ΖΗΚΟΠΗΣ ΕΥΑΓΓΕΛΟΣ
ΑΕΜ: 8808

ΕΝΣΩΜΑΤΩΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ

1η Άσκηση - Υλοποίηση μετρητή δειγματοληψίας

Φοιτητής Ζηκόπης Ευάγγελος(ΑΕΜ: 8808)

Περιγραφή Άσκησης

Σκοπός της παρούσας άσκησης ήταν η υλοποίηση ενός μετρητή που θα χρησιμοποιείται προκειμένου να δειγματοληπτούμε ανα κάποιο χρονικό διάστημα dt. Στόχος ήταν η ελαχιστοποίηση του σφάλματος στην απόσταση που μεσολαβεί μεταξύ δύο δειγμάτων. Επίσης στόχος ήταν η ελαχιστοποίηση της κατανάλωσης ενέργειας, με απώτερο σκοπό το πρόγραμμα του μετρητή να τρέξει σε κάποια ενσωματωμένη συσκευή. Πραγματοποιήθηκαν δύο υλοποιήσεις, μια που δε χρησιμοποιεί τα timestamps με σκοπό κάποια βελτίωση και μια που τα χρησιμοποιεί.

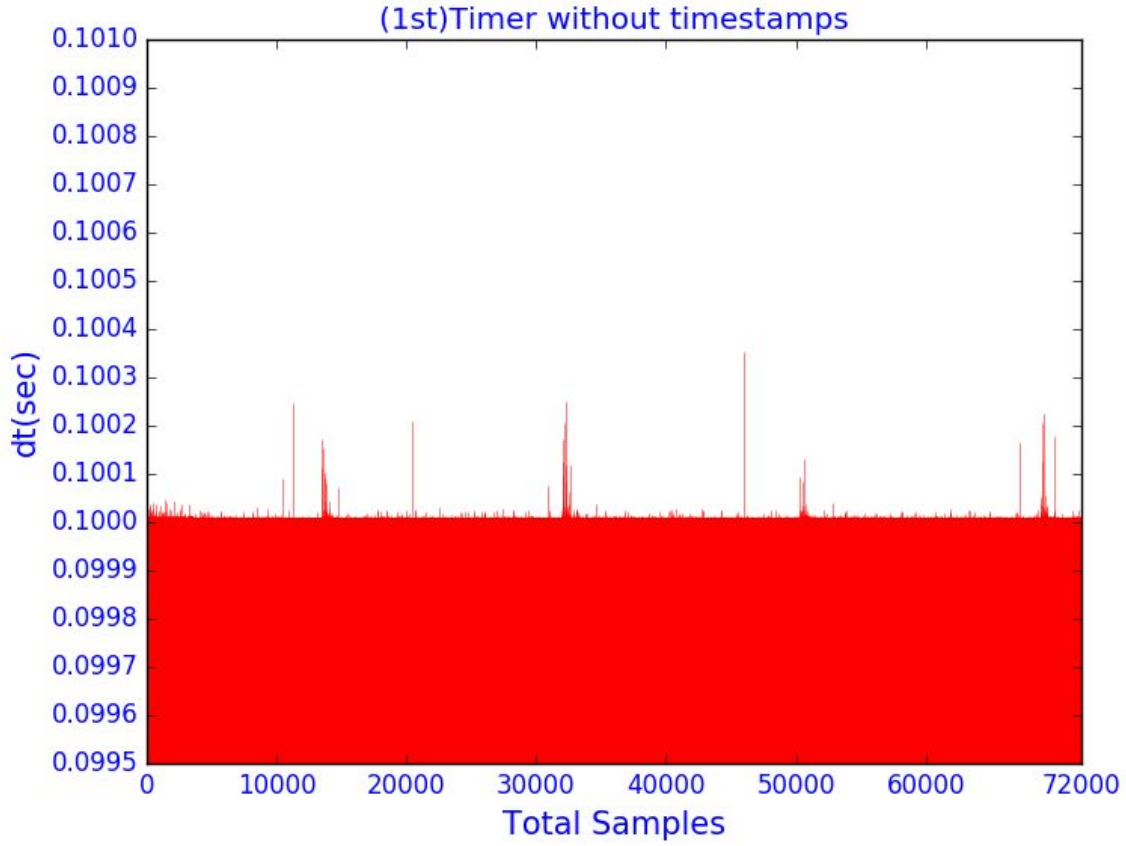
Πρώτη Υλοποίηση

Στην πρώτη υλοποίηση δεν μπορούμε να εκμεταλλευτούμε τις τιμές που επιστρέφει η συνάρτηση gettimeofday(). Αρχικά προσπάθησα να το προσεγγίσω με τη sleep(), ώστε η συσκευή να είναι σε idle mode όσο δε δειγματοληπτεί. Αυτό όμως ήταν αδύνατο, καθώς εξαιτίας του σφάλματος που εισάγει η usleep(), οι ενδιάμεσοι χρόνοι είχαν μεγάλη απόκλιση από το 0.1 second , κι έτσι καταλήξαμε να χάνουμε πολλές μετρήσεις στο διάστημα των 7200 second. Επομένως, αφού δεν μπόρεσα να το υλοποιήσω με τη χρήση της sleep, κατέληξα στο συμπέρασμα πως η πιο ακριβής μέθοδος είναι με τη χρήση της ualarm(). Με αυτόν τον τρόπο το σφάλμα είναι πολύ μικρό. Στο πείραμα των 7200 seconds, μέσω αυτής της υλοποίησης, δεδομένου πως δεν υπάρχει κάποιο άλλο alarm ώστε να διακόψει τη λειτουργία του ακριβώς στα 7200 seconds, η διαδικασία της δειγματοληψίας διήρκησε συνολικά 7201 seconds. Άρα βλέπουμε πως ξεφεύγουμε από το όριο των 7200 seconds κατα 1 second.
Στο πρόγραμμα, αρχικά λαμβάνουμε από τον χρήστη τον συνολικό χρόνο δειγματοληψίας και το χρόνο του ενδιάμεσου διαστήματος και δεσμεύουμε χώρο σε έναν πίνακα για να αποθηκεύσουμε τις μετρήσεις. Έπειτα μπαίνουμε σε μια επανάληψη που θα διαρκέσει μέχρι να γεμίσει ο πίνακας που αποθηκεύουμε τα timestamps. Εκεί, δειγματοληπτούμε και έπειτα ξεκινάμε ένα ualarm() που θα μας ενημερώσει ώστε να πάρουμε το επόμενο δείγμα. Τέλος, δημιουργούμε ένα αρχείο που θα αποθηκεύσει τα δείγματα.

Παρακάτω παρουσιάζονται οι στατιστικές μετρήσεις και το γράφημα της πρώτης υλοποίησης.

- ➔ *Μέσος Όρος*: 0.100005514125
- ➔ *Ελάχιστη Τιμή*: 0.100002
- ➔ *Μέγιστη Τιμή*: 0.100352
- ➔ *Διάμεσος*: 0.100005
- ➔ *Τυπική Απόκλιση*: 4.028595073e-06

ΖΗΚΟΠΗΣ ΕΥΑΓΓΕΛΟΣ
ΑΕΜ: 8808



🔎 Παρατηρήσεις/Συμπεράσματα: Από τις στατιστικές τιμές παρατηρούμε πως ο μέσος όρος αλλά και η διάμεσος είναι πολύ κοντά στην επιθυμητή τιμή του 0.1 second και το σφάλμα κυμαίνεται σε τιμές της τάξης των $(45-55)*10^{-6}$ second. Επίσης από την τυπική απόκλιση εξάγουμε το συμπέρασμα πως οι διακυμάνσεις των τιμών είναι πολύ μικρές. Άρα, μπορούμε να καταλήξουμε στο συμπέρασμα πως η συγκεκριμένη υλοποίηση παρέχει μεγάλη ακρίβεια ως προς το χρόνο, αλλά είναι περισσότερο κοστοβόρα ενεργειακά.

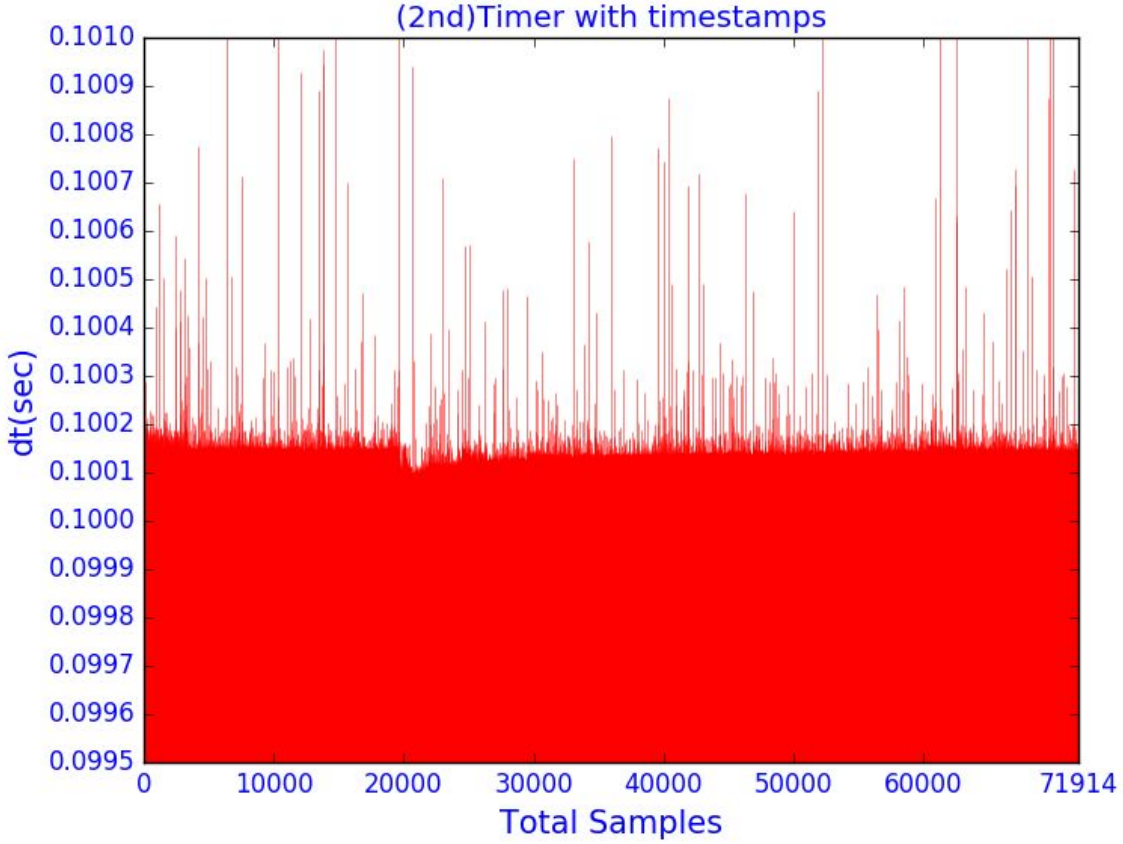
Δεύτερη Υλοποίηση

Στη δεύτερη υλοποίηση έχουμε τη δυνατότητα να εκμεταλλευτούμε τα timestamps που επιστρέφει η συνάρτηση gettimeofday(). Με αυτόν τον τρόπο μπορούμε να εισάγουμε στο πρόγραμμά μας την sleep(), έτσι ώστε να κρατάμε χαμηλά τη χρήση της CPU και έτσι να κυμαινόμαστε σε χαμηλές τιμές κατανάλωσης ενέργειας.
Σε αυτήν την υλοποίηση, τρέχει ένα alarm για 7200 second. Αυτό το alarm θα μας ειδοποιήσει ώστε να σταματήσουμε τη δειγματοληψία. Σε αυτό το διάστημα, δειγματοληπτούμε τα timestamps τα οποία αποθηκεύουμε σε πίνακα και έπειτα καλούμε την usleep() ώστε να περάσουμε σε idle mode. Από αυτές τις τιμές, υπολογίζουμε πόσο διαφέρουν οι διαδοχικές μετρήσεις και έτσι υπολογίζουμε σε κάθε στιγμή δειγματοληψίας εναν μέσο όρο του σφάλματος (δηλαδή πόσο πάνω ή κάτω είμαστε απο την επιθυμητή διαφορά του 0.1 second). Αυτόν τον μέσο όρο τον αφαιρούμε από τα usecond που βάζουμε στη συνάρτηση usleep. Έτσι μπορούμε να θέτουμε δυναμικά το πόσο θα "κοιμάται" η CPU,

και να είμαστε αρκετά κοντά στο επιθυμητό χρονικό διάστημα του 0.1 second. Αυτή η υλοποίηση διήρκησε 7200.03 second και οι μετρήσεις που πήραμε ήταν 71916, δηλαδή χάσαμε συνολικά 84 μετρήσεις, ποσοστό αρκετά χαμηλό αν αναλογιστούμε τον συνολικό αριθμό των δειγμάτων.
Στο πρόγραμμα αυτής της υλοποίησης, ο χρήστης εισάγει το συνολικό χρόνο δειγματοληψίας και το διάστημα που θα παρεμβάλλεται μεταξύ των δειγμάτων. Έπειτα δημιουργείται ο πίνακας που θα αποθηκεύει τα δείγματα και ξεκινάει το alarm που θα τρέξει για 7200 second. Στη συνέχεια δειγματοληπτούμε, υπολογίζουμε τον μέσο όρο του σφάλματος και ξεκινάμε τη usleep(). Αυτή η διαδικασία συνεχίζεται μέχρι να λήξει το alarm. Τέλος μεταφέρουμε τις μετρήσεις σε ένα αρχείο ώστε να τις αποθηκεύσουμε.

Παρακάτω παρουσιάζονται οι στατιστικές μετρήσεις και το γράφημα της δεύτερης υλοποίησης.

- ➔ *Μέσος Όρος*: 0.100133045805
- ➔ *Ελάχιστη Τιμή*: 0.09993
- ➔ *Μέγιστη Τιμή*: 0.982552
- ➔ *Διάμεσος*: 0.100132
- ➔ *Τυπική Απόκλιση*: 0.00329083762821



🔎 Παρατηρήσεις/Συμπεράσματα: Από τις στατιστικές τιμές είναι εύκολο να παρατηρήσουμε πως το σφάλμα της δεύτερης υλοποίησης είναι εμφανώς μεγαλύτερο από αυτό της πρώτης. Επίσης, η ελάχιστη τιμή βλέπουμε πως είναι μικρότερη από 0.1 second γεγονός που σημαίνει πως σε εκείνη τη στιγμή η CPU "κοιμήθηκε" λιγότερο από όσο έπρεπε. Επίσης, από την

τυπική απόκλιση αλλά και από το γράφημα βλέπουμε πως οι διακυμάνσεις είναι περισσότερες από αυτές της πρώτης υλοποίησης. Άρα μπορούμε να συμπεράνουμε πως αυτή η υλοποίηση, δηλαδή ο δυναμικός καθορισμός του χρόνο που δίνουμε στη usleep(),έχει αυξημένο σφάλμα και χάνει κάποιες μετρήσεις (84 στις 72000), αλλά μειώνει αρκετά τη χρήση της CPU απο το πρόγραμμα.