**ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ**

**ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ KΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ**



Εργαστήριο Υπολογιστικών Συστημάτων

**Άσκηση 2:** *Οδηγός ασύρματου δικτύου αισθητήρων στο λειτουργικό σύστημα Linux*

**Ημερομηνία:** *17/04/2016*

**Ομάδα cslabb06**

***Πετρόπουλος-Τράκας Ευθύμης (031 11525)***

***Ροδίτης-Κουτσαντώνης Ορέστης (031 11052)***

**2η Άσκηση**

Αντικείμενο της παρούσας εργατηριακής άσκησης ειναι η υλοποίηση ενός οδηγού συσκευής για ένα ασύρματο δίκτυο αισθητήρων κάτω από το λειτουεγικό σύστημα Linux.Το συγκεκριμένο δίκτυο διαθέτει έναν αριθμό από αισθητήρες (ασύρματες καρτες Crossbow MPR2400CA με αισθητήρες τάσης, θερμοκρασίας και φωτεινότητας MDA100CB)και έναν σταθμό βάσης (ασύρματη κάρτα MPR2400CA και διασύνδεση USB MIB520CB). Ο σταθμός βάσης συνδέεται μέσω USB με υπολογιστικό σύστημα Linux στο οποίο και θα εκτελείται ο ζητούμενος οδηγός συσκευής (ας τον ονομάσουμε Lunix:TNG).

Οι αισθητήες αναφέρουν περιοδικά το αποτέλεσμα τριών διαφορετικών μετρήσεων: της τάσης της μπαταρίας που τους τροφοδοτεί, της θερμοκρασίας και της φωτεινότητας του χώρου που βρίσκονται. Τα δεδομένα μεταφέρονται μέσω ενός δικτύου mesh, από εναλλακτικές διαδρομές, έτσι ώστε το δίκτυο να προσαρμόζεται αυτόματα στην περίπτωση όπου ένας ή περισσότεροι αισθητήρες είναι εκτός της εμβέλειας του σταθμού βάσης.

Ο σταθμός βάσης λαμβάνει πακέτα με δεδομένα μετρήσεων, τα οποία προωθεί μέσω διασύνδεσης USB στο υπολογιστικό σύστημα. Η διασύνδεση υλοποιείται με κύκλωμα Serial over USB, για το οποίο ήδη ο πυρήνας διαθέτει ενσωματωμένους οδηγούς, οπότε τα δεδομένα όλων των μετρήσεων όλων των αισθητήρων εμφανίζονται σε μια εικονική σειριακή θύρα, **/dev/ttyUSB1**.

**Ζητούμενα της Άσκησης**

Ζητείται η κατασκευή οδηγού συσκευής χαρακτήρων (character device driver), ο οποίος θα λαμβάνει τα δεδομένα των μετρήσεων από το δίκτυο αισθητήρων και θα τα εξλαγει στον χώρο χρήστη σε διαφορετικές συσκευές, ανάλογα με το είδος της μέτρησης και τον αισθητήρα από όπου προέρχεται. Έτσι, αντί να υπάρχει ένα και μοναδικό **/dev/ttyUSB1**, θα υπάρχουν τρεις συσκευές για κάθε αισθητήρα, π.χ. Για τον πρώτο αισθητήρα οι **/dev/lunix0-batt, /dev/lunix0-temp, /dev/lunix0-light**.

**Άσκηση 2: Υλοποίηση:**

*/\**

*\* lunix-chrdev.c*

*\**

*\* Implementation of character devices*

*\* for Lunix:TNG*

*\**

*\* Efthymis Petropoulos-Trakas*

*\* Orestis Roditis-Koutsantonis*

*\**

*\*/*

*#include <linux/mm.h>*

*#include <linux/fs.h>*

*#include <linux/init.h>*

*#include <linux/list.h>*

*#include <linux/cdev.h>*

*#include <linux/poll.h>*

*#include <linux/slab.h>*

*#include <linux/sched.h>*

*#include <linux/ioctl.h>*

*#include <linux/types.h>*

*#include <linux/module.h>*

*#include <linux/kernel.h>*

*#include <linux/mmzone.h>*

*#include <linux/vmalloc.h>*

*#include <linux/spinlock.h>*

*#include "lunix.h"*

*#include "lunix-chrdev.h"*

*#include "lunix-lookup.h"*

*/\**

*\* Global data*

*\*/*

*struct cdev lunix\_chrdev\_cdev;*

*/\**

*\* Just a quick [unlocked] check to see if the cached*

*\* chrdev state needs to be updated from sensor measurements.*

*\*/*

*static int lunix\_chrdev\_state\_needs\_refresh(struct lunix\_chrdev\_state\_struct \*state)*

*{*

*struct lunix\_sensor\_struct \*sensor;*

*WARN\_ON ( !(sensor = state->sensor));*

*// return 1;*

*if ((state->buf\_timestamp) < (sensor->msr\_data[state->type]->last\_update)) {*

*return 1;*

*}*

*/\* The following return is bogus, just for the stub to compile \*/*

*return 0;*

*}*

*/\**

*\* Updates the cached state of a character device*

*\* based on sensor data. Must be called with the*

*\* character device state lock held.*

*\*/*

*static int lunix\_chrdev\_state\_update(struct lunix\_chrdev\_state\_struct \*state)*

*{*

*struct lunix\_sensor\_struct \*sensor;*

*sensor = state->sensor;*

*uint32\_t raw\_data;*

*int i, j, power;*

*// if (lunix\_chrdev\_state\_needs\_refresh(state) == 0)*

*// return -EAGAIN;*

*/\**

*\* Grab the raw data quickly, hold the*

*\* spinlock for as little as possible.*

*\*/*

*spin\_lock(&sensor->lock);*

*raw\_data = sensor->msr\_data[state->type]->values[0];*

*state->buf\_timestamp = sensor->msr\_data[state->type]->last\_update;*

*spin\_unlock(&sensor->lock);*

*if (state->type == BATT)*

*raw\_data = lookup\_voltage[raw\_data];*

*else if (state->type == TEMP)*

*raw\_data = lookup\_temperature[raw\_data];*

*else*

*raw\_data = lookup\_light[raw\_data];*

*for (i = 0; i < 8; i ++)*

*state->buf\_data[i] = '0';*

*for (i = 0; i < 6; i++) {*

*power = 1;*

*for (j = 0; j < 4-i; j++)*

*power = power \* 10;*

*state->buf\_data[i] = (raw\_data / power) + '0';*

*raw\_data = raw\_data % power;*

*}*

*for (i = 6; i > 0; i--) {*

*if (i > 2) {*

*state->buf\_data[i] = state->buf\_data[i-1];*

*} else if (i == 2) {*

*state->buf\_data[i] = ',';*

*} else {*

*continue;*

*}*

*}*

*state->buf\_data[7] = '\n';*

*/\* Why use spinlocks? See LDD3, p. 119 \*/*

*/\**

*\* Any new data available?*

*\*/*

*/\**

*\* Now we can take our time to format them,*

*\* holding only the private state semaphore*

*\*/*

*debug("leaving\n");*

*return 0;*

*}*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Implementation of file operations*

*\* for the Lunix character device*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*static int lunix\_chrdev\_open(struct inode \*inode, struct file \*filp)*

*{*

*/\* Declarations \*/*

*enum lunix\_msr\_enum sensor\_type;*

*int ret, minor, sensor\_num, i;*

*debug("entering\n");*

*ret = -ENODEV;*

*if ((ret = nonseekable\_open(inode, filp)) < 0)*

*goto out;*

*/\**

*\* Associate this open file with the relevant sensor based on*

*\* the minor number of the device node [/dev/sensor<NO>-<TYPE>]*

*\*/*

*minor = iminor(inode);*

*sensor\_num = minor / 8;*

*sensor\_type = minor % 8;*

*/\* Allocate a new Lunix character device private state structure \*/*

*struct lunix\_chrdev\_state\_struct \*ptr;*

*ptr = (struct lunix\_chrdev\_state\_struct \*) kmalloc (sizeof(struct lunix\_chrdev\_state\_struct), GFP\_KERNEL);*

*ptr->type = sensor\_type;*

*ptr->sensor = &lunix\_sensors[sensor\_num];*

*ptr->buf\_lim = LUNIX\_CHRDEV\_BUFSZ;*

*for(i = 0; i < ptr->buf\_lim; i++)*

*ptr->buf\_data[i] = NULL;*

*ptr->buf\_timestamp = get\_seconds();*

*sema\_init(&(ptr->lock), 1);*

*/\* Save device private state structure \*/*

*filp->private\_data = ptr;*

*out:*

*debug("leaving, with ret = %d\n", ret);*

*return ret;*

*}*

*static int lunix\_chrdev\_release(struct inode \*inode, struct file \*filp)*

*{*

*kfree(filp->private\_data);*

*return 0;*

*}*

*static long lunix\_chrdev\_ioctl(struct file \*filp, unsigned int cmd, unsigned long arg)*

*{*

*/\* Why? \*/*

*return -EINVAL;*

*}*

*static ssize\_t lunix\_chrdev\_read(struct file \*filp, char \_\_user \*usrbuf, size\_t cnt, loff\_t \*f\_pos)*

*{*

*ssize\_t ret;*

*struct lunix\_sensor\_struct \*sensor;*

*struct lunix\_chrdev\_state\_struct \*state;*

*int i, j, power, temp;*

*state = filp->private\_data;*

*WARN\_ON(!state);*

*sensor = state->sensor;*

*WARN\_ON(!sensor);*

*/\* Lock? \*/*

*/\**

*\* If the cached character device state needs to be*

*\* updated by actual sensor data (i.e. we need to report*

*\* on a "fresh" measurement, do so*

*\*/*

*if (\*f\_pos == 0) {*

*while(lunix\_chrdev\_state\_needs\_refresh(state) == 0){*

*if (wait\_event\_interruptible(sensor->wq, (lunix\_chrdev\_state\_needs\_refresh(state))) == -ERESTARTSYS) {*

*return -EINTR;*

*}*

*}*

*down\_interruptible(&state->lock);*

*lunix\_chrdev\_state\_update(state);*

*up(&state->lock);*

*}*

*down\_interruptible(&state->lock);*

*/\* End of file \*/*

*if (cnt <= 8 - \*f\_pos) {*

*\*f\_pos += cnt;*

*if (\*f\_pos >= 8) {*

*\*f\_pos = 0;*

*}*

*} else {*

*\*f\_pos = 0;*

*cnt = 8 - \*f\_pos;*

*}*

*if (copy\_to\_user(usrbuf, &(state->buf\_data[\*f\_pos]), cnt) != 0) {*

*printk(KERN\_ERR "Could Not Copy all Bytes\n");*

*}*

*/\* Auto-rewind on EOF mode? \*/*

*out:*

*/\* Unlock? \*/*

*up(&state->lock);*

*ret = cnt;*

*return ret;*

*}*

*static int lunix\_chrdev\_mmap(struct file \*filp, struct vm\_area\_struct \*vma)*

*{*

*return -EINVAL;*

*}*

*static struct file\_operations lunix\_chrdev\_fops =*

*{*

*.owner = THIS\_MODULE,*

*.open = lunix\_chrdev\_open,*

*.release = lunix\_chrdev\_release,*

*.read = lunix\_chrdev\_read,*

*.unlocked\_ioctl = lunix\_chrdev\_ioctl,*

*.mmap = lunix\_chrdev\_mmap*

*};*

*int lunix\_chrdev\_init(void)*

*{*

*/\**

*\* Register the character device with the kernel, asking for*

*\* a range of minor numbers (number of sensors \* 8 measurements / sensor)*

*\* beginning with LINUX\_CHRDEV\_MAJOR:0*

*\*/*

*int ret;*

*dev\_t dev\_no;*

*unsigned int lunix\_minor\_cnt = lunix\_sensor\_cnt << 3;*

*debug("initializing character device\n");*

*cdev\_init(&lunix\_chrdev\_cdev, &lunix\_chrdev\_fops);*

*lunix\_chrdev\_cdev.owner = THIS\_MODULE;*

*dev\_no = MKDEV(LUNIX\_CHRDEV\_MAJOR, 0);*

*/\* ? \*/*

*/\* register\_chrdev\_region? \*/*

*ret = register\_chrdev\_region(dev\_no, lunix\_minor\_cnt, "Lunix");*

*/\* end of register\_chrdev\_region \*/*

*if (ret < 0) {*

*debug("failed to register region, ret = %d\n", ret);*

*goto out;*

*}*

*/\* ? \*/*

*/\* cdev\_add? \*/*

*ret = cdev\_add(&lunix\_chrdev\_cdev, dev\_no, lunix\_minor\_cnt);*

*/\* end of cdev\_add \*/*

*if (ret < 0) {*

*debug("failed to add character device\n");*

*goto out\_with\_chrdev\_region;*

*}*

*debug("completed successfully\n");*

*return 0;*

*out\_with\_chrdev\_region:*

*unregister\_chrdev\_region(dev\_no, lunix\_minor\_cnt);*

*out:*

*return ret;*

*}*

*void lunix\_chrdev\_destroy(void)*

*{*

*dev\_t dev\_no;*

*unsigned int lunix\_minor\_cnt = lunix\_sensor\_cnt << 3;*

*debug("entering\n");*

*dev\_no = MKDEV(LUNIX\_CHRDEV\_MAJOR, 0);*

*cdev\_del(&lunix\_chrdev\_cdev);*

*unregister\_chrdev\_region(dev\_no, lunix\_minor\_cnt);*

*debug("leaving\n");*

*}*

Ακολουθεί σύντομη επεξήγηση των επιλογών που κάναμε όσον αφορά τις προσθήκες κώδικα ώστε ο οδηγός να δουλέψει σωστά.

Αρχικά υλοποιήσαμε την *lunix\_chrdev\_init()*. Η συνάρτηση αρχικοποιεί τα structs της συσκευής χαρακτήρων και των συναρτησεων που θα χρησιμοποιηθούν από τα ανοικτά αρχεία της συσκευής, με την *cdev\_init()*. Στη συνέχεια, συνδέει τον οδηγό χαρακτήρων με αρχεία με συγκεκριμένους minor και major αριθμούς, τα οποία έχουμε δημιουργήσει πριν φορτώσουμε τον οδηγό. Συνεπώς, το άνοιγμα των εν λόγω ειδικών αρχείων και κάθε λειτουργία που σχετίζεται με αυτά θα πραγματοποιείται πλέον από τον οδηγό μας. Τέλος, προσθέτει τη συσκευή χαρακτήρων που δημιουργήσαμε στις υπόλοιπες συσκευές χαρακτήρων που υπάρχουν ήδη στο σύστημα.

Έπειτα υλοποιήσαμε τη συνάρτηση *lunix\_chrdev\_open(struct inode \*inode, struct file \*filp)*. Η συνάρτηση αυτή καλείται όταν κάποια διεργασία θέλει να ανοίξει ένα από τα ειδικά αρχεία τα οποία έχουμε κάνει register στον οδηγό μας. Με βάση τον minor αριθμό του αρχείου, συμπεραίνουμε ποιον αισθητήρα και ποιος είδος μέτρησης αφορά. Εν συνεχεία δεσμεύουμε μνήμη για μια δομή *struct lunix\_chrdev\_state\_struct* που θα αφορά το συγκεκριμένο αρχείο, αναθέτουμε στο πεδίο της *sensor* το struct του συγκεκριμένου αισθητήρα από τον οποίο θα διαβάζουμε δεδομένα για το ανοιχτό αρχείο, και τέλος αναθέτουμε τον pointer του *struct lunix\_chrdev\_state\_struct* που δημιουργήσαμε στο πεδίο *private\_data* του *struct\_file* που δημιουργήθηκε με το άνοιγμα του αρχείου. Με το πέρας της εκτέλεσης της συνάρτησης, η συσκευή χαρακτήρων μας είναι έτοιμη να ανταποκριθεί σε λειτουργίες που σχετίζονται με τα ανοιχτά αρχεία για τα οποία είναι υπεύθυνη.

Τελικά φτάσαμε στο σημείο όπου έπρεπε να υλοποιηθεί η *lunix\_chrdev\_read()*. Θα υλοποιήσουμε πρώτα τις συναρτήσεις τις οποίες η read θα χρησιμοποιήσει.

Η πρώτη από αυτές είναι η *lunix\_chrdev\_state\_needs\_refresh()*. Αυτή η συνάρτηση ελέγχει, όταν εξαντλήσουμε τα διαθέσιμα δεδομένα της read, αν υπάρχει μια μέτρηση πιο πρόσφατη από αυτή που χρησιμοποίησε η read τελευταία φορά. Για τον σκοπό αυτό συγκρίνει τα timestamps του *struct lunix\_chrdev\_state\_struct* που ανήκει στο αρχείο και της τελευταίας μέτρησης του αισθητήρα, και αν υπάρχει καινούρια μέτρηση επιστρέφει την αντίστοιχη τιμή.

Σειρά έχει η *lunix\_chrdev\_state\_update()*. Η συνάρτηση αυτη, όταν κληθεί, κλειδώνει τη δομή του αισθητήρα που σχετίζεται με το αρχείο, λαμβάνει την μέτρηση που υπάρχει σε αυτόν όπως ακριβώς είναι εκείνη τη στιγμή, και επιστρέφει το κλείδωμα. Αφού πάρει τα καθαρά δεδομένα, αναλαμβάνει σύφμωνα με το documentation του οδηγού να τα μετατρέψει σε μορφή που να βγάζει νόημα για εμάς. Συνεπώς, έχουμε υλοποιήσει τη λειτουργία της μετατροπής της τιμής του αισθητήρα σε μια νέα κατάλληλη τιμή, χρησιμοποιώντας το generated lookup tables, και έπειτα μετατρέπουμε την αριθμητική τιμή που παράξαμε στην κατάλληλη ακολουθία χαρακτήρων, την οποία αποθηκεύουμε στον character buffer του αρχείου μας, βάζοντας στο κατάλληλο σημείο του πίνακα την υποδιαστολή και στην τελευταία θέση τον χαρακτήρα αλλαγής γραμμής. Αφού τελειώσει λοιπόν η εκτέλεση της συνάρτησης, ο character buffer του αρχείου έχει την τελευταία διαθέσιμη μέτρηση από τον αισθητήρα που του αντιστοιχεί, σε μορφή έτοιμη για διάβασμα ανά χαρακτήρα.

Τέλευταία υλοποιήσαμε την συνάρτηση *lunix\_chrdev\_read()* η οποία καλείται κάθε φορά που μια διεργασία θέλει να διαβάσει δεδομένα για τον αισθητήρα και τη μέτρηση που αντιστοιχεί στο ανοιχτό αρχείο. Δέχεται ως ορίσματα το *struct file* του αρχείου, τη θέση μνήμης του userspace buffer όπου θα διαβαστούν οι χαρακτήρες από το αρχείο,ο αριθμός των bytes που ζητά η διεργασία χρήστη να διαβαστούν, και τέλος η θέση του αρχείου από όπου πρόκειται να ξεκινήσει το διαβασμα χαρακτήρων η διεργασία χρήστη. Από τον buffer που έχουμε στο *private\_data* στο file struct του αρχείου, θα χρησιμοποιήσουμε 8 μόνον χαρακτήρες κάθε δεδομένη στιγμή ώστε να κρατάμε την τρέχουσα μέτρηση που δεν έχει διαβαστεί. Θέλουμε 1 θέση για κάθε ψηφίο της μέτρησης, όπου έχουμε κάθε φορά ακριβώς 6 ψηφία, οπότε για αρχη 6 θέσεις. Μετά θέλουμε άλλη μια θέση για τον χαρακτήρα της υποδιαστολής, και άλλον έναν χαρακτήρα αλλαγής γραμμής στο τέλος.

Η τακτική που ακολουθήσαμε είναι ότι από τη στιγμή που θα γίνει update η αποθηκευμένη μέτρηση του αρχείου, θα πρέπει να διαβαστούν και οι 8 χαρακτήρες της μέτρησης που είναι αποθηκευμένη στον Buffer, πριν ανανεώσουμε τη μέτρηση αυτή από τιν αισθητήρα. Αν λοιπόν η αίτηση για διάβασμα μέσω της read απαιτή bytes που είναι πέρα από τα 8 της μέτρησης, είτε σε μία μόνο κλήση είτε μετά από διαδοχικές κλήσεις, διαβάζονται μόνο όσοι χαρακτήρες απομένουν μέχρι και την αλλαγή γραμμής και έπειτα το \**f\_pos* γίνεται πάλι 0, ώστε στην επόμενη κλήση της συνάρτησης να ανανεωθεί η μέτρηση από τον αισθητήρα και το διάβασμα να ξεκινήσει πάλι από την αρχή του buffer. Αν δεν διαβαστούν όλοι οι χαρακτήρες του buffer με μία μόνο κλήση, η τιμή του *\*f\_pos* αυξάνεται ανάλογα ώστε να ξέρουμε από ποιο σημείο του buffer να ξεκινήσουμε στην επόμενη κλήση της read.

Όπως είπαμε πριν, όταν το *\*f\_pos* ήταν ίσο με 0, η read αναλάμβανε να ανανεώσει την τιμή του buffer απευθείας από τον αισθητήρα. Για αυτόν τον σκοπό έλεγχε αν υπήρχε μέτρηση νεώτερη από αυτή που περιείχε ήδη ο buffer. Αν δεν περιείχε, έμπαινε σε sleep στο *wait\_queue* του αντίστοιχου *struct sensor* περιμένοντας να ξυπνήσει όταν έρθει μια νέα μέτρηση. Στον αρχικό σχεδιασμό του κώδικα, ολόκληρη η λειτουργία του ελέγχου για νέα μέτρηση, της ανανέωσης του buffer και της αποστολής των διαβασμένων bytes χρησιμοποιούσε το κλείδωμα τύπου σημαφόρου του *lunix\_chrdev\_state\_struct* του αρχείου - αυτό απέτρεπε διεργασίες που είχαν ανοιχτό το ίδιο ακριβώς αρχείο με το ίδιο *lunix\_chrdev\_state\_struct* να δημιουργήσουν καταστάσεις συναγωνισμού κατά το διάβασμα. Λόγω όμως προβλημάτων που συνδέονταν με το να κρατάμε κλειδωμένο τον σημαφόρο ενώ παράλληλα η διεργασία περίμενε στο wait\_queue του αισθητήρα, αποφασίσαμε ότι δεν μπορούσαμε να έχουμε όλες τις λειτουργίες κάτω από ένα κλείδωμα. Οπότε, χρησιμοποιώντας τον ίδιο σημαφόρο, δημιουργήσαμε δύο κρίσιμες περιοχές στη συνάρτηση: η περιοχή όπου ανανεώνεται η πληροφορία του buffer, και η περιοχή όπου διαβάζονται οι χαρακτήρες του buffer και αντιγράφονται στον χώρο χρήστη. Μιας και χρησιμοποιήθηκε ο ίδιος σημαφόρος, κάθε στιγμή είτε μια διεργασία θα ανανεώνει τα δεδομένα του buffer, είτε κάποια άλλη διεργασία θα διαβάζει τα δεδομένα του buffer.

Όπως είναι δομημένος ο κώδικας, υπάρχει θεωρητικά η πιθανότητα, αν μια διεργασία δε διαβάσει όλα τα bytes του buffer μαζί, οπότε δε θα γίνει το *\*f\_pos* πάλι 0, η τιμή του buffer να ανανεωθεί μέχρι την επόμενη κλήση της read() από την ίδια διεργασία, οπότε οι εναπομείναντες χαρακτήρες που θα διαβαστούν να είναι λανθασμένοι, καθώς θα προέρχονται από άλλη μέτρηση. Στην περίπτωσή μας όμως, λόγω του ρυθμού ανανέωσης των μετρήσεων, αυτό είναι πρακτικά αδύνατο, και επιπλέον αφορά μόνο διεργασίες που έχουν ανοιχτό το ίδιο *struct file*.

Επίσης, λόγω της δομής του κώδικά μας, κάθε διεργασία που θα φτάσει πάλι σε *\*f\_pos == 0* θα ανανεώσει εκ νέου την τιμή του buffer όταν ξυπνήσει από το wait queue του αισθητήρα, κάτι που είναι περιττό αφού για κοινό *struct lunix\_chrdev\_state\_struct* αρκεί μία μόνο διεργασία να ανανεώσει τον buffer για κάθε νέα μέτρηση, αλλιώς εκτελούμε επαναλαμβανόμενα την ίδια δουλειά. Αυτό θα μπορούσε να αντιμετωπιστεί με έναν επιπλέον έλεγχο για το αν χρειάζεται ανανέωση των τιμών του buffer αμέσως πριν κληθεί η συνάρτηση ανανέωσης του buffer. Αν επιστρεφεί αρνητική απάντηση, αυτό θα σημαίνει ότι μια άλλη διεργασία ανανέωσε την τιμή του buffer πριν προλάβει η δική μας διεργασία, οπότε θα παραλειφθεί το βήμα αυτό.

**Σημαντική σημείωση:** *Όλα τα παραπάνω αρχεία μπορείτε να τα βρείτε, σε περίπτωση ανακρίβειας ή άλλης αμφιβολίας, εαν έρθετε σε επικοινωνία μαζί μας:*

* ***timos@email.com***
* ***orestarod@gmail.com***