MYY601 Operating Systems Lab 1

Vlachos Dimitris 2641

Περιγραφή της εφαρμογης :

► Ο **client** αποστέλει αιτήσεις(PUT/GET) στον server ο οποίος με τη σειρά του φροντίζει για την εξυπηρέτησή τους.
► Ο **server** έχει τη δομή Παραγωγού-Καταναλωτή. Το default νήμα (Δηλαδή η main) λαμβάνει τις αιτήσεις και στη συνέχεια τις εισάγει σε μια δομή circular buffer(queue) .Τα νήματα καταναλωτή εξάγουν από την queue και εκτελούν τις αιτήσεις.
►Η εφαρμογή θα τερματίζει όταν το αποφασίσει ο χρήστης, μέσω του σήματος **SIGTSTP**(number 20). Τότε το default νήμα θα περιμένει να τελειώσουν τη δουλειά τους τα νήματα καταναλωτή, στη συνέχεια υπολογίζει και τυπώνει διάφορα στατιστικά και τέλος τερματίζει.

Μεταβλητές, Δομές και Συναρτήσεις :

⧫ **queue** : Είναι η ουρά που "παίρνει" τους fd’s.
⧫ **head** : Pointer που μας "δείχνει" ποιός fd θα καταναλωθει.
⧫ **rear** : Pointer που μας "δείχνει" ποιός fd θα εισαχθεί στη ουρά.
Σχόλιο : Η ουρά μας θα είναι κυκλική.
Αυτό σημαίνει ότι:
1) Οι pointers head και rear(tail) θα αυξάνονται κυκλικά.
2) Οι pointers μπορούν να δείχνουν στο ιδιο στοιχείο (δηλ. ουρά άδεια).
3) Η τιμη του head μπορεί να είναι μεγαλύτερη από του tail και αντίστροφα.

⧫ **Mutex queue_mutex** : "Προστατεύει" την queue και τις head, tail που είναι κοινόχρηστα δεδομένα.
⧫ **Condition variables condp, condc** : Χρησιμοποιούνται για τον συγχρονισμό Παραγωγού και Καταναλωτών.
Παραγωγος : Όταν η ουρά είναι γεμάτη πες στον Καταναλωτή να εξάγει.
Καταναλωτής : Όταν η ουρά είναι άδεια πες στον Παραγωγο να εισάγει.

⧫ **reader_count** : Δείχνει πόσοι readers ειναι καθε χρονική στιγμή στην αποθήκη
⧫ **writer_count** : Δείχνει αν υπάρχει κάποιος writer στην αποθήκη μια δεδομένη χρονική στιγμή.
⧫ **read_write_mutex, cond_store** : Χρησιμοποιούνται για τον συγχρονισμό με προτεραιότητα στους readers.

⧫ **completed_requests** : Δείχνει ποσα requests έχουν επεξεργαστεί οι καταναλωτές.(Δηλ. πόσα PUT, GET)
⧫ **total_waiting_time** : Συνολικός χρόνος που οι αιτήσεις παραμένουν στην ουρά.
⧫ **total_service_time** : Συνολικός χρόνος που χρειάζεται για την επεξεργασία των αιτήσεων.
⧫ **request_mutex** : Προστατεύει τις παραπάνω κοινόχρηστες μεταβλητές.

■ **function check_error** : Ελέγχει αν υπάρχει κάποιο πρόβλημα στην δημιουργία των νημάτων ή στην αρχικοποίηση των mutexes και των condition variables. Αν υπάρχει τερματίζει το πρόγραμμα τυπώνοντας κατάλληλο μήνυμα.
■ **function calculate_num_elements**: Υπολογίζει και επιστρέφει πόσα στοιχεία υπάρχουν στην ουρά μια δεδομένη στιγμή. Καλείται απο τις συναρτήσεις του παραγωγού και των καταναλωτών.
■ **function enqueue** : Εισάγει ένα στοιχείο στην ουρά. Καλείται απο το default νήμα (main).
■ **function dequeue** : Εξάγει και επιστρέφει ένα στοιχείο απο την ουρά. Καλείται απο τη συνάρτηση καταναλωτή.
■ **function Worker** : Είναι υπεύθυνη για τη λειτουργία των νημάτων καταναλωτή. Καλεί την συνάρτηση dequeue ώστε να πάρει το στοιχείο που έχει εξαχθεί απο την ουρά και στη συνεχεια το επεξεργάζεται με την βοήθεια της process_request.
■ **function total_w_time** : Υπολογίζει τον total_waiting_time.
■ **function total_s_time** : Υπολογίζει τον total_service_time.
■ **function ctr_z_handler** : Είναι υπεύθυνη για τη διαχειρηση του σήματος Control+Z. Όταν ο server λάβει το σήμα, η συνάρτη περιμένει τους καταναλωτές να τελειώσουν(join) κάνει print τα ζητουμενα στατιστικά και τερματιζει την εφαρμογή.

Σωστη λειτουργία Producer – Consumer

Πρέπει να προσέξουμε δύο πράγματα:

1) Υπάρχει το ενδεχόμενο όλα τα νήματα καταναλωτή να περιμένουν μπλοκαρισμένα όταν ο παραγωγος κανει signal για τελευταία φορά. Δηλαδη αν κάνει **pthread_cond_signal** τότε μόνο ένα νήμα καταναλωτή θα ξυπνήσει. Τα άλλα θα μείνουν μπλοκαρισμένα. Συνεπώς πρέπει να κάνει **pthread_cond_broadcast**. Τουλάχιστον αφού θα παράξει το τελευταίο fd.

2) Όταν ένας καταναλωτης ξυπνήσει πρέπει να τσεκάρει αν υπάρχει διαθέσιμος fd προς κατανάλωση (δηλ. πρέπει να καταναλώσει κάτι, αν αυτό το κάτι όντως υπάρχει) αλλα και για το αν ο παραγωγός έχει τελειώσει να πάράγει. Συνεπώς δεν πρέπει να ξανα-περιμένει ένα παραγωγό που δεν προκειτε παράξει κάτι ξανά! Αυτο μπορει να γίνει με μια κοινόχρηστη μεταβλητή **inform_consumer** αρχικοποιημένη σε 0. Οταν ο παραγωγός τελειώσει η τιμή αλλάζει σε 1.
Ο έλεγχος της τιμής του inform_consumer πρέπει να γίνει ακριβώς πριν και ακριβώς μετά το **pthread_cond_wait** !

Λύση του Reader-Writer προβλήματος.

Στο πρόβλημα παραγωγού καταναλωτή, ο παραγωγός και ο καταναλωτής τροποποιούν το **shared resource**. Τώρα μόνο οι writers το τροποποιουν !

Ο κώδικάς μου είναι μέσα στην συνάρτηση **process_request**.
Η λύση δίνει προτεραιότητα στους readers.

Reader:
- Τσεκάρει επαναληπτικά αν υπάρχει writer (μεταβλητή writer_count) στην αποθήκη και αν ναι περιμένει.
- Κανει lock το mutex πριν μπει στην αποθήκη για διάβασμα και κάνει πάλι unlock όταν τελειώσει το διάβασμα. Ετσι εξασφαλίζουμε ότι readers μπορούν να συνυπάρχουν στην αποθήκη.

Writer:
- Τσεκάρει επαναληπτικά αν υπάρχει άλλος writer (μεταβλητή

writer_count) ή readers (μεταβλητή reader_count) στην αποθήκη
και αν ναι περιμένει.
● κάνει lock και unlock σε όλο τον κώδικα εξασφαλίζωντας έτσι ότι
υπάρχει μοναδικός writer στην αποθήκη.

Κάποιες μετρήσεις :

► **Queue size** = 50 **, Consumer threads** = 10

completed requests | 5160 | 5160 | 5160 | 5160 | 5160
total waiting time | 0.000057 | 0.000142 | 0.000102 | 0.000059 | 0.000069
total service time | 0.000098 | 0.000178 | 0.000124 | 0.000099 | 0.000097

► Είναι προφανές ότι αν αυξηθεί το μέγεθος της ουράς, διατηρώντας το πλήθος των νημάτων καταναλωτών ίδιο τότε θα μειωθεί ο χρόνος αναμονής.