

Ενσωματωμένα Συστήματα Πραγματικού Χρόνου Εργασία 1

Ονοματεπώνυμο : Μιχάλης Καρατζάς

AEM : 9137 email: mikalaki@ece.auth.gr , mikalaki@it.auth.gr ,

github για το project: <https://github.com/mikalaki/producer-consumer-multithreading>

1. Το ζητούμενο της Εργασίας

Σε αυτή την εργασία , καλούμαστε να τροποποιήσουμε το δοθέν παράδειγμα [prod-cons.c](#) (στο γερο : [prod-cons_default.c](#)) , το οποίο αποτελεί μια λύση με threads για το γνωστό πρόβλημα *producer- consumer* . Ζητούμενο είναι το νέο πρόγραμμά μας να λειτουργεί με *p* νήματα παραγωγούς (*producer*) και *q* νήματα καταναλωτές (*consumers*) , με τους πρώτους να προσθέτουν στην FIFO ουρά αντικείμενα τύπου *workFunction* (τα οποία ουσιαστικά αποτελούν συναρτήσεις) και τους δεύτερους να λαμβάνουν τα αντικείμενα αυτά και να τα εκτελούν. Στόχος είναι ή εύρεση του αριθμού των νημάτων consumer (*q*), για τον οποίο ελαχιστοποιείται ο μέσος χρόνος αναμονής (*waiting time*) των αντικειμένων *workFunction* μέσα στην FIFO ουρά.

2. Σύνοψη περιγραφή της υλοποίησης

Ο κώδικας του τροποποιημένου προγράμματος , βρίσκεται στο γερο στο αρχείο [prod-cons_multithreading.c](#) . Σε αυτό , έχει υλοποιηθεί όλη η λειτουργικότητα που αναφέρεται στην παραπάνω παράγραφο ενώ σέβεται σε μεγάλο βαθμό το [δοθέν πρόγραμμα](#) . Μερικά πράγματα που αξίζει να σημειωθούν , είναι ότι η εκτέλεση των συναρτήσεων των *WorkFunction* , γίνεται εκτός των *mutexes* , προκειμένου να έχουμε παράλληλη εκτέλεση των συναρτήσεων αυτών από τα *consumer threads* . Επίσης η συνάρτηση [queueDel\(\)](#) του δοθέντος προγράμματος , έχει μετονομαστεί σε [queueExec\(\)](#) , ώστε να περιγράφεται καλύτερα η λειτουργία της. Για τον υπολογισμό των χρόνων αναμονής , χρησιμοποιείται επικουρικά ένας πίνακας τύπου [struct timeval](#) , μήκους ίσου της ουράς (*QUEUE_SIZE*) , προκειμένου μόλις προστεθεί ένα αντικείμενο σε μία θέση της ουράς αναμονής , να προστίθεται σε αυτόν στη αντίστοιχη θέση η χρονική στιγμή του συστήματος (αρχή χρόνου αναμονής). Δίνεται πολύ προσοχή ώστε ο χρόνος αναμονής να λαμβάνεται ως το διάστημα που μεσολαβεί μεταξύ της στιγμής ανάθεσης μιας *WorkFunctions* στην ουρά και της στιγμής που αυτή παραλαμβάνεται (ουσιαστικά πριν την εκτέλεση της) . Οι συναρτήσεις οι οποίες χρησιμοποιούνται , ορίζονται στο αρχείο [myFuctions.h](#) .

3. Μετρήσεις και Συμπεράσματα

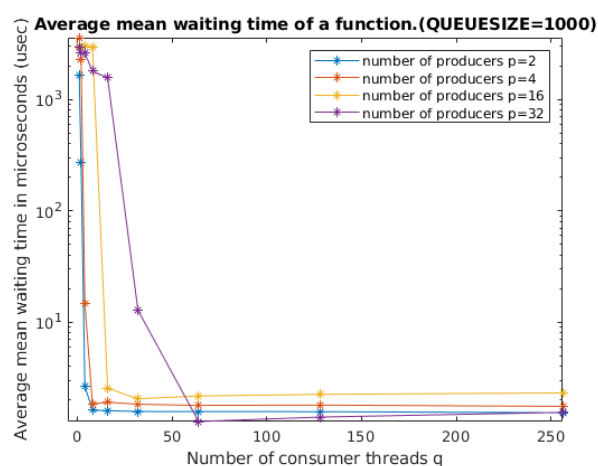
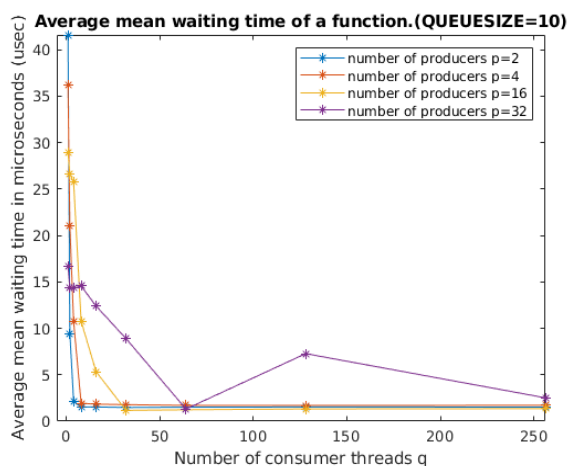
Οι μετρήσεις λήφθηκαν σε λάπτοπ με επεξεργαστή i5-9300H (4 CPUS , 8 Threads) και λειτουργικό Ubuntu 18.10. Για τις μετρήσεις χρησιμοποιήθηκε το αρχείο (script) [bench.sh](#) , το οποίο εκτελώντας το , κάνει compile το πρόγραμμα μας και το εκτελεί για ένα σύνολο από *p* και *q* που ορίζονται μέσα σε αυτό ([bench.sh](#)), ενώ ταυτόχρονα το αρχικό μας πρόγραμμα φροντίζει να κρατά τα στατιστικά μας δεδομένα σε αρχεία. Παρακάτω παρατίθενται πίνακες και γραφήματα μετρήσεων για διάφορους συνδυασμούς των *p* και *q* , [LOOP=50000](#) και για [QUEUE_SIZE=10](#) και [1000](#) , οι χρόνοι είναι σε msec (usec). Οδηγίες για το πως μπορεί κανείς να εκτελέσει το πρόγραμμα υπάρχουν στο [readme.md](#).

Πίνακας μετρήσεων μέσου χρόνου αναμονής για QUESIZE=10 , ο χρόνος δίνεται σε microseconds (usec):

p \ q	q=1	q=2	q=4	q=8	q=16	q=32	q=64	q=128	q=256
p=1	5.7958	3.03288	1.8554	1.53726	1.43384	1.42598	1.37602	1.40502	1.39076
p=2	41.60492	9.42495	2.16174	1.52273	1.5326	1.47396	1.50762	1.52984	1.51872
p=4	36.193245	21.04122	10.69498	1.93716	1.834145	1.78179	1.701505	1.690515	1.71351
p=8	30.530628	31.11056	32.876425	8.968023	3.323615	1.934013	1.98762	1.99421	2.060777
p=16	28.879748	26.630946	25.75025	10.78006	5.325643	1.168432	1.230484	1.320435	1.349479
p=32	16.725566	14.322467	14.326376	14.55789	12.397382	8.891358	1.311868	7.276517	2.519464

Πίνακας μετρήσεων μέσου χρόνου αναμονής για QUESIZE=1000 , ο χρόνος δίνεται σε microseconds (usec):

p \ q	q=1	q=2	q=4	q=8	q=16	q=32	q=64	q=128	q=256
p=1	1068.80452	351.7196	1.95962	1.56394	1.47518	1.44106	1.52814	1.43798	1.42972
p=2	1650.72279	271.88696	2.64672	1.61015	1.59828	1.57043	1.56703	1.56467	1.53095
p=4	3583.21005	2291.27177	14.71434	1.8389	1.90067	1.82522	1.78197	1.79353	1.74645
p=8	3175.76507	3285.484695	3515.961798	3.021867	2.077135	1.9919	2.032303	2.058618	2.08037
p=16	3017.810994	2849.887624	3013.274706	2921.758655	2.537032	2.04339	2.164917	2.237726	2.304903
p=32	2936.610135	2606.420977	2606.372823	1791.290958	1560.450052	12.749311	1.276314	1.398934	1.54117



Συμπεράσματα:

- 1) Από τις παραπάνω μετρήσεις , βλέπουμε ότι κατά κανόνα καθώς αυξάνεται ο *αριθμός των consumers* q για σταθερό *αριθμό producers* p , ο μέσος χρόνος αναμονής μειώνεται έως ότου λάβει μια πολύ μικρή τιμή (1.4-2 usec συνήθως) και έπειτα καθώς αυξάνουμε επιπλέον το q , βλέπουμε μικρές αυξομειώσεις . Στην περίπτωση όπου έχουμε QUESIZE=1000, βλέπουμε ότι καθώς το p αυξάνεται για σταθερό q , μεγαλύτερη μείωση στον μέσο χρόνο αναμονής έχουμε συνήθως όταν το p γίνει ίσο με το q , κάτι που είναι και αναμενόμενο.
- 2) Για μεγαλύτερο QUESIZE , βλέπουμε ότι ο χρόνος αναμονής , παίρνει μεγαλύτερες τιμές , κατι το οποίο είναι λογικό, καθώς σε μια μακρύτερη ουρά , μπορούν να βρίσκονται περισσότερες συναρτήσεις σε κατάσταση αναμονής.
- 3) Η χρήση μεγάλου αριθμού επαναλήψεων σε κάθε producer (LOOP=50000), εξασφαλίζει σταθερότητα των αποτελεσμάτων και ασφάλεια στην εξαγωγή συμπερασμάτων.
- 4) Η παραπάνω ανάλυση , αφορά τα πειράματα που έγιναν με τις συναρτήσεις που ορίζονται στο [myFunctions.h](#) αρχείο και υλοποιούνται στο [myFunctions.c](#) αρχείο , οι συναρτήσεις αυτές έχουν αρκετά μικρο υπολογιστικό φορτίο . Αν είχαμε συναρτήσεις που χρησιμοποιούσαν τους πόρους του συστήματος για μεγαλύτερο χρονικό διάστημα , προφανώς και θα είχαμε μεγαλύτερους χρόνους αναμονής , αλλά και πάλι θα ίσχυαν σε γενικές γραμμές όσα αναφέρθηκαν στα συμπεράσματα 1 και 2.