσθήκης εξαρτάται κυρίως απο το χρόνο μέχρι να παρθεί το mutex απο τον producer. Πρακτικά, η κάθε διεργασία αφαιρείται απο την

ουρά και εκτελείται απο ένα νήμα consumer πολύ γρήγορα(γρηγορότερα ακόμα και απο την περίοδο του timer 0.01sec) με αποτέλεσμα όταν έρχεται η στιγμή ο producer να προσθέσει μια διεργασία, να μην υπάρχει ανταγωνιστής για την ανάληψη του mutex, καθώς η ουρά είναι τις περισσότερες φορές άδεια και οι consumers κοιμούνται(αφού έχουν προλάβει να εκτελέσουν τις συναρτήσεις).

Χρόνος παραμονής μιας διεργασίας στην ουρά

Timer 1sec Period

Μέσος όρος:36.1us **Διάμεσος**:37us **Τυπική απόκλιση**:11.8us **Μέγιστο**:234us **Ελάχιστο**:11us

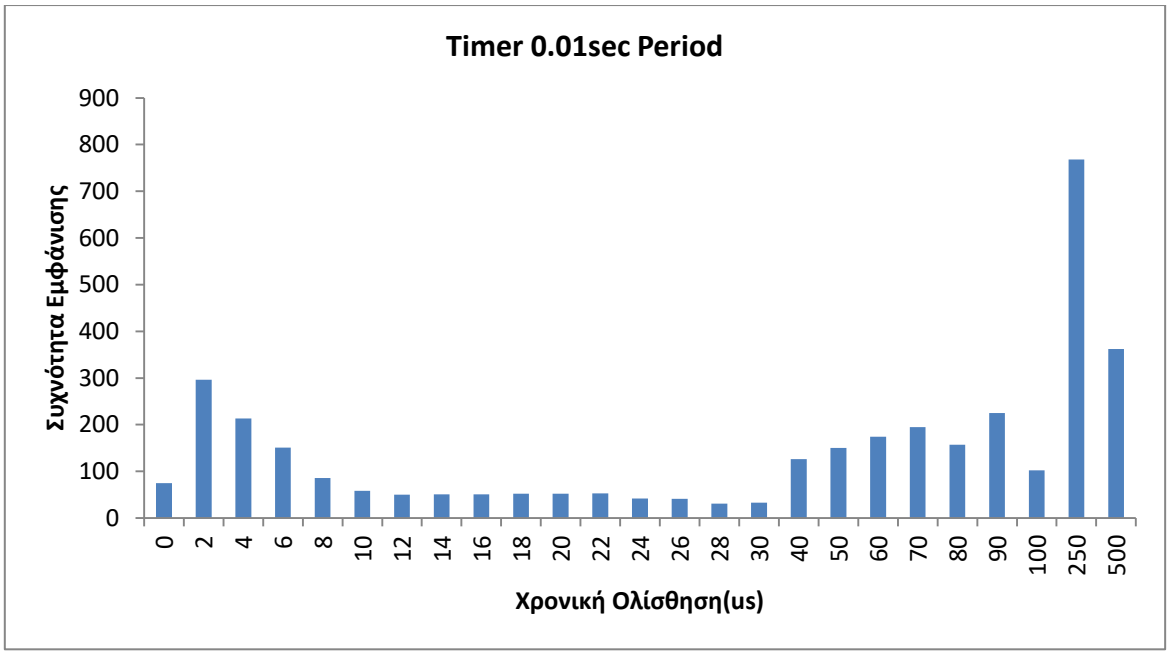
Timer 0.1sec Period

Μέσος όρος:27.4us **Διάμεσος**:30us **Τυπική απόκλιση**:16.57us **Μέγιστο**:2687us **Ελάχιστο**:8us

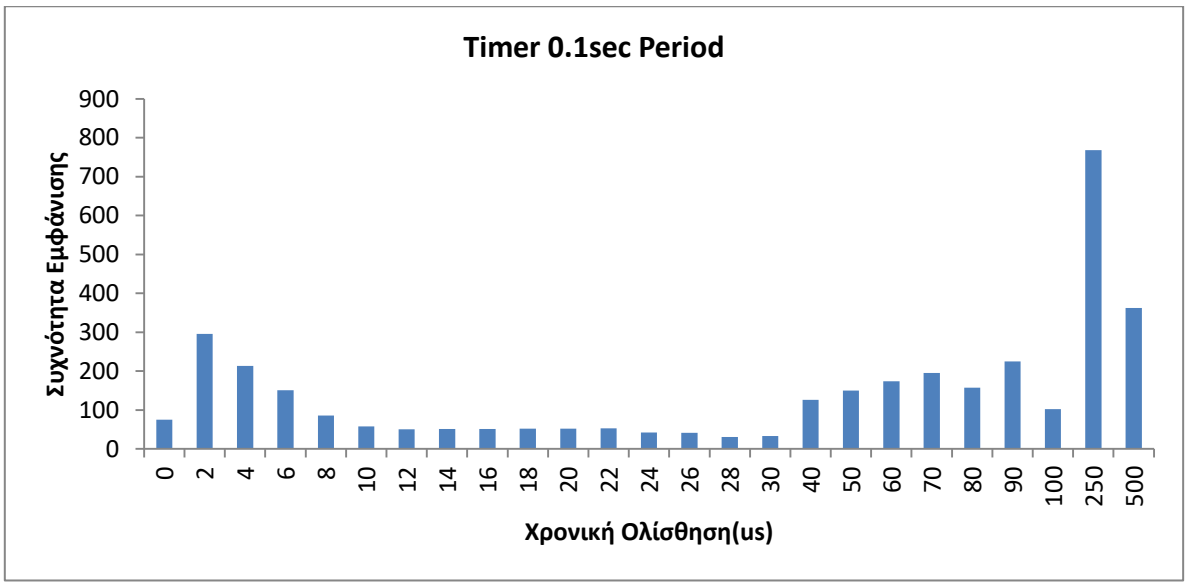
Timer 0.01sec Period

Μέσος όρος:22.8us **Διάμεσος**:20us **Τυπική απόκλιση :** 8.85us **Μέγιστο**:2391us **Ελάχιστο**:7us

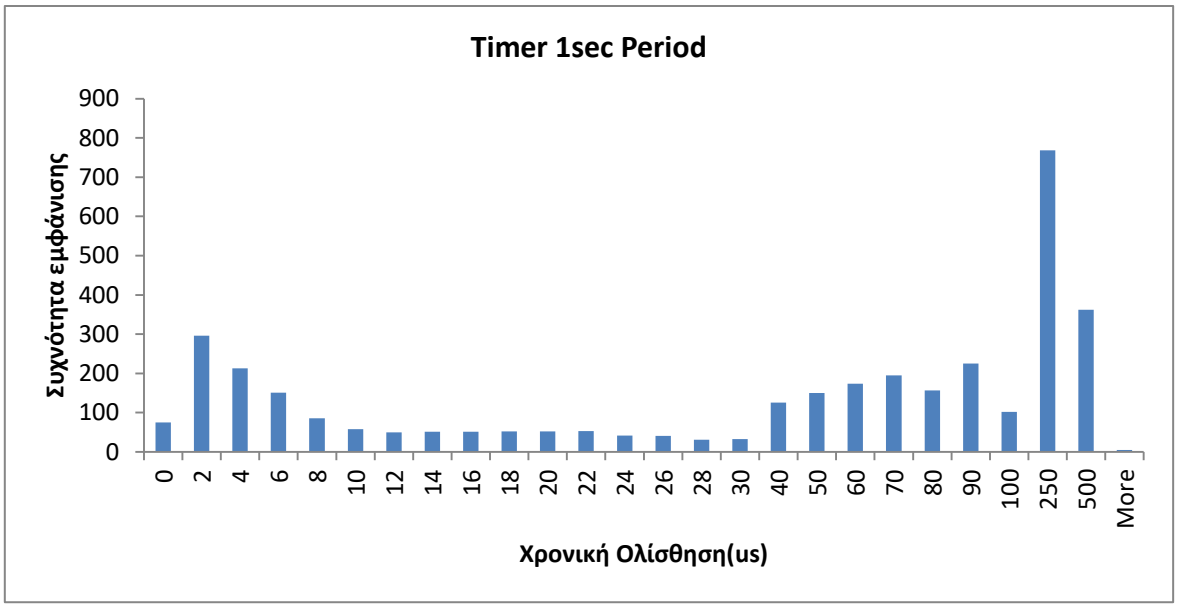
Αποτελέσματα εκτέλεσης με τους 3 timers ταυτόχρονα



Μέσος όρος:39us **Διάμεσος**:4us **Τυπική απόκλιση**: 127.38us **Μέγιστο**:18594us **Ελάχιστο**:0us



Μέσος όρος:51us **Διάμεσος**:22us **Τυπική απόκλιση**: 158.6us **Μέγιστο**:16449us **Ελάχιστο**:0us



Μέσος όρος:91us **Διάμεσος**:61us **Τυπική απόκλιση**: 99.71us **Μέγιστο**:934us **Ελάχιστο**:0us

Κατα τη λειτουργία και των τριών timers ταυτόχρονα οι τιμές της χρονικής ολίσθησης είναι μεγαλύτερες και η συμπεριφορά περισσότερο απρόβλεπτη, με την μέθοδο διόρθωσης να μην τα καταφέρνει εξίσου καλα με πρίν. Πιο συγεκριμένα βλέπουμε ότι ο μέσος όρος του drift αυξήθηκε σε όλους τους timers, με τη μεγαλύτερη αύξηση να σημειώνεται στον timer με περίοδο 1 sec, αμέσως μετά ακολουθεί ο timer με περίοδο 0.1 sec ενώ τη μικρότερη αύξηση παρουσιάζει αυτός με περίοδο 0.01 sec.
.Δεδομένης ,της μεγαλύτερης κινητικότητας στην ουρά λόγω της προσθήκης διεργασιών απο 3 νήματα producers, οι consumers παραμένουν συνολικά περισσότερο ενεργοί. Συνεπώς, υπάρχει μεγαλύτερος ανταγωνισμός μεταξύ των threads για το ποιό θα πάρει το mutex, άρα και μεγαλύτερη απόκλιση μεταξύ διαδοχικών μετρήσεων, γεγονός που επηρεάζει την διόρθωση της usleep().
Επιπλέον, επειδή οι περίοδοι των 3 timers, είναι πολλαπλάσιες η μία της άλλης οι συγκρούσεις είναι αναπόφευκτες.

Ειδικότερα, παρατηρούμε ότι ο timer με περίοδο 1 sec, κάθε φορά που θα προσπαθήσει να προσθέσει κάτι στην ουρά, και συνεπώς να πάρει το mutex, θα ανταγωνιστεί τους άλλους 2 με περιόδους 0.1sec και 0.01sec, γεγονός που δικαιολογεί και την μεγαλύτερη απόκλιση που παρουσίασε το drift σου σχέση με την απομονωμένη εκτέλεση.
Αντίστοιχα, ο timer με περίοδο 0.1 sec θα ανταγωνίζεται συνεχώς με τον timer με περίοδο 0.01 sec και κάθε 10 φορές ,που θα προσπαθεί να προσθέσει κάτι, και τον timer 1sec.
Τέλος, ο timer με περίοδο 0.01 sec,στην πλειοψηφία των προσπαθειών του δεν θα βρίσκει αντίπαλο κάποιον απο τους άλλους timers, καθώς θα συγκρούεται μόνο κάθε 10 προσπάθειες με τον timer περιόδου 0.1 sec και κάθε 100 με τον timer περιόδου 1sec, κάτι που αποτυπώνεται και απο τις μετρήσεις, οι οποίες επηρεάστηκαν λιγότερο, συγκριτικά με την απομονωμένη εκτέλεση.

Χρόνος προσθήκης μιάς διεργασίας στην ουρά

Timer 1sec Period

Μέσος όρος:50us **Διάμεσος**:61us **Τυπική απόκλιση**:17.24us **Μέγιστο**:95us **Ελάχιστο**:7us

Timer 0.1sec Period

Μέσος όρος:3.44us **Διάμεσος**:3us **Τυπική απόκλιση**:0.63us **Μέγιστο**:6us **Ελάχιστο**:1us

Timer 0.01sec Period

Μέσος όρος:3.41us **Διάμεσος**:3us **Τυπική απόκλιση :** 0.92:us **Μέγιστο**:328us **Ελάχιστο**:1us

Χρόνος παραμονής μιας διεργασίας στην ουρά-Ολοι οι timers ταυτόχρονα

Μέσος όρος:31.8us **Διάμεσος**:33us **Τυπική απόκλιση**:29.3us **Μέγιστο**:9688us **Ελάχιστο**:3us

Χρήση της CPU για κάθε πείραμα

Για την εύρεση της χρήσης της CPU, η εκτέλεση των πειραμάτων έγινε με χρήση της εντολής time ./my_code, όπου my_code το αρχείο που προκύπτει μετά το compile του κώδικα. Απο τις τιμές που δίνονται με το πέρας της εκτέλεσης του κάθε πειράματος υπολογίστηκαν τα αποτελέσματα που ακολουθούν για τη χρήση του επεξεργαστή:

i. Εκτέλεση πειράματος για τον **Timer 1sec Period** : Χρήση **CPU 0.01%**
ii. Εκτέλεση πειράματος για τον **Timer 0.1sec Period** : Χρήση **CPU 0.1%**
iii. Εκτέλεση πειράματος για τον **Timer 0.01sec Period** : Χρήση **CPU 1.3%**
iv. Εκτέλεση πειράματος με τους **3 timers ταυτόχρονα**: Χρήση **CPU 1.5%**

Λειτουργία πραγματικού χρόνου- Έγκαιρη έναρξη

Η βασικότερη προυπόθεση για να έχουμε έγκαιρη έναρξη σε ένα σύστημα σαν αυτό που υλοποιήθηκε είναι **η ουρά μας να μην γεμίζει** ποτέ. Το γέμισμα της ουράς ενδέχεται να καθυστερήσει μια επόμενη τοποθέτηση σε αυτήν απο κάποιον timer καθώς θα πρέπει να περιμένει έναν consumer να βγάλει μία διεργασία απο την ουρά, για να καταφέρει να προσθέσει μια καινούρια.
Οι παράγοντες που καθορίζουν σε τί κατάσταση θα βρεθεί η ουρά όμως είναι αρκετοί, καθώς γίνεται αντιληπτό ότι η αλλαγή μεγέθους της ουράς δεν είναι ικανή να λύσει το πρόβλημα απο μόνη της. Ενδεχόμενη αύξηση του μεγέθους της ουράς απο τη μία θα μας επιτρέψει την τοποθέτηση παραπάνω διεργασίων στην ουρά, αλλα αν δεν υπάρχουν αρκετοί consumers, τελικά η διεργασία θα παραμείνει περισσότερο στην ουρα χωρίς τελικά η εκτέλεση της να γίνεται γρηγορότερα. Είναι, λοιπόν απαραίτητο να γνωρίζουμε το πόσο χρόνο καταναλώνει η εκτέλεση της TimerFcn, ώστε ανάλογα να προσαρμόσουμε τον αριθμό των εργατών που θα αναλάβουν την αφαίρεση απο την ουρά και την εκτέλεση της. Εαν οι διεργασίες είναι πολύ χρονοβόρες και καταλήξουμε κάθε εργάτης να έχει αναλάβει απο μία, τότε καθώς οι προσθήκες διεργασιών στην ουρά συνεχίζονται η ουρά θα καταλήξει γεμάτη, με αρνητικές συνέπειες στον χρόνο που τελικά θα εκτελεστούν, οπότε και η αύξηση του αριθμού των εργατών θα ήταν απαραίτητα.

Φυσικά, ρόλο παίζει και ο **ρυθμός προσθήκης** στην ουρά. Μεγάλος ρυθμός προσθήκης, δηλαδή, μικρή περίοδος, πιθανόν να απαιτεί μεγαλύτερο αριθμό εργατών, ή και μεγαλύτερη ουρά εαν αυτό είναι εφικτό καθώς μεγάλη ουρά ίσως σημαίνει και μεγαλύτερος χρόνος αναμονής πρωτού η διεργασία εκτελεστεί. Επομένως, η περίοδος θα πρέπει να είναι τέτοια ώστε ο ρυθμός εισόδου να είναι στη χειρότερη περίπτωση ίσος(εν γένει μικρότερος) με τον ρυθμό εξόδου και η ουρά να μην γεμίζει.

Συμπερασματικά, βλέπουμε ότι το κλειδί για έγκαιρη έναρξη είναι ο ρυθμός με τον οποίο προστίθενται διεργασίες στην ουρά να είναι μικρότερος ή ίσος με τον ρυθμό εξόδου διεργασιών απο αυτή. Ο μέν ρυθμός προσθήκης καθορίζεται απο την περίοδο του timer ενω ο δε ρυθμός εξόδου εξαρτάται, τόσο απο το χρόνο που απαιτεί η εκτέλεση της διεργασίας όσο και απο τον αριθμό των εργατών.

Στη δική μας περίπτωση δεδομένου του μικρού χρόνου που απαιτείται για την εκτέλεση των συναρτήσεων και των περιόδων που δόθηκαν για τους timers, έπειτα απο διαδοχικές εκτελέσεις φάνηκε ότι μια ουρά 5 θέσεων και 10 νήματα-εργάτες ήταν αρκετά ώστε η ουρά να μην γεμίζει.

Ο κώδικα βρίσκεται στο link που ακολουθεί: https://github.com/napoleon98/Embedded-Systems/blob/master/code_apallaktiki_ergasia.c