Ενσωματωμένα Συστήματα Πραγματικού Χρόνου

1η εργασία

Σταύρος Μαλακούδης 9368
samalako@ece.auth.gr
Τα αρχεία τη εργασίας βρίσκονται στο https://github.com/Stavrenas/Embedded-Systems

Σε αυτή την εργασία γίνεται μια κρούση με το πρόβλημα του παραγωγού και του καταναλωτή. Μελετούμε τους χρόνους που περιμένουν τα στοιχεία από τη στιγμή που θα εισαχθούν στην ουρά έως ότου καταναλωθούν. Στην προκειμένη περίπτωση έχουμε πολλούς παραγωγούς και καταναλωτές και προσπαθούμε να βρούμε μια αναλογία παραγωγών και καταναλωτών ώστε να έχουμε τους ελάχιστους χρόνους απόκρισης.

Σχολιασμός του κώδικα

Όπως ζητήθηκε, έγιναν οι αντίστοιχες μετατροπές έτσι ώστε ο κώδικας να τρέχει για P νήματα producer και Q consumer. Σε αντίθεση με τον αρχικό κώδικα, ο χρήστης ορίζει το συνολικό πλήθος των στοιχείων που θέλει να εισάγει στην ουρά. Προφανώς δεν γνωρίζουμε εκ των προτέρων πόσα στοιχεία θα βάλει ο κάθε παραγωγός και πόσα θα καταναλώσει ο κάθε καταναλωτής . Με σκοπό τη μέτρηση του χρόνου, έγινε χρήση του struct timeval και των συναρτήσεων *tic* και *toc*. Για τη μεταφορά των δεδομένων (τη μέτρηση του χρόνου και του argument για τη συνάρτηση *doWork*) δημιουργήθηκε το struct *argument*.

Κάθε παραγωγός “τρέχει” έως ότου να μην υπάρχουν άλλα στοιχεία για να εισαχθούν στην ουρά. Η καταμέτρηση των στοιχείων γίνεται με τη global μεταβλητή *elementsLeft*, η οποία μεταβάλλεται μόνο εντός mutexes ώστε να μην υπάρχουν data races. Ο πίνακας *array* χρησιμοποιείται ώστε κάθε στοιχείο της ουράς να έχει μοναδικό argument για τη συνάρτηση doWork, ώστε να έχουμε μια πιο ρεαλιστική προσέγγιση. Έτσι λοιπόν, στη λούπα ο παραγωγός ορίζει τα στοιχεία του struct *argument* που θα περάσουν στον *void* arg pointer* του *workfunction struct*, που είναι το struct *argument* και ο pointer της συνάρτησης doWork().

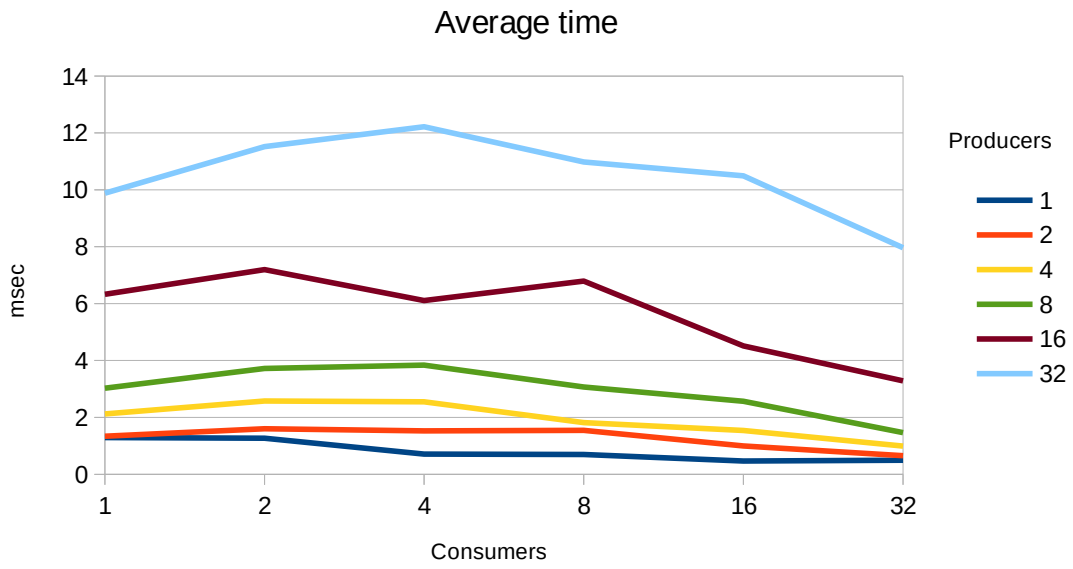
Όμοια, ο καταναλωτής δημιουργεί τοπικά struct ώστε να γίνει η αντιγραφή των δεδομένων και γίνεται η μέτρηση του χρόνου πριν τρέξει η συνάρτηση *doWork*. Οι χρόνοι αποθηκεύονται στον πίνακα times και μετά εγγράφονται σε ένα csv αρχείο ώστε να γίνει η στατιστική τους ανάλυση.

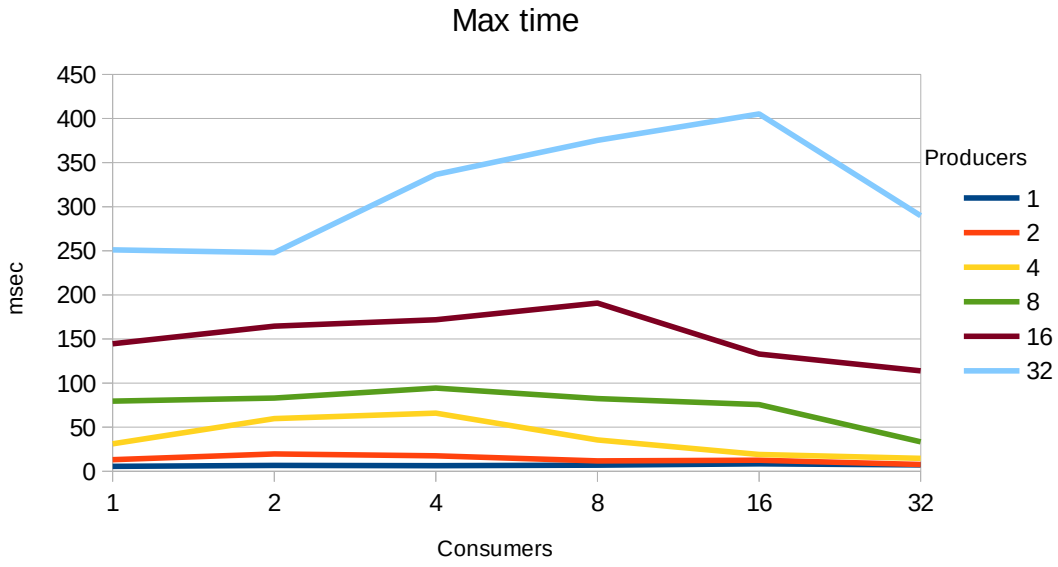
Αποτελέσματα πειραμάτων

Τα πειράματα έγιναν σε 6πύρηνο επεξεργαστή με 12 λογικούς πυρήνες. Σε κάθε πείραμα έγινε εισαγωγή 2000 στοιχείων στην ουρά. Οι εκδόσεις με πολλά νήματα (πάνω από 16) είναι αρκετά ασταθής και συχνά δεν τερματίζουν (τα νήματα “κολλάνε” στις συνθήκες pthread_cond_wait).

Με βάση τον πίνακα που φαίνεται παρακάτω, έχουμε μικρότερο μέσο χρόνο αναμονής στην ουρά για 1 producer και πολλούς consumers. Αυτό συμβαίνει διότι εφόσον τα στοιχεία εισάγονται με μικρότερο ρυθμό στην ουρά, οι consumers “προλαβαίνουν” και τα καταναλώνουν σύντομα. Φαίνεται ότι έπειτα από τα 16 νήματα, ο χρόνος σταθεροποιείται στα 0.46 msec ανά στοιχείο. Ο χρόνος είναι σχετικά μεγάλος διότι η συνάρτηση *doWork*() είναι ελαφρώς απαιτητική. Προφανώς, αν γίνει πιο ελαφριά, οι χρόνοι αναμονής θα είναι μικρότεροι διότι οι consumers θα

τελειώνουν πιο γρήγορα με την κατανάλωση ενός στοιχείου και θα πηγαίνουν στο επόμενο. Σε γενικότερες γραμμές φαίνεται πως όσο περισσότερους consumers έχουμε τόσο μικραίνει ο χρόνος αναμονής, αλλά ο βέλτιστος είναι για 1 producer μόνο.





Συμπεράσματα

Στο πρόβλημα producers-consumers φαίνεται ότι η ιδανική λύση είναι εμφανώς η προσέγγιση με 1 producer και πολλαπλούς consumers. Προσθέτοντας πολλούς producers, πρακτικά κάθε μετρική του συστήματος γίνεται χειρότερη: μέσος χρόνος αναμονής, μέγιστος χρόνος αναμονής, συνολικός χρόνος εκτέλεσης, αστάθεια συστήματος κτλ. Ενδεικτικά, ο μικρότερος συνολικός χρόνος εκτέλεσης ήταν 0.92 sec , για 1 producer και 16 consumers, ενώ ο χειρότερος για 32 producers και 4 consumers 24.43 sec. Μια πιθανή εξήγηση είναι ότι με περισσότερους producers εμπλέκεται στην όλη εξίσωση η επικοινωνία μεταξύ τους, τα pthread_cond_wait, η συχνή αλλαγή global μεταβλητών κτλ. Δηλαδή , το κόστος επικοινωνίας είναι μεγαλύτερο από την επιτάχυνση λόγω παραλληλισμού.