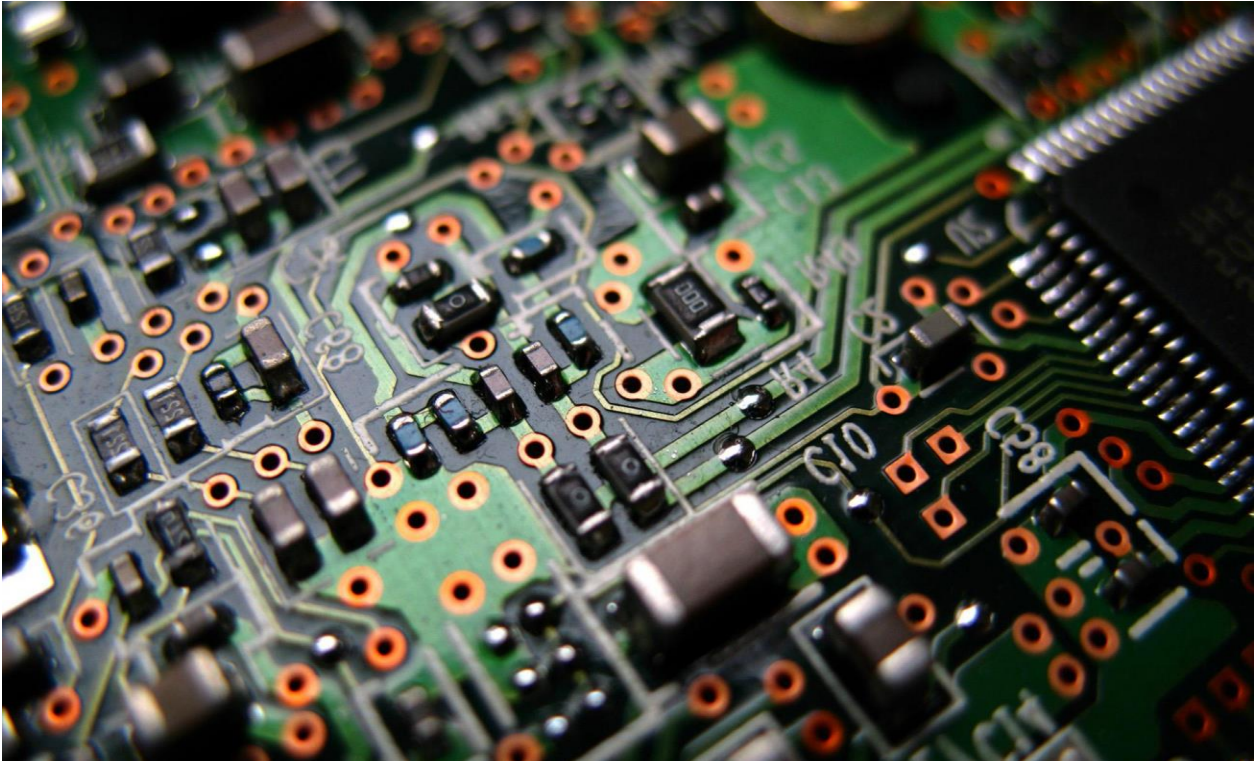


ΕΝΣΩΜΑΤΩΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ

ΕΥΑΓΓΕΛΟΣ ΤΣΙΜΠΟΥΡΗΣ 8257

1



- 1)ΕΙΣΑΓΩΓΗ
- 2)ΑΠΑΡΑΙΤΗΤΑ ΣΧΟΛΙΑ ΓΙΑ ΤΗΝ ΔΙΑΔΙΚΑΣΙΑ ΥΛΟΠΟΙΗΣΗΣ
- 3)ΥΛΟΠΟΙΗΣΗ servermm.c
- 4)ΥΛΟΠΟΙΗΣΗ clientmm.c
- 5)ΑΠΟΤΕΛΕΣΜΑΤΑ ΚΑΙ ΜΕΤΡΗΣΕΙΣ
- 6)ΣΥΜΠΕΡΑΣΜΑΤΑ

ΕΙΣΑΓΩΓΗ

Στην εκφώνηση της 3^{ης} εργασίας ζητήθηκε η ανάπτυξη μιας εφαρμογής client-server, όπου πολλοί clients από ένα τερματικό linux θα ανταλλάσσουν μηνύματα μέσω μιας εφαρμογής server που θα τρέχει στο zsun.

2

Αρχικά θα περιγράψω χωρίς πολλές λεπτομέρειες πως λειτουργεί η εφαρμογή και ποιες παραδοχές έγιναν.

- ➔ Ο client έχει τη δυνατότητα να στείλει όσα μηνύματα θέλει
- ➔ Οι clients εξυπηρετούνται μέσω μιας ουράς, δηλαδή όποιος έρθει πρώτος εξυπηρετείται και πρώτος ανεξάρτητα από το πόση ώρα θα κάνει, η πόσα μηνύματα θα στείλει.
- ➔ Ο server για κάθε client δημιουργεί ένα thread: connection handler. Τα threads συγχρονίζονται μέσω ενός counter για την υλοποίηση της ουράς εξυπηρέτησης.
- ➔ Ο client είναι αποστολέας και λήπτης των μηνυμάτων. Δηλαδή δεν στέλνει σε άλλον client μήνυμα διοτί θα χρειαζόταν την ip address και το port, πράγματα που δεν γίνεται να γνωρίζει.
- ➔ Για να μπορέσουν να γίνουν οι μετρήσεις ο client στέλνει και λαμβάνει το ίδιο μήνυμα πολλές φορές. (Αλλιώς θα έπρεπε να πληκτρολογούμε κάθε φορά άλλο μήνυμα και αυτό θα έπαιρνε υπερβολικά πολύ χρόνο).
- ➔ Η εφαρμογή του client εμφανίζει το περιεχόμενο του μηνύματος και τον αποστολέα (σε μορφή ip address) και στο τέλος εμφανίζει μετρήσεις για την ταχύτητα αποστολής των μηνυμάτων καθώς και για την επιτυχία αποστολής-λήψης.

ΑΠΑΡΑΙΤΗΤΑ ΣΧΟΛΙΑ ΓΙΑ ΤΗΝ ΔΙΑΔΙΚΑΣΙΑ ΥΛΟΠΟΙΗΣΗΣ

3

- ➔ Ο τελικός κώδικας ονομάζεται server/client mm δηλαδή multiple messages/multiple clients.
- ➔ Οι είσοδοι της εφαρμογής clientmm είναι `./clientmm 192.168.1.1 2223 100`
Όπου 1^ο όρισμα ipaddress, 2^ο port number, 3^ο αριθμός μηνυμάτων
- ➔ Οι είσοδοι της εφαρμογής servermm : `./servermm 2223`
όπου το μόνο όρισμα είναι το port number.
- ➔ Τα port numbers client/server προφανώς πρέπει να ταυτίζονται.
- ➔ Για να τρέξει ο κώδικας στο zsun μεταφέρθηκε η βιβλιοθήκη libthread στα αρχεία του zsun. (δεν έγινε static cross-compile).

ΥΛΟΠΟΙΗΣΗ servermm.c

4

- ➔ Όπως αναφέρθηκε και προηγουμένως ο server σε έναν ατέρμων βρόγχο δημιουργεί μια socket για κάθε client που συνδέεται. Στην συνέχεια την αναθέτει σε ένα thread από όπου γίνεται η ανταλλαγή μηνυμάτων.

```
for(;;){
    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
    if (newsockfd < 0)
        error("ERROR on accept");
    args.newsockfd = newsockfd;
    args.linenumber ++;
    pthread_create(&socket_thread, NULL, connectionHandler, (void *) &args);
}
```

- ➔ Στο thread περνιούνται 2 ορίσματα η socket και το linenumber. Το linenumber είναι ένας ακέραιος μετρητής που κρατάει τη θέση του client στην ουρά εξυπηρέτησης. Στην ουσία μέσω αυτής της μεταβλητής γίνεται ο συγχρονισμός των threads ως προς την εξυπηρέτηση.

- ➔ Η connection Handler είναι μια συνάρτηση που χωρίζεται σε 2 κομμάτια. Το πρώτο διαβάζει και τυπώνει από τον client την διεύθυνση του και τον αριθμό των μηνυμάτων που θέλει να στείλει.

```
n = read(newsockfd, (char*) &buflen, sizeof(buflen));
buflen = ntohl(buflen);
char addr[buflen];
bzero(addr, buflen);
n = read(newsockfd, addr, buflen);
printf("Here is the sender address: %s\n", addr);
n = read(newsockfd, &tmp, sizeof(int));
M = ntohl(tmp);
printf("Here is the number of messages: %d\n", M);
datalen = 14; // sizeof(addr) doesnt work well;
tmp = htonl(datalen);
n = write(newsockfd, (char *) &tmp, sizeof(tmp));
```

- ➔ Στο δεύτερο κομμάτι ελέγχει σε ένα ατέρμων βρόγχο αν ο αριθμός εξυπηρέτησής (ανανεώνεται με το κλείσιμο κάθε thread) ταυτίζεται με την θέση του client στην ουρά.

Σε κάθε έλεγχο χρησιμοποιείτε η εντολή `sleep(1)` έτσι ώστε να μην χρησιμοποιεί τον επεξεργαστή συνέχεια και άσκοπα καταναλώνοντας ενέργεια.

Αυτό έχει το αρνητικό βέβαια ότι περιμένει ο κάθε client ένα δευτερόλεπτο ακόμη και αν είναι άδεια η ουρά και θέλει απλά να στείλει ένα μήνυμα.

```
for(;;){
    sleep(1);
    if(linenumbr == args->servicenumbr){
```

- ➔ Όταν έρθει η σειρά εξυπηρέτησης του πελάτη η εφαρμογή διαβάζει σε κάθε loop το μήνυμα και το στέλνει πίσω στον client καθώς και τη διεύθυνση του αποστολέα. Η διαδικασία επαναλαμβάνεται όσες φορές όρισε ο client μέσω του αριθμού μηνυμάτων.

```
for(i=0;i<M;i++){
    bzero(buffer,256);
    n = read(newsockfd,buffer,255);
    if (n < 0) error("ERROR reading from socket");
    printf("Here is the message: %s\n",buffer);
    n = write(newsockfd,buffer,255);
    if (n < 0) error("ERROR writing to socket");

    n = write(newsockfd,addr,datalen);

}
close(newsockfd);
printf("i amd done with %dth customer\n", linenumbr);
args->servicenumbr ++;
pthread_exit(0);
```

- ➔ Τέλος όπως φαίνεται παραπάνω κλείνει το thread, ανανεώνεται η αριθμός εξυπηρέτησης και τυπώνει στο terminal τον αριθμό πελάτη που εξυπηρετήθηκε.

ΥΛΟΠΟΙΗΣΗ clientmm.c

- ➔ Η υλοποίηση του client είναι πιο απλή. Αρχικά συνδέεται στο zsun στην ip/port που δόθηκε και ταυτόχρονα διαβάζει το περιεχόμενο του μηνύματος. (Όπως είπαμε και παραπάνω στην περίπτωση των πολλών μηνυμάτων ο client στέλνει το ίδιο μήνυμα πολλές φορές. Αυτό γίνεται για να μπορούμε να στείλουμε πολλά μηνύματα χωρίς να πληκτρολογούμε ξανά και ξανά διαφορετικά μηνύματα).

```
if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    error("ERROR connecting");
printf("Please enter the message: ");
bzero(buffer, 256);
bzero(messagebuffer, 256);
fgets(messagebuffer, 255, stdin);
```

- ➔ Στη συνέχεια διαβάζει και στέλνει την διεύθυνση του, η οποία θα σταλεί ως η διεύθυνση αποστολέα.

```
struct ifaddrs *ifap, *ifa;
struct sockaddr_in *sa;
char *addr;
getifaddrs (&ifap);
for (ifa = ifap; ifa; ifa = ifa->ifa_next) {
    if (ifa->ifa_addr->sa_family==AF_INET) {
        sa = (struct sockaddr_in *) ifa->ifa_addr;
        addr = inet_ntoa(sa->sin_addr);
    }
}
freeifaddrs(ifap);
//first send data-length then data
int datalen = 14; //sizeof(addr) doesnt work well;
int tmp = htonl(datalen);
n = write(sockfd, (char *)&tmp, sizeof(tmp));
if (n < 0)
    error("ERROR writing to socket");
n = write(sockfd, addr, datalen);
if (n < 0)
    error("ERROR writing to socket");
```

(Ο κώδικας είναι λίγο περίπλοκος γιατί αρχικά πρέπει να μετατρέψουμε την ipaddress σε κατάλληλο τύπο, και κατά δεύτερον για να σταλεί μέσω read/write σε sockets στέλνω πρώτα το μέγεθος της και μετά το περιεχόμενο της. Εξού και τα 2 write στο τέλος).

- ➔ Στη συνέχεια η εφαρμογή στέλνει τον αριθμό μηνυμάτων που θέλει να στείλει

```
// write number of messages
tmp = htonl(M);
n = write(sockfd,&tmp,sizeof(int));
if (n < 0)
    error("ERROR writing to socket");
```

- ➔ Μετά σε έναν βρόγχο στέλνει το μήνυμα στον server. Ταυτόχρονα διαβάζει και τυπώνει τα μηνύματα που του έστειλε ο server καθώς και τη διεύθυνση του αποστολέα. (Όπως αναφέρθηκε αποστολέας/λήπτης των μηνυμάτων ταυτίζεται άρα στέλνει και διαβάζει τα δικά του μηνύματα και διεύθυνση).

```
for(i = 0;i< M;i++){
    n = write(sockfd,messagebuffer,strlen(messagebuffer));
    if (n < 0)
        error("ERROR writing to socket");
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    if (n < 0)
        error("ERROR reading from socket");
    printf("You have a message: %s\n",buffer);

    //n = read(sockfd, (char*)&buflen,sizeof(buflen));
    //buflen = ntohl(buflen);
    //char addr[buflen];
    bzero(address,buflen);
    n = read(sockfd,address,datalen);
    printf("Send by: %s\n\n",address);

    msgrecvd ++;
}
```

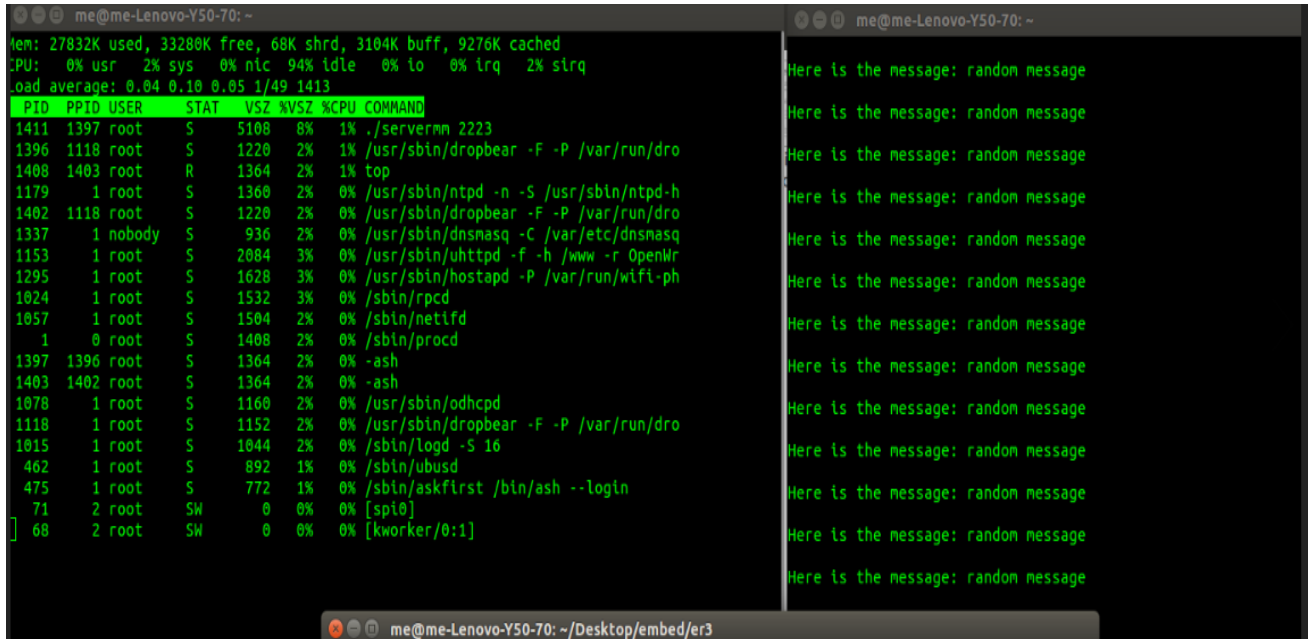
- ➔ Τέλος τυπώνει διάφορες μετρήσεις που έκανε για την ταχύτητα λήψης των μηνυμάτων καθώς και για το εάν δεν στάλθηκε κάποιο από αυτά.

```
printf("messages received = %d \n", msgrecvd);
printf("messages lost = %d \n", M - msgrecvd);
printf("time to receive messages = %f \n", mseconds/1000);
printf("time / number of messages = %f / %d = %f \n", mseconds/1000, M, mseconds/(1000*M));
return 0;
```

ΑΠΟΤΕΛΕΣΜΑΤΑ ΚΑΙ ΜΕΤΡΗΣΕΙΣ

Αρχικά θα παρουσιάσω την χρήση του CPU στο zsun κατά τη διάρκεια αποστολής/λήψης μηνυμάτων από την εφαρμογή servermm.

8



```
mem: 27832K used, 33280K free, 68K shrd, 3104K buff, 9276K cached
CPU:  0% usr  2% sys  0% nic 94% idle  0% io  0% irq  2% srq
load average: 0.04 0.10 0.05 1/49 1413
```

PID	PPID	USER	STAT	VSZ	VSZ%	%CPU	COMMAND
1411	1397	root	S	5108	8%	1%	./servermm 2223
1396	1118	root	S	1220	2%	1%	/usr/sbin/dropbear -F -P /var/run/dro
1408	1403	root	R	1364	2%	1%	top
1179	1	root	S	1360	2%	0%	/usr/sbin/ntpd -n -S /usr/sbin/ntpd-h
1402	1118	root	S	1220	2%	0%	/usr/sbin/dropbear -F -P /var/run/dro
1337	1	nobody	S	936	2%	0%	/usr/sbin/dnsmasq -C /var/etc/dnsmasq
1153	1	root	S	2084	3%	0%	/usr/sbin/uhttpd -f -h /www -r OpenWr
1295	1	root	S	1628	3%	0%	/usr/sbin/hostapd -P /var/run/wifi-ph
1024	1	root	S	1532	3%	0%	/sbin/rpcd
1057	1	root	S	1504	2%	0%	/sbin/netifd
1	0	root	S	1408	2%	0%	/sbin/procd
1397	1396	root	S	1364	2%	0%	-ash
1403	1402	root	S	1364	2%	0%	-ash
1078	1	root	S	1160	2%	0%	/usr/sbin/odhcpd
1118	1	root	S	1152	2%	0%	/usr/sbin/dropbear -F -P /var/run/dro
1015	1	root	S	1044	2%	0%	/sbin/logd -S 16
462	1	root	S	892	1%	0%	/sbin/ubusd
475	1	root	S	772	1%	0%	/sbin/askfirst /bin/ash --login
71	2	root	SW	0	0%	0%	[spi0]
68	2	root	SW	0	0%	0%	[kworker/0:1]

```
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
Here is the message: random message
```

Αριστερά βλέπουμε τις μετρήσεις της εντολής top στο terminal του zsun, δεξιά βλέπουμε ότι την ώρα των μετρήσεων τυπώνωνταν τα μηνύματα που λάμβανε ο server.

Φαίνεται ότι η χρήση του cpu είναι μόλις 1%. Αυτό εξηγείται λόγω του ξοδεύεται πολύ περισσότερος χρόνος στην αποστολή/λήψη των μηνυμάτων παρά στην εκτέλεση των εντολών του κώδικα. Δηλαδή έχουμε μια αποδοτική από άποψη χρήσης cpu εφαρμογή.

Στη συνέχεια θα δούμε τις μετρήσεις για την ταχύτητα λήψης για διαφορετικό αριθμό μηνυμάτων την κάθε φορά.

```
messages received = 1
messages lost = 0
time to receive messages = 1.091453
time / number of messages = 1.091453 / 1 = 1.091453
```

9

```
messages received = 5
messages lost = 0
time to receive messages = 1.260880
time / number of messages = 1.260880 / 5 = 0.252176
```

```
messages received = 50
messages lost = 0
time to receive messages = 2.981213
time / number of messages = 2.981213 / 50 = 0.059624
```

```
messages received = 200
messages lost = 0
time to receive messages = 9.190723
time / number of messages = 9.190723 / 200 = 0.045954
```

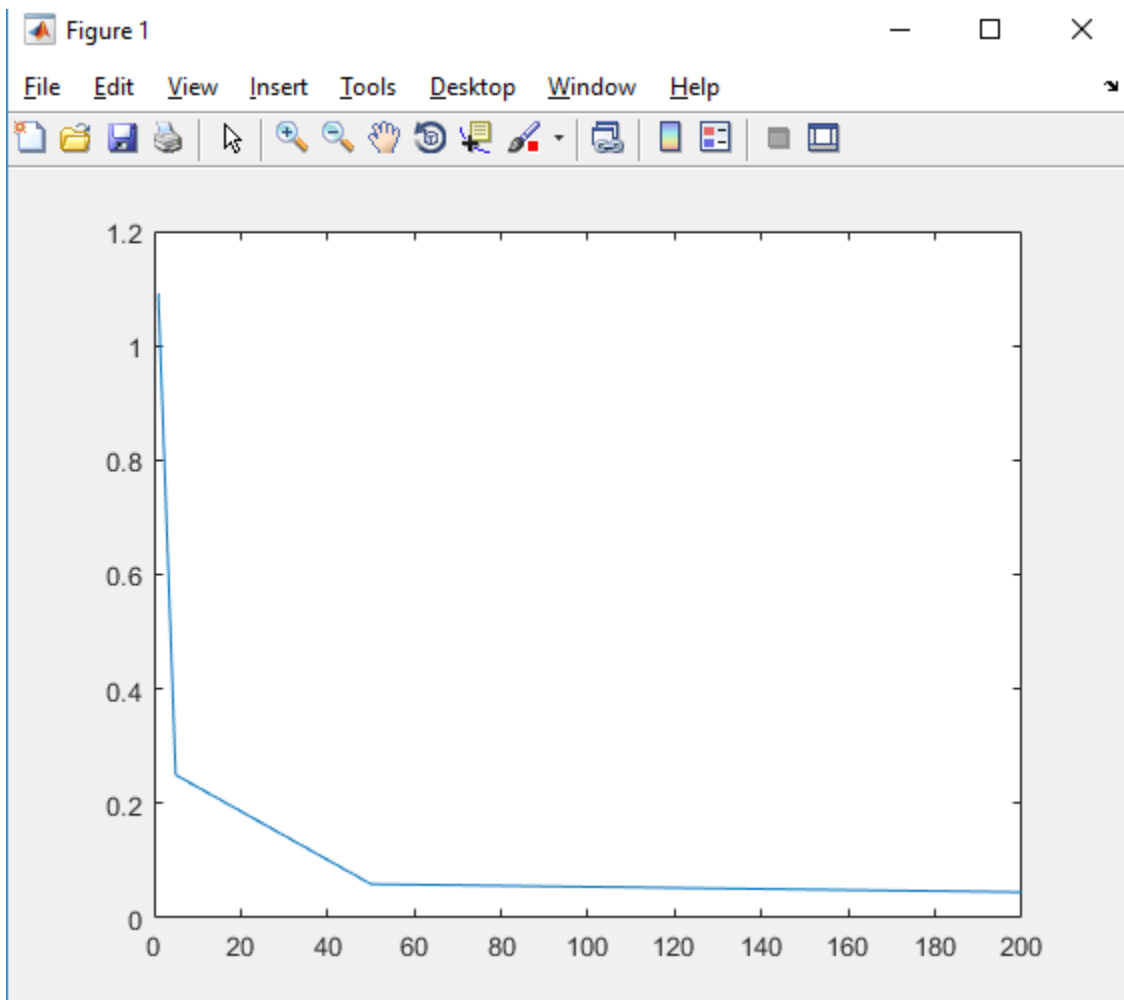
```
messages received = 500
messages lost = 0
time to receive messages = 23.864128
time / number of messages = 23.864128 / 500 = 0.047728
```

```
messages received = 1000
messages lost = 0
time to receive messages = 41.736294
time / number of messages = 41.736294 / 1000 = 0.041736
```

```
messages received = 4000
messages lost = 0
time to receive messages = 169.515610
time / number of messages = 169.515610 / 4000 = 0.042379
```

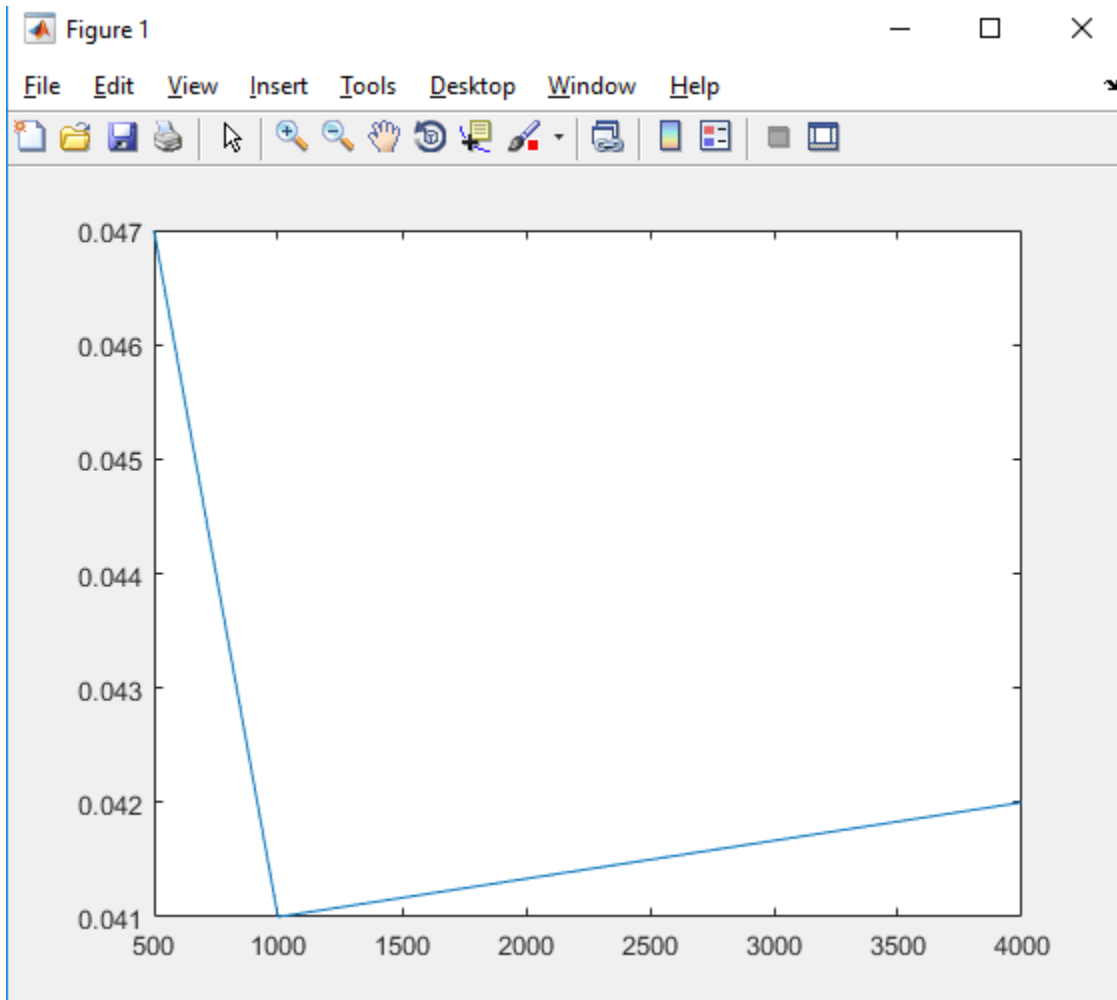
Βλέπουμε όσο αυξάνονται τα μηνύματα ο μέσος χρόνος λήψης μηνύματος μειώνεται δραματικά.

Ας το δούμε και διαγραμματικά για τις μικρότερες τιμές αριθμού μηνυμάτων:



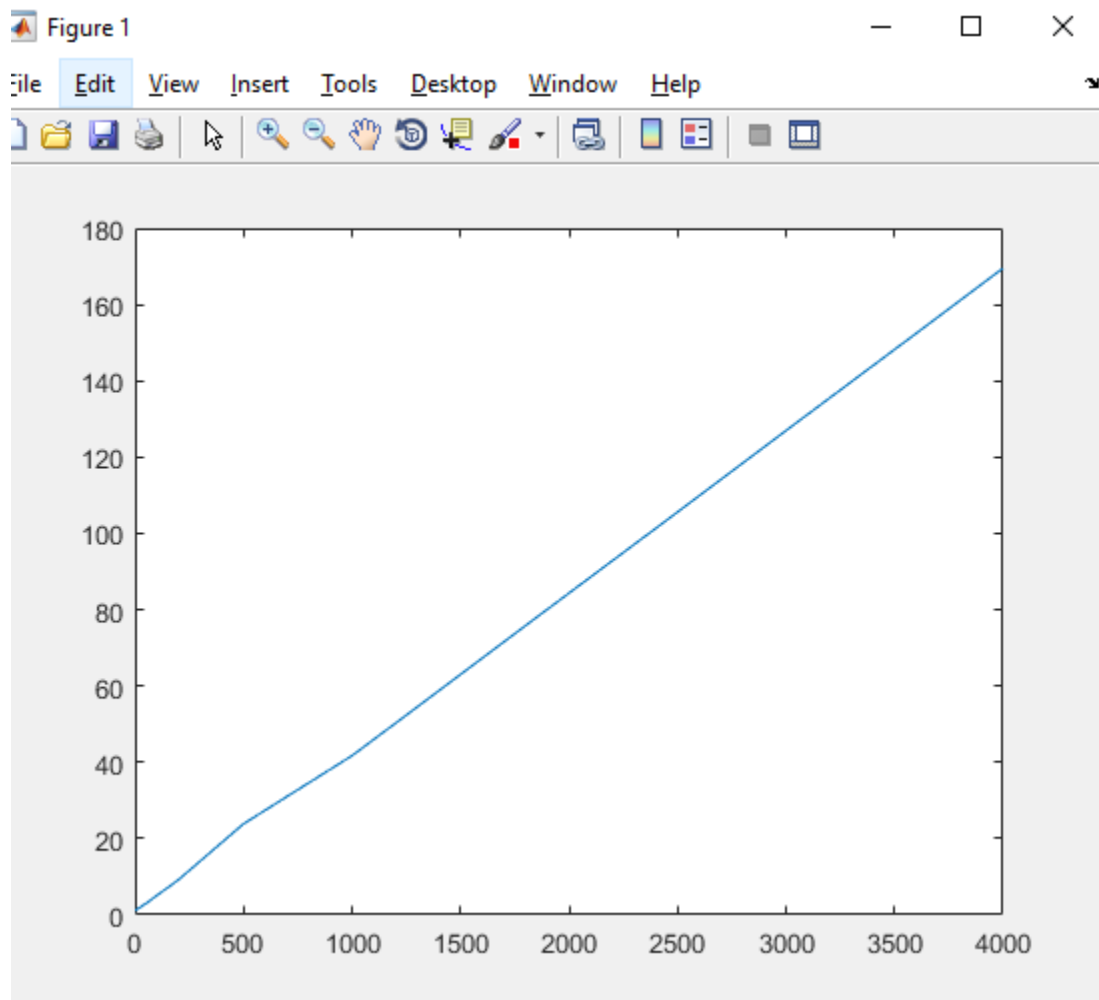
Πράγματι ο χρόνος που χρειάζεται για να σταλεί ένα μήνυμα είναι πολύς (εδώ παίζει μεγάλο ρόλο και η `sleep(1)`). Άρα η εφαρμογή είναι πιο αποδοτική για μεγαλύτερο όγκο μηνυμάτων.

Και για 500-4000:



Η διακύμανση του Y εδώ είναι φυσικά μικρότερη. Πάντως φαίνεται ότι έχουμε ελάχιστο στα 1000 μηνύματα.

Σε συνολικό χρόνο:



Βλέπουμε ότι για μεγάλο αριθμό μηνυμάτων η σχέση $(total_time - number_of_messages)$ είναι γραμμική.

Τέλος θα διερευνήσουμε πως συμπεριφέρεται η εφαρμογή για πολλούς clients συνδεδεμένους ταυτόχρονα. Σκοπός είναι να δούμε κυρίως αν τηρείται η ουρά προτεραιότητας.

```

me@me-Lenovo-Y50-70: ~/Desktop/embed/er3
me@me-Lenovo-Y50-70:~/Desktop/embed/er3$ ./clientmm 192.168.1.1 2223 1
Please enter the message: 
me@me-Lenovo-Y50-70:~/Desktop/embed/er3$ ./clientmm 192.168.1.1 2223 1
Please enter the message: 
me@me-Lenovo-Y50-70:~/Desktop/embed/er3$ ./clientmm 192.168.122.1 2223 1
ERROR connecting: Connection refused
me@me-Lenovo-Y50-70:~/Desktop/embed/er3$ ./clientmm 192.168.1.1 2223 1
Please enter the message: 
  
```

Εδώ βλέπουμε πολλαπλά client-terminals να έχουν συνδεθεί με το zsun.

```

me@me-Lenovo-Y50-70:~/Desktop/embed/er3$ ./clientmm 192.168.1.1 2223 1
Please enter the message: asasaas
me@me-Lenovo-Y50-70:~/Desktop/embed/er3$ ./clientmm 192.168.1.1 2223 1
Please enter the message: asadasdsa
me@me-Lenovo-Y50-70:~/Desktop/embed/er3$ ./clientmm 192.168.122.1 2223 1
ERROR connecting: Connection refused
me@me-Lenovo-Y50-70:~/Desktop/embed/er3$ ./clientmm 192.168.1.1 2223 1
Please enter the message: 
  
```

Εδώ έχει ενδιαφέρον ότι ενώ οι 2 τελευταίοι clients έχουν συνδεθεί και στείλει το κείμενο του μηνύματος τους δεν εξυπηρετούνται διότι λόγω της ουράς περιμένουν τον πρώτο να στείλει το δικό του κείμενο.

```
i amd done with 2th customer  
Here is the message: asasaas  
  
i amd done with 3th customer  
Here is the message: asadasddsa  
  
i amd done with 4th customer  
█
```

Τέλος βλέπουμε τι τυπώνει ο server. Όπως φαίνεται εξυπηρέτησε τους πελάτες με τη σωστή σειρά.

Προκύπτει λοιπόν ότι υποστηρίζεται η εξυπηρέτηση πολλών πελατών ταυτόχρονα και ότι η ουρά προτεραιότητας λειτουργεί άψογα.

Τέλος μετρήθηκε και η κατανάλωση ενέργειας(στο περίπου),
συνδέοντας το zsun σε ένα powerbank χωρητικότητας
10.000 mah.

15



Η κατανάλωση βρέθηκε ότι είναι το $\frac{1}{4}$ του powerbank για 1 ώρα
συνεχής χρήσης. Περίπου 2500 mah/hour. Αρκετά ενεργοβόρο
δηλαδή παρά την πρόβλεψη που έγινε για low cpu usage.
Προφανώς παίζει ρόλο και η κατανάλωση του πομπού/δέκτη.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Η εφαρμογή που αναπτύχθηκε καλύπτει όλες τις προδιαγραφές που είχαν ζητηθεί στην εκφώνηση.

Αναλυτικά:

- ➔ Ο server χειρίζεται πολλαπλές συνδέσεις ταυτόχρονα.
- ➔ Λειτουργική ουρά εξυπηρέτησης
- ➔ Μεγάλος όγκος μικρού σε μέγεθος μηνυμάτων

Τέλος παρουσιάστηκαν όλα τα στατιστικά στοιχεία – έλεγχοι που ζητήθηκαν , και προέκυψαν και τα απαραίτητα συμπεράσματα.
(Βέλτιστη λειτουργία, κατανάλωση cpu,ενέργειας)