Ενσωματωμένα συστήματα πραγματικού χρόνου

Αναφορά – Απαλλακτική Εργασία

- Όνομα: Στασινός Αλκιβιάδης AEM: 9214

- Link: https://github.com/astasinos/University-Projects/blob/master/Real-Time%20Embedded%20Systems/Final%20Project/Timer.c
- Platform: Raspberry Pi Model 3B+

1. Γενική δομή και λειτουργικότητα.

Η εργασία αυτή είχε ως αντικείμενο και στόχο την υλοποίηση ενός **Timer** σε **C**. Το **Timer Object** κατασκευάστηκε με ένα **struct** το οποίο περιέχει τις απαραίτητες μεταβλητές ώστε να είναι δυνατές όλες οι λειτουργίες που προδιαγράφονται από την εκφώνηση (π.χ. ο ορισμός της περιόδου, συνάρτηση για περιοδική εκτέλεση, αριθμός επαναλήψεων, **…**) .

Για την αρχικοποίηση των παραμέτρων του εκάστοτε **Timer Object** δημιουργήθηκε για διευκόλυνση η συνάρτηση ***timerInit(Timer *t, int Period, ...)*** . Η λειτουργία του **Timer** στηρίζεται στο σύστημα **producer – consumer** της πρώτης εργασίας. Κάθε **Timer** αντιστοιχίζεται σε **ένα** νήμα **producer**. Η εκκίνηση του **Timer** γίνεται με τη συνάρτηση ***start(Timer *t)*** η οποία απλά καλεί την **pthread_create()** για την δημιουργία του νήματος αυτού. Κατά την εκκίνηση του νήματος εκτελείται πρώτα η **Timer→StartFcn()** και ύστερα πριν συνεχίσουμε περιμένουμε κατά **StartDelay** δευτερόλεπτα μέσω της **usleep()** . Το κύριο μέρος του **Timer thread** περικλείεται από ένα for loop που εκτελείται όσες φορές πρέπει να εκτελεστεί το **task** *(**TasksToExecute***).

Σε κάθε επανάληψη, ο producer προσπαθεί να “πάρει” το **mutex** της ουράς. Αν αυτή είναι γεμάτη, εκτελείται η T**imer→ErrorFcn()** και ο timer περιμένει να αδειάσει κάποια θέση. Αν όμως δεν είναι γεμάτη τότε ο **Timer** (δηλ. ο producer) προσθέτει στην ουρά ένα **task**, ενώ μέχρι την επόμενη εκτέλεση του προσπαθεί μέσω της **usleep()** να περιμένει τόσο χρόνο όσο πιο κοντά γίνεται στην περίοδο του. Μετά από την τελευταία προσθήκη στην ουρά , άρα ουσιαστικά από την τελευταία εκτέλεση της **Timer→Fcn()** , εκτελείται η **Timer→StopFcn()** και ο **Timer** σταματά. Τα **tasks** στην ουρά είναι στη πραγματικότητα **workFunction structs**. Ο τύπος του struct αυτού είναι ο ίδιος με την πρώτη εργασία και περιλαμβάνει απλά ένα δείκτη προς την **Timer→Fcn()** που θα εκτελείται περιοδικά, τα **δεδομένα** για τη συνάρτηση αυτή και ένα **timeval struct** στο καταγράφεται ο χρόνος που προστέθηκε το **task** στην ουρά.

Οι **consumers** περιμένουν κάθε φορά να προστεθεί ένα **task** στην ουρά. Ο **consumer** που θα πάρει το **mutex** θα αναλάβει να “βγάλει” και ένα αντικείμενο από την ουρά και έπειτα να εκτελέσει την **Timer→Fcn()** που σχετίζεται με το task. Σημειώνεται πως οι συναρτήσεις για αρχικοποίηση/διαγραφή της ουράς ή προσθήκης/αφαίρεσης αντικειμένων από αυτή είναι οι ίδιες όπως φαίνονται στην πρώτη εργασία.

2. Διόρθωση χρονικής ολίσθησης.

Όπως ήδη αναφέρθηκε για την επίτευξη της περιοδικότητας στην εκτέλεση των συναρτήσεων χρησιμοποιήθηκε η συνάρτηση **usleep()**. Ωστόσο, όπως αναφέρεται και στο **man page** της “**The sleeptime may be lengthened slightly by any system activity or by the time spent processing the call or by the granularity of system timers**” . Επομένως, η χρήση της **usleep()** δε προσφέρει μεγάλη χρονική ακρίβεια και ερχόμαστε αντιμέτωποι με μία χρονική ολίσθηση η οποία σε βάθος χρόνου αποδεικνύεται σημαντική. Για την ελαχιστοποίηση της ολίσθησης από την περίοδο και την αποφυγή της συσσώρευσης μεγάλης απόκλισης εφαρμόστηκε η παρακάτω τεχνική:

Ορίζεται αρχικά ένα σημείο χρονικής αναφοράς (μετά από κάθε εκτέλεση της **queueAdd**) ως προς το οποίο θα μετράμε το χρόνο που παρήλθε από τη προηγούμενη φορά που ο **Timer** βρέθηκε σε αυτό το σημείο. Μέσω της **gettimeofday()** δηλαδή υπολογίζεται η χρονική διαφορά μεταξύ δύο iterations του for loop του **Timer**. Από τον χρόνο αυτόν αφαιρείται η περίοδος και το αποτέλεσμα αποθηκεύεται στη μεταβλητή **timedrift**. Εν ολίγος, η τιμή του **timedrift** ανάλογα με το πρόσημο της μας δείχνει πόσο “πάνω” ή “κάτω” αποκλίναμε από την περίοδο για κάθε task. Το όρισμα τελικώς της **usleep** θα είναι η μεταβλητή **waittime** για την τιμή της οποίας ξεχωρίζουμε 3 περιπτώσεις.

A) Στην πρώτη εκτέλεση του loop τίθεται ίση με την περίοδο **Timer→Period**. **B)** Σε κάθε επόμενο iteration είναι **waittime = waittime – timedrift**. Για παράδειγμα αν είχαμε περίοδο **10 ms** και την πρώτη φορά αντί για 10 περιμέναμε λανθασμένα **11** τότε στην επόμενη θα προσπαθήσουμε να περιμένουμε **10 – (11-10) = 9 ms** για να αντισταθμιστεί η καθυστέρηση που προσθέτει το σύστημα και ο πραγματικός χρόνος να καταλήξει **10 ms = Period** . Αν τώρα πάλι αντί

για **10** τελικά καταλήξουμε να περιμένουμε **9.6 ms** τότε ο νέος χρόνος αναμονής (waittime) γίνεται **9 – (9.6-10)** = **9.4 ms** . Ο χρόνος αναμονής δηλαδή προσαρμόζεται συνεχώς για να μειώσει τις αποκλίσεις προς οποιαδήποτε κατεύθυνση. Θα μπορούσαμε να πούμε πως μέθοδος αυτή προσομοιάζει σε έναν ελεγκτή με ανατροφοδότηση. **Γ)** Σε περίπτωση που το σφάλμα είναι πάνω από την περίοδο, π.χ. αν με περίοδο **10 ms** ο χρόνος που περάσει είναι **21** ms τότε το **waittime** για την επόμενη επανάληψη θα έπαιρνε αρνητική τιμή (αδύνατο). Γι' αυτό τότε το **waittime** τίθεται στο **0** δηλαδή το επόμενο **task** ξεκινά άμεσα. Τέτοιες περιπτώσεις είναι σπάνιες.

3. Σχολιασμός και σύγκριση πειραματικών αποτελεσμάτων.

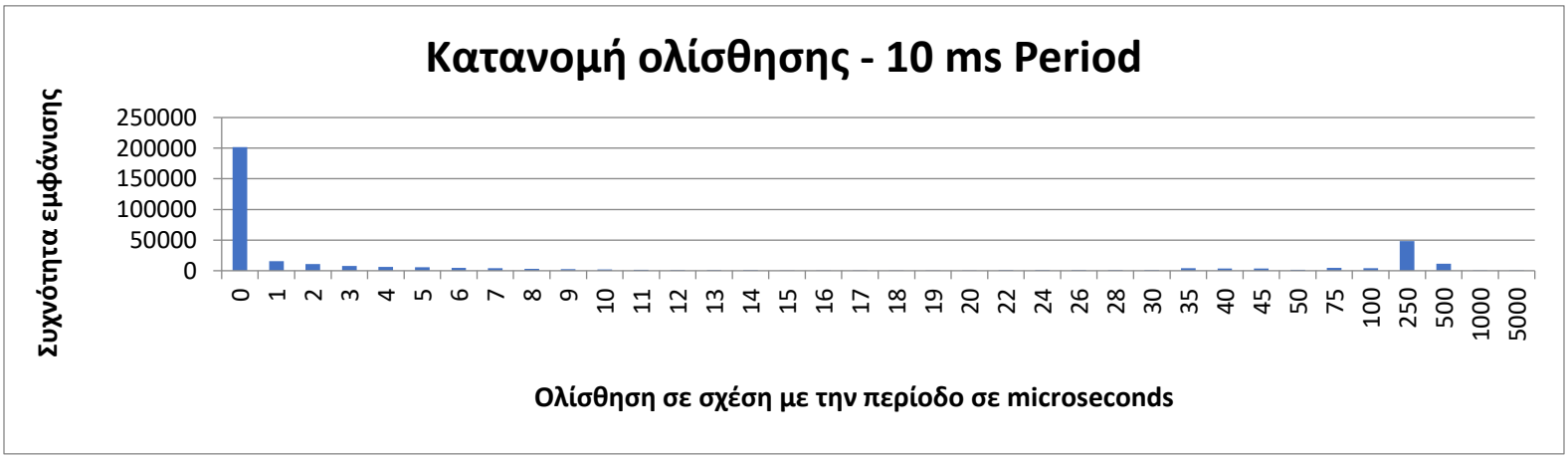
Συνολικά διενεργήθηκαν **4** πειράματα με αριθμό consumers **100**. Ένα στο οποίο λειτουργούσαν παράλληλα **και οι 3** Timers και άλλα τρία στα οποία έτρεχε ο καθένας μόνος του. Σε κάθε πείραμα έγιναν τρείς διαφορετικές μετρήσεις.

1. Η χρονική ολίσθηση (drift) για κάθε task. **2**. Ο χρόνος που ξοδεύει ο producer (Timer) να βάλει κάθε φορά το task στην ουρά. Ο χρόνος αυτός υπολογίζεται από την πρώτη προσπάθεια που κάνει ο producer να κάνει lock το mutex της ουράς, μέχρι και να εκτελεστεί η **queueAdd**. **3.** Ο χρόνος παραμονής του task στην ουρά, δηλαδή ο χρόνος από όταν το αντικείμενο προστεθεί στην ουρά έως ότου παραληφθεί από έναν consumer και εκτελεστεί η **queueDel**. Μετά την queueDel το mutex γίνεται unlock και εκτελείται η **Timer→Fcn**. Ουσιαστικά, ο τελευταίος αυτός χρόνος **είναι και ο χρόνος καθυστέρησης (latency)** για την εκτέλεση της **Timer→Fcn()**.

3.1 Λειτουργία και των 3 Timer μαζί.

3.1.1 Καταγραφή χρονικών μετατοπίσεων.

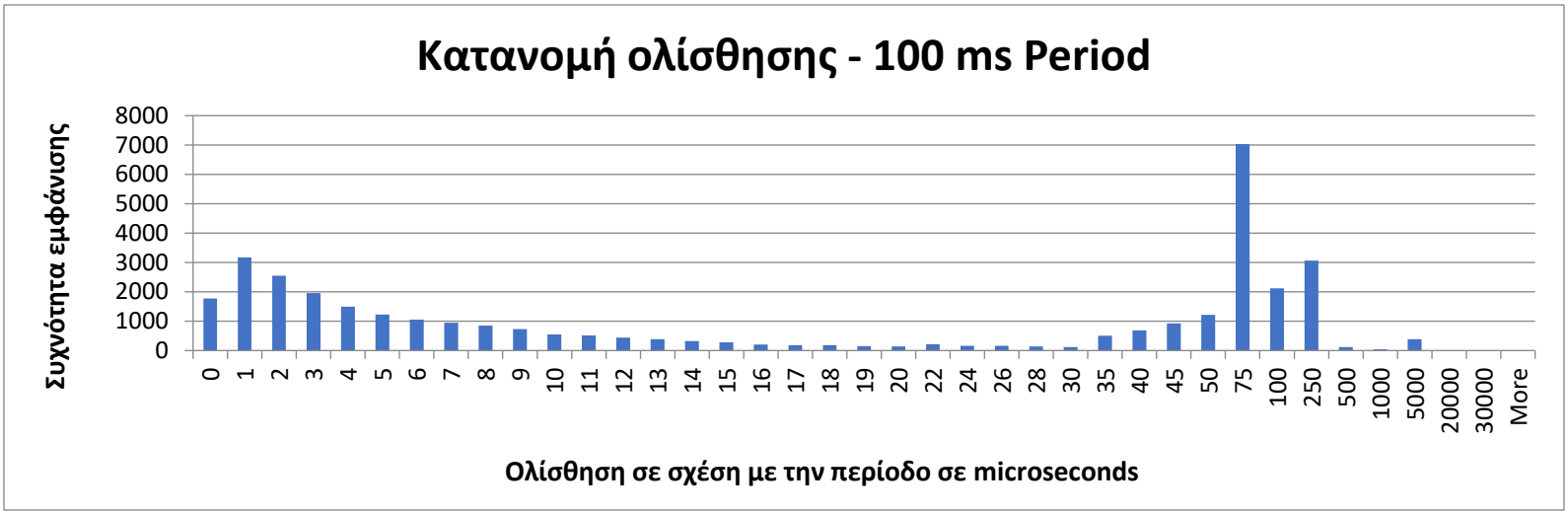
Πρώτα παρουσιάζονται τα στατιστικά στοιχεία των drift για τους **3 Timers** μαζί. Στο ακόλουθο διάγραμμα φαίνεται η κατανομή της **απόλυτης τιμής** time drift **για τον timer με περίοδο 10 ms** όταν λειτουργούσαν και οι άλλοι δύο μαζί του.



Βλέπουμε πως τις περισσότερες φορές λόγω της διόρθωσης, η χρονική απόκλιση ήταν **μηδενική**. Την αμέσως μεγαλύτερη συχνότητα είχαν ολισθήσεις **150 – 250 us** με λίγες να φτάνουν στην περιοχή των **500 us**. Οι υπόλοιπες κινήθηκαν στην περιοχή **1 ως 10 us**, ενώ **πολύ** σπάνια υπήρξαν αποκλίσεις πάνω από **5000 us**.

Μέσος όρος Drift : 79.405 us , Μέγιστο : 13120 us , Ελάχιστο : 0 us , Διάμεσος : 15 us Τυπική απόκλιση : 135.9718 us

Για τον δεύτερο **Timer (100 ms Period)** τα αντίστοιχα αποτελέσματα ήταν:

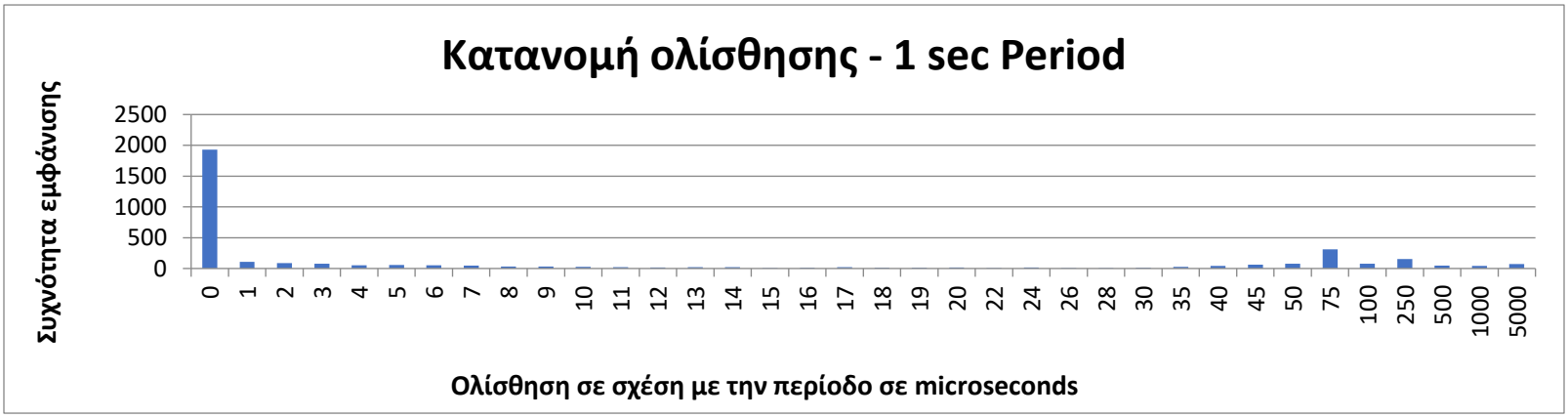


Στο διάγραμμα παρατηρούμε πως ως επί τω πλείστων η χρονικές μετατοπίσεις σε σχέση με την περίοδο κινούνται γύρω από την περιοχή των **75 us** . Την αμέσως μεγαλύτερη συχνότητα εμφάνισης είχαν ολισθήσεις **150 – 250 us** με λίγες να φτάνουν στην περιοχή των **500 us** και άνω. Οι υπόλοιπες κινήθηκαν στην περιοχή **1 ως 15 us** με τις πιο συχνές στα **1** και **2 us ,** ενώ όπως πρίν **πολύ** σπάνια υπήρξαν αποκλίσεις πάνω από **5000 us**.

Μέσος όρος Drift : 66.99 us , Μέγιστο Drift : 14147 us , Ελάχιστο Drift : 0 , Διάμεσος : 15 us

Τυπική απόκλιση : 317.57 us

Τέλος παρουσιάζονται τα στατιστικά στοιχεία για τον **3ο Timer** με περίοδο ενός δευτερολέπτου. Όπως και πριν τα ακόλουθα αποτελέσματα αναφέρονται σε παράλληλη λειτουργία του **Timer** μαζί με τους άλλους δύο.



Μέσος όρος Drift : 131.47 us , Μέγιστο Drift : 5707 us , Ελάχιστο Drift : 0 , Διάμεσος : 39 us

Τυπική απόκλιση : 366.651 us

Εδώ παρατηρούμε και πάλι πως οι **μηδενικές** χρονικές μετατοπίσεις κυριαρχούν ενώ οι αμέσως πιο συχνές είναι κοντά στα **75 us** και μερικές στη περιοχή των **250 us.** Ωστόσο παρατηρούνται περίπου 75 μετρήσεις πάνω από **1000 us** . Το γεγονός πως αυτός ο **Timer** έχει περίοδο ενός δευτερολέπτου άμεσα σημαίνει πως τοποθετεί μόνο **3601** tasks στην ουρά (Το πρώτο όπως ζητείται τοποθετείται απευθείας ενώ κάθε επόμενο μετά από **1 sec**) . Αυτό έχει ως αποτέλεσμα οι λιγοστές υψηλές χρονικές αποκλίσεις να επηρεάζουν σημαντικά τον μέσο όρο.

Μια σημαντική παρατήρηση είναι πως αν αθροίσουμε όλες τις τιμές των drift (με τα πρόσημα) για κάθε timer μπορούμε να βρούμε κατά πόσο στο τέλος του πειράματος ξεπεράσαμε τη προδιαγεγραμμένη διάρκεια της μίας ώρας. Συγκεκριμένα για τον **1ο Timer** το άθροισμα είναι **3535 us** δηλαδή η λειτουργία του διήρκησε 1 ώρα και 3 ms. Ο **2ος Timer** είχε άθροισμα drift **643 us** ενώ ο **3ος 2431 us .** Συμπερασματικά το πρόγραμμα τερματίστηκε όταν σταμάτησε ο πρώτος **Timer** και συνολικά αποκλίναμε **3 ms** .

3.1.2 Καταγραφή χρόνου προσθήκης task στην ουρά.

Όπως εξηγήθηκε ο χρόνος αυτός ορίζεται από την πρώτη φορά που ένα **Timer – Producer** προσπαθεί να πάρει το mutex της ουράς, μέχρι να ολοκληρωθεί η queueAdd για το task. Έτσι, για λειτουργία και των τριών Timer μαζί έχουμε:

Timer 1 – Period 10 ms

Μέσος Όρος: 2.94 us , Μέγιστος χρόνος: 13113 us , Ελάχιστος : 0 us , Διάμεσος: 1 us , Τυπική Απόκλιση: 68.59 us

Timer 2 – Period 100 ms

Μέσος Όρος: 17.37 us , Μέγιστος χρόνος: 14114 us , Ελάχιστος : 1 us , Διάμεσος: 1 us , Τυπική Απόκλιση: 229.09 us

Timer 3 – Period 1 sec

Μέσος Όρος: 47.8 us , Μέγιστος χρόνος: 5716 us , Ελάχιστος : 1 us , Διάμεσος: 2 us , Τυπική Απόκλιση: 276.53 us

Οι μετρήσεις που κατεγράφησαν κινήθηκαν στη **συντριπτική** πλειοψηφία τους σε χρόνους **1 – 5 us .** Ωστόσο όπως φαίνεται υπήρξαν κάποια έντονα **"spikes"** στις τιμές. Το φαινόμενο αυτό εξηγείται αργότερα.

3.1.3 Καταγραφή χρόνου καθυστέρησης (latency) εκτέλεσης της Timer→Fcn .

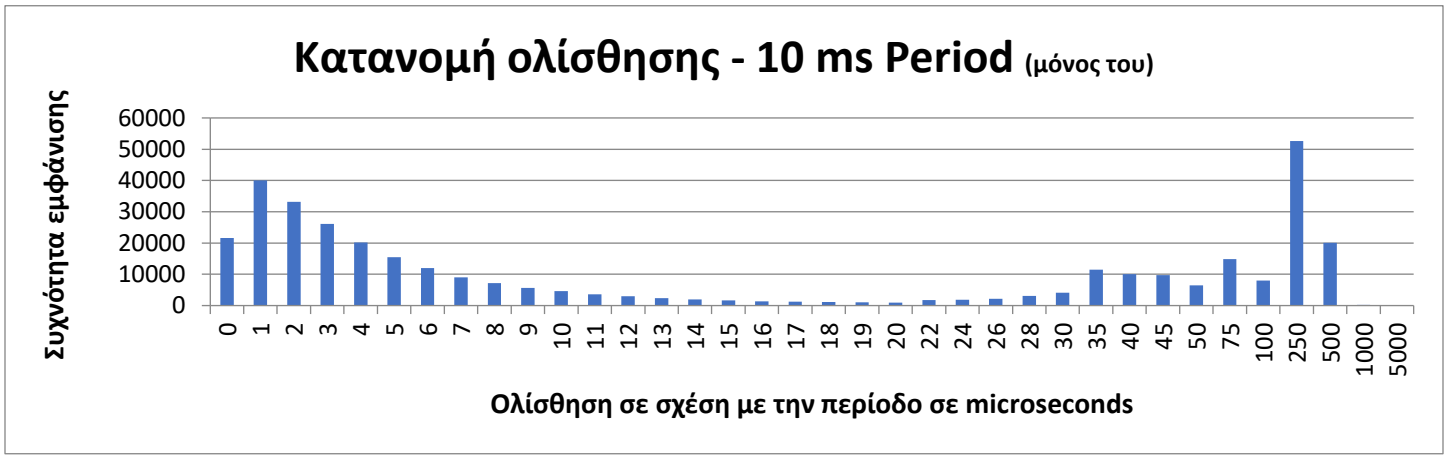
Τα παρακάτω αποτελέσματα αντιστοιχούν στο **χρόνο καθυστέρησης (latency)** από όταν ένα task μπει στην ουρά μέχρι την εκτέλεση της **Timer→Fcn()** . Συνολικά προστέθηκαν 360001 + 36001 + 3601 = 399603 tasks στην ουρά ώστε κάθε timer να έχει διάρκεια μιας ώρας.

Μέσος όρος καθυστέρησης: 37.566 us , Μέγιστη : 1484 us , Ελάχιστη: 2 us , Διάμεσος: 35 us , Τυπική Απόκλιση: 16.13

3.2 Λειτουργία κάθε Timer ξεχωριστά.

3.2.1 Καταγραφή χρονικών μετατοπίσεων.

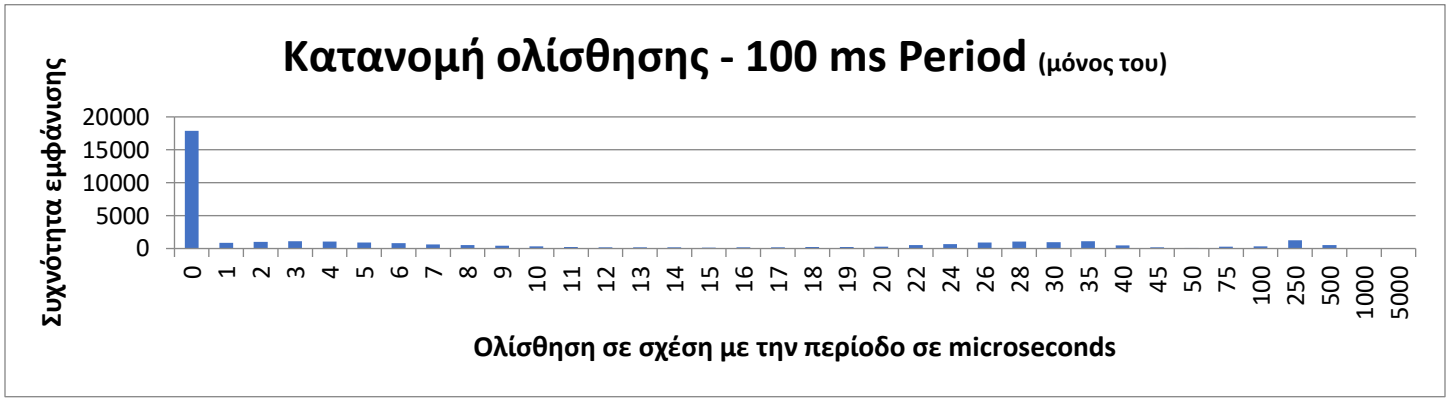
Πρώτα για τον **Timer** με περίοδο **10 ms** έχουμε:



Μέσος όρος Drift : 55.19 us , Μέγιστο Drift : 1847 us , Ελάχιστο Drift : 0 , Διάμεσος : 17 us Τυπική απόκλιση : 90.33 us

Εδώ το 75 % των μετρήσεων κινήθηκε σε τιμές κάτω από **20 us** ενώ οι υπόλοιπες τιμές κινήθηκαν κοντά στα **250 us**.

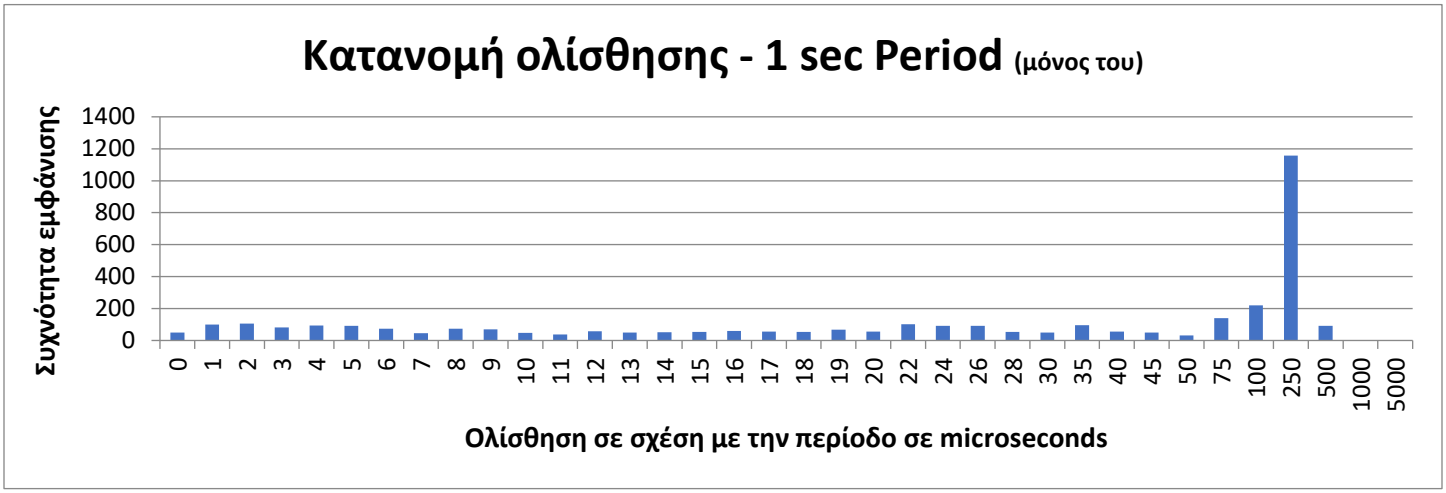
Για τον **Timer** με περίοδο **100 ms** σε απομονωμένη λειτουργία έχουμε:



Μέσος όρος Drift : 35.66 us , Μέγιστο Drift : 566 us , Ελάχιστο Drift : 0 , Διάμεσος : 22 us Τυπική απόκλιση : 60.81 us

Παρατηρούμε πως κυριαρχεί η **μηδενική** απόκλιση ενώ δεν υπάρχει **καμία** μέτρηση πάνω από την περιοχή των **500 us**.

Τέλος για τον τρίτο **Timer** με περίοδο **1 sec** είναι :



Από το διάγραμμα καταλαβαίνουμε πως οι μισές μετρήσεις ήταν από **0** έως **75 us** και οι υπόλοιπες κοντά στα **250 us**. Σημαντικό το ότι δεν υπάρχει καμία μέτρηση πάνω από **500 us**.

Μέσος όρος Drift : 79.23 us , Μέγιστο Drift : 354 us , Ελάχιστο Drift : 0 , Διάμεσος : 32 us

Τυπική απόκλιση : 81.12 us

3.2.2 Καταγραφή χρόνου προσθήκης task στην ουρά.

Timer 1 – Period 10 ms

Μέσος Όρος: 1.26 us , Μέγιστος χρόνος: 1573 us , Ελάχιστος : 0 us , Διάμεσος: 1 us , Τυπική Απόκλιση: 2.70 us

Timer 2 – Period 100 ms

Μέσος Όρος: 1.48 us , Μέγιστος χρόνος: 4 us , Ελάχιστος : 1 us , Διάμεσος: 1 us Τυπική Απόκλιση: 0.52 us

Timer 3 – Period 1 sec

Μέσος Όρος: 1.75 us , Μέγιστος χρόνος: 448 us , Ελάχιστος : 1 us , Διάμεσος: 2 us , Τυπική Απόκλιση: 7.46 us

3.2.3 Καταγραφή χρόνου καθυστέρησης (latency) εκτέλεσης της Timer→Fcn.

Timer 1 – Period 10 ms

Μέσος Όρος: 34.99 us , Μέγιστος χρόνος: 370 us , Ελάχιστος : 10 us , Διάμεσος: 23 us , Τυπική Απόκλιση: 17.24 us

Timer 2 – Period 100 ms

Μέσος Όρος: 52.51 us , Μέγιστος χρόνος: 115 us , Ελάχιστος : 21 us , Διάμεσος: 61 us Τυπική Απόκλιση: 16.64 us

Timer 3 – Period 1 sec

Μέσος Όρος: 62.38 us , Μέγιστος χρόνος: 125 us , Ελάχιστος : 33 us , Διάμεσος: 65 us , Τυπική Απόκλιση: 12.98 us

3.3 Σύγκριση και ερμηνεία αποτελεσμάτων.

3.3.1 Σύγκριση Timer με περίοδο 10 ms.

Στο διάγραμμα των χρονικών μετατοπίσεων για εκτέλεση με τρείς **Timers μαζί** υπάρχουν πολλές μηδενικές τιμές και οι δεύτερες πιο συχνές είναι αυτές στα **250 us**. Στο αντίστοιχο διάγραμμα για απομονωμένη εκτέλεση παρατηρείται πως η **usleep** έκανε περισσότερα λάθη **αλλά** αυτά ήταν συγκεντρωμένα από **0 ως 10 us**, και περίπου ίδιο αριθμό λαθών κοντά στα **250 us** σε σχέση με την προηγούμενη περίπτωση. Η **βασικότερη** διαφορά είναι πως κατά την απομονωμένη εκτέλεση, κοντά στη περιοχή των **5000 us** υπήρξαν μόνο **4** μετρήσεις. Αντίθετα η εκτέλεση με όλους τους Timers μαζί είχε ως αποτέλεσμα την εμφάνιση 243 μετρήσεων κοντά στα **5000 us** ενώ υπήρξαν και 6 μετρήσεις από **10000** ως **15000 us**. Εδώ παρατηρείται εύκολα η εμφάνιση μερικών **αρκετά υψηλών** τιμών, φαινόμενο που δεν εμφανίστηκε

τόσο έντονα στην απομονωμένη εκτέλεση. Κατά συνέπεια όπως είδαμε ο μέσος όρος των drift εδώ ήταν **79.405 us** ενώ στην απομονωμένη εκτέλεση **55.19 us**.

Κατά την απομονωμένη εκτέλεση υπάρχει μόνο ένα νήμα **producer** το οποίο κλειδώνει το mutex της ουράς, εναποθέτει task σε αυτή και αφού παραδώσει το mutex "κοιμάται" μέσω της **usleep** τόσα **us** όσα ορίζει η περίοδος του **(εδώ 10 ms)**. Αφού τοποθετήσει το task στην ουρά, άμεσα ένας από τους **consumers** θα παραλάβει το task και θα εκτελέσει την **Timer→Fcn**. Δεδομένης της πολύ μικρής επεξεργαστικής δυσκολίας που έχουν τα task που έχουμε επιλέξει, τα **10 ms** περιόδου που έχει αυτός ο **Timer** είναι υπεραρκετά για τη διεκπεραίωση του task από έναν **consumer**.

Από την άλλη, όταν λειτουργούν **και οι τρείς Timers** διάφορα φαινόμενα μπορούν όπως είδαμε να επηρεάσουν αισθητά τις τιμές της χρονικής μετατόπισης (drifting). Πιο συγκεκριμένα, τώρα συνυπάρχουν **3** νήματα **producer** με το ρυθμό τοποθέτησης task στην ουρά να αυξάνει. Κάθε φορά που ένας **Timer** προσπαθεί να βάλει κάτι στην ουρά πρέπει να κλειδώσει το **mutex**. Αυτό είναι πολύ πιθανό να τον φέρει σε σύγκρουση με κάποιον άλλο **Timer** που θέλει να κάνει το ίδιο εκείνη τη στιγμή. Επιπλέον η αυξημένη κίνηση στην ουρά σημαίνει πως είναι τώρα πιο πιθανό ένας **consumer** να "κλέψει" το **FIFO mutex** από κάποιον producer που θα έπρεπε εκείνη τη στιγμή να εκτελεστεί. Σημειώνεται επίσης πως κατά τις μετρήσεις τα νήματα ήταν επιβαρυμένα και με εντολές απαραίτητες για τις ίδιες τις μετρήσεις.

Οι περίοδοι των timers είναι πολλαπλάσια η μία της άλλης, οπότε εύκολα ο timer με περίοδο 10 ms μπορεί να συγκρουστεί με τον timer με περίοδο 100 ms η και αυτόν με περίοδο ενός δευτερολέπτου. Παραδειγματικά αναφέρεται η περίπτωση όπου ο **Timer** με περίοδο **1 sec** θέλει να εκτελεστεί. Τη στιγμή εκείνη λόγω της σχέσης των περιόδων **και οι υπόλοιποι Timers** επιδιώκουν να προσθέσουν task στην ουρά. Στη χειρότερη περίπτωση ο **Timer** με περίοδο **10 ms** θα πρέπει να περιμένει την εκτέλεση των άλλων δύο **Timer** και την παραλαβή των task από τους **consumers** πριν μπορέσει να ξεκινήσει. Η καθυστέρηση αυτή μπορεί να γίνει σημαντική. Οι συγκρούσεις γενικά μπορεί να είναι αρκετά συχνές και με διαβαθμίσεις στην επίπτωση που έχουν σε κάθε συγκεκριμένο **Timer**. Οι πιθανές συγκρούσεις είναι μεταξύ **Timers 10 ms – 100 ms , 10 ms – 1 sec , 100 ms – 1sec , 10 ms – 100 ms – 1 sec**.

Η καθυστέρηση που προκαλούν τα φαινόμενα αυτά προστίθεται με την ανακρίβεια της **usleep** με αποτέλεσμα να αυξάνουν το χρόνο μεταξύ των εκτελέσεων ενός **Timer**, άρα κατ' επέκταση και τη χρονική μετατόπιση από την περίοδο (drifting).

3.3.2 Σύγκριση Timer με περίοδο 100 ms.

Για τον **Timer** με περίοδο **100 ms** όπως φαίνεται και από τα διαγράμματα, στην απομονωμένη εκτέλεση η συντριπτική πλειοψηφία των μετρήσεων είναι κοντά στο **0** με κάποιες να βρίσκονται στην περιοχή των **25 – 35 us** και λίγες στα **250 us.** Αντίθετα στην μαζική εκτέλεση βλέπουμε πως οι τιμές συγκεντρώνονται στα **75 us** και είναι αισθητές μερικές υψηλότερες τιμές. Συγκεκριμένα υπήρξαν 385 τιμές κοντά στα **5000 us** και 10 τιμές στη περιοχή **10000 – 15000 us**.

Αντίθετα η υψηλότερη καταγεγραμμένη τιμή για την απομονωμένη εκτέλεση ήταν μόλις **569 us**. Καταλαβαίνουμε πως για τους ίδιους λόγους που αναλύθηκαν στη προηγούμενη σύγκριση, η απομονωμένη εκτέλεση είχε καλύτερη και συνεπέστερη χρονική συμπεριφορά. Μέσος όρος απομονωμένης: **35.66** , Μέσος όρος μαζικής εκτέλεσης: **66.99 us**.

3.3.3 Σύγκριση Timer με περίοδο 1 sec.

Όπως και στη σύγκριση του πρώτου **Timer** εδώ στην εκτέλεση και των τριών timer μαζί βλέπουμε μεγαλύτερη συγκέντρωση στις μηδενικές τιμές, **αλλά** και την ύπαρξη αρκετών υψηλών τιμών κοντά στα **5000 us** και 10 τιμών στα **10000 – 15000 us**. Η απομονωμένη εκτέλεση εμφανίζει μεγαλύτερη συχνότητα στις τιμές **100** – **250 us** αλλά η μέγιστη τιμή drift εδώ είναι **354 us**. Επίσης ο μέσος όρος των drift σε αυτόν τον timer επηρεάζεται πιο εύκολα λόγω των υποπολλαπλάσιων task που προσθέτει στην ουρά (αρά και λιγότερες μετρήσεις drift). Όμοια με πριν, η απομονωμένη εκτέλεση αποδεικνύεται αποδοτικότερη ως προς τη χρονική μετατόπιση. Μέσος όρος απομονωμένης: **79.23 us** , Μέσος όρος μαζικής εκτέλεσης: **131.47 us**.

4. Μέτρηση καταναλισκόμενης επεξεργαστικής ισχύος για την εκτέλεση.

Για τη μέτρηση της χρήσης της **CPU** για κάθε πείραμα χρησιμοποιήθηκε η εντολή **time** ./program όπου program το όνομα του executable για κάθε πείραμα. Βάσει των αποτελεσμάτων της εντολής, για την εκτέλεση με τη παράλληλη λειτουργία **και των τριών** Timer η μέση χρήση της CPU προέκυψε **0.9 %** . Για τον Timer με περίοδο **10 ms** με τον ίδιο τρόπο, η χρήση προέκυψε **0.817 % .**

Αντίστοιχα για την απομονωμένη εκτέλεση του Timer με περίοδο **100 ms** η χρήση ήταν **0.088 %** και τέλος για τον Timer με περίοδο **1 sec** η χρήση της CPU ήταν **0.0098 %** . Είναι φανερό πως όσο μεγαλύτερη η περίοδος του Timer τόσο λιγότερη η χρήση της CPU. Παρατηρούμε πως οι εκτελέσεις χρησιμοποιούν πολύ μικρό ποσοστό της CPU, μικρότερο του **0.** Μεγαλύτερη περίοδος σημαίνει μεγαλύτερα διαστήματα κατά τα οποία ο Timer "κοιμάται" αλλά και σε μια

προκαθορισμένη διάρκεια λειτουργίας (μίας ώρας) ο Timer με μεγαλύτερη περίοδο τοποθετεί λιγότερα tasks στην ουρά, άρα και λιγότερες εκτελέσεις της Timer→Fcn. Γι' αυτό και οι timers με περίοδο **100 ms και 1 sec** εμφανίζουν αρκετά χαμηλότερα ποσοστά.

5. Βελτιστοποίηση χρόνου αναμονής για λειτουργία πραγματικού χρόνου.

Η λειτουργία πραγματικού χρόνου αναφέρεται στην ικανότητα του συστήματος, μόλις υπάρξει ένα αίτημα για την εκτέλεση ενός task, το αίτημα να εξυπηρετείται άμεσα. Πρακτικά, τα tasks πρέπει να μπαίνουν απρόσκοπτα στην ουρά χωρίς καθυστερήσεις και αφού εισέλθουν θα πρέπει να εκτελούνται όσο το δυνατόν πιο άμεσα.

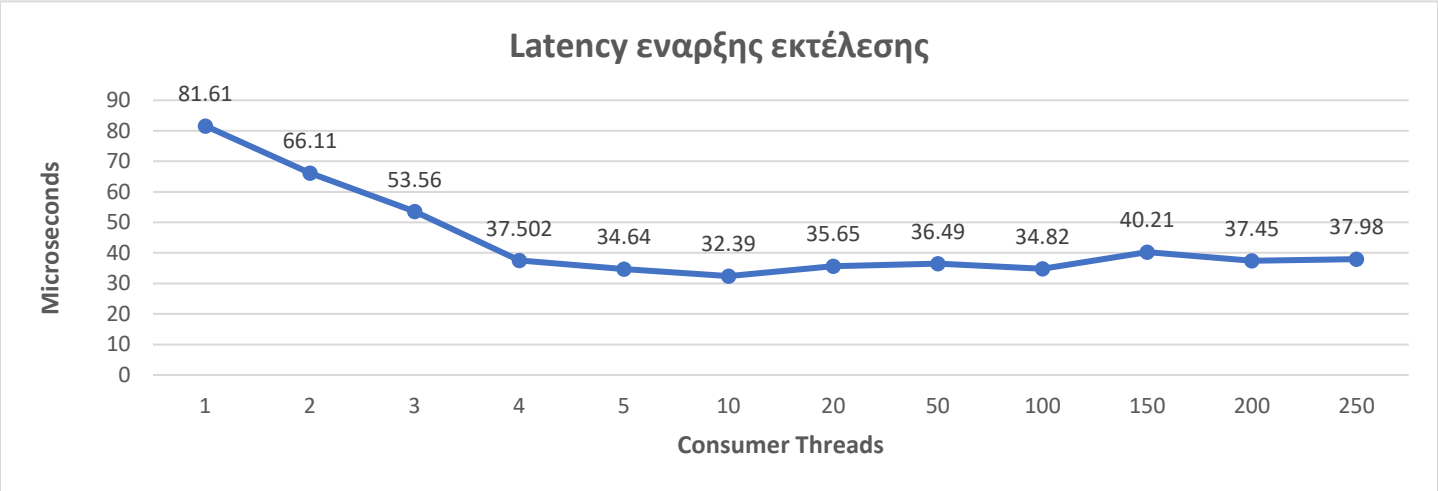
Όπως αναφέρεται και στην εκφώνηση, οι παράμετροι που επηρεάζουν το χρόνο αναμονής για την εκτέλεση μιας **Timer→Fcn** είναι **α)** Το μέγεθος της ουράς **β)** ο χρόνος που ξοδεύουμε για την εκτέλεση της **Timer→Fcn γ)** με ποια περίοδο τοποθετεί ο **Timer** τα task στην ουρά και τέλος **δ)** ο αριθμός από τους consumers.

Για λειτουργία πραγματικού χρόνου απαραίτητη προϋπόθεση είναι η ουρά **να μη φτάνει στα όρια της δηλαδή να μη γεμίζει συχνά**. Θα πρέπει να υπάρχει ομαλή ροή εισόδου και εξόδου των tasks. Αν η ουρά παραμένει γεμάτη για αρκετό χρόνο τότε είναι πιθανό ένας **Timer** που δε βρίσκει ελεύθερο χώρο να τοποθετήσει ένα task να **μπλοκάρει** έως ότου να αδειάσει μια θέση. Φυσικά κάτι τέτοιο θα έχει μεγάλο κόστος στα πλαίσια του στόχου μας για έγκαιρη εκτέλεση. Σημειώνεται πως μια **πολύ** μεγάλη ουρά θα βοηθούσε στο να μη δημιουργείται συμφόρηση, **αλλά** ο μέσος χρόνος αναμονής θα αυξανόταν. Καλύτερη λύση θα ήταν η αύξηση των εργατών – consumers. Επομένως μια ιδανική ουρά θα πρέπει να μη γεμίζει αλλά να είναι και αρκετά μικρή για έγκαιρη έναρξη ακόμα και του τελευταίου task της ουράς.

Ο χρόνος που ξοδεύουμε για την εκτέλεση της κάθε **Timer→Fcn** , δηλαδή ο χρόνος που απασχολείται κάθε consumer μπορεί να επηρεάσει σημαντικά το latency εκτέλεσης των συναρτήσεων. Αν η πολυπλοκότητα των συναρτήσεων είναι τέτοια ώστε **όλοι** οι consumers να μένουν απασχολημένοι καθώς προστίθενται συνεχώς νέα tasks στην ουρά, τότε υπάρχει πάλι ο κίνδυνος αυτή να γεμίσει. Σαφώς σε τέτοιες περιπτώσεις η πιο εύκολη μέθοδος επίλυσης είναι η αύξηση του αριθμού των consumers ή αν δε ξεπερνάμε τα όρια της έγκαιρης εκτέλεσης η αύξηση του μεγέθους της ουράς.

Όμοια με τα προηγούμενα καταλαβαίνουμε πως η περίοδος που έχει ο κάθε **Timer** δηλαδή ο ρυθμός εισόδου των tasks στην ουρά είναι ένας παράγοντας που πρέπει να λάβουμε υπόψη μας. Αν ο ρυθμός πρέπει να αυξηθεί, θα πρέπει και να αντισταθμιστεί με κάποια από τις άλλες παραμέτρους (πχ αύξηση consumers , μεγέθυνση ουράς).

Στη δική μας περίπτωση, οι συναρτήσεις **Timer→Fcn** είναι πολύ απλές και δεν απασχολούν για σημαντικό χρόνο τους consumers. Επομένως μια ουρά **5** θέσεων κρίνεται αρκετή. Με ουρά 5 θέσεων, πραγματοποιήθηκαν μετρήσεις για τους χρόνους αναμονής των task αλλάζοντας κάθε φορά τον αριθμό των **consumers**.



Παρατηρήθηκε πως η ουρά δε γέμιζε ποτέ με **10 consumers** ενώ περαιτέρω αύξηση του αριθμού των consumers δεν προσέφερε ιδιαίτερα πλεονεκτήματα. Επομένως, δεδομένων των συναρτήσεων που έχουμε και των περιόδων των Timers, μπορούμε να πούμε πως μια ουρά **τουλάχιστον** 5 θέσεων μαζί με **10** και άνω consumers είναι ικανοποιητικά χαρακτηριστικά του συστήματος για να επιτύχουμε έγκυρη εκκίνηση εκτέλεσης και λειτουργία πραγματικού χρόνου.