Ενσωματωμένα Συστήματα Πραγματικού Χρόνου
Producers-Consumers

Θεόδωρος Κατζάλης
AEM:9282
katzalis@auth.gr

28/03/2021

1 Εισαγωγή

Το πρόβλημα **producers-consumers** αποτελεί χαρακτηριστικό παράδειγμα ανάγκης συγχρονισμού σε multithreading εφαρμογές. Η περιγραφή του σχετίζεται με την προσθήκη και την αφαίρεση στοιχείων σε μια ουρά, χρησιμοποιώντας threads τα οποία αναλαμβάνουν το καθένα έναν απο τους δύο ρόλους ως producer ή consumer αντίστοιχα . Στην δικιά μας περίπτωση, τα στοιχεία της ουράς είναι αντικείμενα τύπου `struct` (`workFunction`), τα οποία καλούν μια τυπική συνάρτηση όταν πρόκειται κάποιος producer να προσθέσει ένα στοιχείο ή ένας consumer να το αφαιρέσει. Σε αυτήν την αναφορά θα μελετήσουμε τον χρόνο αναμονής που μεσολαβεί μεταξύ της προσθήκης ενός αντικειμένου στην ουρά μέχρι να καλεστεί κάποιος consumer για να το αφαιρέσει, χωρίς όμως να συμπεριληφθεί η εκτέλεση της συνάρτησης φόρτου που έχει να αναλάβει ο consumer στην κλήση του. Αξίζει να σημειωθεί, ότι στην συγκεκριμένη υλοποίηση έχουμε ορίσει ένα συγκεκριμένο αριθμό επαναλήψεων για τον οποίο ο κάθε producer προσπαθεί να προσθέσει ένα στοιχείο, ενώ απο την πλευρά των consumers έχουμε ορίσει έναν ατέρμονο βρόγχο με σκοπό να αδειάσουμε την ουρά.

Η **τεχνική παραλληλοποίησης** του συγκεκριμένου προβλήματος βασίζεται στην χρήση **pthreads** με **mutexes** και συνδυασμό blocking και waking up εντολών, `pthread_cond_wait()` και `pthread_cond_signal()` αντίστοιχα. Η χρήση τους θεωρείται απαραίτητη, διότι η ουρά είναι κοινή και προσβάσιμη μεταξύ των νημάτων, το οποίο θα παρουσίαζε race condition αν δεν υπήρχε συγχρονισμός μεταξύ της προσθήκης και της αφαίρεσης στοιχείων.

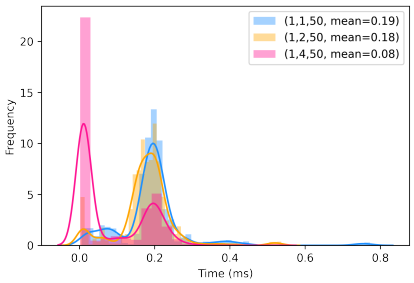
Η σχεδιαστική απόφαση του **ατέρμονου βρόγχου** απο την πλευρά των consumers δημιουργεί μια πρόκληση η οποία είναι η εύρεση τρόπου τερματισμού (break from the loop). Φυσικά ο στόχος μας είναι να τερματίσουν οι consumers όταν αδειάσει η ουρά. Ωστόσο, ας μην ξεχνάμε ότι οι consumers, όταν η ουρά είναι άδεια, περιμένουν waking up signal απο κάποιον producer. Τι θα γίνε λοιπόν αν έχουν τελειώσει οι producers και κάποιος consumer παραμένει blocked, αναμένοντας κάποιο signal? Θα μπορούσαμε ενδεχομένως να αποφύγουμε μια τέτοια κατάσταση ελέγχοντας απο πριν αν θα πρέπει ο consumer να περιμένει ή όχι με το αν υπάρχουν producers. Ωστόσο όμως επειδή το scheduling των threads δεν γίνεται με γνωστό τρόπο, υπάρχει περίπτωση κάποιοι consumer να έχουν "διαβάσει" νωρίτερα ότι υπάρχουν producers και να παραμένουν blocked και στη συνέχεια να τελειώσουν όλοι producers αφήνοντας κάποιους consumers σε blocked state. Οπότε πρέπει να στείλουμε waking up signal στους blocked consumers, αφότου τελειώσουν οι producers. Για να το κάνουμε αυτό, μετά την κλήση `pthread_join()` των producers, ο master thread, ελέγχοντας μια global μεταβλητή που σηματοδοτεί αν έχουν τερματίσει όλοι consumers, στέλνει waking up signals επανειλημμένα μέχρι η συνθήκη τερματισμού των consumers να είναι αληθής. Βέβαια αυτός ο τρόπος δεν είναι βέλτιστος, διότι δεν ανιχνεύουμε αν κάποιο thread είναι blocked και αν όντως χρειάζεται waking up signal για να τερματίσει, αλλά στέλνουμε συνέχεια waking up signals αφού τελειώσουν οι producers μέχρι να αδειάσει η ουρά απο τους consumers.

2 Διαγράμματα

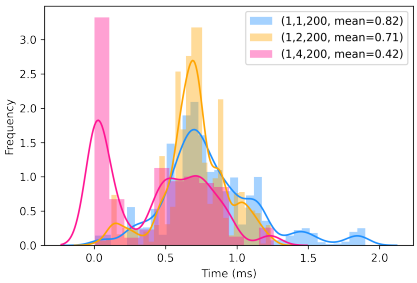
Για την μελέτη του χρόνου αναμονής, παρουσιάζουμε ένα πλέγμα διαγραμμάτων. Τα διαγράμματα έχουν παραμετροποιηθεί ως προς τον αριθμό των producers, consumers και το μέγεθος της ουράς. Πιο συγκεκριμένα τρέξαμε το πρόγραμμα για όλους τους συνδυασμούς producers, consumers για τιμές 1,2 και 4. Για την ουρά επιλέξαμε τις τιμές 50, 200, 1000. Σε κάθε διάγραμμα βλέπουμε για σταθερό αριθμό producers και queuesize, τρία διαφορετικά ιστογράμματα του χρόνου αναμονής που αντιστοιχούν σε διαφορετικό αριθμό consumers.

Στη συνέχεια, ως προς τις στήλες του πλέγματος, μεταβάλλεται η τιμή του queuesize και ως προς τις γραμμές η τιμή των producers. Σε κάθε διάγραμμα απεικονίζεται και η μέση τιμή.

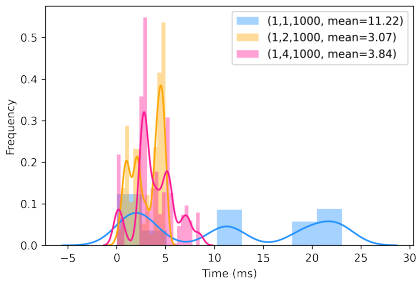
Αξίζει να σημειωθεί ότι σχετικά με τον αριθμό των επαναλήψεων απο την πλευρά των producers, δηλαδή πόσες φορές ένας producer θα προσθέσει στοιχεία στην ουρά, επιλέξαμε μια σταθερή τιμή 1000. Το σύστημα στο οποίο έγιναν οι μετρήσεις έχει τέσσερις πυρήνες.



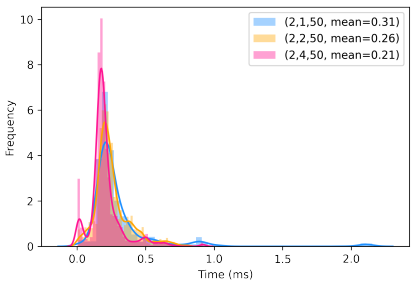
(a) (1,x,50)



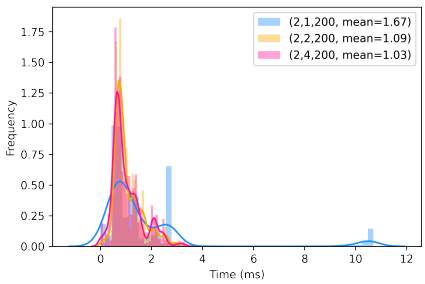
(b) (1,x,200)



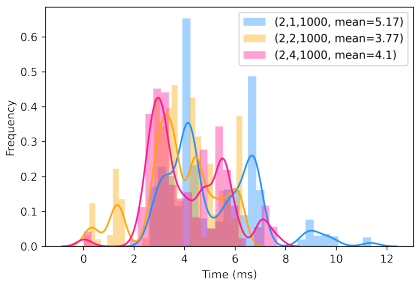
(c) (1,x,1000)



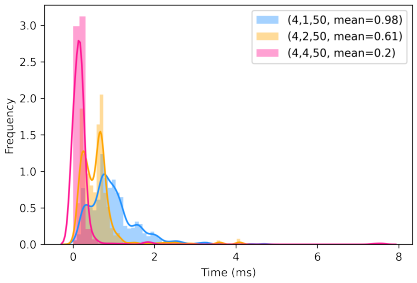
(a) (2,x,50)



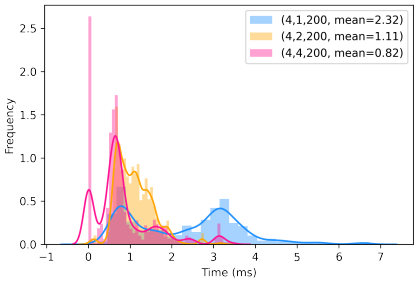
(b) (2,x,200)



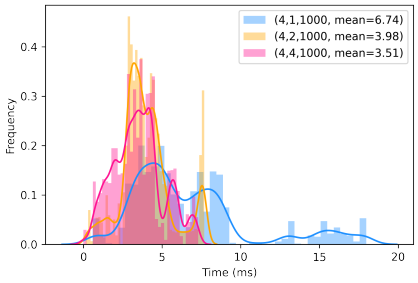
(c) (2,x,1000)



(a) (4,x,50)



(b) (4,x,200)



(c) (4,x,1000)

Figure 3: (αριθμός producers, μεταβλητή consumers, μέγεθος ουράς)

3 Συμπεράσματα

Με βάση τα παραπάνω διαγράμματα, μπορούμε να εξάγουμε τα εξής συμπεράσματα:

- Παρατηρούμε ότι για σταθερό αριθμό producers, όσο αυξάνονται οι consumers, ο χρόνος αναμονής μειώνεται. Αυτό είναι αναμενόμενο μιας και υπάρχουν περισσότεροι consumers για να "εξυπηρετήσουν" τα στοιχεία που προστέθηκαν στην ουρά.
- Με ανάλογο σκεπτικό, αυξάνοντας τον αριθμό των producers και κρατώντας σταθερό τον αριθμό των consumers, ο χρόνος αναμονής αυξάνεται.
- Σχετικά με το μέγεθος της ουράς, παρατηρούμε ότι όσο αυξάνεται το μέγεθος της, τόσο αυξάνεται και ο χρόνος αναμονής. Επειδή η ουρά είναι πλέον μεγάλη και έχουμε κρατήσει σταθερό των αριθμό των επαναλήψεων 1000 σε κάθε run, πλέον ένα producer-thread, όταν αναλαμβάνει δουλεία, έχει την δυνατότητα να προσθέσει έναν μεγάλο αριθμό στοιχείων. Αυτή η καθυστέρηση που οφείλεται στην προσθήκη περισσότερων στοιχείων στην ουρά, οδηγεί στην αύξηση του χρόνου αναμονής. Αξίζει να σημειωθεί ότι κάποια threads εξαιτίας συστήματος τρέχουν ψευδοπαράλληλα, οπότε για να τρέξει ένας consumer, θα πρέπει συνήθως να ολοκληρωθεί ένα τμήμα δουλειάς ενός άλλου thread.

Ο κώδικας για την παραπάνω υλοποίηση μπορεί να βρεθεί εδώ.