ΜΥΥ601
Λειτουργικά Συστήματα
Πολυνηματικός server
(2018 Εαρινό εξάμηνο)

Μήτος-Κώτσης Αντώνης (Α.Μ. 3028)
Συμεωνίδης Θεόδωρος (Α.Μ. 3083)

Λειτουργία

Πρόκειται για μια υλοποίηση παραγώγου-καταναλωτή με έναν server ο οποίος μπορεί να εκτελεί ταυτόχρονα πολλαπλές αιτήσεις .

Ο client αποστέλει αιτήσεις στον server από όπου τις δέχετε ο παραγωγός(main function),αυτός στην συνέχεια είναι υπεύθυνος να εισάγει τις αιτήσεις σε μια δομή τύπου ουράς και να στείλει σήμα στον καταναλωτή οτι υπάρχουν αιτήσεις προς εκτέλεση.

Ο καταναλωτής από την άλλη αποτελείται από ένα σύνολο νημάτων,αποθηκευμένα σε έναν πίνακα ,τα οποία είναι υπεύθυνα για την λήψη και εκτέλεση των αιτήσεων από την ουρά.

Μόλις ένα νήμα εξάγει μια αίτηση από την ουρά τότε το πρώτο βήμα που γίνεται είναι, η αναγνώρισει του είδους της αίτησης. Μετά από αυτό, και αφού τηρηθούν οι κανόνες προτεραιότητας των αιτήσεων,το νήμα οδεύει προς την εκτέλεση της αίτησης. Τέλος αφού η εκτέλεση τερματιστέι με επιτυχία το νήμα πρέπει να αποστέιλει το αποτέλεσμα πίσω στον client και να επιστρέψει στον πίνακα ώστε να εκτελέσει μια νέα αίτηση.

Όταν ο χρήστης αποφασίσει να τερματίσει το πρόγραμμα στέλνωντας σήμα τερματισμού (SIGTSTP) τότε ο server περιμένει την λήξη της εκτέλεσης όλων των νημάτων , εμφανίζει τον μέσο χρόνο αναμονής των αιτήσεων στην ουρά ,τον μέσο χρόνο εκτέλεσης μιας αίτησης και τον συνολίκο αριθμό αιτήσεων που υλοποιήθηκαν και μετά τερματίζει.

Δομές και Συναρτήσεις

Οι συναρτήσεις που υλοποίησαμε μαζί με την χρήση τους αναφαίρονται παρακάτω:

- thread_start: Είναι η συνάρτηση με την οποία ξεκινούν την λειτουργία τους όλα τα νήματα καταναλωτή .
- check_bounds: Ελέγχει τις μεταβλητές head και tail αν ξεπερνούν τα όρια της ουράς αιτήσεων και αν το κάνουν τις επαναφέρουν στην αρχή της.
- is_Full: Ελέγχει αν η ουρά των αιτήσεων είναι γεμάτη και επιστρέφει 1 ,αλλιώς επιστρέφει 0.
- is_Empty: Ελέγχει αν η ουρά των αιτήσεων είναι άδεια και επιστρέφει 1 ,αλλιώς επιστρέφει 0.
- calculate_time: Χρησιμοποιείται για τον υπολογισμό του χρόνου αναμονής μιας αίτησης στην ουρά ,καθώς και για τον υπολογισμό του χρόνου εξυπηρέτησης μιας αίτησης.
- put_request: Χρησιμοποιείται από τον παραγωγό για να εισάγει νέες αιτήσεις στην ούρα
- get_request: Επιστρέφει στον καταναλωτή έναν κόμβο ουράς (ο οποίος περιέχει τον περιγραφέα του socket καθώς και τον χρόνο έναρξης της αίτησης) ωστέ να υλοποιηθεί η εξυπηρέτηση της αίτησης.

- TSTP_handler: Αναλαμβάνει τη διαχείρηση του σήματος τερματισμού (SIGSTP) που μπορεί να δεχτεί ο server. Εκκινεί τη διαδικασία κλεισίματος, του τελικού σταδίου της διάρκειας ζωης του server.

Οι δομές που χρησιμοποιήσαμε μαζί με τη χρήση τους αναφέρονται παρακάτω:

- Condition variables:
  - not_full, not empty : Ελέγχουν το τρόπο με τον οποίο συνεργάζοντε μεταξύ τους οι καταναλωτές και οι παραγωγοί και την αλληλεπίδραση τους (ώθηση, απώθηση απο τη στοίβα)
  - reader_inQueue, writer_inQueue: Συγχρονίζουν τους readers και τους writers (PUT και GET requests) δίνοντας προτεραιότητα στους readers. Λειτουργούν σαν flags για να μπορεί μια νεοαφιχθής αίτηση να ελέγξει αν μπορεί να εκτελέσει τη λειτουργία της ή αν πρέπει να περιμένει.
- Mutexes/Locks:
  - queue_mutex : Συγχρονίζει τις εισαγωγές και τις αποθήσεις στην ουρά
  - while : Χρησιμοποιείται

- Αλλες δομές:
  - total_waiting_time : Αποθηκεύει το συνολικό χρόνο αναμονής των αιτήσεων στην ουρά
  - total_service_time : Αποθηκέυει το συνολικό χρόνο επεξεργασίας των αιτήσεων (από τη στιγμή που εξαχθεί από την ουρά και ύστερα).
  - complete_requests: Αποθηκεύει τις συνολικές αιτήσεις που έχει δεχθεί ο server από την στιγμή εκκίνησης του.

Εκσφαλμάτωση και corner cases:

- Τι γίνεται σε περίπτωση που ο signal handler έχει κάνει broadcast στην condition variable not_empty (αφού έχει θέσει πρώτα ένα flag το οποίο μπορούν να διαβάσουν όλοι οι threads, το while_breaker) και δεν έχουν μπλοκάρει όλοι οι worker threads σε αυτη τη μεταβλητή; Με άλλα λόγια τι γίνεται σε περίπτωση που από το broadcast αυτό δεν ενημερωθούν όλοι οι καταναλωτές ώστε να κλείσουν; Η περίπτωση να ξυπνήσει ένας thread από την broadcast μπορεί να αντιμετωπιστεί με έναν απλό έλεγχο του flag, αμέσως μετά την κλήση που μπλόκαρε και έβαλε τον thread στη “λίστα αναμονής”. Το σημαντικό είναι ότι πρέπει ο ίδιος έλεγχος να επαναληφθεί και πριν την κλήση της συνάρτησης που βάζει το thread στη λίστα αναμονής. Διότι η επιθυμητή συμπεριφορά είναι να μην ξαναμπεί ο thread στη λίστα αναμονής αφού ο handler έχει καλέσει την join(). Έτσι με αυτό το τρόπο σε περίπτωση που κάποιος thread εξυπηρετεί μια αίτηση τότε αφού την ολοκληρώσει θα κάνει τον έλεγχο και στη συνέχει θα τερματίσει.

- Απελευθερώνονται αυτόματα τα locks που διαθέτει ένας thread κατά τον τερματισμό του;

  Σε περίπτωση που κάποιος thread διαθέτει κάποιο lock τότε θα πρέπει να το απελευθερώσει πριν τερματίσει διότι το σύστημα δεν τα απελευθερώνει αυτόματα.

Μετρήσεις και επιδόσεις:

- Παρατηρούμε ότι για μεγέθη ουράς που κειμένονται σε κλίμακα δεκάδων και συνολικό αριθμό από καταναλωτές τον μισό από το μέγεθος της ουράς, ο μέσος χρόνος αναμονής είναι 0,1 sec και ο μέσος χρόνος εξυπηρέτησης είναι 0,3 sec. Φυσικά αν αυξήσουμε το μέγεθος της ουράς και διατηρήσουμε τον αριθμό τον καταναλωτών σταθερό ή το αντίθετο, τότε μειώνεται ο μέσος χρόνος αναμονής.
- Παρατηρούμε πως ο μέσος χρόνος εξυπηρέτησης δεν άλλαξε σε σχέση με υλοποίηση που χρησιμοποιούσε ένα νήμα, θεωριτικά στη δεύτερη περίπτωση θα έπρεπε να είναι ελαφρώς μικρότερος. Όμως ο χρόνος αναμονής κάθε αίτησης στην υλοποίηση με ένα νήμα είναι σίγουρα μεγαλύτερος από τον χρόνο εξυπηρέτησης. Οπότε συγκρίνωντας τους δύο αυτούς χρόνους αναμονής αντιλαμβανόμαστε πόσο πιο αποδοτικός είναι ο πολυνηματικός server, της ταξης >300%.