Ενσωματωμένα Συστήματα Πραγματικού Χρόνου -Τελική Εργασία: C timer

Ονοματεπώνυμο : Μιχάλης Καρατζάς AEM : 9137 email: mikalaki@ece.auth.gr

Github repo για το project: https://github.com/mikalaki/producer-consumer-timer

1 Εισαγωγικά

Ένας *timer*, αποτελεί μια υπορουτίνα που αυτόματα εκτελεί κώδικα ανά τακτά χρονικά διαστήματα. Σε αυτή την εργασία καλούμαστε να υλοποιήσουμε έναν *timer* στην C, βασισμένο στην προηγούμενη εργασία του μαθήματος, η οποία ήταν η υλοποίηση μια *πολυνηματικής* (*multithreading*) υλοποίησης του προβλήματος *producer-consumer* με χρήση *ουράς FIFO*.

Αξιοποιώντας την αρχιτεκτονική της προηγούμενης εργασίας, πρέπει κάθε νήμα(*thread*) *producer* να αντιστοιχεί σε ένα αντικείμενο *timer* και να προσθέτει περιοδικά, την συνάρτηση προς εκτέλεση στην κοινή ουρά. Με την σειρά τους τα νήματα *consumers*, θα βρίσκονται σε αδράνεια, έως ότου προστεθεί μια συνάρτηση στην κοινή ουρά. Μόλις συμβεί αυτό, στόχος είναι η όσο τον δυνατόν πιο άμεση λήψη και εκτέλεση της συνάρτησης, από ένα *νήμα consumer*. Όταν ένας *timer* προσθέσει την συνάρτηση του όσες φορές απαιτείται, τερματίζει και καταστρέφεται.

Στα πλαίσια τις εργασίας είναι και η αντιμετώπιση της χρονικής μετατόπισης των πειραμάτων (*time-drifing)*, ώστε να ελαχιστοποιείται η χρονική απόκλιση των διαδοχικών αναθέσεων του *timer* στην *ουρά*, από την επιθυμητή περίοδο(*Period)* του timer. Η έγκαιρη και αποτελεσματική αντιμετώπιση του φαινομένου, εξασφαλίζει την διατήρηση της πραγματικής περιόδου μεταξύ των αναθέσεων στον επιθυμητό χρόνο, ενώ αποτρέπει την αθροιστική καθυστέρηση που συσσωρεύεται στα πειράματα.

Τέλος καλούμαστε να καταγράψουμε τους χρόνους που απαιτούνται για την ανάθεση συναρτήσεων στην *ουρά* από τα *producer threads*, τους χρόνους αναμονής μιας συνάρτησης στην ουρά καθώς και να παρουσιάσουμε στατιστικά και συμπεράσματα για τους χρόνους αυτούς.

2 Σύντομη περιγραφή της υλοποίησης του *timer.*

Η υλοποίηση του *timer*, βρίσκεται στο αρχείο timer.c του project, το οποίο βασίζεται πλήρως στις αρχές του prod-cons_multithreading.c, της προηγούμενης εργασίας και έχει αναπτυχθεί πάνω σε αυτό. Διαφοροποιήσεις υπάρχουν, κυρίως στον τρόπο με τον οποίο οι *timers* και τα *producer threads* αναθέτουν συναρτήσεις για εκτέλεση στην ουρά, ενώ τα *consumer threads* λειτουργούν ακριβώς με τον ίδιο τρόπο με αυτόν της προηγούμενης εργασίας. Κοινή παραμένει επίσης η χρήση των *mutexes* για συγχρονισμό μεταξύ των νημάτων, καθώς και η συνθήκη τερματισμού του προγράμματος.

Για την υλοποίηση των ζητούμενων προδιαγραφών(*Specification)* του *timer*, έχει δημιουργηθεί η *δομή (struct) Timer* (εικόνα 1), στην οποία δηλώνονται οι απαιτούμενες μεταβλητές και συναρτήσεις με κατάλληλο τρόπο. Για την δημιουργία ενός αντικειμένου τύπου *Timer*, έχει δημιουργηθεί η μέθοδος timerInit(), με κατάλληλα ορίσματα, ενώ για την διαγραφή ενός αντικειμένου τύπου *timer* έχει δημιουργηθεί η μέθοδος timerDelete().

```
typedef struct {
  unsigned int Period; //Period given in milliseconds
  unsigned int TasksToExecute; //How many times the workFunc will by executed by times
  unsigned int StartDelay;// StartDellay is in seconds, the initial dellay of workFunc execution
  pthread_t producer_tid; // the producer thread id correspoding to timer

  struct workFunction * TimerFcn;// TimerFcn

  void * (* StartFcn)(void *);
  void * (* StopFcn)(void *);
  void * (* ErrorFcn)(void *);

  void * userData;

  queue *Q;
} Timer;
```

Εικόνα 1: Η δομή (struct) του timer.

Αντίστοιχα η μέθοδος , start(t) , δέχεται ως όρισμα έναν *pointer* σε αντικείμενο τύπου timer, μεταβιβάζει τα δεδομένα του *timer* στο *producer thread* που του αντιστοιχεί ώστε να εκκινήσει η περιοδική εκτέλεση της *TimerFcn*. Με όμοιο τρόπο λειτουργεί και η startat(t,y,m,d,h,min,sec) , ωστόσο στα ορίσματα της, εισάγεται η ημερομηνία και ώρα (*d/m/y h:min:sec*) στην οποία θέλουμε να εκκινήσει ο *timer*. Αυτό επιτυγχάνεται στο πρόγραμμα με την χρήση μεταβλητών τύπου *struct tm* και με την συνάρτηση difftime(), η οποία επιστρέφει σε δευτερόλεπτα την διαφορά μεταξύ της παρούσας χρονικής στιγμής και της επιθυμητής χρονικής στιγμής για την εκκίνηση του *timer*.

Στην εφαρμογή που μας ζητείται να αναπτυχθεί θέλουμε να εκτελέσουμε 4 πειράματα μιας ώρας. Θέλουμε να τρέξουμε 3 πειράματα με έναν *timer* κάθε φορά όπου αυτός θα έχει περίοδο 1 δευτερόλεπτο, 0.1 και 0.01 δευτερόλεπτα αντίστοιχα καθώς και 1 πείραμα όπου οι 3 *timers* εκτελούνται ταυτόχρονα. Προς τον σκοπό αυτό έχει αναπτυχθεί και η συνάρτηση main() του προγράμματος, η οποία δέχεται ως ορίσματα (*command line arguments*) το μέγεθος της ουράς (QUEUESIZE) και το πλήθος των νημάτων καταναλωτών(Q). Στην συνέχεια εκτυπώνει, μέσω της συνάρτησης printExecutionMenu(), μενού από το οποίο επιλέγει ο χρήστης ποιο από τα παραπάνω πειράματα επιθυμεί να εκτελεστεί. Στην συνέχεια, με βάση την επιλογή, αρχικοποιούνται τα δεδομένα του προγράμματος, των *timer* , τα *νήματα producers* και εκκινεί το πρόγραμμα. Η συνάρτηση που επιλέχθηκε να εκτελείται περιοδικά (δηλαδή η *TimerFcn*), υπολογίζει 10 συνημίτονα , ωστόσο εύκολα μπορεί να αλλάξει με κάποια άλλη συνάρτηση που δηλώνεται στο αρχείο myFunctions.h , αν και όλες έχουν παρόμοιο και μικρό φόρτο εργασίας. Οι χρόνοι που χρειάζονται για την στατιστική ανάλυση που μας ζητείται, υπολογίζονται με χρήση της gettimeofday()(για τους χρόνους αναμονής χρησιμοποιήθηκε και η clock_gettime(), για ακρίβεια σε μικρούς χρόνους(< 5 *microseconds*) αν προκύψουν) και αποθηκεύονται σε αρχεία *csv*, με κατάλληλο όνομα που δηλώνει τις παραμέτρους εκτέλεσης του προγράμματος από τις οποίες προέκυψαν .

3 Ορθότητα

Η επαλήθευση ορθότητας του προγράμματος του *timer* προκύπτει από τον έγκαιρο τερματισμό του προγράμματος στο αναμενόμενο χρονικό διάστημα καθώς και από πραγματικό πλήθος των εκτελέσεων των συναρτήσεων το οποίο είναι ίδιο με το αναμενόμενο. Η ορθότητα (αξιοπιστία) ως προς τον χρόνο σχολιάζεται εκτενώς σε παρακάτω ενότητες.

4 Αντιμετώπιση time-drifting

Η αντιμετώπιση του φαινομένου *time-drifting*, πραγματοποιείται με τον παρακάτω τρόπο. Κατά την πρώτη περίοδο το νήμα *producer* ορίζεται να “κοιμάται” μέσω της usleep(), για χρόνο ίδιο με αυτόν της επιθυμητής περιόδου. Στο τέλος κάθε περιόδου υπολογίζεται η απόκλιση της πραγματικής περιόδου, δηλαδή του χρόνου μεταξύ δυο διαδοχικών αναθέσεων από το *νήμα producer* στην ουρά, από την επιθυμητή περίοδο και τέλος αυτή η απόκλιση αφαιρείται από το χρονικό διάστημα αδράνειας (χρόνος εντός της usleep()) του *producer* της προηγούμενης περιόδου και έτσι ορίζεται το νέο διάστημα αδράνειας, για την νέα περίοδο. Προφανώς με αυτήν την μέθοδο της **“ανάδρασης”, ενδέχεται να έχουμε αρνητική απόκλιση**(π.χ. περίοδο 0.095 *seconds* αντί 0.1) , **η οποία αντιμετωπίζετε ανάλογα.** Τα αποτελέσματα αυτής της μεθόδου παρουσιάζονται και αναλύονται σε επόμενη ενότητα (5.3).

5 Εκτέλεση σε Pi, μετρήσεις και στατιστικά

5.1) Προεργασία

Προτού το πρόγραμμα εκτελεστεί στο raspberry Pi, εφαρμόστηκαν πειράματα των 2 λεπτών, με ταυτόχρονη εκτέλεση των 3 *timer*, για ένα σύνολο διαφορετικών τιμών των

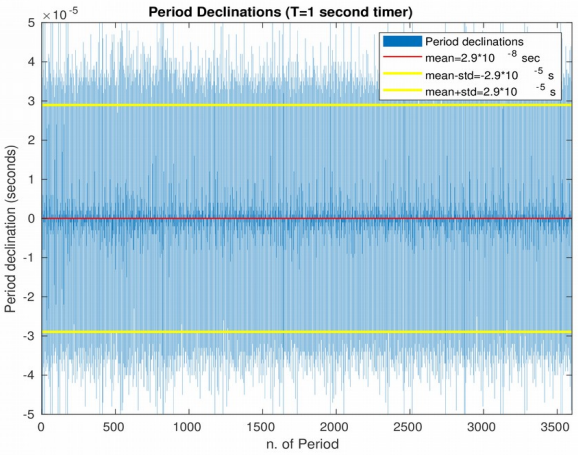
μεταβλητών *QUEUESIZE* και *Q*, με την βοήθεια του bench.sh στο *development pc* (i5-9400H, 8gb ram, Ubuntu 18.04) ώστε να εξασφαλίσουμε ότι κατά την εκτέλεση στο *pi*, δεν θα έχουμε φαινόμενα πλήρωσης (γέμισμα) στην κοινή ουρά αλλά και ότι ο μέσος χρόνος αναμονής θα είναι όσο δύναται μικρότερος. Έτσι καταλήξαμε στο ότι για *QUEUESIZE >= 4* δεν έχουμε γέμισμα της ουράς, ενώ ο μέσος χρόνος αναμονής δεν επηρεάζεται ιδιαίτερα , συμπεράσματα αναμενόμενα, καθώς ο χρόνος εκτέλεσης των συναρτήσεων είναι πολύ μικρότερος σε σχέση με τις περιόδους των *timer*. Επομένως επιλέξαμε για τα πειράματα μας στο raspberry Pi , *QUEUESIZE =4* και *Q=4*. Προφανώς αυτές οι τιμές προκύπτουν για τις ισχύουσες παραμέτρους των πειραμάτων , εάν είχαμε άλλες παραμέτρους (π.χ. πλήθος *timer* αντικειμένων, φόρτος συναρτήσεων *TimerFcn*, περίοδοι κτλ.), ενδέχεται να άλλαζε το επιθυμητό *QUEUESIZE* και *Q*.

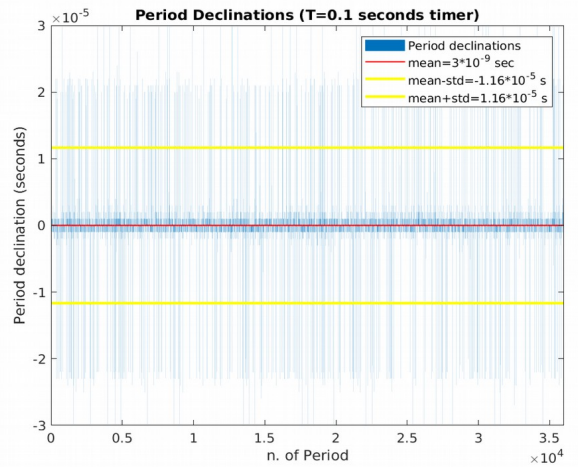
5.2) Εκτέλεση στο Raspberry Pi

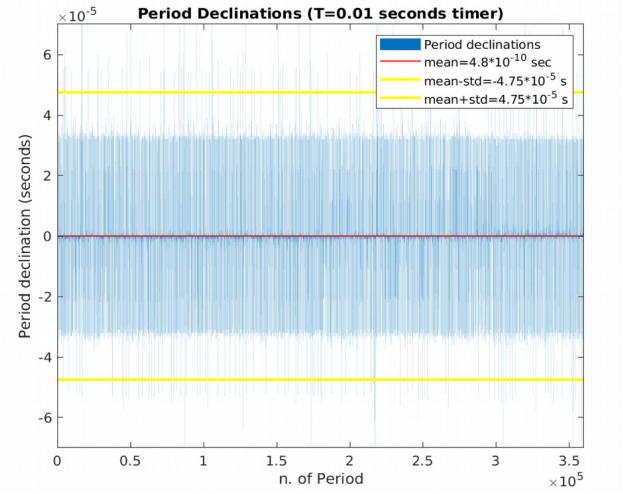
Τα πειράματα πραγματοποιήθηκαν σε Raspberry Pi 4 model B με ram 4 GB, χρησιμοποιήθηκε το *img* που μας δόθηκε με μικρές παραμετροποιήσεις ώστε να είναι λειτουργικό για το μοντέλο Pi 4 και χρησιμοποιήθηκε ο *cross compiler* που μας δόθηκε (link εδώ). Το *compile* έγινε με την εντολή :*"`arm-linux-gnueabihf-gcc`"*, στο dev pc, ενώ στην συνέχεια το εκτελέσιμο μεταφέρθηκε με την εντολή *scp* στο pi και εκτελέστηκε μέσω *ssh*.

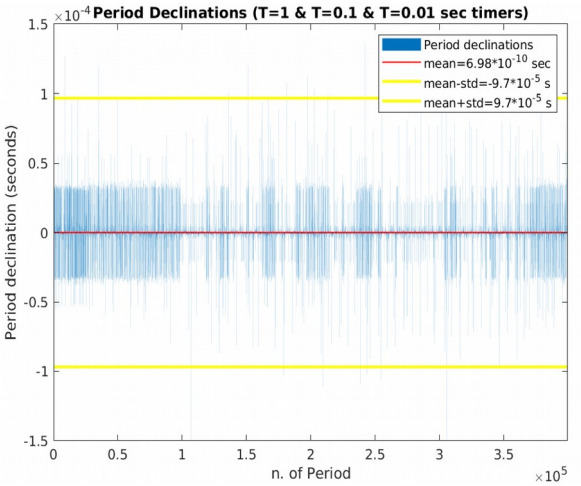
5.3) Μετρήσεις και στατιστικά - Χρόνος ανάθεσης εργασιών από νήμα producer στην ουρά

Σε κάθε πείραμα, υπολογίστηκε για κάθε συνάρτηση (*timerFcn*), η απόκλιση-διαφορά του χρόνου ανάθεσης της στην ουρά από τον επιθυμητό χρόνο (δοσμένη περίοδο). Προφανώς *απόκλιση=(πραγματικός_μεταξύ_διαδοχικών_ανάθέσεων) - (επιθυμητή_περίοδος)*. Στα γραφήματα και στους πίνακες παρακάτω παρουσιάζονται αυτές οι αποκλίσεις σε δευτερόλεπτα. Στα γραφήματα έχουν κοπεί ακραίες τιμές (*spikes)*, που ξεφεύγουν από την κατανομή των δεδομένων.









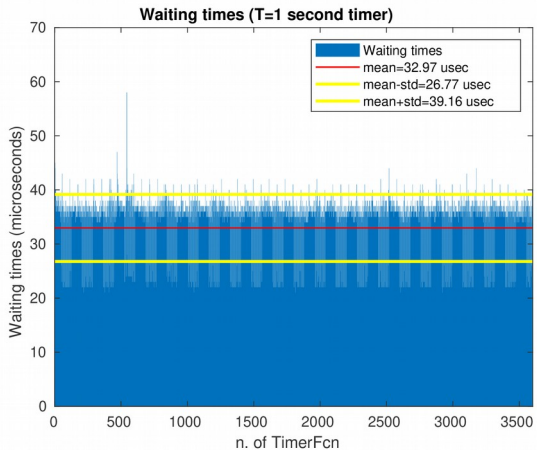
Στατιστικά Απόκλισης πραγματικών περιόδων από επιθυμητές (period declination-drift):

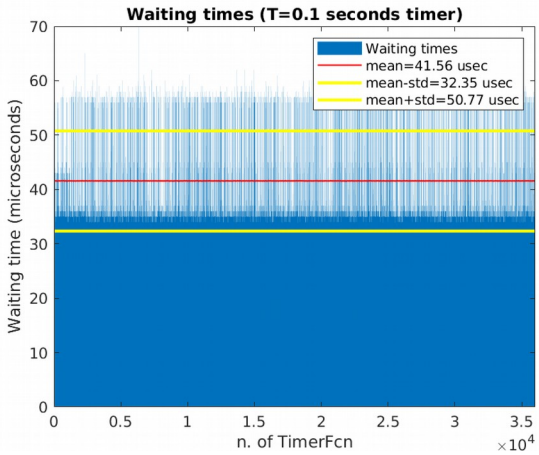
Απόκλιση Περιόδων	Ελάχιστο κατά απόλυτο	Μέγιστο	Μέση Τιμή	Τυπική Απόκλιση	Μέση Τιμή απολύτων	Διάμεσος
Πείραμα 1 (T=1 sec)	0	0.000171 sec	$2.9 * 10^{-8}$ sec	28.9 μs (10^{-6} sec)	21 μs (10^{-6} sec)	0
Πείραμα 2 (T=0.1 sec)	0	0.000165 sec	$3 * 10^{-9}$ sec	11.7 μs (10^{-6} sec)	3.5 μs(10^{-6} sec)	0
Πείραμα 3 (T=0.01 sec)	0	0.007574 sec	$4.8 * 10^{-10}$ sec	47.5 μs (10^{-6} sec)	18.7 μs(10^{-6} sec)	0
Πείραμα 4 (and the 3 timers)	0	0.009127 sec	$6.9 * 10^{-10}$ sec	96.8 μs (10^{-6} sec)	14.2 μs(10^{-6} sec)	0

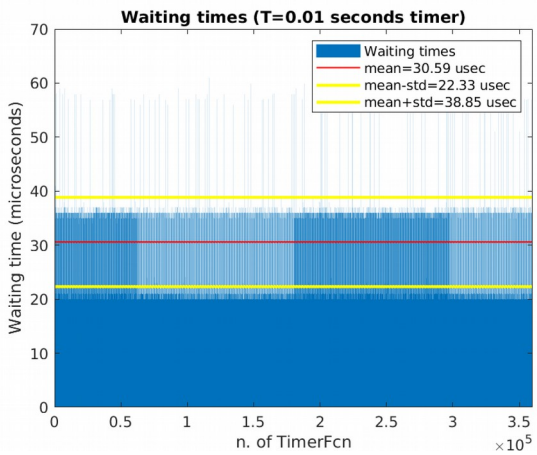
Από τα παραπάνω στατιστικά βλέπουμε, ότι η απόκλιση των πραγματικών περιόδων(***drift* ανά περίοδο**) από τις επιθυμητές περιόδους, παρουσιάζει κατά κάποιο τρόπο μια ταλάντωση γύρω από το 0, ενώ κατά κανόνα λαμβάνει τιμές τάξεις μεγέθους μικρότερες από αυτές των επιθυμητών περιόδων, καθώς σε κάθε πείραμα έχουμε πρακτικά μέση τιμή 0 και τυπική απόκλιση της τάξης των 10^{-5} *seconds* , δηλαδή περίπου 100-1000 φορές μικρότερη από την μικρότερη περίοδο (T=0.01 sec). Αυτό σημαίνει ότι στην συντριπτική τους πλειονότητα οι αποκλίσεις, είναι τόσο μικρές που δεν επηρεάζουν την περιοδική εκτέλεση των συναρτήσεων μας ακόμα και για συστήματα πραγματικού χρόνου όπου η ακρίβεια στους χρόνους είναι μείζονος σημασίας. Βέβαια, οι μέγιστες τιμές τους ειδικά για τα πειράματα 3,4 θα μπορούσαν να μας ανησυχήσουν καθώς είναι πολύ κοντά στην περίοδο T =0.01 *seconds*, ωστόσο δεν παύει παρά να είναι μερικές μεμονωμένες ακραίες τιμές, οι οποίες αποτελούν σπάνιες εξαιρέσεις, κάτι που επιβεβαιώνεται και από τα στατιστικά μας. Επίσης, το γεγονός ότι η μέση τιμή είναι πρακτικά 0, και οι αποκλίσεις των περιόδων, ταλαντώνονται ελαφρώς γύρω από το 0, βοηθάει στην αποφυγή φαινομένων συσσώρευσης και αθροιστικής αύξησης της συνολικής καθυστέρησης μεταξύ ενός συνόλου διαδοχικών περιόδων (***total drift***), κάτι το οποίο ενδέχεται να ήταν καταστροφικό για την αξιοπιστία του προγράμματος μας. Έτσι ο μηχανισμός αντιμετώπισης του φαινομένου *time-drifting* που παρουσιάστηκε στην ενότητα 4, φαίνεται ότι πληροί τους στόχους του σε μεγάλο βαθμό.

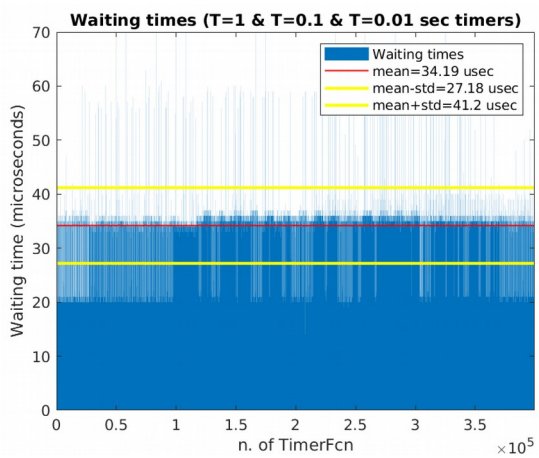
Χρόνος αναμονής μιας συνάρτησης στην ουρά.

Παρακάτω παρουσιάζονται τα στατιστικά των χρόνων αναμονής των συναρτήσεων στην ουρά για τα 4 πειράματα. Ο χρόνος αυτός ισοδυναμεί με τον χρόνο που μεσολαβεί από την ανάθεση μιας συνάρτησης έως ότου αυτή εκτελεστεί. Οι χρόνοι στα γραφήματα και στους πίνακες παρακάτω είναι σε *microseconds.*









Στατιστικά Χρόνων αναμονής στην ουρά:

	Ελάχιστο	Μέγιστο	Μέση Τιμή	Τυπική Απόκλιση
Πείραμα 1 (T=1 sec)	21 usec	58 usec	32.96 usec	6.19 usec
Πείραμα 2 (T=0.1 sec)	32 usec	254 usec	41.56 usec	9.21 usec
Πείραμα 3 (T=0.01 sec)	17 usec	230 usec	30.59 usec	8.26 usec
Πείραμα 4 (and the 3 timers)	13 usec	273 usec	34.19 usec	7 usec

Από τα στατιστικά των χρόνων αναμονής παραπάνω, βλέπουμε ότι αυτοί κυμαίνονται γενικά μεταξύ 30-40 *microseconds*,ενώ σπάνια παρουσιάζουν ελαφρώς μεγαλύτερες αποκλίσεις . Η τιμή λοιπόν του χρόνου αναμονής στην ουρά είναι πάντα τάξεις μεγέθους μικρότερη από τις περιόδους μας(περίπου 500-1000 μικρότερη από την μικρότερη περίοδο μας T=0.01), οπότε μπορούμε να θεωρήσουμε ότι από την στιγμή που ένας *producer* αναθέτει μια εργασία στην κοινή ουρά, αυτή εκτελείται πρακτικά άμεσα, γεγονός που καθιστά το σύστημα μας αξιόπιστο. Σε διαφορετική περίπτωση, για χρόνους αναμονής συγκρίσιμους των περιόδων το σύστημα μας πρακτικά δεν θα ικανοποιούσε τους επιθυμητούς χρονικούς περιορισμούς .

5.4) Ποσοστό χρήσης CPU.

Για την εύρεση του ποσοστού της χρήσης CPU, χρησιμοποιήσαμε στις εκτελέσεις μας την εντολή *time,* (πχ. *time ./programm.out*), έπειτα από την έξοδο της, αθροίσαμε το *user* και το *system time*, τα οποία αθροιστικά αποτελούν τον συνολικό χρόνο του CPU και έπειτα το διαιρέσαμε με τον συνολικό – πραγματικό χρόνο (*real),* ώστε να πάρουμε το ποσοστό χρήσης CPU. Τα αποτελέσματα για τα 4 πειράματα έχουν ως εξής:

	user time	system time	real	CPU usage %
Πείραμα 1 (T=1 sec)	0m0.002s	0m0.320s	60m0.520s	0,00%
Πείραμα 2 (T=0.1 sec)	0m0.002s	0m2.655s	60m3.329s	(0.002+2.65)/(60*60+3.33) = 0.07%
Πείραμα 3 (T=0.01 sec)	0m0.002s	0m30.351s	60m10.482s	(0.002+30.35)/(60*60+10.48) = 0.8%
Πείραμα 4 (and the 3 timers)	0m0.001s	0m33.779s	60m5.627s	(0.001+33.78)/(60*60+5.627) = 0.9%

Όπως ήταν αναμενόμενο, λόγω της αδράνειας βλέπουμε ότι και στα 4 πειράματα μας , το ποσοστό χρήσης CPU, είναι πολύ κοντά στο 0.

6 Τελικά Σχόλια – Συμπεράσματα – Σκέψεις

Το πρόγραμμα που υλοποιήθηκε στα πλαίσια της εργασίας, δηλαδή ο *timer* στην C, αποτελεί μια γενική λύση, που θα μπορούσε με μικρή παραμετροποίηση (κυρίως της *main()*), να προσαρμοστεί σε ένα πλήθος παρόμοιων προβλημάτων όπου είναι επιθυμητή η περιοδική εκτέλεση εργασιών ή ακόμα και η εκτέλεση ή εκκίνηση μιας περιοδικής διεργασίας σε συγκεκριμένη χρονική στιγμή. Ωστόσο ενδέχεται η επιλογή μεγέθους ουράς (*QUEUESIZE*) καθώς και το πλήθος των νημάτων *consumer (Q)*, να μεταβαλλόταν ώστε να είναι πιο κοντά στις ανάγκες του εκάστοτε προβλήματος. Επίσης η στατιστική ανάλυση που προηγήθηκε ενδέχεται να έδινε διαφορετικά ελαφρώς αποτελέσματα. Ωστόσο η εμφάνιση κατά κανόνα πολύ μικρών τιμών στους χρόνους αναμονής στην ουρά και η αρκετά μικρή απόκλιση ως προς τις περιόδους, για τους χρόνους ανάθεσης, σε συνδυασμό με την μηδενική αθροιστική χρονική απόκλιση στα πειράματα, οδηγεί σε εκτέλεση των εργασιών στον επιθυμητό χρόνο, δείχνοντας έτσι ότι το πρόγραμμα που υλοποιήθηκε είναι αξιόπιστο ως προς τον χρόνο, κάτι αποτελεί βασική και κύρια απαίτηση των Ενσωματωμένων συστημάτων πραγματικού χρόνου.

Το αρχείο matlab μέσω του οποίου έγινε η στατιστική ανάλυση των χρόνων:

https://github.com/mikalaki/producer-consumer-timer/blob/master/stats_and_data/statistical_analysis_graphs.m

ενώ τα δεδομένα βρίσκονται στον φάκελο:
https://github.com/mikalaki/producer-consumer-timer/tree/master/stats_and_data/roughData

Project's Repo: https://github.com/mikalaki/producer-consumer-timer

Οδηγίες εκτέλεσης στο Readme.