Σταύρος Μαλακούδης samalako@ece.auth.gr 9368

Τελική εργασία

Ενσωματωμένα Συστήματα Πραγματικού Χρόνου

- link για το github

Τρόπος Υλοποίησης

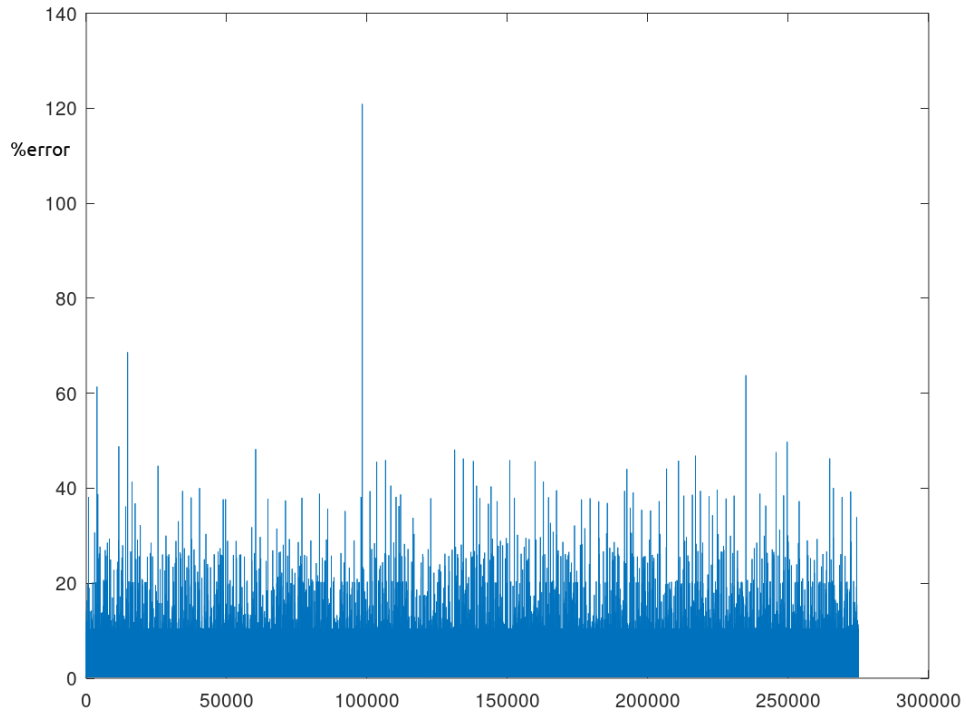
Ο κώδικας χρησιμοποιεί Pthreads για την εκτέλεση των functions που διαχειρίζονται τα Mac Addresses. Ένα thread (loader) κάθε 10 δευτερόλεπτα επιστρέφει ένα MacAddress (από το αρχείο Addresses.bin) και ελέγχει αν υπάρχει ήδη στην ουρά με τις πρόσφατες διευθύνσεις .Αν δεν υπάρχει ήδη, το τοποθετεί εκεί, αλλιώς το τοποθετεί στην ουρά με τις κοντινές επαφές. Το άλλο thread (unloader) ελέγχει το πιο παλιό στοιχείο που μπήκε στην ουρά με τις πρόσφατες και το σβήνει αν είναι μέσα σε αυτή πάνω από 20 λεπτά. Επίσης, κάθε 4 ώρες εκτελεί ένα covid test και αποθηκεύει τις στενές επαφές σε ένα binary αρχείο, αν το τεστ είναι θετικό, ενώ κάθε 14 μέρες αρχικοποιεί ξανά τη λίστα των στενών επαφών.Τα Mac Addresses αποθηκεύονται στις ουρές με τη χρήση ενός struct που περιέχει ένα string για τη διεύθυνση και ένα struct timeval για τη μέτρηση του χρόνου.

Ζητούμενα

Η προσομοίωση που αναφέρθηκε, μπορεί να τρέξει 100 φορές πιο γρήγορα, αναθέτοντας στη μεταβλητή QUANTUM (στο αρχείο utilities.h) την τιμή 0.01 . Στην προκειμένη περίπτωση, επιστρέφεται μια νέα τιμή Mac Address κάθε 0.1 δευτερόλεπτο και γενικότερα επιταχύνονται όλες οι διαδικασίες.

V0: έκδοση με polling

Αν μετρήσουμε την καθυστέρηση σε κάθε εκτέλεση της συνάρτησης που επιστρέφει τη διεύθυνση επί 7:30 ώρες, προκύπτει το παρακάτω διάγραμμα:



Σε αυτό το διάγραμμα παρουσιάζεται συναρτήσει του χρόνου το ποσοστιαίο σφάλμα, δηλαδή η καθυστέρηση σε κάθε εκτέλεση. Εξαιτίας του πλήθους των δειγμάτων όμως, δεν προκύπτει κάποιο ιδιαίτερο συμπέρασμα οπτικά. Το μόνο που μπορούμε να παρατηρήσουμε είναι κάποια peaks, τα οποία οφείλονται στις λειτουργίες που συμβαίνουν πιο αραιά, όπως η εγγραφή των κοντινών επαφών σε αρχείο, η επαναφορά των ουρών κτλ. Κάνοντας όμως στατιστική ανάλυση , παρατηρούμε ότι η μέση τιμή του **ποσοστιαίου** σφάλματος έγκειται στο 0.88%, δηλαδή 0.88ms (στα 100ms) το οποίο είναι αρκετά ικανοποιητική τιμή. Το πρόβλημα που έγκειται όμως, είναι η χρήση του επεξεργαστή, η οποία βρίσκεται στο 100% καθ' όλη τη διάρκεια του πειράματος:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1282	pi	20	0	20972	1464	1224	S	98.7	0.3	3:49.30	main
1371	**pi**	**20**	**0**	**7636**	**2644**	**2208**	**R**	**1.0**	**0.6**	**0:00.60**	**top**

(Η διαδικασία είναι η main)

Αυτό συμβαίνει διότι ο έλεγχος για το πόσος χρόνος πέρασε, γίνεται μέσω μιας while λούπας , η οποία συνεχώς περιμένει μέχρι να περάσουν 0.1 δευτερόλεπτα. Παρόλα αυτά, η χρήση της μνήμης παραμένει σταθερά στο 0.5-1% (από τα συνολικά 430MB διαθέσιμα) , δηλαδή 20 -40 Mb (ανάλογα με το μέγεθος των ουρών) Αυτό είναι αρκετά σημαντικό πλεονέκτημα, δεδομένων των περιορισμένων πόρων. Όμως, σε περίπτωση όπου χρειαστεί να εκτελέσει και άλλη λειτουργία η συσκευή παράλληλα, μπορεί να προκύψει πολύ μεγαλύτερη καθυστέρηση στις αποκρίσεις.

Βελτίωση

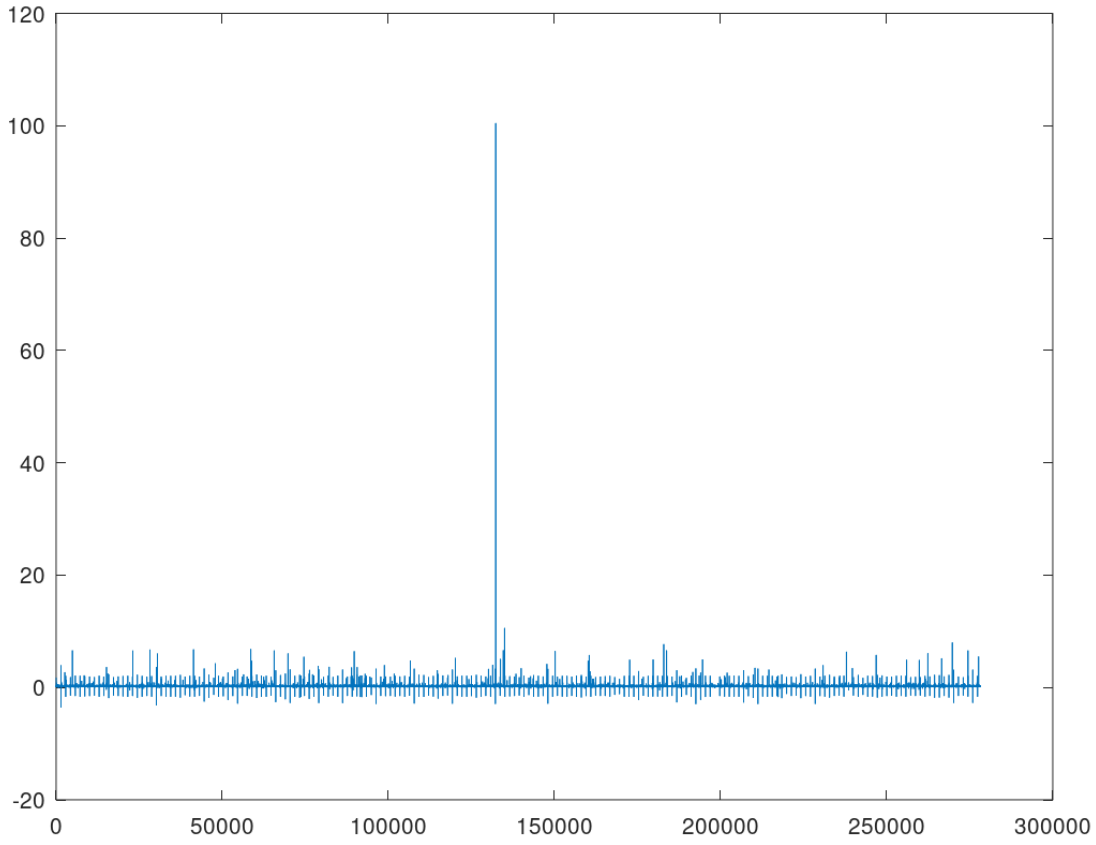
V1: έκδοση με νέο timer thread

Με σκοπό να μειωθεί η χρήση του επεξεργαστή, δημιουργούμε ένα νέο thread του οποίου μοναδική δουλειά είναι κάθε 0.1 δευτερόλεπτα να στέλνει σήμα στα άλλα δύο ώστε να εκτελέσουν τις συναρτήσεις. Γίνεται sleep και μόλις τελειώσει ενημερώνει τα άλλα threads μέσω της pthread_signal ενώ τα άλλα περιμένουν το σήμα μέσω της pthread_cond_wait. Η χρήση του επεξεργαστή φαίνεται παρακάτω:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5561	**pi**	**20**	**0**	**7636**	**2600**	**2156**	**R**	**1.3**	**0.6**	**0:02.99**	**top**
5562	pi	20	0	29168	1500	1284	S	0.7	0.3	0:01.76	v1

(Η διαδικασία είναι η v1)

Αυτό μειώνει δραματικά τη χρήση του CPU από το 99% στο 1%. Η χρήση μνήμης παραμένει ίδια και προφανώς λειτουργικά δεν αλλάξαμε κάτι άλλο, πέρα από τον τρόπο που επιβάλλουμε στις συναρτήσεις να εκτελεστούν συστηματικά κάθε 0.1 δευτερόλεπτα. Μάλιστα, η μέση τιμή του ποσοστιαίου σφάλματος πέφτει στο 0.2%, μια μείωση της τάξης του 75%. Ενδεικτικά, το διάγραμμα των **ποσοστιαίων** χρόνων καθυστέρησης φαίνεται παρακάτω:



Εδώ τυχαίνει να παρατηρήσουμε και ορισμένους αρνητικούς χρόνους, φαινόμενο για το οποίο δεν γνωρίζω την αιτιολόγηση. Ίσως έχει να κάνει με τη λειτουργία των συναρτήσεων sleep και pthread_cond_signal- pthread_cond_wait . Σε γενικότερες γραμμές όμως , παρατηρούμε ότι η ποσοστιαία καθυστέρηση είναι αρκετά πιο μικρή, με μικρότερη διακύμανση και λιγότερες ακραίες τιμές σε σχέση με την πρώτη υλοποίηση.

Συμπέρασμα: Στη συγκεκριμένη συσκευή , με το συγκεκριμένο κώδικα , έχουμε στη v0 μέσο χρόνο κλήσης των συναρτήσεων 100.8 ms ενώ στο v1 100.2 ms. Θέλοντας να πετύχουμε κατά μέσο όρο ακριβώς αυτό το νούμερο, θα πρέπει μειώσουμε το threashold στη v0 και το χρόνο της sleep στη v1. Το rpi zero έτρεχε το stock image χωρίς επιπλέον πακέτα (εκτός από fish και zip), και όχι αυτό που δόθηκε, διότι με το 2ο υπήρχαν πολλά προβλήματα με το setup του WiFi. Επίσης, έλειπαν πολλές εντολές και η διαχείριση της συσκευής ήταν πιο δύσκολη.

Σταύρος Μαλακούδης samalako@ece.auth.gr 9368