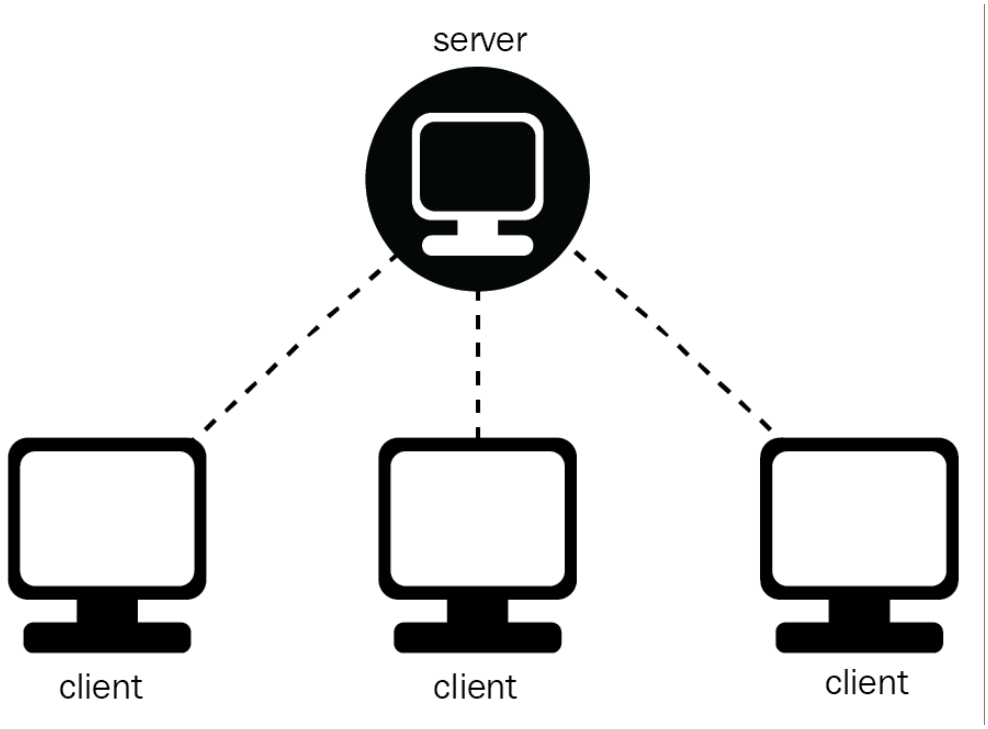
Τμήμα Μηχ. Η/Υ & Πληροφορικής Εαρινό 2019
Πανεπιστήμιο Ιωαννίνων

ΜΥΥ601 Λειτουργικά Συστήματα

ΟΜΑΔΑ
1)Χαρίσης Γκέκας
2)Δημήτριος Σαμουρέλης

Εργαστήριο 1:Υλοποίηση ενός απλού πολύνηματικού διακομιστή αποθήκευσης ζευγών κλειδιού-τιμής



Η **βασική εργασία** της άσκησης μας είναι να οικοδομήσουμε τον πολυνηματικό διακομιστή αποθήκευσης κλειδιού-τιμής. Ο διακομιστής μας έχει δομή παραγωγού-καταναλωτή.Στο αρχείο server.c στην συνάρτηση main εκμεταλευόμαστε το νήμα της και υλοποίουμε τη

δομή του παραγωγού.Δημιουργούμε επίσης τα νήματα του καταναλωτή **pthread_create(&consumers[i],NULL,consumers_serve,NULL)** έχοντας επιπλέον αρχικοποιήσει μια global μεταβλητή **threads_num.**
Το νήμα (ένα νήμα) παραγωγού περιμένει για εισερχόμενες αιτήσεις σύνδεσης. Όταν ο διακομιστής λάβει μία νέα σύνδεση, το νήμα παραγωγού προσθέτει σε μια κοινόχρηστη ουρά μία περιγραφή της σύνδεσης **int new_connection_fd**; και ενα **struct timeval starting_time**; οπού αναφέρεται στο χρόνο έναρξης της σύνδεσης δημιουργώντας έτσι ένα struct :**description_t queue[queue_size].**

Δομή ουράς(από την πλευρά του παραγωγού): Όταν έρχεται μία νέα σύνδεση ο παραγωγός προσθέτει στην δομή ουράς αυξάνοντας τον δείκτη **tail** κατά 1 tail=(tail+1)%queue_size (κυκλική δομή ουράς π.χ οτάν φτάνει στη θέση 29 ο δείκτης tail να πηγαίνει στην αρχή)για κάθε νέα σύνδεση. Αρχικά το tail=0 (global μεταβλητή) . Η ουρά αιτήσεων είναι προσβάσιμη σε όλα τα νήματα του διακομιστή. Προκειμένου να διατηρηθεί η ουρά σε συνεπή κατάσταση, μόνο ένα νήμα επιτρέπεται να την προσπελάζει κάθε φορά. Για να το διασφαλίσουμε αυτό προσθέσαμε στον παραγωγό κλειδαρια **pthread_mutex_t mutex_queue pthread_mutex_unlock(&mutex_queue).** Ανάμεσα στον κώδικά τους (κρίσιμη περιοχή) έχουμε προσθέσει κώδικα συγχρονισμού για τις συνθήκες άδειας και γεμάτης ουράς. Έχουμε αρχικοποιήσει τις **pthread_cond_t cond_non_full_queue pthread_cond_t cond_non_empty_queue** (μεταβλητές συνθήκης) . Επίσης έχουμε αρχικοποίσει την global μεταβλητή **head**=0 . Όσο η διαφορά tail – head == queue_size (δηλαδή όσο η ουρά είναι γεμάτη) ο παράγωγός κάνει wait μέχρι να λάβει από τον καταναλωτή το **signal cond_non_full_queue** δηλαδή μέχρι να τον ενημερώσει ο άλλος οτι δεν ειναί γεμάτη η ουρά . Επίσης έχουμε και την **gettimeofday** για να πάρουμε τον χρόνο έναρξης της κάθε σύνδεσης .Εδώ θα καλέσουμε την **add_to_queue.** Αφού αυξήσω το tail ειδοποιώ με **signal cond_non_empty_queue** οτι δεν είμαι άδειο (δηλαδή ότι έχω request να εξυπηρέτησει ο καταναλώτης ο οποίος περίμενει οσό η ουρά ειναί άδεια) . Αφού έχω νέα σύνδεση ειδοποιώ τον καταναλωτή ότι δεν είμαι αδείος (**cond_non_empty_queue**). Στην συνέχεια κάνω unlock για να βγω από την κρίσιμη περιοχή **pthread_mutex_unlock(&mutex_queue).**Ένα νήμα δεσμεύει κάποιους (περιορισμένους σε σχέση με μία διεργασία) πόρους στο σύστημα. Όταν ένα νήμα τερματίσει οι πόροι του πρέπει να αποδεσμευτούν. Η pthread_join() μπλοκάρει μέχρι το νήμα να τερματίσει. **pthread_join(consumers[i],NULL).**

Πλευρά του καταναλώτη (void *consumer(void *arg)) .Εδώ δημιουργούμε την μέθοδο του πολυνηματικού καταναλωτή και τις διεργασίες που θα εκτελέσει .Καταρχήν αρχικοποιούμε μια μεταβλητή **int new_connection_fd** την οποία θα την χρειαστούμε σαν όρισμα στην **process_request**(παίρνει σαν όρισμα μια **int socket_fd**) όπου επεξεργάζεται το client request.Συνεχίζούμε μπαίνοντας στον βρόγχο while που εκτελείται συνεχώς.Βάζουμε το **mutex_queue** για να προστατέυσουμε την κοινόχρηση ουρά μου αφού θέλουμε να την εκτελούν ενα ενα τα νήματα.Έπειτα βάζουμε μέσα στον βρόγχο while την συνθήκη tail-head
(τις οποίες tail και head τις έχουμε θέσει σαν global και τις βάλαμε να δείχνουν στην αρχή του πίνακα).Όπου η αφαίρεση του **tail-head** εαν ισούται με το μηδέν σημαίνει ότι είναι άδεια.Μέσα σε αυτόν τον βρόγχο χρησιμοποιούμε το σήμα wait με όρισμα την μεταβλητή συνθήκης
cond_non_empty_queue και το mutex της ουράς **mutex_queue**.Εδω κάνουμε wait διότι εαν είναι άδεια η ουρά ο καταναλωτής δεν μπορεί να κάνει κάτι οπότε περιμένει ενα signal απο τον παραγωγό που θα τον ειδοποιήσει πως προστέθηκε request στην ουρά αρα συνέχισε την δουλεία σου.Με το που βγαίνουμε απο τον βρόγχο αυξάνουμε το head κατά 1 για τον λόγο οτι απομακρύνουμε ενα request (head=(head+1)%queue_size για να

γυρίζει στην αρχή της ουράς) .Στην συνέχεια υπάρχει το if το οποίο μας δείχνει εαν έχουμε φτάσει στο τέλος της ουράς ώστε να ξαναγυρίσουμε στην αρχή κυκλικά.Αφού αύξησαμε το head, η επόμενη υποχρέωση που έχουμε είναι να θέσουμε το **new_connection_fd** ίσο με **queue[tail – head].connection_fd**.Τώρα στην μεταβλητή αυτήν έχουμε τον περιγραφέα αρχείου σύνδεσης πελάτη.Στις δύο global Μας μεταβλητές **secs** και **usecs(**μέσα στο lock της **mutex_for_time**,διότι είπαμε οτι θέλουμε ενα mutex για όλους τους χρόνους) αποθηκεύουμαι μέσω της ουράς απο struct την ώρα έναρξης της σύνδεσης σε seconds kai mseconds αντίστοιχα.Μετά κάνουμε signal με όρισμα την μεταβλητή συνθήκης **cond_non_full_queue**
στον παραγώγο που είναι σε κατάσταση wait όταν είναι γεμάτος,δηλαδή εννούμε ότι εμείς σαν καταναλωτής επεξεργαστήκαμε ένα request εσύ(παραγωγός) δεν είσαι πια γεμάτος ,μπορείς να συνεχίσεις οτι κάνεις.Το επόμενο βήμα ειναι να κάνουμε unlock το mutex της ουράς **pthread_mutex_unlock(&mutex_queue)**,αφού προστατεύσαμε τους κοινόχρηστους πόρους μας.Μας ζητήθηκε να βρούμε τον χρόνο επεξεργασίας των request,με την βοήθεια της συνάρτησης **gettimeofday** με όρισματα **before_execute,NULL,** αφού θέλουμε να βρούμε τον χρόνο επεξεργασίας ετοίματος (process_request) και από κάτω
gettimeofday με ορίσματα **after_execute,NULL.**Βάζουμε **pthread_mutex_lock** με όρισμα
mutex_for_time για να προστατεύσουμε τους υπολογισμους των χρόνων μας.Εδώ κάνουμε την πράξη **total_waiting_time** += (**before_execute.tv_sec***1000000 + **before_execute.tv_usec**) -(**secs***1000000 – **usecs**);.*(επί 1000000 για την μετατροπή των δευτερολέπτων σε μδευτερολεπτα.Θέσαμε για ευκολία (**sec_to_usec 1000000**).Βρίσκουμε το **total_service_time** με παρόμοιο τρόπο.Κάθε φορά που εισερχόμαστε στο βρογχο while προσθέτουμε +1 στο **completed_requests**.Αφού τελειώσαμε με τους υπολογισμούς των κοινόχρηστων μεταβλητών ήρθε η ώρα να ξεκλειδώσουμε **pthread_mutex_unlock(&mutex_for_time)**;.Επιστρέφουμε στον πελάτη το αποτέλεσμα της λειτουργίας και τερματίζουμε.

Για την add_to_queue(int new_fd , struct timeval starting_time) :

Αποθηκεύω στη δομή ουράς στην θέση **tail-head** τον περιγραφέα της σύνδεσης και το χρόνο έναρξης (π.χ για tail= 0 αρχικα θα αποθηκεύσω στην θέση 0 στη δομή ουράς και όσο ο καταναλώτης δεν εξυπηρετεί τα request η μεταβλητή tail θα αυξάνεται) . Αποθηκέυω τον χρόνο έναρξης σε δύο πεδία του struct timeval το **starting_time.tv_sec** και το **starting_time.tv_usec** για τον υπολογισμό των χρόνων αργότερα.

Για το πρόβλημα των readers-writers :
Εδω υλοποιούμε μια απλή λύση συγχρονισμού αναγνώστων-γραφέων χρησιμοποιώντας pthreads.Στην συνάρτηση **process_request** πηγαίνουμε στην **case GET** όπου πραγματοποιείται η ανάγνωση.Χρησιμοποιούμε την
pthread_mutex_lock(&mutex_put_get) για να εξασφαλίσουμε οτι θα εκτελεστεί απο ένα νήμα κάθε φορά.Έχουμε βάλει εναν βρόγχο που τρέχει όσο το writer_count ! = 0 θα κάνουμε wait με την συνάρτηση
pthread_cond_wait(&cond_enter,&mutex_put_get) .Δηλαδή για όσο δεν δεν γράφουμε περιμένουμε.Έπειτα αυξάνω την μεταβλήτη **reader_count** και ξεκλειδωνω το **mutex pthread_mutex_unlock(&mutex_put_get) .**Τώρα συμβαίνει η διαδικασία του διαβάσματος.Κλειδώνουμε με την **pthread_mutex_lock(&mutex_put_get)** και χρησιμοποιώ σαν όρισμα την **mutex_put_get** ξανά αφού είπαμε οτι θέλουμε ενα mutex για readers-writers.Μειώνουμε την reader_count αφου κάναμε ότι θέλαμε να

κάνουμε.Έπειτα καλώ την **pthread_cond_signal(&cond_enter)** για να "ξυπνίσουμε" την αντίστοιχη wait που έχουμε στο case PUT
thread_cond_wait(&cond_enter,&mutex_put_get); .Και τελικά κάνω unlock **pthread_mutex_unlock(&mutex_put_get);** αφού την προστατευσαμε επιτυχώς.
case PUT:Κλειδώνουμε με **pthread_mutex_lock(&mutex_put_get)(**Ξανά με ίδιο όρισμα αφου θέλουμε να χρησιμοποιήσουμε το ίδιο για όλα.Μπαίνουμε στον βρόγχο while με συνθήκη reader_count || writer_count δηλαδή για όσο reader_count !=0 ή writer_count !=0 είμαστε σε κατάσταση αναμονής.Μόλις βγω απο τον βρόγχο αυξάνω την μεταβλητή μου writer_count και κάνουμε το **pthread_mutex_unlock(&mutex_put_get),**αφού ασφαλίσαμε την μεταβλητή μας write_count.Τώρα έχουμε την λειτουργία του γραψίματος .Όταν τελειώσει το γράψιμο κάνουμε **pthread_mutex_lock(&mutex_put_get)** ,αυξανουμε τον writer_count κάνουμε **pthread_cond_signal(&cond_enter)**,για να δηλώσουμε οτι τελείωσαμε το γράψιμο αφου μειώσαμε την writer_count ωστε ο βρογχος που έχουμε πανω στην case GET να σταματήσει να ισχύει και να βγούμε.Τελειώνοντας κάνουμε **pthread_mutex_unlock(&mutex_put_get).**
Παρατηρηση :Έχουμε δώσει προτεραιότητα στους **readers**.

Για τον υπολογισμό των χρόνων έχουμε :

1)**long compute_average_total_waiting_time**

2)**long compute_average_total_service_time** οι οποιες παίρνους ως ορίσματα **long total_waiting_time,int completed_requests**.

Η πρώτη υπολογίζει το **average_total_waiting_time** διαιρώντας το **total_waiting_time** με τον αριθμό των εκτελεσμένων request (**completed_requests**).
Η δεύτερη υπολογίζει το **average_total_service_time** διαιρώντας το **total_service_time** με τον αριθμό των εκτελεσμένων request (completed_requests).

Η **void my_handler(int sig)** τυπώνει 1)**Completed requests** 2)**Average total waiting time** 3)**Average total service time**. Επίσης έχουμε προσθέσει στην main την κλήση της **my_handler** όταν λάβει το **ctrl + z signal(SIGTSTP,my_handler);**

Εντολές για την διαδικασία τρεξίματος ./client -a localhost -i 1 -p →για αποθήκευση και ./client -a localhost -i 1 -g για ανάγνωση.

queue_size = 10

ΝΗΜΑΤΑ

1 Average total waiting time in usecs: 1785600.00
Average total service time in usecs: 52.00
2
Average total waiting time in usecs: 1274053.00
Average total service time in usecs: 66.00
3
Average total waiting time in usecs: 135682.00
Average total service time in usecs: 62.00
5
Average total waiting time in usecs: 915426.00

Average total service time in usecs: 62.00

10
Average total waiting time in usecs: 370060.00
Average total service time in usecs: 67.00
15
Average total waiting time in usecs: 1785079.00
Average total service time in usecs: 67.00
20
Average total waiting time in usecs: 175804.00
Average total service time in usecs: 62.00

25 Average total waiting time in usecs: 1468450.00
Average total service time in usecs: 63.00

30
Average total waiting time in usecs: 279218.00
Average total service time in usecs: 63.00

Παρατηρώ οτι για αριθμό νημάτων = 3 στην αρχική φάση φόρτωσης ο χρονός αναμονής ειναί ο μικρότερος και επίσης οτι ο χρόνος εξυπηρέτησης για ένα νήμα ειναί ο μικρότερος.

queue_size = 20;

NHMATA

1
Average total waiting time in usecs: 1283352.00
Average total service time in usecs: 63.00
2
Average total waiting time in usecs: 761138.00
Average total service time in usecs: 66.00
3
Average total waiting time in usecs: 1736909.00
Average total service time in usecs: 67.00
5
Average total waiting time in usecs: 838148.00
Average total service time in usecs: 60.00
10
Average total waiting time in usecs: 1434172.00
Average total service time in usecs: 64.00
15
Average total waiting time in usecs: 279479.00

Average total service time in usecs: 55.00
20
Average total waiting time in usecs: 1505022.00
Average total service time in usecs: 53.00
25
Average total waiting time in usecs: 173568.00
Average total service time in usecs: 68.00
30

Average total waiting time in usecs: 1080846.00
Average total service time in usecs: 52.00

Παρατηρώ ότι για αριθμό νημάτων=25 ο μέσος χρόνος αναμονής είναι ο μικρότερος ενώ ο χρόνος εξυπηρέτησης είναι μικρότερος για 30 νήματα.

queue_size = 30;

ΝΗΜΑΤΑ

1
Average total waiting time in usecs: 309574.00
Average total service time in usecs: 65.00
2
Average total waiting time in usecs: 213057.00
Average total service time in usecs: 64.00
3
Average total waiting time in usecs: 230958.00
Average total service time in usecs: 62.00

5
Average total waiting time in usecs: 663105.00
Average total service time in usecs: 56.00
10
Average total waiting time in usecs: 959775.00
Average total service time in usecs: 58.00
15
Average total waiting time in usecs: 360201.00
Average total service time in usecs: 71.00
20
Average total waiting time in usecs: 193735.00
Average total service time in usecs: 52.00
25
Average total waiting time in usecs: 388218.00
Average total service time in usecs: 71.00
30
Average total waiting time in usecs: 978350.00
Average total service time in usecs: 60.00

Παρατηρώ ότι για αριθμό νημάτων=20 ο μέσος χρόνος αναμονής είναι ο μικρότερος

ενώ ο χρόνος εξυπηρέτησης είναι μικρότερος για 5 νήματα.

Οπότε συνολίκα για την αρχική φόρτωση για queue_size = 10 ο server είναι πιο γρήγορος με αριθμό νημάτων = 3 , για queue_size = 20 ο server είναι πιο γρήγορος με αριθμό νημάτων = 20 , για queue_size = 30 ο server είναι πιο γρήγορος με αριθμό νημάτων = 25. Καθώς μεγαλώνει το μέγεθος της ουράς αυξάνονται οι απαιτήσεις για νήματα για να έχουμε καλύτερη απόδοση του server. Επίσης παρατηρούμε οτι για σταθερό αριθμό νημάτων π.χ νήματα =2 αυξάνοντας το μέγεθος της ούρας έχουμε καλύτερος χρόνους εξυπηρέτησης και ο εξυπηρέτης είναι πιο αποδοτικός .
Καθώς αυξάνουμε τον αριθμό των νημάτων οι αιτήσεις εξυπηρετούνται πιο γρήγορα .
Επίσης για μέγεθος ουράς= 10 καθώς αυξάνεται ο αριθμός των νημάτων ο χρόνος εξυπηρετησης μειώνεται και ο εξυπηρέτης γίνεται πιο αποδοτικός .

queue_size = 30 παράλληλα put- get από εναν client με σταθερό μέγεθος ούρας με την εντολή **./client -a localhost -i 1 -p get & ./client -a localhost -i 1 -g get** για να ελέγξουμε τον συγχρονισμό αναγνωστών γραφέων . **Δεν έχουμε τροποποιήσει τον πέλατη να στέλνει πολλαπλές αιτήσεις ταυτόχρονα για τον έλεγχο του κώδικά μας .**

ΝΗΜΑΤΑ

10
Average total waiting time in usecs: 1912694.00
Average total service time in usecs: 47.00

15
Average total waiting time in usecs: 1207021.00
Average total service time in usecs: 50.00

20
Average total waiting time in usecs: 1557221.00
Average total service time in usecs: 50.00

25
Average total waiting time in usecs: 1369088.00
Average total service time in usecs: 49.00

30
Average total waiting time in usecs: 1257572.00
Average total service time in usecs: 56.00

Παρατήρω ότι όσο αυξάνονται τα νήματα μείωνεται ο χρόνος εξυπηρέτησης για σταθερό μέγεθος ουράς = 30 .