Εργασία 1: Ενσωματωμένα ΣΠΧ

Τριαρίδης Κωνσταντίνος 9159 triaridis@ece.auth.gr
Github link : https://github.com/kostino/EmbeddedSystems1

Για τα benchmarks χρησιμοποιήθηκε laptop με επεξεργαστή intel 5ης γενιάς με 4 physical cores και hyperthreading.

Περιγραφή κώδικα:

Αρχικά, η εκφώνηση λέει ότι η επανάληψη στη συνάρτηση των consumers πρέπει να είναι while(1) οπότε έπρεπε να βρεθεί κάποιο λογικό trigger που να τους σταματάει. Ένα τέτοιο φαινόμενο είναι το τέλος της παραγωγής και για να το παρακολουθούμε πιο εύκολα και να ενημερώνονται οι consumers προσθέτουμε αντίστοιχη μεταβλητή στη struct της FIFO έτσι ώστε μόλις τελειώσει η παραγωγή οι consumers να συνεχίσουν να δουλεύουν μέχρι αυτή να αδειάσει και μετά να σταματήσουν. Για να γίνει αυτό πρέπει να βγάλουμε τους consumers από το cond_wait(notEmpty) που θα βρίσκονται όταν αδειάσει η ουρά, οπότε πρώτα θα κάνουμε spawn τα consumer και producer threads, έπειτα θα κάνουμε join τα producer threads και θα ενημερώσουμε για τέλος της παραγωγής μετά θα ενημερώσουμε τα consumer threads με το condition notEmpty για να βγουν απο το wait και τέλος θα κάνουμε join τα consumer threads. Για να είμαστε σίγουροι ότι ενημερώθηκαν όλοι οι consumers να βγουν από το wait χρησιμοποιούμε μια global μεταβλητή που κρατάει των αριθμό των τελειωμενων consumer και κάνουμε broadcast μέχρι αυτή να γίνει ίση με τον αριθμό των consumers.

Στη συνέχεια έπρεπε να υλοποιηθούν κάποιες συναρτήσεις pointers στις οποίες θα προστίθενται στην FIFO ώστε να εκτελεστούν από τους consumers. Επιλέχθηκε να υλοποιηθεί μια απλή συνάρτηση που τυπώνει στην οθόνη πολλές φορές και τρείς που κάνουν μαθηματικές πράξεις. Για να είναι εύκολη η τυχαία επιλογή απο τα threads κατασκευάστηκε global πίνακας με pointers σε αυτές.

Για να κρατιούνται οι στατιστικές του χρόνου αναμονής από τη στιγμή πού ένας producer βάζει στην ουρά μέχρι ένας consumer να παραλάβει προστέθηκε ο αρχικός χρόνος στο struct που τοποθετείται στην ουρά στα void * arg μαζί με τα ορίσματα της συνάρτησης. Αυτό υλοποιείται στην workFunctionInit που ακολούθησε τη λογική της QueueInit που υπήρχε ήδη. Ο χρόνος που μεσολαβεί υπολογίζεται όταν το παραλαμβάνει ο consumer και γράφεται σε έναν global πίνακα.

Ο global πίνακας για τους χρόνους, η μεταβλητή finished και ο global πίνακας με function pointer βρίσκονται στο ίδιο critical section με τη FIFO οπότε χρησιμοποιείται το ίδιο mutex.

Benchmarks :

	C=1	C=2	C=4	C=8	C=16	C=32
Mean(usec)	54.38	81.04	84.72	84.55	95.09	76.29

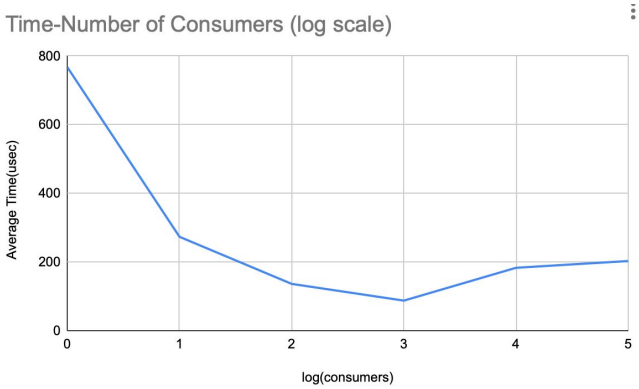
Τα πρώτα αποτελέσματα (Με P=4) ήταν ιδιαίτερα περίεργα. Περίμενα να βλέπω να μειώνεται ο μέσος χρόνος όσο αυξάνονται τα threads στην αρχή και μετά όταν βάλω αρκετά πολλά να αρχίσει να αυξάνεται (να μειώνεται η αποδοτικότητα της παράλληλης υλοποίησης). Αντίθετα ο ελάχιστος χρόνος ήταν για 1 consumer. **Το μόνο συμπέρασμα στο οποίο μπόρεσα να καταλήξω είναι ότι το workload των συναρτήσεων είναι τόσο μικρό (λιγότερο από αυτό της**

εναλλαγής των mutex), ωστε οι consumers είναι ήδη έτοιμοι όταν προστίθεται το επόμενο στοιχείο στη FIFO.

Για να το αντιμετωπίσω πρόσθεσα μια πλασματική καθυστέρηση με usleep, όπως προτείνεται εδώ: https://elearning.auth.gr/mod/forum/discuss.php?d=78650#p113161 και χρησιμοποιήθηκε για πειράματα ένας producer όπως προτείνει εδω ο κ. Πιτσιάνης: https://elearning.auth.gr/mod/forum/discuss.php?d=77215#p113360

Τα αποτελέσματα πλέον βγάζουν πλέον περισσότερο νόημα (από την άποψη ότι η προσθήκη περισσότερων consumer threads βοηθάει στην αρχή μέχρι να ξεπεράσω ένα όριο). Τα αποτελέσματα επίσης θεωρώ έχουν σταθεροποιηθεί καθώς δεν παρατηρούνται μεγάλες διαφορές από την αλλαγή του αριθμού του producer loop από 2000 σε 3000 και των πειραμάτων απο 10 σε 15. **Παρατηρώ ότι έχω ελαχιστοποίηση του μέσου χρόνου για C=8** που είναι διπλάσιο από τον αριθμό core του συστήματος άρα συμπεραίνω ότι για το σύστημα μου έχει νόημα να χρησιμοποιήσω περισσότερα threads από τον αριθμό των cores και θα δω βελτίωση μέχρι ένα όριο. Επίσης βγάζει απόλυτο νόημα ότι για μια υλοποίηση ουράς με 10 θέσεις η χρήση πάνω από 10 καταναλωτών σιγά σιγά θα έχει μειωμένα αποτελέσματα.

#TEST for P=1 with usleep#	C=1	C=2	C=4	C=8	C=16	C=32
Mean(usec)	767.49	273.29	136.15	87.39	183.07	202.72
std Dev.(usec)	253.73	83.98	25.99	20.09	80.28	89.30



Παρατηρώ ότι οι κατανομές των χρόνων είναι **κανονικές** όπως θα περίμενα για χρόνους αναμονής εκτός από αυτήν για C=1 που έχει συγκέντρωση τιμών σε 3 σημεία(δεν μπόρεσα να το αιτιολογήσω...)