Εργασία 3

ΕΝΣΩΜΑΤΩΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ

Σεπτέμβριος 2018

του φοιτητή

Παπαδόπουλου Κωνσταντίνου
ΑΕΜ 8677

Εισαγωγή:

Σκοπός της παρούσας εργασίας είναι η δημιουργία ενός συστήματος ανταλλαγής απλών μηνυμάτων, με την επιπρόσθετη ιδιότητα ότι θα αποτελεί ενσωματωμένο σύστημα πραγματικού χρόνου.
Πολλαπλοί clients θα πρέπει να έχουν την δυνατότητα να εξυπηρετηθούν από έναν server για να λάβουν και να στείλουν μηνύματα.
Στη συνέχεια αναλύεται η υλοποίηση των εφαρμογών client (compiled για PC, x86 αρχιτεκτονική) και server (compiled για το ZSUN), καθώς και άλλα στοιχεία του προγράμματος.

Client-Server:

Η λειτουργία του συστήματος ανταλλαγής μηνυμάτων που σχεδιάσθηκε προσομοιάζει με ένα σύστημα διανομής της κλασικής αλληλογραφίας.
Συνεχίζοντας τη συγκεκριμένη αναλογία, μπορούμε να παρομοιάσουμε τους υπαλλήλους στο ταχυδρομείο με τα threads του συστήματος, τους πελάτες με τους clients που θέλουν να δουν αν έχουν καινούργια μηνύματα ή/και να στείλουν αυτοί κάποιο message και τα ράφια που κρατούν τα γράμματα με τη δομή που χρησιμοποιούμε για να αποθηκεύσουμε ηλεκτρονικά τα μηνύματα των χρηστών.
Με αυτόν τον τρόπο, κοιτάζοντας δηλαδή την κατάλληλη γραμμή και στήλη στον πίνακά μας (βλ. παρακάτω), τα στοιχεία αποστολέα / παραλήπτη ταυτίζονται για να γίνει η παράδοση μηνυμάτων.

<u>Server program</u>
Η εφαρμογή πρέπει να δέχεται ένα όρισμα εισόδου με τον αριθμό της πόρτας στο μηχάνημα. Τον αριθμό τον επιλέγει ο διαχειριστής και πρέπει να είναι γνωστός έτσι ώστε και η εφαρμογή client να ακούει στην ίδια πόρτα.
Η εφαρμογή server είναι υπεύθυνη να ανοίξει ένα socket σε μία καθορισμένη πόρτα της συσκευής όπου εκτελείται. Η εφαρμογή εκτελείται αενάως αναμένοντας κάποιο χρήστη να συνδεθεί στη συγκεκριμένη πόρτα ώστε να ξεκινήσει η επικοινωνία.
Αναλυτικά στοιχεία του κώδικα:

- Δημιουργούνται διαφορετικά νήματα για την εξυπηρέτηση κάθε πελάτη.

- Δημιουργήσαμε μια δομή *Box (int exist; char mail)* που αποθηκεύει τα μηνύματα κάθε client και την ύπαρξη τους ή όχι (vacancy).

- Στον πίνακα *Box mailbox[99][99]* αποθηκεύονται τα μηνύματα όλων των χρηστών. Κάθε γραμμή αντιπροσωπεύει τα μηνύματα που προορίζονται για ένα συγκεκριμένο client και κάθε στήλη καθορίζει από που έχουν σταλεί αυτά (δες σχήμα).

- Ένας πίνακας *int sum[99]* κρατάει τον συνολικό αριθμό μηνυμάτων για κάθε χρήστη. Με αυτόν τον τρόπο μειώνουμε το χρόνο προσπέλασης του πίνακα *Box mailbox[99][99]*.

- Αρχικά ο server δέχεται την διεύθυνση του client που εξυπηρετεί (parsing του string που έχει σταλθεί ως διεύθυνση από τον client). Κοιτάζει στη συνέχεια για τυχόν μηνύματα που προορίζονται για αυτόν. Έπειτα, ο πελάτης μπορεί να στείλει καινούργιο μήνυμα ή να ανανεώσει τα εισερχόμενα του. Η διαδικασία επαναλαμβάνεται.

Έχουν προστεθεί σχόλια στον κώδικα του προγράμματος που εξηγούν αναλυτικότερα τη λειτουργία του.

Client program

Η εφαρμογή client δέχεται τρία ορίσματα εισόδου: την IP της συσκευής όπου εκτελείται το πρόγραμμα server, την αντίστοιχη πόρτα και τη διεύθυνση/όνομα του client.
Η εφαρμογή στέλνει στο server τη διεύθυνσή της. Προβάλλονται τα εισερχόμενα μηνύματα του χρήστη, αν υπάρχουν. Στη συνέχεια, ζητάει από τον χρήστη να εισάγει κάποιο μήνυμα και τη διεύθυνση του παραλήπτη ή απλά να ανανεώσει τα εισερχόμενα του. Η διαδικασία επαναλαμβάνεται.
Ο πρώτος client που εισέρχεται στο σύστημα έχει ρόλο διαχειριστή και πρέπει να έχει τη διεύθυνση *CL00*.
Είναι σημαντικό ο client πριν στείλει κάποιο μήνυμα να κάνει refresh για να ελέγξει ότι δεν έχει λάβει αυτός κάποιο εισερχόμενο μήνυμα.

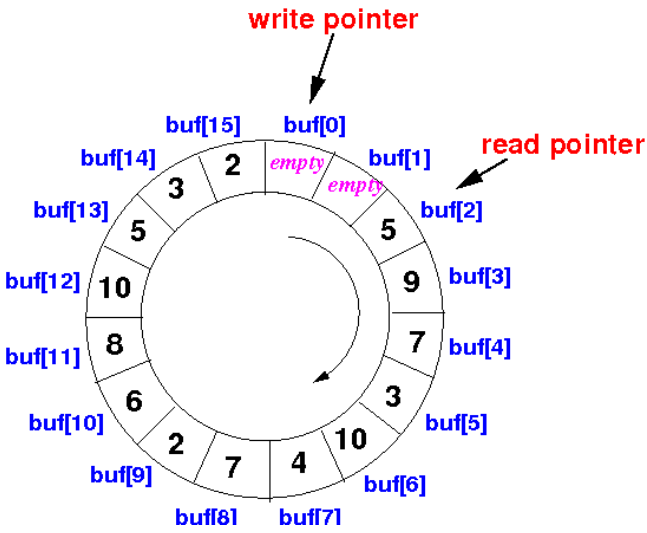
Έχουν προστεθεί σχόλια στον κώδικα του προγράμματος που εξηγούν αναλυτικότερα τη λειτουργία του.

Στα παρακάτω σχήματα φαίνεται η λειτουργία του client-server system.

CLIENT		SERVER
Send client address	→	Receive sender's address
Receive possible messages	←	Check for messages for sender
Send a recipient address and message or refresh	→	Receive message and address
Receive confirmation message	←	Send message status

Μεγάλος φόρτος – Ουρές:

Με τη σύνδεση κάθε client στο σύστημα, αυτός εισάγεται σε μία FIFO (κυκλική) ουρά προτεραιότητας. Αυτό θα βοηθήσει έτσι ώστε όταν η ουρά είναι άδεια το σύστημα να ελέγχει για σύνδεση κάθε 2 sec, μπαίνοντας σε sleep.
Όταν το σύστημα εξυπηρετεί 50 clients, δηλαδή όταν φτάνει περίπου σε 50% του maximum load, τότε με τη σύνδεση κάθε καινούργιου client δεν δημιουργείται καινούργιο thread. Αντ' αυτού, τίθεται σε λειτουργία η σειρά εξυπηρέτησης με βάση την ουρά προτεραιότητας και κάθε πελάτης που εξυπηρετείται μπαίνει στο τέλος της ουράς, δίνοντας προτεραιότητα σε όλους τους άλλος που μπήκαν μετά από αυτόν.



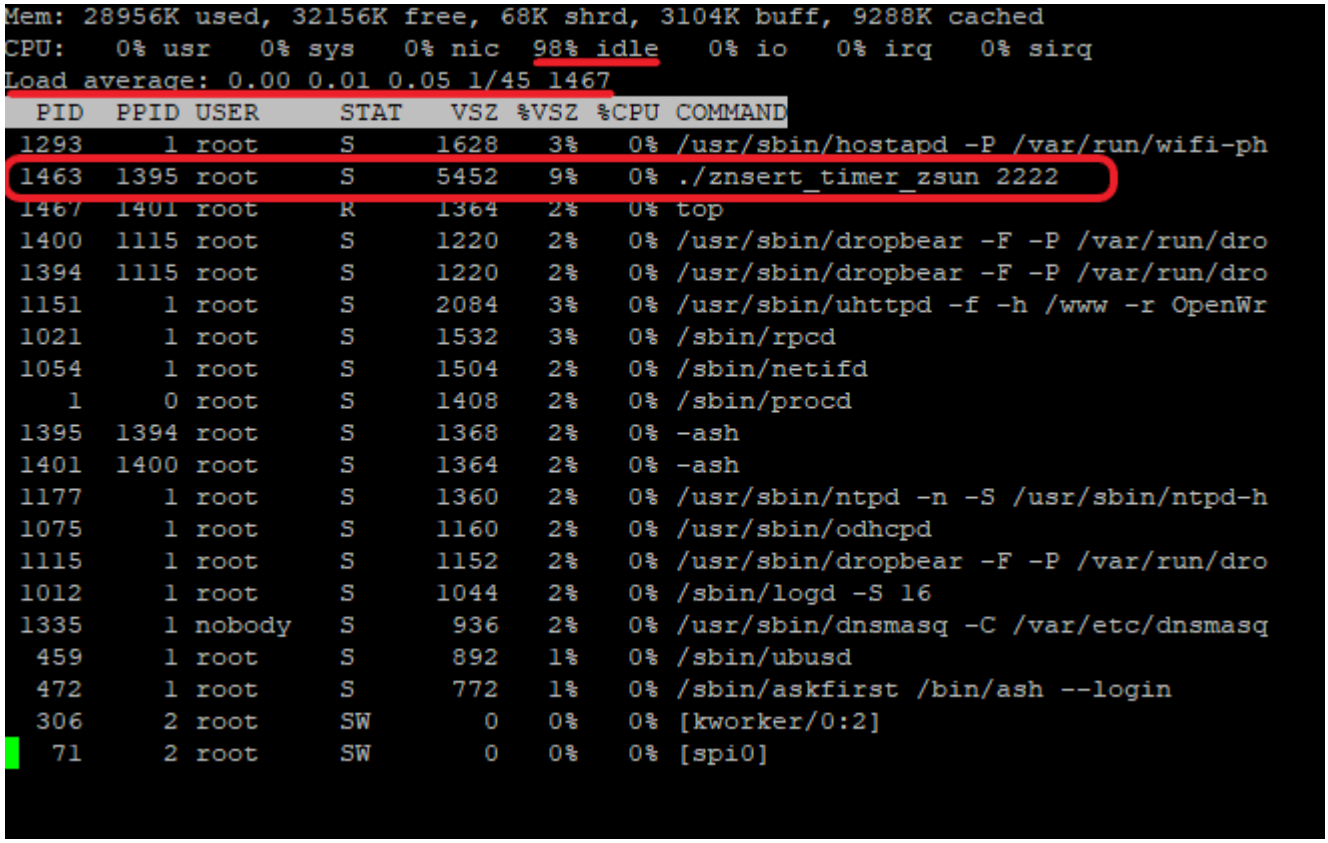
Έλεγχος ορθότητας:

Πραγματοποιήθηκε έλεγχος ορθότητας στέλνοντας/λαμβάνοντας μηνύματα πειραματικά από διάφορους clients σε άλλους clients μέσω του server. Επιπλέον μετρήθηκε ο χρόνος απόκρισης του συστήματος και η ενεργειακή κατανάλωσή του.
Τα αποτελέσματα βρίσκονται στο σύνδεσμο που δίνεται στο τέλος της εργασίας.

Ενεργειακή κατανάλωση:

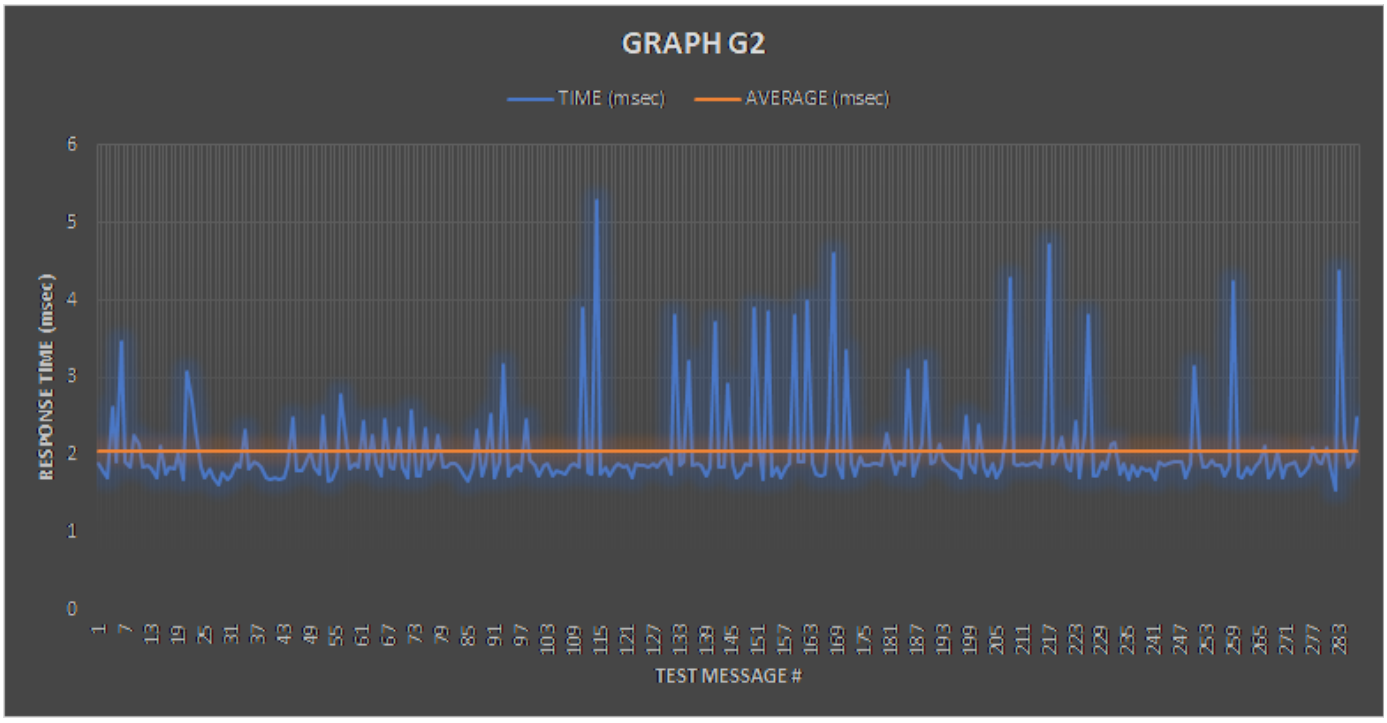
Όπως αναφέρθηκε και παραπάνω το σύστημα ελέγχει αν η ουρά προτεραιότητας είναι άδεια και τότε μπαίνει σε *sleep* με περίοδο 2 sec καταναλώνοντας με αυτόν τον τρόπο λιγότερη ενέργεια.

Όταν η ουρά δεν είναι άδεια, σε κανονική δηλαδή λειτουργία του συστήματος, το σύστημα ελέγχει για νέα αιτήματα των πελατών κάθε 1 sec (με *sleep, nanosleep(&delay,NULL)*).
Οι παραπάνω χρόνοι φαίνεται να δημιουργούν καλή απόκριση για τη συγκεκριμένη εφαρμογή.
Τρέχοντας την εντολή *uptime* και *top* παρατηρούμε στη μέτρηση για το φόρτο των τελευταίων 5 λεπτών, ότι αυτός δεν υπερβαίνει το 9%.
Τα αποτελέσματα βρίσκονται στο σύνδεσμο που δίνεται στο τέλος της εργασίας.



Ταχύτητα λήψης/αποστολής μηνυμάτων:

Όπως εξηγήθηκε και προηγουμένως, για λόγους εξοικονόμησης ενέργειας, θέσαμε στο ενσωματωμένο σύστημα 1 sec sleep περίοδο για την αναζήτηση καινούργιων client και 1 sec sleep περίοδο για τον έλεγχο λήψης/αποστολής μηνυμάτων. Αυτό βέβαια δε σημαίνει ότι το σύστημα είναι σε sleep κάθε 1 sec, διότι η *recv()* είναι μία blocking συνάρτηση, στην οποία το πρόγραμμα θα περιμένει για να στείλει ο client.
Άρα, το worst case scenario απόκρισης του συστήματος είναι περίπου 2 sec (ο χρόνος που μεσολαβεί ανάμεσα στις δύο προηγούμενες περιόδους, όπου δεν είναι σε sleep το πρόγραμμα, είναι αμελητέος σε σχέση με τα 2 δευτερόλεπτα) και αυτό συμβαίνει όταν συνδέεται για πρώτη φορά ένας client.
Στις μετρήσεις μας ο client έστελνε κάθε 2 sec ένα μήνυμα *CL01-HELLO* και τη χρονική στιγμή εκείνη παίρναμε ένα timestamp (*gettimeofday()*). Όταν ο server λάμβανε το μήνυμα έστελνε με τη σειρά του ένα ACK message. Με τη λήψη του ACK ο client κρατούσε πάλι timestamp. Με τη διαφορά των δύο timestamps μετρήσαμε την απόκριση του συστήματός μας.
Κατά μέσο όρο παρατηρήθηκε ταχύτητα λήψης/αποστολής μηνυμάτων ίση με περίπου 2 msec.
Περισσότερα βρίσκονται στο σύνδεσμο που δίνεται στο τέλος της εργασίας.



Βέλτιστος (αριθμός μηνυμάτων) / W:

Θεωρούμε ότι υπάρχει μια συσχέτιση μεταξύ του φόρτου εργασίας του επεξεργαστή (CPU cycles) και του W και ότι όσο αυξάνει το ένα θα υπάρχει κάποια αύξηση και στο άλλο.
Ιδανική, για τη μέτρηση του W, θα ήταν η χρήση ενός USB Power Meter ή κάποιας άλλη μετρητικής διάταξης ισχύος. Έχοντας όμως υπόψη τις προηγούμενες μετρήσεις της απόκρισης του συστήματός μας, διακρίνουμε την περίπτωση ενός client που στέλνει μηνύματα και την περίπτωση περισσότερων του ενός client.

- 1 client – thread: Είδαμε προηγουμένως ότι ο μέσος χρόνος απόκρισης του συστήματος ήταν 2 msec / μήνυμα. Αν λοιπόν στέλνουμε κατά μέσο όρο 500 (500 = 1/0.002) μηνύματα το δευτερόλεπτο το σύστημα θα αποδίδει το βέλτιστο και από άποψη φόρτου, αλλά και ενεργειακής κατανάλωσης.

- Many clients – threads: Το ZSUN είναι ένα μονοπύρηνο σύστημα, μη superscalar (χωρίς hyper-threading), συνεπώς η αύξηση των clients θα έχει ως αποτέλεσμα να αποκλίνουμε από το βέλτιστο αναζητούμενο λόγο.

Σχόλια-Παρατηρήσεις:

- Η σύνδεση client - server είναι μέσω WiFi.
- Θεωρούμε 99 ως το μέγιστο αριθμό χρηστών.
- Τα μηνύματα είναι σύντομα (σαν SMS), πίνακας *char mail[30]*, ώστε να μην απαιτούνται πολλοί πόροι για την αποστολή και την αποθήκευσή τους.
- Ο server χειρίζεται πολλαπλές συνδέσεις ταυτόχρονα μέσω νημάτων-threads.
- Δεν παρατηρήθηκαν προβλήματα με race conditions. Παρόλα αυτά μπορούμε να χρησιμοποιήσουμε mutex (*pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER, pthread_mutex_lock(&mutex), pthread_mutex_unlock(&mutex)*).
- Ζητήματα όπως η εγκατάσταση του OpenWrt και το cross-compiling αναφέρονται αναλυτικά στις δύο προηγούμενες εργασίες.
- Παρατηρήθηκε ότι κλείνοντας κάποιο terminal που φιλοξενεί client τερματίζει και το πρόγραμμα server, κάτι το οποίο είναι προφανώς ανεπιθύμητο. Η λύση που βρέθηκε είναι η χρήση του *Ctrl-Z* και μετά *$ bg*, *$ disown*, *$ exit.*

*****PDF της Εργασίας 1 και της Εργασίας 2**

*****Ο κώδικας της εργασίας, καθώς και άλλο υλικό βρίσκονται στο:**
https://www.dropbox.com/sh/fbmas98kdbg80uv/AACfTO-Bk1KnUv1OpAj2YXjva?dl=0