Ενσωματωμένα Συστήματα Πραγματικού Χρόνου

Απαλλακτική Εργασία 2020

Υλοποίηση Timer στη C

Αμοιρίδης Βασίλειος 8772

Θεσσαλονίκη, Δεκέμβριος 2020

Table of Contents

vamoirid@auth.gr Βασίλειος Αμοιρίδης 8772

1. Ανάλυση του προβλήματος

Το πρόβλημα της εργασίας που χρειάζεται να επιλυθεί, είναι η σχεδίαση ενός Timer για ένα Real-Time Embedded System περιβάλλον. Αυτός ο Timer θα πρέπει να χρησιμοποιεί διαφορετικά threads με σκοπό την εισαγωγή tasks σε μια κοινή ουρά πολλών tasks τα οποία περιμένουν την εκτέλεση τους. Πιο συγκεκριμένα, ο κάθε Timer θα πρέπει να τοποθετεί tasks μέσα στην ουρά με μια συγκεκριμένο περίοδο. Οι consumers είναι υπεύθυνοι για να παίρνουν αυτά τα tasks από την ουρά και να τα εκτελούν. Το πρόγραμμα μοιάζει με την αρχική έκδοση της εργασίας με τη διαφορά ότι πλέον τα tasks τοποθετούνται στην ουρά με συγκεκριμένη περίοδο και από συγκεκριμένο producer.

2. Προσθήκες στο πρόγραμμα

Όπως ζητήθηκε από την εκφώνηση δημιουργήθηκε μια δομή timer η οποία περιέχει μέσα όλα όσα ζητάει η εκφώνηση. Παράλληλα δημιουργήθηκε μια ακόμα δομή producerDataType οποία περιέχει έναν timer και ένα queue έτσι ώστε να τα συνδέσει χωρίς να χρειαστεί να βάλει καινούρια μεταβλητή μέσα στη δομή του timer.

3. Τρόπος λειτουργίας

Για την αρχικοποίηση όλων των παραμέτρων της δομής του timer χρησιμοποιείται μια καινούρια συνάρτηση με όνομα timerInit() η οποία είναι υπεύθυνη για την αρχικοποίηση του timer. Οι συναρτήσεις startFcn(), stopFcn(), errorFcn() καλούνται αντίστοιχα στην εκκίνηση στο τέλος και σε περίπτωση σφάλματος κατά τη διάρκεια της εκτέλεσης του producer.

4. Διόρθωση timedrift

Ο τρόπος με τον οποίο επιτυγχάνεται η περιοδική τοποθέτηση εργασιών μέσα στην ουρά είναι η χρήση της συνάρτησης usleep(). Με τη βοήθεια της συνάρτησης αυτής, το κάθε νήμα producer «κοιμάται» για ένα χρονικό διάστημα (περίοδος) και «ξυπνάει» με σκοπό να βάλει κάτι καινούριο μέσα στη λίστα. Σύμφωνα όμως με το man page της συνάρτησης usleep() αναγράφεται το εξής: "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 συνάρτησης usleep() προσφέρει μια απροσδιοριστία στο σύστημα η οποία σε συνδυασμό με το χρόνο εκτέλεσης των εντολών και τον χρόνο για τοποθέτηση ενός task στην ουρά καταλήγει να μην έχει σταθερό χρονικό διάστημα στην περίοδο.

Για τη διόρθωση τώρα του συγκεκριμένου προβλήματος χρησιμοποιήθηκε η εξής μέθοδος. Αρχικά ορίζεται ένα σημείο μέσα στη συνάρτηση το οποίο είναι το timestamp της κάθε περιόδου. Έτσι, χρησιμοποιώντας την μέτρηση του χρόνου σε 2 διαδοχικές περιόδους, είναι δυνατό να βρεθεί ο συνολικός χρόνος που πέρασε από την προηγούμενη εκτέλεση. Αν δηλαδή η περίοδος ήταν 100ms και από λάθος περίμενε 105ms τότε η καινούρια usleep() θα δεχτεί τιμή 100 - (105 – 100) = 95ms. Παρόλα αυτά αν και η επόμενη περίοδος έχει πάλι σφάλμα 98ms τότε αυτό σημαίνει ότι η καινούρια περίοδος θα έχει 95 – (98 – 100) = 97ms. Στην ουσία δηλαδή αυτό

vamoirid@auth.gr Βασίλειος Αμοιρίδης 8772

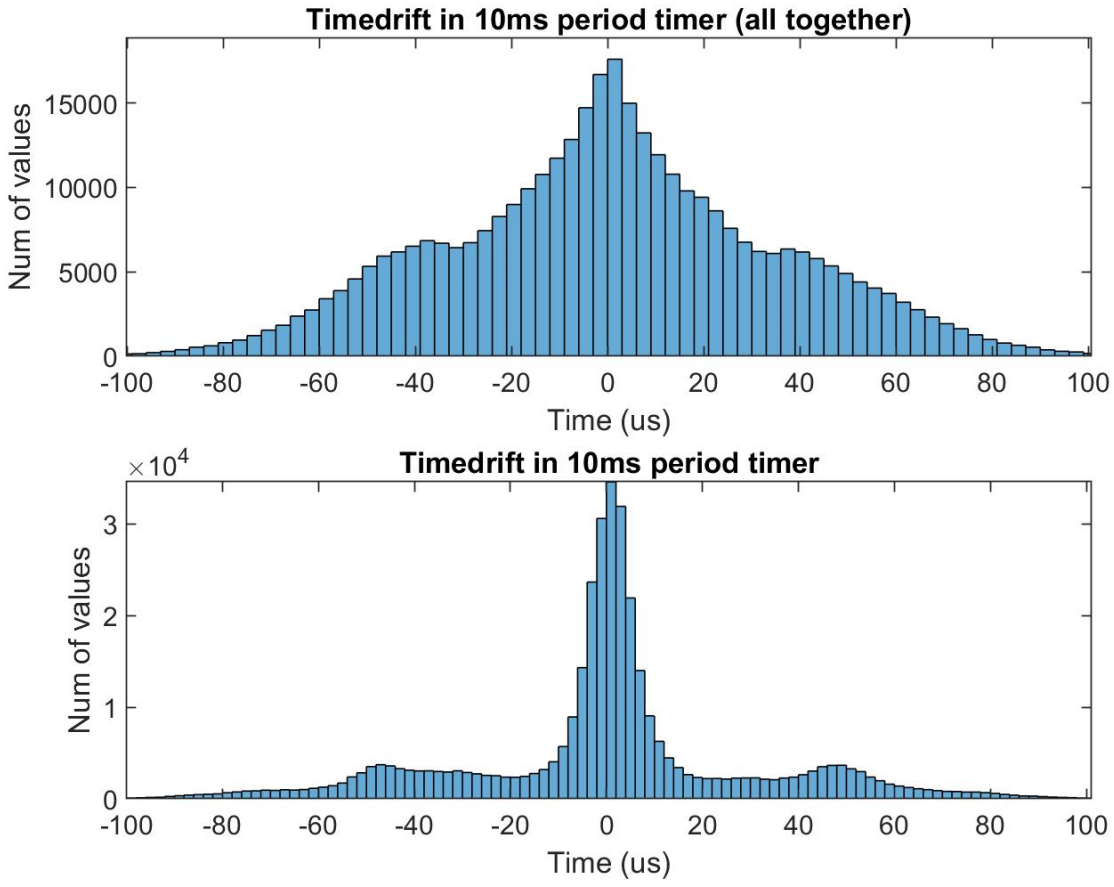
που συμβαίνει είναι ότι δεν αφαιρείται ο χρόνος από τη σταθερή τιμή της περιόδου, αλλά αφαιρείται από τη διορθωμένη τιμή της προηγούμενης περιόδου. Στην ουσία λοιπόν αυτή η μέθοδος θα μπορούσε κανείς να πει ότι μοιάζει με τον έλεγχο ενός feedback control loop. Πράγματι λοιπόν τα παρακάτω αποτελέσματα αποδεικνύουν ότι η συγκεκριμένη μέθοδος διόρθωσης του timedrift αποφέρει πολύ καλά αποτελέσματα.

5. Πειράματα

Για το πειράματα χρησιμοποιήθηκαν συνολικά 4 consumers γιατί όπως θα αναφερθεί και στη συνέχεια, το είδος των συναρτήσεων σε συνδυασμό με τη CPU του Raspberry Pi 4 δεν επιτρέπουν την βελτίωση με μεγαλύτερο αριθμό από consumers.

5.1 Timer 10ms Period

Ο συγκεκριμένος timer έχει περίοδο 10ms δηλαδή εκτελείται συνολικά 100 φορές το δευτερόλεπτο και 360.000 φορές για μια ολόκληρη ώρα πειράματος. Τα στοιχεία που προκύπτουν από την απομονωμένη εκτέλεση και από την εκτέλεση παράλληλα με τους άλλους 2 είναι τα εξής.



Εικόνα 1. Αποτελέσματα των εκτελέσεων του Timer με περίοδο 10ms.

vamoirid@auth.gr Βασίλειος Αμοιρίδης 8772

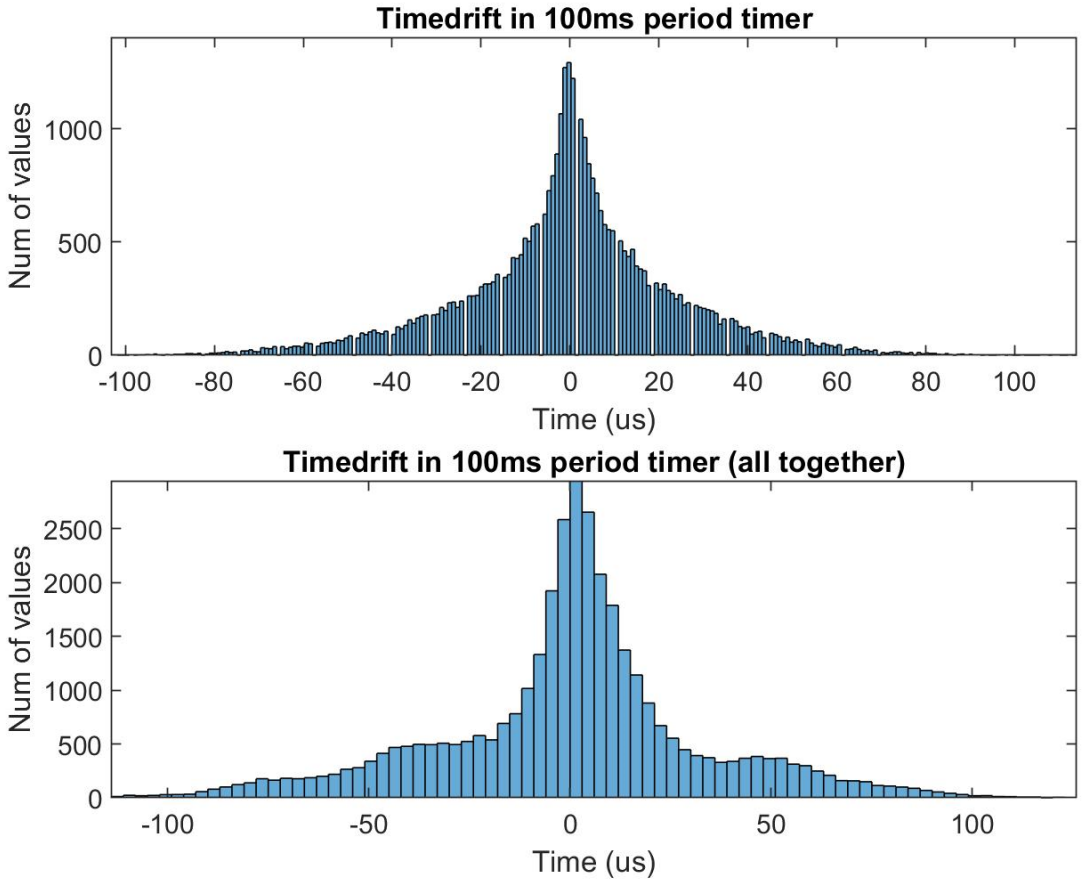
Timer 10ms	**Μέση Τιμή**	**Τυπ. Απόκλιση**	**Ελάχιστη**	**Διάμεση**	**Μέγιστη**	**Timedrift**
Απομονωμένη	22	40,51	-3779	0	3738	146
Όλοι μαζί	34	72,89	-6220	0	6163	172

**Η μέση τιμή υπολογίστηκε αφού προστέθηκαν οι απόλυτες τιμές των μετρήσεων*
**Όλες οι τιμές είναι μετρημένες σε us.*

Σε ένα μικρό ποσοστό των μετρήσεων, ~0.1% για την απομονωμένη εκτέλεση και ~1% για την εκτέλεση όλων μαζί, υπήρχαν κάποια «καρφιά» στις μετρήσεις, όπως επαληθεύεται και από τον πίνακα, τα οποία φτάνανε άλλοτε μέχρι και τα 6000us. Το φαινόμενο αυτό θα εξηγηθεί παρακάτω.

5.2 Timer 100ms Period

Ο συγκεκριμένος timer έχει περίοδο 100ms δηλαδή εκτελείται συνολικά 10 φορές το δευτερόλεπτο και 36.000 φορές για μια ολόκληρη ώρα πειράματος. Τα στοιχεία που προκύπτουν από την απομονωμένη εκτέλεση και από την εκτέλεση παράλληλα με τους άλλους 2 είναι τα εξής.



Εικόνα 2. Αποτελέσματα των εκτελέσεων του Timer με περίοδο 100ms.

vamoirid@auth.gr Βασίλειος Αμοιρίδης 8772

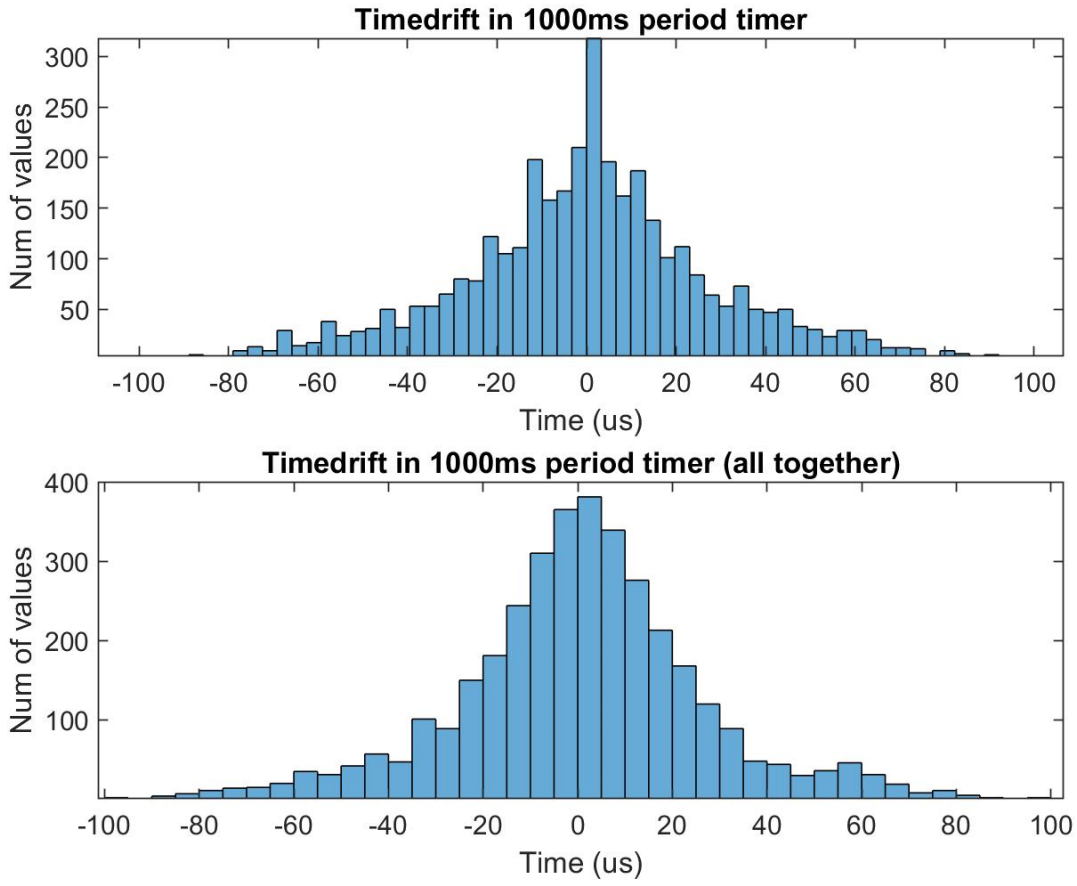
Timer 100ms	**Μέση Τιμή**	**Τυπ. Απόκλιση**	**Ελάχιστη**	**Διάμεση**	**Μέγιστη**	**Timedrift**
Απομονωμένη	32	33,89	-215	0	239	132
Όλοι μαζί	44	52,35	-768	1	742	167

**Η μέση τιμή υπολογίστηκε αφού προστέθηκαν οι απόλυτες τιμές των μετρήσεων*
**Όλες οι τιμές είναι μετρημένες σε us.*

Σε ένα μικρό ποσοστό των μετρήσεων, ~0.2% για την απομονωμένη εκτέλεση και ~1% για την εκτέλεση όλων μαζί, υπήρχαν κάποια «καρφιά» στις μετρήσεις, όπως επαληθεύεται και από τον πίνακα, τα οποία φτάνανε άλλοτε μέχρι και τα 750us. Το φαινόμενο αυτό θα εξηγηθεί παρακάτω.

5.3 Timer 1000ms Period

Ο συγκεκριμένος timer έχει περίοδο 1000ms δηλαδή εκτελείται συνολικά κάθε 1 δευτερόλεπτο και 3.600 φορές για μια ολόκληρη ώρα πειράματος. Τα στοιχεία που προκύπτουν από την απομονωμένη εκτέλεση και από την εκτέλεση παράλληλα με τους άλλους 2 είναι τα εξής.



Εικόνα 3. Αποτελέσματα των εκτελέσεων του Timer με περίοδο 1000ms.

vamoirid@auth.gr Βασίλειος Αμοιρίδης 8772

Timer 1000ms	**Μέση Τιμή**	**Τυπ. Απόκλιση**	**Ελάχιστη**	**Διάμεση**	**Μέγιστη**	**Timedrift**
Απομονωμένη	18	26,62	-116	0	182	156
Όλοι μαζί	25	75,78	-1487	1	1804	183

**Η μέση τιμή υπολογίστηκε αφού προστέθηκαν οι απόλυτες τιμές των μετρήσεων*
**Όλες οι τιμές είναι μετρημένες σε us.*

Σε ένα μικρό ποσοστό των μετρήσεων, ~1% για την εκτέλεση όλων μαζί, υπήρχαν κάποια «καρφιά» στις μετρήσεις, όπως επαληθεύεται και από τον πίνακα, τα οποία φτάνανε άλλοτε μέχρι και τα 1800us. Το φαινόμενο αυτό θα εξηγηθεί παρακάτω.

5.4 Σχολιασμός των timedrift των πειραμάτων

Όπως ήταν προφανές, οι στατιστικές του χρόνου παρουσιάζουν πολύ πιο γρήγορες τιμές στην απομονωμένη εκτέλεση παρά στη συνολική. Σε όλες τις περιπτώσεις υπήρχε χαμηλότερη μέση τιμή των αποτελεσμάτων στην απομονωμένη εκτέλεση ενώ αντίστοιχα η τυπική απόκλιση ήταν μικρότερη πάλι στην απομονωμένη εκτέλεση.

Αυτό που φαίνεται να έχει ιδιαίτερο ενδιαφέρον είναι η τιμή της τυπικής απόκλισης σε συνδυασμό με τις μέγιστες/ελάχιστες τιμές και επίσης το πολύ μεγαλύτερο ποσοστό % των «καρφιών» που υπήρχε στην εκτέλεση όλων μαζί σε σχέση με την απομονωμένη. Ο λόγος που συμβαίνει το συγκεκριμένο γεγονός έγινε κατανοητός όταν παρατηρήθηκαν τα timeseries των μετρήσεων και παρατηρήθηκε μια περιοδικότητα στην εμφάνιση των καρφιών. Αυτή η περιοδικότητα συμπίπτει με την περιοδικότητα που έχουν οι timers μεταξύ τους. Έτσι λοιπόν γίνεται κατανοητό ότι κατά ένα πολύ μεγάλο ποσοστό τα συγκεκριμένα καρφιά οφείλονται στο γεγονός ότι οι κάθε 10 περίοδοι του 10ms Timer είναι 1 περίοδος του 100ms Timer και αντίστοιχα κάθε 10 περίοδοι του 100ms Timer είναι 1 περίοδος του 1000ms timer. Επίσης ένας ακόμα σημαντικός παράγοντας είναι ο scheduler και ο τρόπος με τον οποίο επιλέγει να κατανείμει τα tasks ανάλογα με τις ανάγκες του συστήματος την προκειμένη στιγμή.

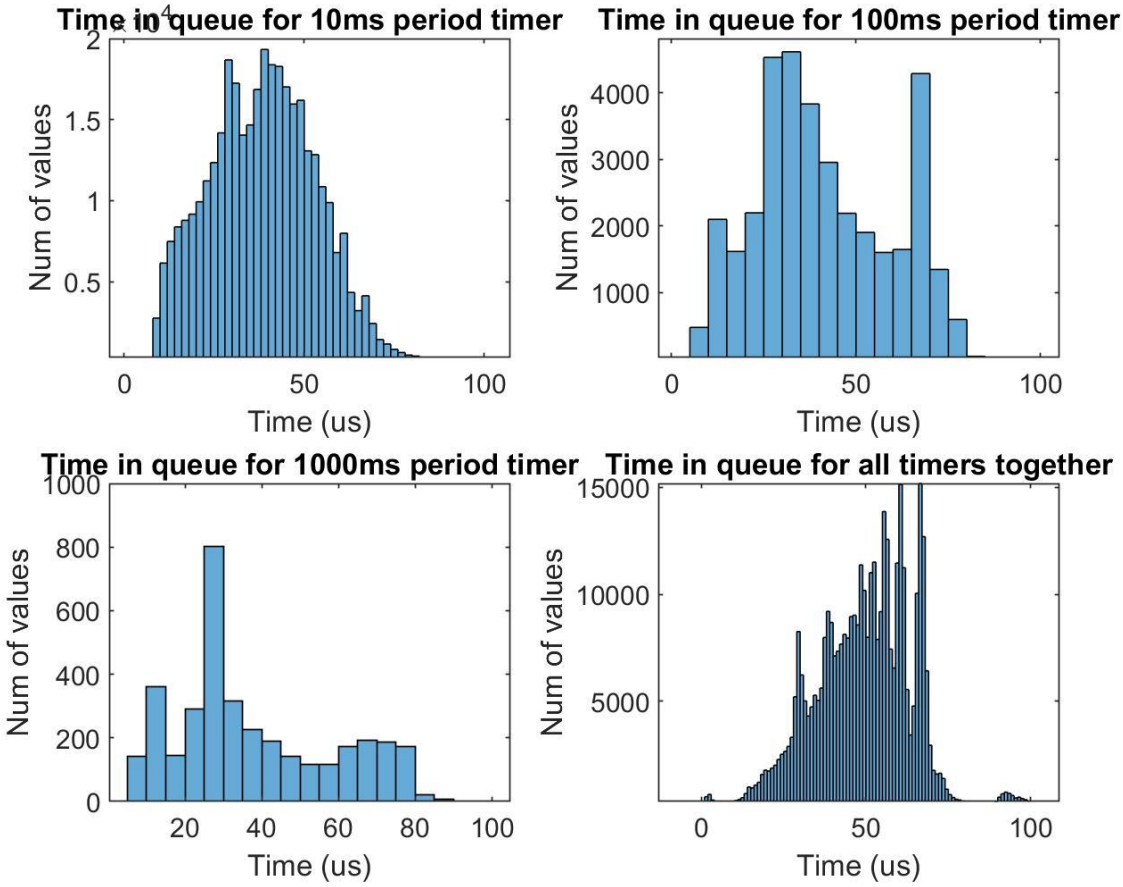
Ιδιαίτερο ενδιαφέρον προκαλεί το γεγονός ότι στην περίπτωση του Timer με περίοδο 1000ms δεν παρατηρήθηκε κανένα καρφί κατά την απομονωμένη εκτέλεση. Η συγκεκριμένη εκτέλεση περιείχε 3600 μετρήσεις σε αντίθεση με τις 360000 που είχε αυτή με τον Timer 10ms. Ακόμα μεγαλύτερο ενδιαφέρον παρουσιάζει το γεγονός ότι το συνολικό timedrift του προγράμματος δεν ξεπέρασε τα 200us. Σε όλες τις περιπτώσεις δηλαδή, ο τρόπος που χρησιμοποιήσαμε για τη διόρθωση επέτρεψε μετά από 1 ολόκληρη ώρα πειραμάτων το συνολικό timedrift να είναι κάτω από τα 200us δηλαδή να έγκειται και στην ακρίβεια της usleep.

5.5 Χρόνος αναμονής στην ουρά

Οι χρόνοι αναμονής στην ουρά είναι κάτι το οποίο σίγουρα μπορεί να διαφοροποιηθεί από πείραμα σε πείραμα. Παρόλα αυτά στην περίπτωση που επιλέχθηκε, ένα πρόγραμμα δηλαδή που υπολογίζει μονάχα ημίτονα και συνημίτονα από γωνίες κάτι τέτοιο δεν θα έπρεπε να έχει

vamoirid@auth.gr Βασίλειος Αμοιρίδης 8772

και μεγάλη διαφοροποίηση στα 3 διαφορετικά πειράματα. Η μόνη περίπτωση που θα μπορούσε να υπάρχει διαφοροποίηση είναι αυτή του 4ου πειράματος στο οποίο πλέον ο φόρτος αυξάνεται περαιτέρω και υπάρχουν παραπάνω tasks τα οποία θα πρέπει να εκτελεσθούν.



Εικόνα 4. Χρόνοι αναμονής στην ουρά, για κάθε ένα από τα 4 πειράματα.

QUEUETIMES (us)	Μέση τιμή	Τυπ. Απόκλιση	Ελάχιστη	Διάμεση	Μέγιστη
Period 1000ms	37,45	20,36	7	30	99
Period 100ms	41,93	43,03	6	38	1888
Period 10ms	38,83	38,69	4	38	4500
All together	49,50	21,35	2	50	3150

**Όλες οι τιμές είναι μετρημένες σε us.*

Τα παραπάνω διαγράμματα σε συνδυασμό με τα στατιστικά στοιχεία στον πίνακα δείχνουν χαρακτηριστικά ότι η μέση τιμή επηρεάζεται όταν υπάρχουν πολλές εργασίες ταυτόχρονα. Αυτό προφανώς είναι και αποτέλεσμα του ότι οι 3 timers χρησιμοποιούν πόρους από τη CPU τους οποίους πόρους εκείνη τη στιγμή δεν μπορεί να χρησιμοποιήσει κάποιος consumer για να μπορέσει να εκτελέσει τη συνάρτηση που του ανατέθηκε. Αξιοσημείωτο είναι πως ακόμα και στην περίπτωση του timer με περίοδο 10ms η μέση τιμή που του χρόνου αναμονής για ένα task είναι στα 38,83us δηλαδή 0,38% της περιόδου.

vamoirid@auth.gr Βασίλειος Αμοιρίδης 8772

6. Σχολιασμός μεγεθών

Το μέγεθος της ουράς είναι κάτι που επηρεάζει άμεσα τον τρόπο λειτουργίας του συστήματος. Αυτό γιατί πολύ απλά μια πολύ μικρή ουρά η οποία γεμίζει πιο γρήγορα από όσο αδειάζει, προφανώς και θα εμποδίσει τους timers να τοποθετούν tasks και να «χάνονται» tasks που πρέπει να γίνουν. Αυτό είναι κάτι που ανάλογα με τη χρήση του Ενσωματωμένου Συστήματος, μπορεί να αποβεί μοιραίο. Παρόλα αυτά η αλόγιστη αύξηση των θέσεων της ουράς δεν συνεπάγεται απαραίτητα και με την σωστή χρονική εκτέλεση των εργασιών. Αυτό εξαρτάται επίσης και από το είδος των tasks και από τον αριθμό των consumers.

Το συγκεκριμένο πρόβλημα αναλύθηκε περισσότερο στην πρώτη εργασία από την οποία παράχθηκαν τα εξής συμπεράσματα:

- Αν το πρόγραμμα τρέχει μονάχα με συναρτήσεις CPU-bound, οι οποίες δεν ξοδεύουν χρόνο στο να περιμένουν κάποιο I/O να τους απαντήσει, τότε ο μέγιστος αριθμός consumers που επιφέρουν αποτέλεσμα είναι ίσος με τον μέγιστο αριθμό threads του επεξεργαστή.
- Αν στο πρόγραμμα υπάρχουν και συναρτήσεις I/O-bound, οι οποίες περιμένουν απαντήσεις από περιφερειακά και είναι ένα πολύ συνηθισμένο είδος συναρτήσεων για Ενσωματωμένα Συστήματα Πραγματικού Χρόνου, τότε η περαιτέρω αύξηση των consumers πέρα από τον αριθμό των threads του επεξεργαστή μπορεί να επιφέρει αύξηση στην ταχύτητα.

Στην περίπτωση του προγράμματος και οι 3 εργασίες που αναλαμβάνουν να τοποθετήσουν στη λίστα οι timers, υπολογίζουν μονάχα το ημίτονο/συνημίτονο/εφαπτομένη μιας γωνίας πράγμα που σημαίνει πως ο αριθμός δεν μπορεί να ξεπερνάει τον αριθμό των νημάτων. Με όλα τα παραπάνω δεδομένα κρίθηκαν αποτελεσματικοί οι αριθμοί n=10 για το queue και n=4 για τους consumers.

Τα tasks όμως έχουν ακόμα έναν παράγοντα ο οποίος μπορεί να επηρεάσει σημαντικά την απόκριση του προγράμματος. Αυτός είναι ο χρόνος που χρειάζεται συνολικά για την εκτέλεση τους. Αν λοιπόν τα tasks χρειάζονται για να εκτελεστούν έναν χρόνο εκτέλεσης συγκρίσιμο με τον χρόνο της περιόδου τότε το σύστημα κινδυνεύει να γεμίσει. Κινδυνεύει δηλαδή, όλοι οι consumers να είναι απασχολημένοι με την εκτέλεση των tasks τους και να μην είναι δυνατόν να ξεκινήσουν τα επόμενα tasks στον χρόνο που πρέπει. Σε μια εφαρμογή όπου θα υπήρχαν πολλοί επεξεργαστές και πολλά threads για να αναλάβουν τέτοιες δουλειές, μια απλή αύξηση του αριθμού των consumers θα μπορούσε να λύσει το συγκεκριμένο πρόβλημα, πάλι όμως υπό την προϋπόθεση ότι ο χρόνος εκτέλεσης αν και μεγαλύτερος της περιόδου, θα έπρεπε να είναι σε συγκρίσιμα μεγέθη με τον αριθμό των threads. Στην περίπτωση του Raspberry Pi 4 όμως, το οποίο έχει μια 4-core CPU μόνο αυτό είναι μια λύση η οποία δεν είναι δυνατή. Έτσι λοιπόν, το κάθε task θα πρέπει να μπορεί να εκτελείται σε χρόνο μικρότερο από αυτόν της περιόδου του, έτσι ώστε να μπορεί να μην «γεμίζει» το σύστημα.

vamoirid@auth.gr Βασίλειος Αμοιρίδης 8772