Ενσωματωμένα συστήματα πραγματικού χρόνου

Υπατία Δάμη

8606

05/10/2020

Σκοπός εργασίας :

Σκοπός αυτής της εργασίας ήταν η δημιουργία ενός timer μέσω νημάτων producer και consumer. Η υλοποίηση αυτή αποσκοπεί στη μεγαλύτερη ακρίβεια του timer όσον αφορά την τήρηση της περιόδου του. Τα νήματα producer είναι υπεύθυνα για την τοποθέτηση των συναρτήσεων προς εκτέλεση σε μία ουρά, και η ρύθμιση της περιόδου για να έχουμε όσο το δυνατόν μικρότερο drift . Τα νήματα consumer είναι υπεύθυνα για την ανάκτηση των συναρτήσεων από την ουρά και για την εκτέλεσή τους.

Ανάλυση κώδικα :

Η υλοποίηση της εργασίας έγινε στη γλώσσα C . Υλοποιήθηκε η δομή timer και οι συναρτήσεις

- startFcn(int* FunChoice, int* ArgChoice)
- Timer *timerInit(int Per,int Tasks,int Delay,int FunChoice,int Args);
- StopFcn(int ArgChoice , queue *q)
- Start(void *args,pthread_t prod)
- Startat(int t,int y,int m,int h,int min ,int sec,void *args,pthread_t prod)
- ErrorFcn(queue *q);
- calculate_metrics(int procon,int arraySize ,long int elapse[]);

Επίσης διαμορφώθηκαν κατάλληλα δομές και συναρτήσεις της προηγούμενης εργασίας ,όπως τα νήματα producer και consumer , και οι δομές για την εκτέλεση των συναρτήσεων και των ορισμάτων τους.

Void startFcn(int* FunChoice, int* ArgChoice):

Η συνάρτηση startFcn καλείται πριν τη δημιουργία του timer ,για να επιλαγεί η συνάρτηση που θα εκτελείται περιοδικά και τα ορίσματα που θα δέχεται . Στα πλαίσια της εργασίας η επιλογή της συνάρτησης και των ορισμάτων γίνεται τυχαία , επιλέγοντας με τη χρήση της rand() , μια από τις 3 συναρτήσεις που είναι αποθηκευμένες σε ένα πίνακα δεικτών σε συναρτήσεις ‘Functions’ και ορίσματα που είναι αποθηκευμένα στην καθολική δομή ‘AllArgs’.Οι επιλογές για τη συνάρτηση και τα ορίσματα αποθηκεύονται στις μταβλητές FunChoice και ArgChoice.

Timer *timerInit(int Per,int Tasks,int Delay,int FunChoice,int Args):

Η συνάρτηση timerInit είναι υπεύθυνη για την αρχικοποίηση του timer. Δέχεται ως ορίσματα την επιθυμητή περίοδο, τον αριθμό των συναρτήσεων που πρόκειται να εκτελεστούν , την επιθυμητή καθυστέρηση πριν την εκκίνηση του και τις μεταβλητές που έχει ήδη επεξεργαστεί η startFcn, με την επιλογή της συνάρτησης προς εκτέλεση και τα ορίσματά

της.Η μεταβλητή ‘wrk’ του timer είναι μια δομή η οποία διαθέτει ένα δείκτη προς συνάρτηση ‘*TimerFcn ‘, και ένα δείκτη στη δομή με τα ορίσματα ‘*arg’.

void Start(void *args,pthread_t prod):

Η συνάρτηση αυτή είναι υπεύθυνη για την εκκίνιση του timer. Δέχεται ως ορίσματα τη δομή args και το νήμα prod που θα είναι υπεύθυνο για την τήρηση της περιόδου του timer.Η δομή args είναι τα ορίσματα που θα περαστούν στην συνάρτηση pthread_create() ,και περιέχει ένα δείκτη στο αντικείμενο timer και ένα δείκτη στην ουρά fifo που θα τοποθετούνται οι συναρτήσεις.

void Startat(int y,int m,int d,int h,int min ,int sec,void *args,pthread_t prod)

Η συνάρτηση Startat εκκινεί τον timer σε μια συγκεκριμένη ημερομηνία και ώρα. Χρησιμοποιώντας τη δομή ‘tm’ και την δομή τύπου ‘time_t’, αποθηκεύουμε την παρούσα ημερομηνία και ώρα. Έπειτα υπολογίζουμε τη διαφορά με την επιθυμητή χρονολογία και τη μετατρέπουμε σε δευτερόλεπτα. Τέλος βάζουμε το νήμα να κοιμηθεί για το χρονικό διάστημα που υπολογίσαμε και έπειτα να εκκινήσει τον timer.

void *producer (void *args):

Το νήμα producer είναι υπεύθυνο για την τοποθέτηση των εργασιών στην ουρά fifo αλλά και για την τήρηση της ακρίβειας της περιόδου του timer. Η διατήρηση της περιόδου ακολουθεί την εξής λογική. Μετά από πειράματα για διαφορετικές περιόδους παρατηρήθηκε πως το σφάλμα της συνάρτησης usleep() είναι το πολύ μια τάξη μεγέθους κάτω από την τιμή που πήρε ως όρισμα, αλλά συνήθως αρκετές τάξεις μεγέθους μικρότερη. Επιλέγεται λοιπόν σύμφωνα με αυτή την παρατήρηση μια τιμή, ανάλογα με την τάξη μεγέθους της περιόδου , η οποία αντιπροσωπεύει το διάστημα που θέλουμε να κοιμηθεί το νήμα μέσω της usleep.Πχ για επιθυμητή περίοδο 1000000 us , το νήμα θα κοιμηθεί 950000 us. Το διάστημα που υπολείπεται για να ολοκληρωθεί η περίοδος , μετριέται σε ένα βρόχο while με τη χρήση της gettimeofday() ,μέχρι η μεταβλητή που ελέγχεται να πάρει τιμή ίδια με το το διάστημα που υπολείπεται. Η υλοποίηση αυτή προσθέτει κάποια επιβάρυνση στη CPU , αλλά την μικρότερη δυνατή έτσι ώστε να πετύχουμε μεγάλη ακρίβεια στον timer. Στη μεταβλητή time_drift αποθηκεύεται η απόκλιση από την επιθυμητή περίοδο όταν προστεθεί η εργασία στην ουρά. Συγκεκριμένα με τη χρήση της gettimeofday μετριέται το διάστημα που μεσολαβεί από τη στιγμή που συμπληρώνεται η περίοδος μέχρι να τοποθετηθεί η εργασία στην ουρά. Στην σπάνια περίπτωση που η usleep κοιμηθεί παραπάνω από την επιθυμητή περίοδο, τότε παρακάμπτουμε το βήμα του υπολογισμού του εναπομείνοντος διαστήματος και προσθέτουμε το επιπλέον διάστημα που κοιμήθηκε το νήμα στη μεταβλητή time_drift .Επιπλέον , το νήμα producer είναι υπεύθυνο για το διαχειρισμό της περίπτωσης που η ουρά θα γεμίσει. Στην περίπτωση αυτή καλείται η συνάρτηση ErrorFcn, η οποία υπολογίζει τον χρόνο που περνάει μέχρι να ελευθερωθεί η ουρά , και τον επιστρέφει στο νήμα producer. Ο

producer έπειτα υπολογίζει σύμφωνα με το χρόνο αυτό τις περιόδους που χάθηκαν στο διάστημα αυτό. Το λιγότερο είναι μία περίοδος , διότι ο έλεγχος για τη γεμάτη ουρά γίνεται αναγκαστικά αφού θα έχει κοιμηθεί το νήμα .Το ζήτημα είναι να μπορέσει το νήμα να κρατήσει το βήμα της περιόδου έτσι ώστε αν χαθεί μια περίοδος, να τοποθετηθεί η εργασία στην επόμενη .Η διαδικασία που ακολουθείται είναι η ίδια με την κανονική , το νήμα κοιμάται και υπολογίζεται ο εναπομείνων χρόνος, αφαιρώντας το διάστημα που μέτρησε η ErrorFcn από το χρόνο που θα κοιμηθεί το νήμα και υπολογίζοντας ανάλογα με το χρόνο που πέρασε, το time_drift μέχρι να τοποθετηθεί η επόμενη συνάρτηση στην ουρά.

void *consumer (void *q):

Το νήμα consumer είναι υπεύθυνο για την παραλαβή των συναρτήσεων από την ουρά και την εκτέλεσή τους. Το νήμα εκτελεί συναρτήσεις συνεχώς , όσο η ουρά είναι γεμάτη , και τερματίζει όταν τερματίσουν όλοι οι timers. Μπορούν να υπάρχουν επίσης πολλά νήματα consumers που τρέχουν παράλληλα και εκτελούν συναρτήσεις από την ίδια ουρά. Ο consumer επίσης μετράει τον χρόνο που μεσολαβεί από τότε που τοποθετήθηκε η συνάρτηση στην ουρά , μέχρι να εκτελεστεί .Η στιγμή που τοποθετήθηκε η συνάρτηση μετριέται από τον producer και αποθηκεύεται στη δομή με τα ορίσματα της δομής workFunction , έτσι να έχει πρόσβαση σε αυτήν ο consumer και να υπολογίσει το επιθυμητό διάστημα.

long int ErrorFcn(queue *q) :

Η συνάρτηση αυτή καλείται από τον producer στην περίπτωση που η ουρά γεμίσει με συναρτήσεις .Περιμένει με τη χρήση της pthread_cond_wait το σήμα που θα σταλεί όταν ελευθερωθεί η ουρά, υπολογίζει τον χρόνο που πέρασε , και τον επιστρέφει στον producer.

void StopFcn(int ArgChoice,queue *q):

Η συνάρτηση StopFcn καλείται όταν κάποιος timer τερματίσει .Ελευθερώνει τα κατειλλημένα από τον εκάστοτε timer ορίσματα της δομής Args και μειώνει κατά 1 την μεταβλητή timer_finished. Στη μεταβλητή αυτή προστίθεται μια μοναδα για κάθε timer που ξεκινάει και αφαιρείται μια για κάθε timer που τερματίζει. Έτσι όταν θα έχει τιμή 0 σημαίνει πως όλοι οι timers θα έχουν τερματίσει και οι consumers ελέγχοντάς την θα τερματίσουν επίσης .

Στατιστική Ανάλυση:

Πραγματοποιήθηκαν 4 πειράματα διάρκειας μιας ώρας, 3 με ξεχωριστές εκτελέσεις των timer με περιόδους 0.01,0.1,1 sec και μια φορά και με τους τρεις timer μαζι και έγινε στατιστική ανάλυση για τους εξής χρόνους:

- χρόνος που μεσολαβεί από τη στιγμή που υπολογίζεται η περίοδος μέχρι να βάλει ο producer την εργασία στην ουρά (στον πίνακα drift)
- χρόνος που μεσολαβεί από τη στιγμή που ο producer τοποθετεί την εργασία στην ουρά μέχρι να την παραλάβει ο consumer (στον πίνακα consumer)

Τα αποτελέσματα των πειραμάτων φαίνονται στους παρακάτω πίνακες,με τους χρόνους να είναι εκφρασμένοι σε usecs. Παρατηρούμε πως ο μέσος χρόνος του είναι πολύ μικρότερος όταν οι timers τρέχουν ξεχωριστα, πράγμα που είναι λογικό , καθώς χρησιμοποιούν την ίδια ουρά και όταν κλειδώνει από κάποιον , δεν μπορεί να χρησιμοποιηθεί από τους υπόλοιπους. Παρόλα αυτά η διάμεσος είναι μικρή και στις 4 περιπτώσεις , πράγμα που σημαίνει πως τις περισσότερες φορές το drift είναι μικρό.

1s period:

	drift	consumer
average	1.051	25.45
median	1	24
min	1	1
max	4	78
st_dev	0.253	5.81

0.1s period:

	drift	consumer
average	0.416	31.65
median	1	22
min	1	19
max	4	76
st_dev	96.27	11.87

0.01s period:

	drift	consumer
average	0.041	21.74
median	1	21
min	2	20
max	9142	93
st_dev	198.55	5.598

All timers together:

	drift	consumer1
average	51.79	31.21
median	1	30
min	1	1
max	13531	10444
st_dev	782.34	42.93

Τέλος, διερευνήθηκε ο βέλτιστος αριθμός consumers . Έγιναν πειράματα για 1 , 2 και 3 consumers. Οι χρόνοι για δύο consumers ήταν ελαφρώς καλύτερη απότι για έναν, ενώ με τρείς δεν υπήρξε σημαντική διαφορά. Επομένως κρίνεται κατάλληλος αριθμός consumers το 2.

Τα πειράματα διεξήχθησαν σε Raspberry pi 3 b.