

# Ενσωματωμένα Συστήματα Πραγματικού Χρόνου Εργασία 1

Όνοματεπώνυμο : Μιχάλης Καρατζάς

AEM : 9137

email: [mikalaki@ece.auth.gr](mailto:mikalaki@ece.auth.gr)

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

## 1. Σύντομη περιγραφή των ζητούμενων και της υλοποίησης

Σε αυτή την εργασία , καλούμαστε να τροποποιήσουμε το δοθέν παράδειγμα [prod-cons.c](#) (στο repo : [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 *producers* και *q* νήματα *consumers* , με τους πρώτους να προσθέτουν στην FIFO ουρά αντικείμενα τύπου *workFunction* 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 των αντικειμένων *workFunction* μέσα στην FIFO ουρά.

Ο κώδικας του τροποποιημένου προγράμματος ,βρίσκεται στο repo 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](#) . Αξίζει να σημειωθεί, ότι υπάρχουν 5 συναρτήσεις και 10 ορίσματα (προκαθορισμένα) και κάθε φορά ένας producer προσθέτει έναν τυχαίο συνδυασμό από μια συνάρτηση και ένα όρισμα στην ουρά. Επίσης η [εκτέλεση των συναρτήσεων των WorkFunction από τους consumers](#) , γίνεται εκτός των *mutexes* , προκειμένου να έχουμε [παράλληλη εκτέλεση των συναρτήσεων αυτών](#). 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 (QUEUESIZE) , προκειμένου μόλις προστεθεί ένα αντικείμενο σε μία θέση της FIFO ουράς , να προστίθεται σε αυτόν στη αντίστοιχη θέση η χρονική στιγμή του συστήματος (αρχή χρόνου αναμονής)(επίσης χρησιμοποιήθηκε και *struct timespec* μαζί με την *clock\_gettime()*,για ακρίβεια σε μικρούς χρόνους). Ο χρόνος αναμονής λαμβάνεται ως το διάστημα που μεσολαβεί μεταξύ της στιγμής ανάθεσης μιας WorkFunction στην ουρά από ένα παραγωγό και της στιγμής που αυτή παραλαμβάνεται από έναν καταναλωτή ( ουσιαστικά πριν την εκτέλεση της) . Οι συναρτήσεις οι οποίες χρησιμοποιούνται στις *workFunctions* , ορίζονται στο αρχείο [myFuctions.h](#) . Για τον τερματισμό του προγράμματος , χρησιμοποιείται η μεταβλητή *terminationStatus* , η οποία , αυξάνεται κατά 1 μόλις ένας producer ολοκληρώσει την συνολική του ανάθεση (τοποθετήσει συνολικά στην ουρά LOOP σε πλήθος στοιχεία) , μόλις αυτή η μεταβλητή λάβει τιμή ίση με τον αριθμό των producers και αφού αδειάσει η ουρά από στοιχεία , σηματοδοτείται το τέλος της διαδικασίας των producers-consumers και οι δεύτεροι επιστρέφουν στην *main()*. Η ορθότητα του προγράμματος , επαληθεύεται εκτυπώνοντας κατά την λήξη του, τον συνολικό αριθμό των συναρτήσεων οι οποίες εκτελέστηκαν , ο οποίος αποθηκεύεται στην μεταβλητή *functionsCounter*. Η τιμή της μεταβλητής αυτής αυξάνεται μόλις ένας consumer αναλάβει μια συνάρτηση για εκτέλεση, όταν στο τέλος του προγράμματος, είναι ίση με **p\*LOOP**, αντιλαμβανόμαστε ότι στο πρόγραμμα μας εκτελούνται επιτυχώς όλες οι συναρτήσεις που πρέπει να εκτελεστούν.

## 2. Μετρήσεις και Συμπεράσματα

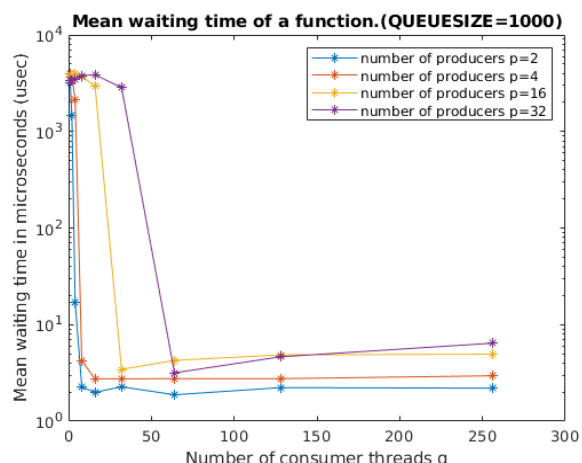
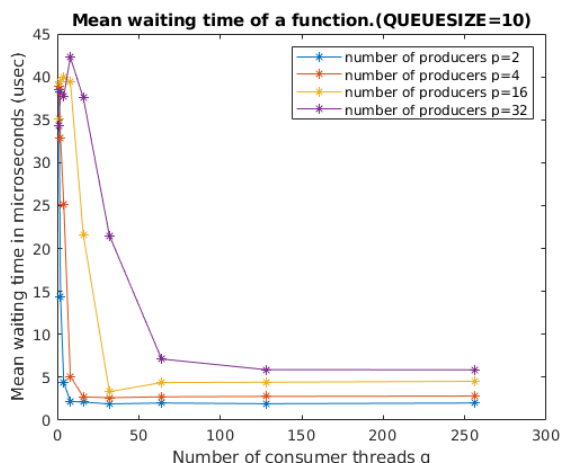
Οι μετρήσεις λήφθηκαν σε λάπτοπ με επεξεργαστή i5-9300H (4 CPUS , 8 Threads) και λειτουργικό Ubuntu 18.10. Για τις μετρήσεις χρησιμοποιήθηκε το αρχείο (script) [bench.sh](#) , το οποίο εκτελώντας το , κάνει compile το πρόγραμμα μας και το εκτελεί για ένα σύνολο από *p* και *q* που ορίζονται μέσα σε αυτό. Ταυτόχρονα το αρχικό μας πρόγραμμα φροντίζει να κρατά στατιστικά δεδομένα σε αρχεία, καθώς και όλους του χρόνους αναμονής των *workFunction* σε csv , οι οποίο έπειτα θα επεξεργαστούν από το αρχείο [statistics.m](#) , για λήψη χρήσιμων στατιστικών. Παρακάτω παρατίθενται πίνακες και γραφήματα μετρήσεων για διάφορους συνδυασμούς των *p* και *q* , **LOOP=10000** και για **QUEUESIZE=10** και **1000** , οι χρόνοι είναι σε *μsec (usec)*. Οδηγίες για την εκτέλεση του προγράμματος 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	7.907	6.35	2.498	1.973	1.799	1.943	1.771	1.848	1.983
p=2	38.515	14.399	4.401	2.206	2.107	1.899	2	1.92	1.999
p=4	38.902	32.924	25.158	5.104	2.715	2.621	2.723	2.768	2.816
p=8	36.647	39.232	41.225	30.68	5.897	3.688	3.715	3.647	3.643
p=16	35.066	39.449	40.005	39.42	21.619	3.326	4.382	4.426	4.526
p=32	34.347	38.305	37.723	42.26	37.552	21.514	7.128	5.886	5.872
p=64	33.96	32.923	36.79	41.615	37.223	26.883	21.195	13.339	13.585

**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	107.228	682.416	4.572	2.115	1.959	1.93	1.903	1.988	2.034
p=2	3230.505	1464.026	16.89	2.253	1.989	2.272	1.877	2.226	2.198
p=4	3997.201	3654.78	2120.548	4.224	2.741	2.748	2.754	2.758	2.952
p=8	3909.871	4726.561	4603.474	3086.12	3.148	3.254	3.6	3.829	3.659
p=16	3604.914	3677.429	4003.022	3688.1	2958.501	3.439	4.262	4.861	4.941
p=32	3374.09	3394.415	3493.08	3803.286	3858.888	2856.644	3.17	4.641	6.421
p=64	3349.608	3373.243	3679.007	4137.892	3930.922	3526.686	1846.586	3.548	7.643



## Συμπεράσματα:

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 και έπειτα καθώς αυξάνουμε επιπλέον το *q* , βλέπουμε μικρές αυξομειώσεις(πρακτικά ο μέσος χρόνος αναμονής παραμένει σταθερός) , κάτι το οποίο είναι εμφανές και στα παραπάνω διαγράμματα. Οπότε για συγκεκριμένο αριθμό παραγωγών *p* , έχουμε ελαχιστοποίηση του μέσου χρόνου αναμονής για συγκεκριμένο αριθμό καταναλωτών *q* (με πράσινο στους πίνακες) και η επιπλέον αύξηση του αριθμού των καταναλωτών μετά από αυτό το σημείο δεν μας δίνει πρακτικά κάποιο όφελος.

Η σταθεροποίηση αυτή σε μια πολύ μικρή τιμή , σημαίνει ότι πλέον οι *producers* πρακτικά δεν προλαβαίνουν να προσθέτουν νέα αντικείμενα και να γεμίζουν την ουρά καθώς τα παραλαμβάνουν και τα εκτελούν σχεδόν άμεσα οι *consumers*. Αντίθετα αριστερότερα στα γραφήματα (για μικρότερες τιμές του *q*) , όπου ακόμα έχουμε σημαντικές τιμές για τον μέσο χρόνο αναμονής (π.χ. *p=8* και *q=1*), μπορούμε να συμπεράνουμε ότι κατά κάποιο τρόπο οι *producers* “υπερισχύουν” των *consumers* και γεμίζουν την ουρά με αντικείμενα. Αυτό μπορούμε να το διαπιστώσουμε και από το γεγονός ότι η αύξηση του μεγέθους της ουράς 100 φορές ( 10 -> 1000) , αύξησε τον μέσο χρόνο αναμονής στα αριστερά του οριζοντίου άξονα (π.χ. *p=4* και *q=2*) επίσης κατά 100 φορές περίπου, που σημαίνει ότι πλήθος αντικειμένων τοποθετήθηκαν στη ουρά (ενδεχομένως να γέμισε κιάλας), ενώ οι τιμές στις οποίες οι μέσοι χρόνοι αναμονής σταθεροποιούνται (για μεγαλύτερα *q*, π.χ. *p=2* και *q=8*) , δεν άλλαξαν πρακτικά. Αυτό μας δείχνει ότι όταν οι *producers* προσθέτουν αντικείμενα με μεγαλύτερο ρυθμό από ότι καταναλώνουν οι *consumers* , έχει νόημα η χρήση μιας μεγαλύτερης ουράς, καθώς οι θέσεις της θα γεμίζουν με νέα αντικείμενα , διαφορετικά μια μεγαλύτερη ουρά θα ήταν σπατάλη.

Βέβαια τα συμπεράσματα αυτά, αφορούν πειράματα που έγιναν με τις συναρτήσεις που ορίζονται στο αρχείο [myFunctions.h](#) , οι οποίες έχουν αρκετά χαμηλό υπολογιστικό φορτίο. Για συναρτήσεις που θα αξιοποιούσαν τους πόρους του συστήματος για μεγαλύτερο χρονικό διάστημα θα είχαμε σε γενικές γραμμές μεγαλύτερους χρόνους αναμονής , ενώ η σταθεροποίηση τους θα συνέβαινε δεξιότερα (για μεγαλύτερο *q*) .

Η χρήση μεγάλου αριθμού επαναλήψεων σε κάθε *producer* (LOOP=10000), εξασφαλίζει ασφάλεια στην εξαγωγή συμπερασμάτων . Αυτό επαληθεύεται συγκρίνοντας τον ολικό μέσο χρόνο αναμονής με το μέσο χρόνο αναμονής για τα 3/4 των δεδομένων, και βρίσκοντας παραπλήσιες τιμές. \*Ο μέσος χρόνος αναμονής για τα 3/4 των δεδομένων καθώς και αλλά στατιστικά σχετικά με τον χρόνο αναμονής, περιγράφονται και αναλύονται στο αρχείο [statistics.pdf](#) (λινκ στο αρχείο).\*

