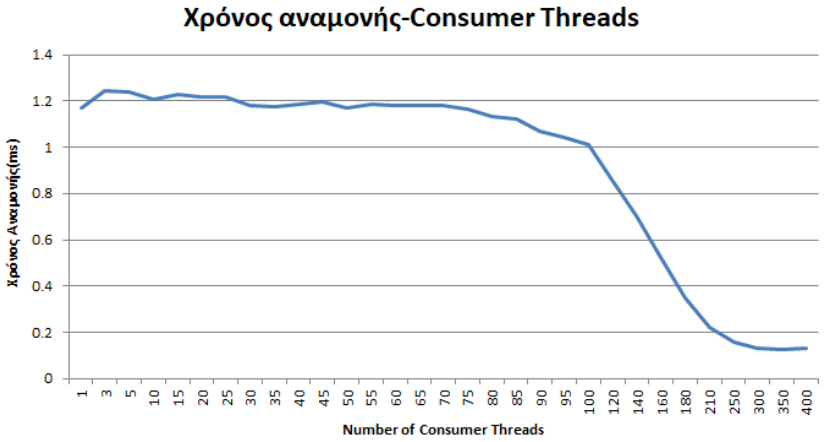
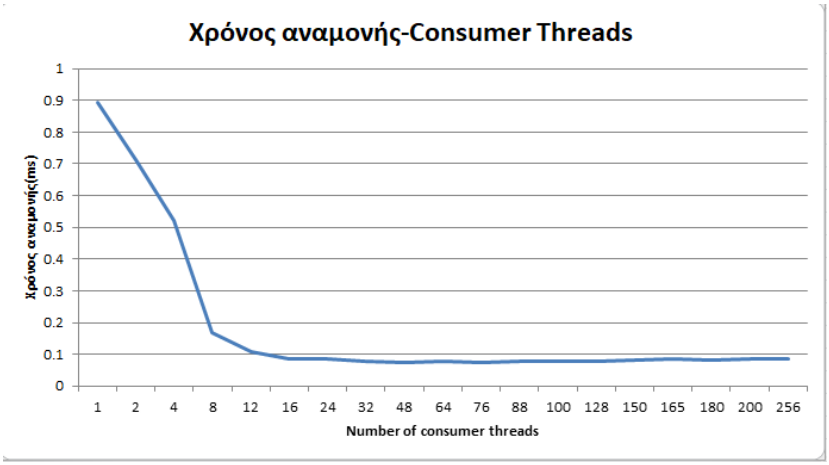
ΑΝΑΦΟΡΑ –ΕΡΓΑΣΙΑ Ι

Στην πρώτη εργασία κληθήκαμε να διαμορφώσουμε το παράδειγμα prod-cons.c ώστε να λειτουργεί με πολλαπλά *threads* τόσο για τον *consumer* όσο και για τον *producer*, με τα νήματα του *consumer* να παίρνουν δείκτες σε συνάρτηση απο μια κοινή ουρά και να εκτελούν διάφορες συναρτήσεις. Οι συναρτήσεις εκτελούν απλούς υπολογισμούς και δέχονται σαν όρισμα μια τιμη που υπολογίζεται με βάση το *threadıd* και τον *counter* που μετρά τον αριθμό των δεικτών που έχουν εισαχθεί στην ουρα πετυχαίνοντας έτσι να εισάγεται διαφορετικό όρισμα για κάθε κλήση της συνάρτησης ,με τις δύο απο αυτες να υπολογίζουν τριγωνομετρικους αριθμούς γωνιών(*tan,cos*),αλλες δύο να εκτυπώνουν απλα το όρισμα που δέχθηκαν και μια να υπολογίζει το τετράγωνο του αριθμού που δέχθηκε σαν όρισμα. Επιλέον, ελέγχοντας οτι η τιμή του counter στο τέλος του προγράμματος είναι ίση με LOOP*producers,βεβαιωνόμαστε ότι εισήχθη στην ουρά ο αναμενόμενος αριθμός δεικτών.Για τον υπολογισμό του χρόνου που παρήλθε, απο τη στιγμη που το αντικείμενο τύπου *workfunction* εισήχθη στην ουρά μέχρι να το παραλάβει απο αυτή ο *consumer* και προτού εκτελεστεί η συνάρτηση, **προστέθηκε** στο *struct workfunction* μια μεταβλητή τυπου ***unsigned long long*** για την αποθήκευση του χρόνου που το κάθε αντικείμενο εισήχθη στην ουρα. Αυτή η μεταβλητή απλοποιεί τον υπολογισμό του χρόνου που ζητείται καθώς, για κάθε αντικείμενο αρκεί ενας υπολογισμός του χρόνου παράληψης απο τον *consumer* και μια απλή αφαίρεση της τιμής που έχει αποθηκευτεί στην *extra* μεταβλητή που προστέθηκε, απο την τιμή που υπολογίστηκε στον *consumer*. **Η εκτέλεση** των **συναρτήσεων** έγινε **εκτός**(και όχι ενδιάμεσα) των *lock - unlock* του ***mutex*** για την εκμετάλλευση της παραλληλίας, καθώς οι συναρτήσεις δεν μεταβάλλουν *global* δεδομένα γεγονός που θα καθιστούσε αναγκαία την σειριοποίηση τους(και συνεπώς τη χρήση κάποιου αλλου ξεχωριστου *mutex*).Το πρόγραμμα που προέκυψε, εκτελέστηκε σε σύστημα με **2 πυρήνες** ενω ο αριθμός του ***LOOP*** που αυτός περιείχε στο εσωτερικο του **5000**. Ο αριθμός αυτος για την τιμή του LOOP κρίθηκε αρκετός καθώς παρατηρήθηκε οτι οι μετρήσεις του μέσου χρόνου αναμονής σταθεροποιούνται. Για μικρότερες τιμές στο LOOP(της τάξης του 100,200,1000κλπ), το σύστημα δεν προλάβαινε να περάσει το μεταβατικό στάδιο κατα το οποίο οι producers μονοι τους διαγωνίζονται για το mutex (λόγω του οτι φτιάχνονται πρώτοι) καθώς οι consumers φτιάχονται λίγο αργότερα,με αποτέλασμα οι χρόνοι αναμονής να είναι αρκετά μεγαλύτεροι και όχι τοσο αντιπροσωπευτικοί. Κρατώντας αυτούς τους αριθμούς αλλα και το **μέγεθος της ουράς σταθερό και ίσο με 10** έγιναν κάποια πειράματα μεταβάλλοντας τον αριθμό των νημάτων του *consumer* με σκοπό την εύρεση του αριθμού που θα βελτιστοποιεί το μέσο χρόνο αναμονής,για πολλούς *producers*(70) αλλά και για έναν.Θεωρητικά για την επίτευξη του καλύτερου μέσου χρόνου θα πρέπει να πετύχουμε τον ίδιο ρυθμό παραγωγής και κατανάλωσης. Ντετερμινιστικά, αν ο ρυθμός παραγωγής είναι μεγαλύτερος απο τον ρυθμό κατανάλωσης η ουρά κάποια στιγμή θα γεμίσει, αντίθετα αν ο ρυθμός κατανάλωσης είναι μεγαλύτερος απο τον ρυθμό παραγωγής η ουρά θα αδειάσει. Στο πρόγραμμα που έχει υλοποιηθεί ,η επιθυμητή ισορροπία προσπαθεί να επιτευχθεί μέσω του *mutex* και των *pthread_cond_wait*() και *pthread_cond_signal*() με τα τελευταία δύο να αναλαμβάνουν το σταμάτημα των παραγωγών(όταν η ουρά είναι γεμάτη) και των καταναλωτών (όταν η ουρα είναι άδεια) αλλα και την επανέναρξη τους όταν οι συνθήκες το επιτρέπουν. Λόγω του γεγονότος οτι η ανάληψη του *mutex* απο τα *threads* καταναλωτών και παραγωγών είναι τυχαίο γεγονός (εκτός την περίπτωση της άδειας ουρά στην οποία το επόμενο νήμα θα είναι σίγουρα κάποιος παραγωγός ή της γεμάτης ουράς όπου το επόμενο νήμα θα είναι σίγουρα καταναλωτής) είναι λογικό κάποιες φορές ο ρυθμός παραγωγής να είναι μεγαλύτερος απο την ρυθμό κατανάλωσης και αντίστροφα, για αυτό το λόγο κρίνεται απαραίτητη μια τιμή για το *queuesize*(για το σύστημα μας αλλα και αυτες τις συναρτήσεις η τιμη 10) ικανή να χειριστεί αυτές τις αυξομειώσεις χωρίς

να οδηγειται η ουρα σε κατάσταση *full*. Ωστόσο, να σημειωθεί οτι μια πολύ μεγάλη ουρά ενδέχεται να προκαλέσει μεγαλύτερους χρόνους αναμονής διότι κάθε «αντικείμενο» θα παραμένει περισσότερη ώρα μέσα στην ουρά. Για να είνα τα αποτελέσματα περισσότερο αντιπροσωπευτικά, για κάθε αριθμό απο *consumer threads* έγιναν δυο *runs* και πάρθηκε ο μέσος όρος αυτων. Στο πρώτο διάγραμμα οι μετρήσεις έχουν γίνει για 70 *producers* ενω στο δέυτερο με 1 *producer*.



Παρατηρούμε ότι για τιμές απο 1-70 *consumer threads*, όταν δηλαδή ο αριθμός ειναι μικρότερος απο τον αριθμό των νημάτων του producer ο χρόνος παραμένει σχεδόν σραθερός και κυμαίνεται γύρω απο τα 1.2 *ms*. Όταν ο αριθμός των νημάτων του *consumer* ξεπεράσει τον αντίστοιχο του *producer* η μείωση του χρόνου αναμονής γίνεται πλέον αισθητή, ενω μετά τα 100 *threads consumer* υπάρχει κατακόρυφη πτώση του χρόνου με **το ελάχιστο** του μέσου χρόνου αναμονής να επιτυγχάνεται κοντά στα **300 *consumer threads*.** Από εκείνο τον αριθμό και έπειτα οι χρόνοι σταθεροποιούνται χωρίς να υπάρχει κάποια βελτίωση καθώς τα παραπάνω *threads* του *consumer* παραμένουν αδρανή. Με άλλες συναρτήσεις που θα εκτελούσαν περισσότερο χρονοβόρες διαδικασίες λόγω του ότι θα κρατούσαν το αντίστοιχο *thread* δεσμευμένο περισσότερο χρόνο ,πιθανότατα θα απαιτούνταν μεγαλύτερος αριθμός απο *consumer threads* για να επιτευχθεί ο ελάχιστος χρόνος αναμονής. Όμοια για το διάγραμμα με τον ***1 producer***,με ελάχιστο στα 48 ***consumer threads***



Ο κώδικας βρίσκεται στο *link* που ακολουθει : https://github.com/napoleon98/Embedded-Systems/blob/master/code_ergasia_1.c