Ενσωματωμένα Συστήματα Πραγματικού Χρόνου
Εργασία 1, ΗΜΜΥ ΑΠΘ, 8ο Εξάμηνο

Κίτσιος Κωνσταντίνος 9182

Στην πρώτη εργασία του μαθήματος επεκτείναμε τον ήδη υπάρχον και λειτουργικό κώδικα για το πρόβλημα Producer-Consumer ώστε να δουλεύει με πολλαπλούς Producers και πολλαπλούς Consumers. Αυτό επιτυγχάνεται με την χρήση της βιβλιοθήκης παράλληλου προγραμματισμού **pthreads.**

Στο νεό πρόβλημα Multiple Producer-Multiple Consumer συμβολίζουμε με P τον αριθμό των παραγωγών(producers), με Q τον αριθμό των καταναλωτών(consumers) και με LOOP τον αριθμό των επαναλήψεων που εκτελέι κάθε νήμα παραγωγών. Τα νήματα των καταναλωτών δεν έχουν κάποιο συγκεκριμένο αριθμό επαναλήψεων αλλά λειτουργούν με εναν βρόγχο while(1) από τον οποίο βγαίνουν υπό συγκεκριμένες συνθήκες τερματισμού(Βλ. Παρακάτω).

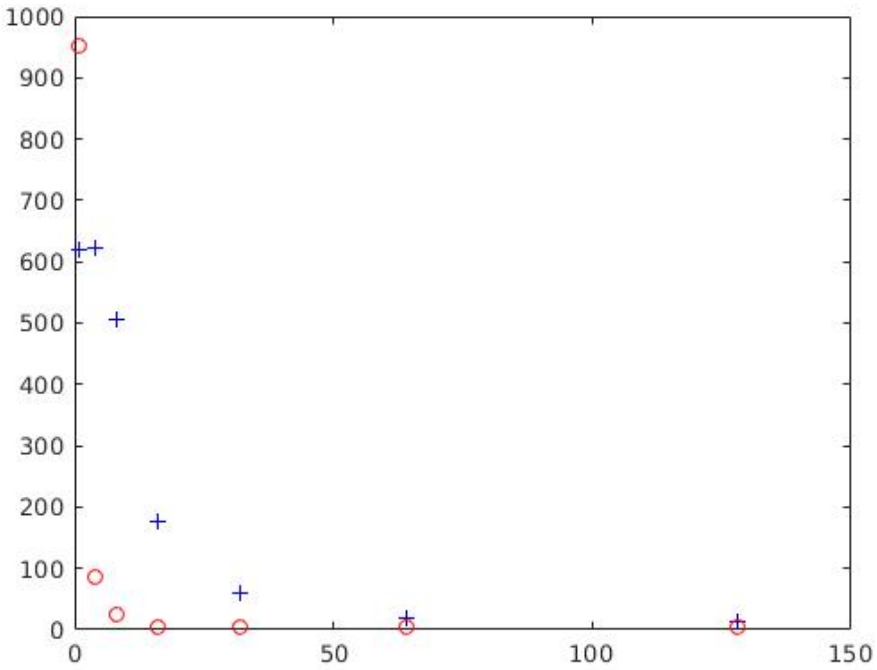
Η ουρά FIFO είναι κοινή και όλοι οι παραγωγοί τοποθετούν σε αυτήν συναρτήσεις τις οποιές οι καταναλωτές εκτελούν μόλις παραλάβουν. Στο τέλος, γίνεται καταγραφή και ανάλυση του χρόνου που απαιτείται απο την στιγμή που ένας παραγωγός τοποθετεί μία συνάρτηση στην ουρά μέχρι ένας καταναλωτής να την παραλάβει, αλλά όχι να την εκτελέσει. Τα σχετικά αποτελέσματα φαίνονται στην επόμενη σελίδα. Σχετικά με την υλοιποίηση έγιναν οι εξής αλλαγές σε σχέση με το απλο P-C προβλημα:

- Αντί για integers η ουρά FIFO έχει πλέον συναρτήσεις που δέχονται ως όρισμα τον thread id του producer και υπολογίζουν τα ημίτονα 10 γωνιών που εξαρτώνται απο το id αυτο, και συγκεκριμένα υπολογίζει το *sin((tid + i)*PI/10)* για i=1,2,...,10 όπου tid το thread id. Επίσης σαν όρισμα δίνεται και το starting time ώστε να υπολογιστεί ο ζητούμενος χρόνος.
- Προστέθηκαν συνθήκες τερματισμού για να σπάσουν τον βρόγχο while(1) όταν **τερματίσουν όλα τα producer threads ΚΑΙ η ουρά είναι άδεια.** Για την πρώτη συνθήκη χρησιμοποιείται η global μεταβλητή *finishedProducers* με τιμές απο 1...p η οποία προστατεύεται από το MUTEX *prodCountMut.*
- Για την χρονομέτρηση χρησιμοποιείται επίσης ένα MUTEX, το *timeMut*, το οποίο προστατεύει την global μεταβλητή *TOTAL_TIME_G.* Αυτήν γίνεται κάθε φορά που ένα thread **consumer** τερματίζει, καθώς ο συνολικός χρόνος σε κάθε επανάληψη χωρίς κίνδυνο για race, και αποθηκεύεται στην τοπική ως προς τα threads μεταβλητή *total_time.*

Για την εύρεση του Q που ελαχιστοποιεί τον χρόνο παραμονής στην ουρά, μετρήθηκε ο **μέσος όρος απο έναν αριθμό πειραμάτων(~10)** για διάφορες τιμες του Q με σταθερές τις παραμέτρους **QUEUESIZE=100, LOOP=1000** ενώ χρησιμοποιήθηκαν δύο τιμές **P=4** και **P=32**. Οι μέσοι όροι παρουσιάζονται στους πίνακες παρακάτω, ενώ οι μετρήσεις έγιναν σε μηχάνημα με επεξεργαστή *intel i7* που επιτρέπει μεγάλο βαθμό παραλληλοποιήσεις και πολλά threads.

Q	1	4	8	16	32	64	128	256	512	1024	2048
P=4	953	85.6	26.2	3.8	4.1	4.7	4.9	3.8	4.6	3.9	4.2
P=32	620	623	507	175	59.3	18	13.2	6.81	5.75	5.9	5.5

Time average in **us** for different values of P and Q



Βλέπουμε ότι για P=4 η τιμή που σταθεροποιείται ο μέσος χρόνος αναμονής είναι **Q=16** ενώ για P=32 είναι **Q=256**(στο Q=512 έχουμε ελάχιστο αλλά με μικρή διαφορά). Τέλος παραθέτουμε και το διάγραμμα μεταβολής του χρόνου ως προς το Q(μέχρι Q=128).

Οι παραπάνω τιμές προέκυψαν όπως είπαμε ως μέσοι όροι απο περίπου 10 μετρήσεις. Όσον αφορά την διασπορά των μετρήσεων αυτών αναφέρουμε πως όλο και μειωνόταν καθώς αυξάνοταν το **LOOP** και για **LOOP=1000** που εγιναν οι παραπάνω μετρήσεις είχε φτάσει σε πολύ μικρή τιμή, δηλαδή όλες οι μετρήσεις ήταν κοντά στον μέσο όρο, δεν υπήρχαν outliers. Αυτό συμβαίνει γιατί η κάθε μια μέτρηση είναι από μόνη της ένας μέσος όρος **LOOP** σε αριθμό στοιχείων άρα για μεγαλύτερο **LOOP** έχουμε πιο αντιπροσωπευτικό δείγμα.

Repo link: https://github.com/kitsiosk/Multiple-Producer-Consumer