Άσκηση 3

Ενσωματωμένα Συστήματα Πραγματικού Χρόνου

Αντώνης Παππάς (7002)
antonigp@auth.gr

1 Υλοποίηση βασικής λειτουργίας

Το πρόγραμμα δεν υλοποιήθηκε με το μοντέλο producer-consumer άλλα με τη χρήση του *inotify* API. Όπως ζητήθηκε έχουμε 2 νήματα, το κύριο νήμα αποθήκευσης και το νημα σάρωσης. Το νήμα σάρωσης γράφει σε ένα αρχείο τα ssids και το νήμα αποθήκευσης παρακολουθει τις αλλαγές στο εν λόγω αρχείο και κάθε φορά που κλείνει το διαβάζει. Πιο αναλυτικά:

1.1 Πίνακας SSID και timestamps

Για την αποθήκευση των SSID μόνο μία φορά, κι όχι κάθε φορά που εμφανίζεται, χρησιμοποιείται ένας πίνακας κατακερματισμού(hash table) με βάση αυτόν που διαθέτει η GNU C Library και την συνάρτηση κατακερματισμού της. Ο πίνακας κατακερματισμού προσφέρει ιδανική πολυπλοκότητα αναζήτησης O(1), αν δεν υπαρχουν συγκρούσεις. Γι'αυτό φροντίζουμε το πλήθος των κλειδιών να μην υπερβαίνει το 80% της μέγιστης χωρητικότητας.

Για την αντιστοίχηση των κλειδιών στα δεδομένα υλοποιήθηκε η δομή *observations*:

```
struct observations {
        struct timespec *observation_times; //Pointer to array of timestamps
        size_t size; //Current number of elements
        size_t capacity; //Preallocated memory
};
```

Επειδή η διεπαφή που παρέχει η βιβλιοθήκη για τον πινακα κατακερματισμού είναι περιορισμένη και δύσχρηστη, έγραψα τις συναρτήσεις *hashcreate()*, *hashsearch()*, *hashadd()* και *hashresize()*. Ειδικά η συνάρτηση *hashresize()* είναι απαραίτητη καθώς ο πίνακας κατακερματισμού της βιβλιοθήκης έχει σταθερό μέγιστο μέγεθος, που όταν ξεπεραστεί πρέπει να δημιουργηθεί αντίγραφο με μεγαλύτερη χωρητικότητα. Οι παραπάνω συναρτήσεις παίρνουν σαν όρισμα την δομή:

```
struct hash_struct{
        struct hsearch_data *hashtable; //Library-provided hash structure
        char **keys;
        struct observations *data;
        size_t size;  //Current number of elements
        size_t capacity; //Preallocated memory
};
```

Η υλοποίηση των παραπάνω βρίσκεται στο αρχείο *myhash.c* και στο αρχείο header *myhash.h*.

1.2 Νήμα Σάρωσης

Το νήμα σάρωσης είναι σχετικά απλό και αποτελείται από μόνο έναν βρόχο. Στην αρχή του βρόχου καταγράφεται ο πραγματικός χρόνος, μετά καλείται το script *searchWifi.sh*, υπολογίζεται ο χρόνος μέχρι την επόμενη σάρωση και καλείται η συνάρτηση *nanosleep()*.

Το νήμα σάρωσης υλοποιείται με την συνάρτηση *timer_thread()*,στο αρχείο main.c, και χρησιμοποιεί βοηθητικές συναρτήσεις που βρίσκονται στο αρχείο *timer_thread.c* κι ορίζονται στο *timer_thread.h*[1]. Το script που δόθηκε χρησιμοποιήθηκε σχεδόν ως έχει. Η μόνη αλλαγή που έγινε είναι ότι αντί τα ssids να στέλνονται στο stdout, αποθηκεύονται στο αρχείο */tmp/wifiout*.

1.3 Νήμα Αποθήκευσης

Το νήμα αποθήκευσης είναι το κύριο νήμα. Η λειτουργία του νήματος αποθήκευσης βασίζεται στο *inotify* API. Για να ελέγχουμε πότε είναι έτοιμα τα ssids, αντιστοιχούμε το αρχείο */tmp/wifiout* στη λίστα παρακολούθησης ενός στιγμιοτύπου *inotify* με τη μάσκα `IN_CLOSE_WRITE` και παίρνουμε έναν file descriptor. Καλώντας τη συνάρτηση *poll* με όρισμα τον file descriptor το νήμα μπλοκάρει χωρίς να καταναλώνει χρόνο cpu. Όταν το script *searchWifi.sh*

[1] `https://www.gnu.org/software/libc/manual/html_node/Elapsed-Time.html`

κλείσει το αρχείο */tmp/wifiout*, το λειτουργικό σύστημα ξυπνάει το νήμα αποθήκευσης και καλείται η συνάρτηση *handle_events()* η οποία αναλύει το αρχείο και «αντλεί» τα ssids.

Η έκδοση του προγράμματος προβολής της κατάστασης των ασύρματων συσκευών *iwinfo* που παρέχεται από το OpenWRT, δεν καταγράφει τον χρόνο ανίχνευσης, για αυτό ώς χρόνος ανίχνευσης ορίζουμε τον χρόνο τελευταίας επεξεργασίας του */tmp/wifiout*. Μετά από κάθε σάρωση αποθηκεύουμε σε ένα αρχείο το ssid και το αντιστοιχο timestamp σε μορφή csv, επιπλέον το ssid αναζητείται στον πίνακα κατακερματισμού με τη συνάρτηση *hashsearch()*. Αν βρεθεί, αποθηκεύουμε το timestamp καλώντας την *add_observation()*, αλλιώς το προσθέτουμε στον πίνακα κατακερματισμού με την *hashadd()*.

Ο κώδικας για τη χρήση του *inotify* API βασίζεται στο παράδειγμα που δίνεται στη σελίδα man του *inotify*[2]. Η συνάρτηση *handle_events()* βρίσκεται στο αρχείο `main.c`.

2 Περιορισμοί ZSUN

Το πρόγραμμα αναμένεται να τρέχει συνεχώς, κι όχι για συγκεκριμένο χρονικό διάστημα οπότε πρέπει να λάβουμε υπόψη τους περιορισμούς της συσκευής. Υποθέτουμε ότι στο zSun υπάρχει μία κάρτα SD 8GB ως χώρος αποθήκευσης, προσβάσιμη από το φάκελο */mnt/sda1/*, και μνήμη RAM το πολύ 32MB.

2.1 Περιορισμοί χώρου αποθήκευσης

Τα timestamps αποθηκεύονται στο αρχείο */mnt/sda1/ssid_logs/timestamps*. Αρχικά αποθηκεύονται στο αρχείο *timestamps*. Πριν προστεθούν καινούργια timestamps, ελέγχεται το μέγεθος του αρχείου κι αν είναι μεγαλύτερο από 25MB, προστίθεται στο όνομα του αρχείου ο πραγματικός χρόνος και δημιουργείται εκ νέου ένα άδειο αρχείο *timestamps*. Αυτό γίνεται με τη συνάρτηση *track_filesize()*. Στη συνέχεια ελέγχουμε το χώρο που απομένει στην κάρτα SD. Αν είναι λιγότερος από 25MB, διαγράφουμε τα αρχεία που έχουν δημιουργηθεί το πολύ μέχρι να μείνουν 250MB. Αν δεν έχουν μείνει αρχεία το πρόγραμμα τερματίζει. Τα παραπάνω γίνονται με τη συνάρτηση *check_for/storage()*.

2.2 Περιορισμοί μνήμης

Στη συνέχεια ελέγχουμε πόση μνήμη καταναλώνει το πρόγραμμα από το σύστημα `/proc`. Αν είναι παραπάνω από 25MB, διαγράφουμε τον πίνακα κατακερματισμού από τη μνήμη, και δημιουργούμε έναν καινούργιο. Ο έλεγχος γίνεται με τη συνάρτηση *get_kb_occupied()*.

3 Εκτέλεση στο zSun

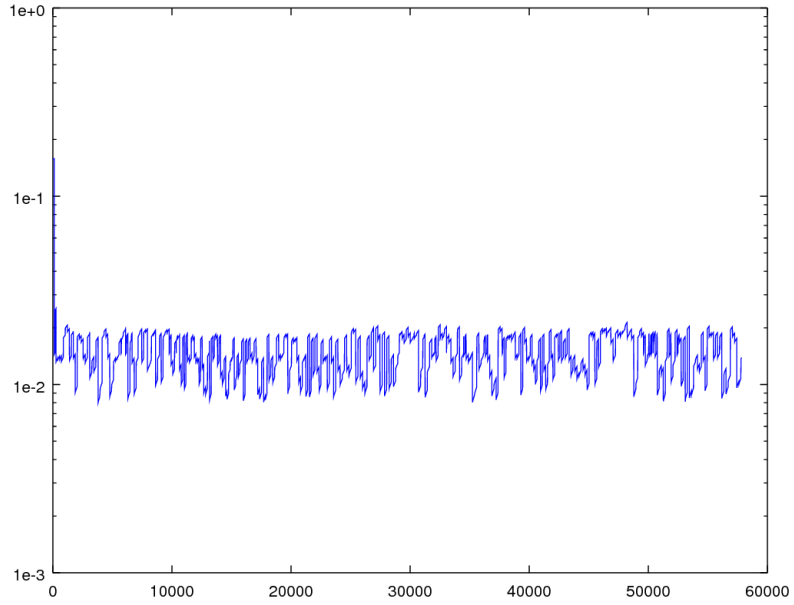
Στο σχήμα 1 καταγράφεται η καθυστέρηση εισαγωγής των δεδομένων στον πίνακα κατακερματισμού για περίοδο δειγματοληψίας 10 δευτερολέπτων. Βλέπουμε ότι η καθυστέρηση είναι γύρω στα 0.01s. Επιπλέον το ποσοστό του χρόνου που η CPU χρησιμοποιείται από το πρόγραμμα είναι 0.13%.

Στο σχήμα 2 καταγράφεται η αντίστοιχη καθυστέρηση για 1 δευτερόλεπτο. Ένα πρόβλημα που παρουσιάστηκε είναι ότι το zSun χρειάζεται περίπου 2.5 δευτερόλεπτα για να σαρώσει, με συνέπεια να αυξάνεται λίγο το διάστημα που χρειάζεται για να γίνει η εισαγωγή στον πίνακα. Παρόλα αυτά δε χάνονται ssids καθώς η καθυστέρηση παραμένει 10 φορές μικρότερη από το χρόνο σάρωσης. Όμως επειδή το zSun είναι σχεδόν μόνιμα σε κατάσταση σάρωσης, η ασύρματη σύνδεση είναι ασταθής, γι'αυτό το πρόγραμμα πρέπει να χρησιμοποιείται μέσα από έναν πολυπλεκτη τερματικών όπως το *tmux*. Το ποσοστό του χρόνου που η CPU χρησιμοποιείται από το πρόγραμμα είναι 0.20%.

Το πρόγραμμα εκτελείται στην κάρτα SD, στον ίδιο φάκελο με το *searchWifi.sh* με την εντολή `.\parser [time]`, όπου [time] τα δευτερόλεπτα μεταξύ σαρώσεων.

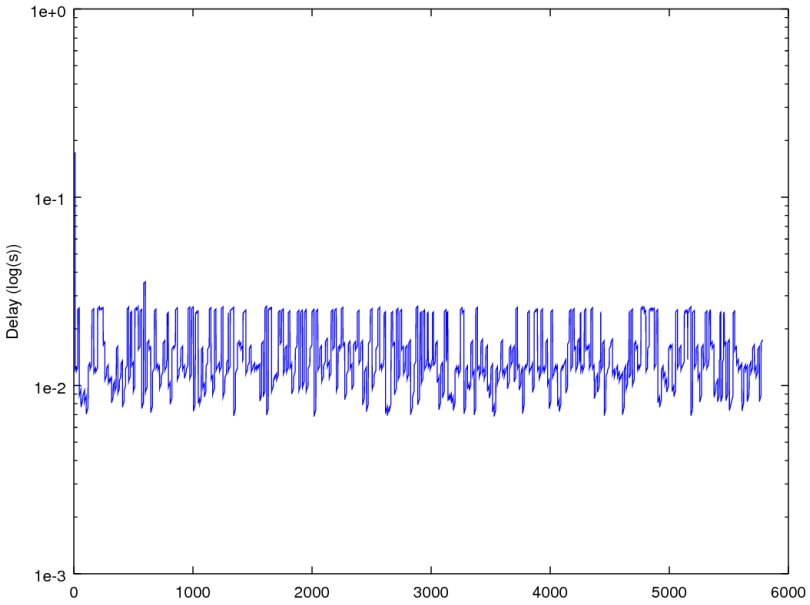
Με την αναφορά επισυνάπτεται ο πηγαίος κώδικας της εργασίας, το Makefile και το εκτελέσιμο πρόγραμμα για το zSun.

[2] `http://manpages.ubuntu.com/manpages/xenial/en/man7/inotify.7.html`



Σχήμα 1: Διαφορά των χρόνων από την στιγμή ανίχνευσης μέχρι τη στιγμή που προστίθεται το timestamp στον πίνακα. (10 δευτερόλεπτα)

.



Σχήμα 2: Διαφορά των χρόνων από την στιγμή ανίχνευσης μέχρι τη στιγμή που προστίθεται το timestamp στον πίνακα. (1 δευτερόλεπτα)

.