Αναφορά Τελικής Εργασίας Ενσωματωμένων Συστημάτων

Άρης Ελευθέριος Παπαγγέλης – ΑΕΜ: 8883

Ανάλυση λειτουργίας προγράμματος

Το πρόγραμμα υλοποιήθηκε με τις εξής συναρτήσεις:

- main
- server
- *client (thread)
- *receiveMsg (thread)
- produceMsg (SIGALRM handler)
- sendMsgs
- catch_int_term (SIGINT και SIGTERM handler)
- logger

Ορίζουμε κάποιες σταθερές αρχικά, καθώς και κάποιες καθολικές μεταβλητές για τη λειτουργία του προγράμματος. Οι καθολικές μεταβλητές είναι:

- λίστα με τις IPs
- λίστα με τα ΑΕΜ
- λίστα με τα ASCII μηνύματα
- δύο πίνακες μεγέθους STUDENTS, που περιλαμβάνουν το τελευταίο μήνυμα του buffer που στάλθηκε σε συγκεκριμένο φοιτητή, καθώς και το index στο οποίο ήταν το εν λόγω μήνυμα, ώστε να συνεχίσουμε να στέλνουμε απο το σημείο που σταματήσαμε στην τελευταία επικοινωνία
- μια μεταβλητή buff που περιλαμβάνει τα 2000 πιο πρόσφατα μηνύματα, καθώς και μια μεταβλητή count που δείχνει σε ποια θέση του πίνακα θα αποθηκευτεί το επόμενο στοιχείο
- ένα flag ονόματι fullBuffer, που δείχνει αν έχει γεμίσει ήδη μια φορά ο πίνακας buffer ή όχι
- δύο μεταβλητές, log_buffer και curr_log_count, που χρησιμοποιούνται για να γράφουν σε αρχείο κάθε LOG_BATCH_SIZE (=10) μηνύματα που λαμβάνονται
- δύο mutexes για το συγχρονισμό μεταξύ των threads, ένα που αφορά την πρόσβαση στις μεταβλητές buff και count, και ένα που αφορά τον file descriptor που ανοίγει ο server όταν κάνει accept νέα σύνδεση και δημιουργεί το thread *receiveMsg

main

Αρχικά, αρχικοποιούμε τις λίστες IPsLastMsgSentIndex και IPsLastMsgSent με αρχικές τιμές, θέτουμε τους handlers για τα αντίστοιχα μηνύματα που αναφέρονται στην αρχή της αναφοράς, και κάνουμε seed την rand(). Ύστερα, καλούμε την produceMsg για να παράξουμε έστω ένα μήνυμα στον buffer πριν αρχίσει να λειτουργεί το σύστημα, δημιουργούμε ένα thread *client και καλούμε και τη συνάρτηση server(). Επομένως, οι συναρτήσεις client και server τρέχουν σε παράλληλα νήματα.

catch_int_term

Ο handler των σημάτων SIGINT και SIGTERM, το μόνο που κάνει είναι να καλεί τη συνάρτηση logger για να γράψει όσα αρχεία υπάρχουν στο buffer και δεν έχουν γραφεί ακόμα σε αρχείο. Ύστερα τερματίζει το πρόγραμμα. Χρησιμοποιείται για να κάνουμε catch ένα σήμα kill που στέλνεται μετά από 2 ώρες λειτουργίας του προγράμματός μας.

server

Κάνουμε τις απαραίτητες αρχικοποιήσεις για την επικοινωνία μέσω TCP sockets, με διαδοχικές κλήσεις στις συναρτήσεις socket, setsockopt, bind, listen με κατάλληλα ορίσματα. Δηλαδή, δημιουργούμε ένα listening socket στο port 2288, που αναμένει οποιαδήποτε σύνδεση. Μετά, μέσα σε ένα infinite loop, έχουμε την εντολή newfd = accept(...), που θα δημιουργήσει νέο socket μόλις δεχτεί οποιαδήποτε σύνδεση μέσω της συνάρτησης connect(...) στον client (αφού έχουμε κλειδώσει πρώτα το fd_mutex, θα φανεί στην επόμενη συνάρτηση γιατί). Μόλις αυτό γίνει, φτιάχνουμε νέο thread *receiveMsg στο οποίο περνάμε σαν όρισμα το file descriptor newfd που δημιουργήθηκε από την accept(...). Το infinite loop θα συνεχίσει να εκτελείται παράλληλα με το thread *receiveMsg και να δέχεται νέα πιθανά connections, περιμένοντας ωστόσο μέχρι να ξεκλειδωθεί το fd_mutex από το thread.

***receiveMsg**

Η συνάρτηση αυτή, που τρέχει σαν ξεχωριστό thread, είναι υπεύθυνη για την λήψη και αποθήκευση των μηνυμάτων από τον client. Θέτουμε αρχικά την τιμή του file descriptor που περάσαμε σαν όρισμα by reference σε μια μεταβλητή sock. Ύστερα, μπορούμε να ξεκλειδώσουμε το fd_mutex και να συνεχίσει ο server να αναμένει νέες συνδέσεις. Λαμβάνουμε το πρώτο μήνυμα με μια recv. Ύστερα, θα συνεχίσουμε να λαμβάνουμε για όσο το ληφθέν μήνυμα δεν είναι “Exit”, οπότε και θα σταματήσει η λήψη. Μέσα στο loop λήψης μηνυμάτων, ελέγχουμε καταρχάς αν έχει γεμίσει το buffer, οπότε θα θέσουμε το count ξανά ίσο με 0 και το flag fullBuffer ίσο με 1. Ύστερα, θα ελέγξουμε για διπλότυπα, είτε μέχρι το τωρινό count, αν δεν έχει γεμίσει ο πίνακας μηνυμάτων, είτε σε όλο τον πίνακα αν έχει γεμίσει. Αν εν τέλει δε βρεθεί διπλότυπο μήνυμα, τότε κλειδώνουμε το mutex του buffer (αφού θα τροποοποιηθεί η buff και η count), το ληφθέν μήνυμα αντιγράφεται στον πίνακα μας και αυξάνεται η count. Καλείται επίσης η συνάρτηση logger() για να αποφασίσει αν θα γραφούν σε αρχείο οι νέες αλλαγές. Ύστερα ξεκλειδώνουμε το mutex του buffer. Αν βρεθεί διπλότυπο, τότε απλά προχωράμε στη λήψη του επόμενου μηνύματος, χωρίς να αντιγράψουμε το προηγούμενο στον πίνακά μας. Ο server στέλνει το μήνυμα “OK” στον client για το συγχρονισμό των αποστολών/λήψεων (μήνυμα ACK), και ύστερα λαμβάνει το επόμενο μήνυμα. Η παραπάνω διαδικασία επαναλαμβάνεται μέχρι να λάβουμε “Exit” ή μέχρι να κάνει timeout το socket. Στο τέλος, θα κλείσουμε το socket και θα τερματίσουμε την εκτέλεση του thread *receiveMsg.

***client**

Κάνουμε τις απαραίτητες αρχικοποιήσεις για την επικοινωνία μέσω TCP sockets, θέτοντας αρχικά το PORT επικοινωνίας. Ύστερα, έχουμε ένα loop από 0 μέχρι STUDENTS (αριθμός φοιτητών/ΑΕΜ) που θα τρέχει κάθε 60 δευτερόλεπτα για να μην καταναλώνει άσκοπη ενέργεια. Μέσα στο loop, δημιουργούμε αρχικά socket, μετατρέπουμε την IP του j-στου student από τη λίστα IPs σε binary μορφή, και προσπαθούμε να κάνουμε connect(...) σε αυτή την IP. Αν αυτό επιτύχει, τότε καλούμε τη συνάρτηση sendMsgs(sock,j) για την αποστολή των μηνυμάτων, αλλιώς εκτυπώνουμε πως η σύνδεση με την εν λόγω IP απέτυχε. Σε κάθε περίπτωση, στη συνέχεια θα κλείσουμε το socket, και θα ελέγξουμε το επόμενο ΑΕΜ στη λίστα. Αν έχουμε ήδη ελέγξει όλα τα ΑΕΜ, τότε το thread θα κάνει sleep για 60 δευτερόλεπτα και θα γίνει η ίδια διαδικασία ξανά από την αρχή. Δηλαδή, το thread προσπαθεί διαδοχικά να συνδεθεί και να στείλει μηνύματα σε κάθε ΑΕΜ, κάθε 60 δευτερόλεπτα.

sendMsgs

Η συνάρτηση αυτή είναι υπεύθυνη για την αποστολή μηνυμάτων στον παραλήπτη στον οποίο συνδεθήκαμε μέσω της *client. Δέχεται ως ορίσματα το socket επικοινωνίας και το receiver, που είναι το index του j-στου φοιτητή της λίστας. Χρησιμοποιούμε τους πίνακες IPsLastMsgSent και IPsLastMsgSentIndex που ορίστηκαν στην αρχή του προγράμματος, και περιέχουν το τελευταίο μήνυμα που στάλθηκε στον receiver καθώς και το index στο οποίο βρισκόταν το μήνυμα. Θέτουμε το endIndex για το loop που ακολουθεί ίσο με την τωρινή τιμή του count. Ύστερα, ελέγχουμε , για να στείλουμε μηνύματα ή όχι, το εξής: αν το index του τελευταίου μηνύματος που στείλαμε στον εν λόγω φοιτητή είναι διάφορο του endIndex, οπότε και σημαίνει πως το count είχε προχωρήσει στο ενδιάμεσο και θα πρέπει να του στείλουμε τα καινούρια μηνύματα, ή αν το τελευταίο μήνυμα που στείλαμε στον εν λόγω φοιτητή είναι διάφορο του buff[endIndex]. Ο δεύτερος έλεγχος καλύπτει την περίπτωση να έχει γεμίσει ΞΑΝΑ το ολόκληρο το buff απο μηνύματα δικά μου ή άλλων αποστολέων, χωρίς να έχει σταλέι μήνυμα εν τω μεταξύ στον εν λόγω παραλήπτη. Σε αυτή την περίπτωση, ενώ μπορεί το index του τελευταίου μηνύματος να είναι ίσο με το endIndex, το μήνυμα στην εν λόγω θέση θα διαφέρει οπότε και πάλι θα πρέπει να ξαναστείλουμε μηνύματα. Μέσα στο loop, το index του τελευταίου μηνύματος θα αυξάνεται (ελέγχεται ύστερα αν ξεπέρασε το μέγιστο μήκος του buff) και θα στέλνεται το μήνυμα που βρίσκεται στην συγκεκριμένη θέση του buff στον παραλήπτη. Μετά, αναμένεται μήνυμα ACK από τον παραλήπτη για τη συνέχεια της αποστολής. Τέλος, αλλάζει και το τελευταίο μήνυμα που στάλθηκε, last_msg_sent, και το loop ξανατρέχει. Το loop αποστολής θα επαναλαμβάνεται όσο το index του τελευταίου μηνύματος για τον εν λόγω αποστολέα είναι διάφορο του endIndex. Αφού τερματίσει το loop, θα σταλεί μήνυμα “Exit” στον server, και θα ενημερωθούν οι πίνακες IPsLastMsgSent και IPsLastMsgSentIndex στη θέση του receiver με τις καινούριες τιμές. Ύστερα η συνάρτηση επιστρέφει.

produceMsg

Η συνάρτηση αυτή καλείται κάθε φορά που καταφθάνει στο πρόγραμμα σήμα SIGALRM. Σκοπός της είναι να παράγει νέο μήνυμα και να το προσθέτει στο buffer. Αρχικά, προσπαθεί να κλειδώσει το buffer mutex με pthread_mutex_trylock, εφόσον θα τροποποιήσει το buff και το count παρακάτω. Αν δε το καταφέρει, αναβάλεται για 3 δευτερόλεπτα και ξανακαλείται. Αν το καταφέρει, τότε παράγει το μήνυμα με τη μορφή που δίνεται στην εκφώνηση μέσω της sprintf, με αποστολέα το δικό μου AEM (=8883), τυχαίο παραλήπτη από τη λίστα, timestamp δημιουργίας του μηνύματος και τυχαίο ASCII μήνυμα απο την προκαθορισμένη λίστα. Ύστερα, αντιγράφει το μήνυμα στο buff και αυξάνει το count, αφού προτέρως έχει ελέγξει αν είναι ήδη γεμάτο το buff. Καλείται επίσης η συνάρτηση logger() για να αποφασίσει αν θα γραφούν σε αρχείο οι νέες αλλαγές. Ύστερα, αποφασίζεται τυχαία ένα διάστημα από 20 έως 60 δευτερόλεπτα * για να παραχθεί το επόμενο μήνυμα μετά από τόσο χρόνο μέσω της alarm. Επιπλέον, ο εν λόγω χρόνος γράφεται στο αρχείο “timebetweenmessages.log” για την εξαγωγή στατιστικών. Τέλος, το buffer mutex ξεκλειδώνεται και ο signal handler επιστρέφει.

*Η εκφώνηση έλεγε κάθε 1 με 5 λεπτά, αλλά επειδή είμασταν λίγα άτομα θεώρησα σκόπιμο να μειώσω το χρόνο παραγωγής μηνυμάτων ώστε να υπάρχουν εν τέλει περισσότερα μηνύματα στο buffer και να δοκιμαστεί σωστά το σύστημα. Παρόλα αυτά, η λογική θα ήταν η ίδια για οποιοδήποτε διάστημα.

logger

Η συνάρτηση αυτή καλείται στα σημεία που περιγράφηκαν στις παραπάνω συναρτήσεις. Σκοπός της είναι να γράφει σε αρχείο τα μηνύματα που έχουν ληφθεί, για την εξαγωγή στατιστικών αργότερα από αυτά. Δέχεται ως ορίσματα το μήνυμα το οποίο αφορά η κλήση, καθώς και μια int μεταβλητή flush_remaining μέσω της οποίας θα γραφούν τα υπολειπόμενα μηνύματα σε αρχείο όταν φθάσει στο πρόγραμμα σήμα KILL. Λειτουργεί ως εξής: Αρχικά, αντιγράφεται το μήνυμα στο log_buffer και αυξάνεται το curr_log_count. Ύστερα, ελέγχουμε αν το curr_log_count ισούται με LOG_BATCH_SIZE (=10) ή αν flush_remaining ισούται με 1 για να αρχίσουμε να γράφουμε το log_buffer σε αρχείο. Αυτό το κάνουμε διότι το FILE I/O είναι μια αργή διαδικασία λόγω του ότι πρέπει να έχει πρόσβαση στη μονάδα αποθήκευσης, που ειναι πολλαπλάσια πιο αργή από τη RAM. Επομένως, για να μην έχουμε αυτό το performance overhead, προτιμούμε να γράφουμε κάθε 10 (ή και περισσότερα αν χρειαστεί) μηνύματα στο αρχείο και όχι κάθε μήνυμα που στέλνεται ένα-ένα. Η εξαίρεση είναι όταν η logger κληθεί από την catch_int_term με όρισμα flush_remaining = 1, οπότε και θα γραφούν τα μηνύματα που απομένουν στο log_buffer και το πρόγραμμα θα τερματίσει.

Παραδοχές Επικοινωνίας και εκτέλεσης

Η επικοινωνία έγινε μεταξύ τριών Raspberry Pi, τα δύο από αυτά Raspberry Pi Zero W και το ένα Raspberry Pi 3 Model B+. Οι συσκευές συνδέθηκαν ασύρματα, έχοντας IPs με τη μορφή που περιγράφεται στην εκφώνηση.

Το compilation του αρχείου έγινε μέσω make. Το makefile περιλαμβάνεται μαζί με τον πηγαίο κώδικα της εργασίας.

Το πρόγραμμα εκτελέστηκε μέσω του εξής bash script, με τίτλο run2hours.sh:

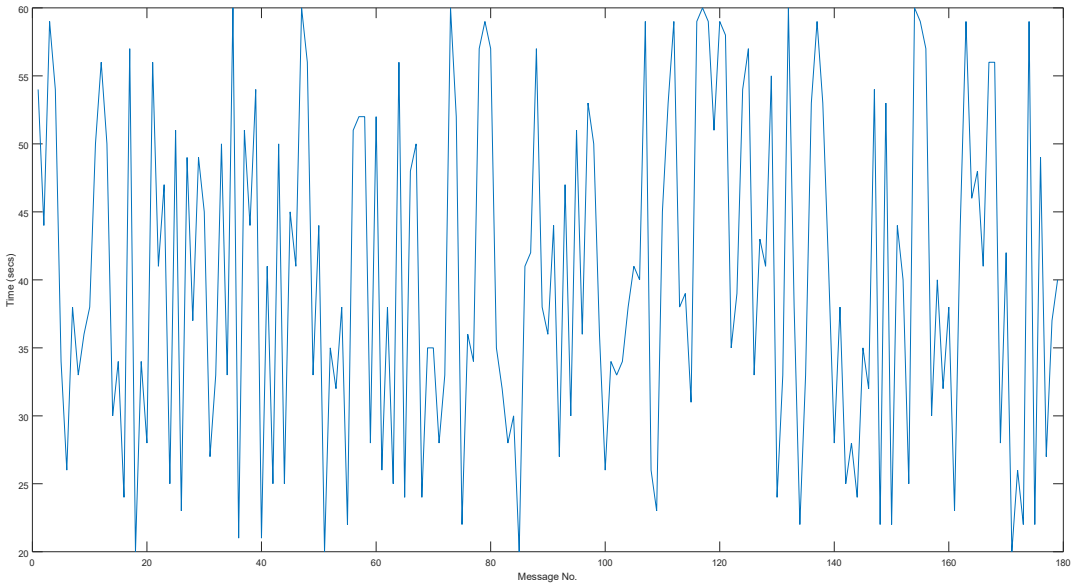
```
./bin/espx &
pid=$!
sleep 7200
kill $pid
sed "s/_/,/g;" statistics.log > statistics.csv
sed -i '1s/^/sender,receiver,timestamp,msg\n/' statistics.csv
```

Το παραπάνω script, αφου ξεκινά την εκτέλεση του προγράμματος, θα στείλει σήμα διακοπής μετά από 2 ώρες λειτουργίας. Ύστερα, θα παράγει αρχείο .csv από το αρχείο .log .

Τελικά, μετά την εκτέλεση του προγράμματος έχουν παραχθεί 3 αρχεία:
statistics.log
statistics.csv
timebetweenmessages.log
Τα αρχεία αυτά αναλύονται παρακάτω για την παραγωγή στατιστικών.

Διαγράμματα και στατιστικές μετρήσεις

α) Χρόνος μεταξύ των παραχθέντων μηνυμάτων



Φαίνεται η τυχαιότητα του χρόνου παραγωγής μηνυμάτων, από 20 έως 60 δευτερόλεπτα. Τα μηνύματα που παρήγαγα, δηλαδή τα μηνύματα με αποστολέα 8883, κατά τη διάρκεια του δίωρου λειτουργίας, ήταν 179.

Στατιστικά:
Mean = 40.3296
Median = 39
Std = 12.2574

Τα μηνύματα αυτά υπάρχουν στο αρχείο sender8883.txt .

β) Συχνότερο ASCII μήνυμα από όλους τους αποστολείς

When The Sun Goes Down , μήνυμα που παρήχθει από τον φοιτητή με ΑΕΜ 8906.

Στο τέλος λειτουργίας του προγράμματος το μήνυμα αυτό βρισκόταν στο buffer συνολικά 44 φορές από τα 719 μηνύματα, δηλαδή είχε συχνότητα εμφάνισης περίπου 6%.

γ) Συχνότερο ASCII μήνυμα που είχε ως αποστολέα εμένα

Si tu novio te deja sola. , μήνυμα που παρήχθει από τον φοιτητή με ΑΕΜ 8883.

Στο τέλος λειτουργίας του προγράμματος το μήνυμα αυτό βρισκόταν στο buffer συνολικά 23 φορές από τα 719 μηνύματα, δηλαδή είχε συχνότητα εμφάνισης περίπου 3,2%.

Το ποσοστό αυτού του μηνύματος σε σχέση με τα 179 μηνύματα που παρήγαγα, είναι περίπου 12,8%.

δ) Συχνότερο ASCII μήνυμα που είχε ως παραλήπτη εμένα

Morning Glory , μήνυμα που παρήχθει από τον φοιτητή με ΑΕΜ 8906.

Στο τέλος λειτουργίας του προγράμματος το μήνυμα αυτό βρισκόταν στο buffer συνολικά 22 φορές από τα 719 μηνύματα, δηλαδή είχε συχνότητα εμφάνισης περίπου 3%.

Το ποσοστό αυτού του μηνύματος σε σχέση με τα 321 μηνύματα που παρήγαγε ο αποστολέας 8906, είναι περίπου 6,8%.

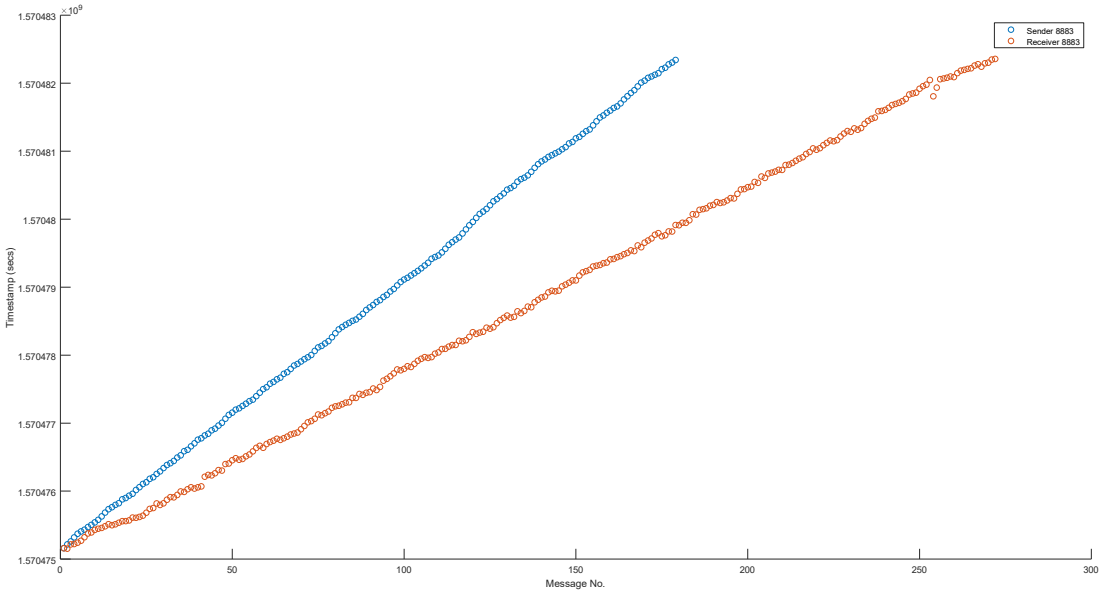
ε) Πλήθος παραγμένων μηνυμάτων από κάθε φοιτητή

Ο φοιτητής 8883 παρήγαγε 179 μηνύματα.
Ο φοιτητής 8906 παρήγαγε 321 μηνύματα.
Ο φοιτητής 8858 παρήγαγε 219 μηνύματα.

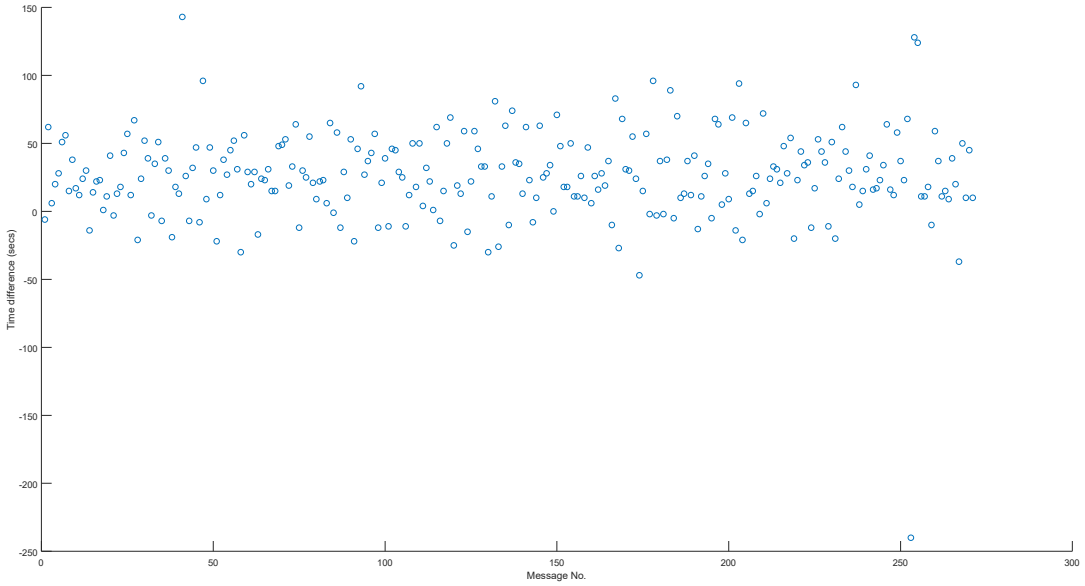
στ) Πλήθος μηνυμάτων που είχαν ως παραλήπτη τον κάθε φοιτητή

Ο φοιτητής 8883 ήταν παραλήπτης 271 μηνυμάτων.
Ο φοιτητής 8906 ήταν παραλήπτης 186 μηνυμάτων.
Ο φοιτητής 8858 ήταν παραλήπτης 261 μηνυμάτων.

ζ) Timestamps για το κάθε μήνυμα που εμπλέκει το ΑΕΜ μου



η) Χρονικές διαφορές ανάμεσα στα παραχθέντα μηνύματα που είχαν ως παραλήπτη εμένα



Τέλος, υπάρχουν και τα αρχεία sender8883.txt και receiver8883.txt, που έχουν μόνο τα μηνύματα στα οποία είμαι εγώ αποστολέας και παραλήπτης αντίστοιχα. Βρίσκονται μαζί με τα αρχεία που αναφέρονται στην προηγούμενη ενότητα.

Screenshots και φωτογραφίες

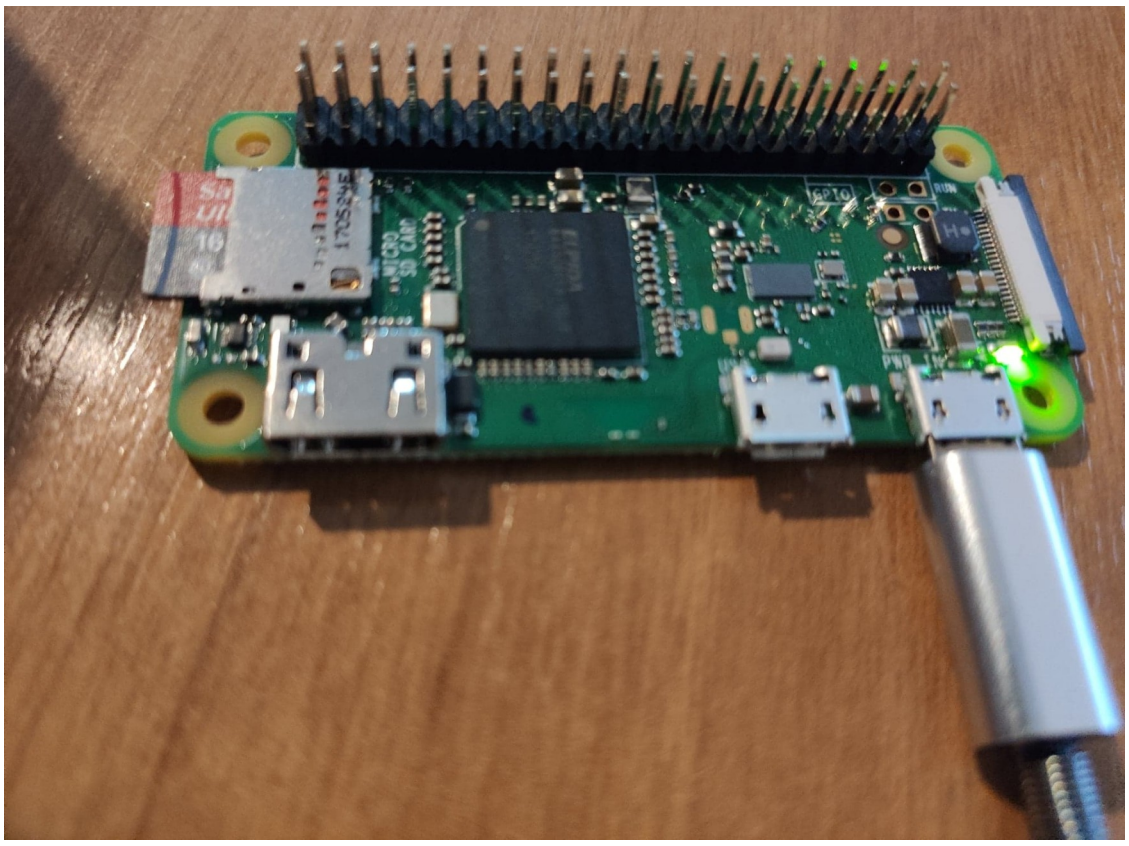
```
aris@FrenzyCawk-PC:~/espx$ ./a.out
        8883_8858_1570649870_Eat clen tren hard test your limits.

Count:2
CLIENT: Connection Failed with IP 192.168.100.15
        8883_8858_1570649870_Eat clen tren hard test your limits.

        8883_8858_1570649883_Me lo pide sin gorrito.
Count:3
CLIENT: Connection Failed with IP 192.168.100.15
        8883_8858_1570649870_Eat clen tren hard test your limits.

        8883_8858_1570649883_Me lo pide sin gorrito.
        8883_8858_1570649903_Compound lifts are better than isolation.
        8883_8906_1570649908_Reggaeton is the best musical genre.
        8883_8858_1570649913_Me lo pide sin gorrito.
        8883_8858_1570649926_Si tu novio te deja sola.
Count:7
```

Η παραπάνω εικόνα είναι απο τοπική δοκιμή του προγράμματος, παρέλειψα να πάρω screenshots του command line κατά την λειτουργία στα Raspberry Pis. Για το screenshot εισήγαγα την IP του PC μου και μια τυχαία τοπική IP. Παρόλα αυτά η λογική ήταν ακριβώς η ίδια στα Pis, έχοντας φυσικά τις αντίστοιχες IPs των Pis.



Στιγμιότυπο απο τη δοκιμή του προγράμματος τοπικά, με σύνδεση σε PC για τροφοδοσία.

Στιγμιότυπο από την τελική εκτέλεση του προγράμματος ανάμεσα στα 3 Pi. Τροφοδοσία μέσω Powerbank.

Σχόλια και παρατηρήσεις

Παρατηρούμε πως το σύστημα λειτουργεί σωστά ανάμεσα σε 3 άτομα. Δυστυχώς δεν είχα τη δυνατότητα να δοκιμάσω με περισσότερους φοιτητές, αλλά εφόσον λειτουργεί για n>2 εικάζω πως θα λειτουργεί σωστά και με περισσότερα άτομα.

Ο κώδικας της εργασίας, τα αρχεία .txt και .csv με τα στατιστικά, το makefile και το script run2hours.sh βρίσκονται στο ακόλουθο repository του Github:

https://github.com/ArisPapangelis/Real-Time-Embedded-Systems-ECE/tree/master/final-project

Τα προαναφερθέντα συμπεριλαμβάνονται φυσικά και στο .zip αρχείο στο οποίο βρίσκεται και η παρούσα αναφορά.