ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εργαστήριο Λειτουργικών Συστημάτων: 2η Εργαστηριακή Άσκηση
Ομάδα 40

Ονοματεπώνυμο:	Τόφαλος Φίλιππος	Χρήστος Τσούφης
Αριθμός Μητρώου:	03117087	03117176
Ημερομηνία Επίδειξης:	03 – 12 – 2020	
Ημερομηνία Παράδοσης:	17 – 12 – 2020	

Εισαγωγή

Ζητούμενο της παρούσας εργαστηριακής άσκησης είναι η υλοποίηση ενός οδηγού συσκευής χαρακτήρων για το λειτουργικό σύστημα Linux, που αφορά την άντληση τιμών από δίκτυο αισθητήρων που επικοινωνούν με έναν σταθμό βάση, ο οποίος επικοινωνεί με τον υπολογιστή μας με σειριακή σύνδεση μέσω USB. Έχοντας διαθέσιμο τον κώδικα που αφορά την συλλογή δεδομένων από την σειριακή θύρα (lunix-ldisk και lunix-attach), την μετάφραση των ακατέργαστων δεδομένων βάσει του πρωτοκόλλου που ακολουθεί το σύστημα των αισθητήρων (lunix-protocol) και την προσωρινή αποθήκευση των δεδομένων ανά αισθητήρα (lunix-sensors), καλούμαστε να συμπληρώσουμε τον κώδικα που σχετίζεται με την συσκευή χαρακτήρων μέσω της οποίας θα λαμβάνει ο χρήστης τα δεδομένα του δικτύου μορφοποιημένα, ανά αισθητήρα και μέτρηση (lunix-chrdev).
Για τον σκοπό ανάπτυξης του οδηγού και προσομοίωσης ενός υπολογιστή που είναι άμεσα συνδεδεμένος με την συσκευή υπό εξέταση, αξιοποιούμε το λογισμικό εικονικοποίησης QEMU-KVM σε συνδυασμό με το script utopia.sh που διατίθεται για την παρούσα άσκηση, και με το οποίο επιτυγχάνουμε την προώθηση, στην σειριακή θύρα S0 της εικονικής μηχανής, των μετρήσεων που διαθέτει ένας εξυπηρετητής TCP/IP ο οποίος βρίσκεται στο εργαστήριο και σε σύνδεση με το δίκτυο αισθητήρων που μας ενδιαφέρει.

Εγκατάσταση του Οδηγού

Εφόσον εγκατασταθεί επιτυχώς το QEMU-KVM, ρυθμίσουμε και τοποθετήσουμε τα αρχεία του utopia στον κατάλογο της επιλογής μας (εν προκειμένω ~/utopia), και τοποθετήσουμε τα αρχεία του οδηγού στον επιθυμητό κατάλογο /path/to/lunix/soruce/code/ εντός της εικονικής μηχανής, ακολουθούμε κάθε επόμενη φορά τα ακόλουθα βήματα για την σύνδεση στην εικονική μηχανή και την εγκατάσταση του οδηγού.

```
[GUEST]   cd ~/utopia
[GUEST]   ./utopia.sh
[GUEST]   ssh -p 22223 root@localhost # Από δεύτερο τερματικό
[HOST]    cd /path/to/lunix/soruce/code/
[HOST]    make
[HOST]    insmod lunix.ko
[HOST]    ./lunix_dev_nodes.sh
[HOST]    ./lunix-attach /dev/ttyS0
```

Σε αυτήν την υποενότητα είναι καίριο να σημειωθεί ότι ο οδηγός μας εκτελείται αμέσως μετά την εντολή insmod. Ωστόσο, είναι αναγκαίο να εγκαταστήσουμε το line discipline μέσω της εντολής ./lunix-attach /dev/ttyS0 ώστε να δεχτούμε μετρήσεις τις οποίες μπορεί να επεξεργαστεί ο οδηγός.

Διαδικασία Συγγραφής της Λύσης

Συνάρτηση lunix_chrdev_init

Ξεκινάμε την συμπλήρωση του αρχείου lunix-chrdev.c από την συνάρτηση lunix_chrdev_init(), με την οποία γίνεται η δέσμευση της ζητούμενης περιοχής αριθμών για την συσκευή μας και αρχικοποιείται η συσκευή χαρακτήρων. Συγκεκριμένα, αξιοποιούμε τον πειραματικό αριθμό 60 ως major, και επειδή χρησιμοποιούμε τα 3 τελευταία bits κάθε minor number για να προσδιορίσουμε την μέτρηση που αντιστοιχεί σε κάθε έναν από τους 16 αισθητήρες, εν τέλει αποφασίζουμε να δεσμεύσουμε μια περιοχή $16 << 3 = 16 \cdot 2^3$ αριθμών, όπως προσδιορίζεται από την τιμή lunix_minor_cnt. Πρώτη συμπλήρωση λοιπόν που κάνουμε είναι η κλήση της register_chrdev_region με την οποία δεσμεύουμε το επιθυμητό πλήθος αριθμών, ξεκινώντας από αυτόν που ορίζει το αντικείμενο dev_no, για όνομα driver αυθαίρετα επιλεγμένο ως lunix. Ύστερα, αρκεί να εκχωρήσουμε αυτούς τους αριθμούς σε μια καινούργια συσκευή χαρακτήρων την οποία κατασκευάζουμε μέσω του αντικειμένου lunix_chrdev_cdev που έχει οριστεί globally στην αρχή του αρχείου, με την κλήση της cdev_add, όπως φαίνεται στην αμέσως επόμενη και τελευταία προσθήκη για αυτό το αρχείο.
Η συνάρτηση που αναλύθηκε καλείται με την εκτέλεση της εντολής insmod lunix.ko κατά την εγκατάσταση του οδηγού (μέσω της κλήσης της συνάρτησης __init lunix_module_init στο αρχείο lunix-module.c) και η εκτέλεση της γίνεται σε process context δεδομένου της χρήσης της εντολής insmod από τον ίδιο τον χρήστη. Επίσης, οι κόμβοι του συστήματος αρχείων που αντιστοιχούν στην περιοχή αριθμών που δεσμεύσαμε παραπάνω, δημιουργούνται με την εκτέλεση του lunix_dev_nodes.sh, με ονόματα που περιγράφτηκε στην εκφώνηση της εργασίας. Αξίζει να σημειωθεί ότι η επιλογή αυτών των ονομάτων δεν επηρεάζει τον οδηγό, γιατί αυτός αρκείται στους major και minor numbers που του δίνουμε. Τα ονόματα στον κατάλογο /dev/, όπως αυτά προκύπτουν από το lunix_dev_nodes.sh, αφορούν μόνο το filesystem και το πως θα έχουμε πρόσβαση στην συσκευή μέσω κάποιου προγράμματος στο userspace.

Σημείωση: Οι προσθήκες και αλλαγές στον κώδικα – σκελετό που μας έχει δοθεί, σημειώθηκαν με το σχόλιο `/* ! */`.

```
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;
        /* For every sensor we want at least 3 minor numbers,
           the measurement info is contained at the 3 LSB
           of the minor number, and the region must be
           consecutive, so we end up with 16 << 3 minor numbers */
        unsigned int lunix_minor_cnt = lunix_sensor_cnt << 3;

        debug("initializing character device\n");

        /* We initialize the global cdev structure, specifying
           the file operations right above to be used */
        cdev_init(&lunix_chrdev_cdev, &lunix_chrdev_fops);
```

```
        lunix_chrdev_cdev.owner = THIS_MODULE;

        /* Produce a Device ID for the pair (Major = 60, Minor = 0) */
        dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0);

        /* ! */
        /* register_chrdev_region? */
        /* We register the wanted range, starting from (Major = 60, Minor = 0), up
           to (Major = 60, Minor = 16 << 3) */
        ret = register_chrdev_region(dev_no, lunix_minor_cnt, "lunix");

        if (ret < 0) {
                debug("failed to register region, ret = %d\n", ret);
                goto out;
        }

        /* ! */
        /* cdev_add? */
        /* After the above registration, we are ready to add the
           char device for the corresponding cdev structure and
           the defined range */
        ret = cdev_add(&lunix_chrdev_cdev, dev_no, lunix_minor_cnt);

        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;
}
```

Συνάρτηση lunix_chrdev_state_needs_refresh

Στην συνέχεια προχωράμε με την συμπλήρωση της συνάρτησης που ελέγχει (χωρίς την χρήση κλειδώματος, και όσο πιο σύντομα γίνεται λόγω αυτού) για το αν είναι απαραίτητο να ανανεωθεί ο buffer ενός αντικειμένου τύπου lunix_chrdev_state_struct. Αυτό πραγματοποιείται εύκολα συγκρίνοντας τα αποθηκευμένα timestamps του προαναφερθέντος αντικειμένου και του αντικειμένου τύπου lunix_sensor_struct που αντιστοιχεί σε αυτό, όπως φαίνεται στην συμπλήρωση που έχουμε πραγματοποιήσει παρακάτω. Σημειώνεται ότι η ανανέωση του πρώτου timestamp γίνεται εντός της συνάρτησης lunix_chrdev_state_update. Τελικά επιστρέφουμε το αποτέλεσμα αυτής της σύγκρισης προκειμένου να αξιοποιηθεί από τις lunix_chrdev_state_update και lunix_chrdev_open όπως θα φανεί στην συνέχεια.

```
static int lunix_chrdev_state_needs_refresh(struct lunix_chrdev_state_struct *state)
{
        /* ! */
        /* We declare the (boolean) return value */
        int ret;
        struct lunix_sensor_struct *sensor;

        WARN_ON ( !(sensor = state->sensor) );
        /* ! */
        /* We simply compare the sensor timestamp with the most recent one stored
           in the measurement state struct */
```

```
        ret = (state->buf_timestamp != sensor->msr_data[state->type]->last_update);

        /* The following return is bogus, just for the stub to compile */
        /* The return is not bogus anymore, but equal to the result of the
           above comparison */
        return ret; /* ! */
}
```

Συνάρτηση lunix_chrdev_state_update (και η βοηθητική συνάρτηση format_value)

Μένει να συμπληρώσουμε την βοηθητική συνάρτηση lunix_chrdev_state_update προτού συμπληρώσουμε τις συναρτήσεις της δομής file_operations. Το πρώτο πράγμα που επιχειρούμε εντός της συνάρτησης είναι να ελέγξουμε αν υπάρχουν νέα δεδομένα στον αισθητήρα που αντιστοιχεί στο state που εξετάζουμε. Για αυτόν τον σκοπό όμως είναι απαραίτητο να διεκδικήσουμε το spinlock του αντίστοιχου sensor, για να λάβουμε αποκλειστική πρόσβαση στα δεδομένα αυτού, δηλαδή να επιτύχουμε τον αποκλεισμό των υπόλοιπων διεργασιών όσο διαρκεί ο έλεγχος για την ύπαρξη και ενδεχομένως την ανάγνωση νέων δεδομένων, και ιδανικά να αποφύγουμε την ανανέωση των δεδομένων του αισθητήρα μέσω κάποιας διακοπής, όπως πράγματι κάνουμε με την χρήση της spin_lock_irqsave(). Εφόσον λοιπόν ληφθεί το κλείδωμα, τότε εξετάζουμε το αποτέλεσμα της συνάρτησης lunix_chrdev_state_needs_refresh, και:
1) Αν όντως υπάρχει νέα μέτρηση (η οποία θα είναι αποθηκευμένα στην θέση sensor→msr_data[state→type]→values, δηλαδή για τον sensor του αντικειμένου state, και για την μέτρηση state→type που αντιστοιχεί σε αυτό), τότε κάνουμε ανάθεση σε μια προσωρινή μεταβλητή και ανανεώνουμε το timestamp του state
2) Διαφορετικά, αν δεν υπάρχουν νέα δεδομένα τότε τελειώνει η συνάρτηση υπό εξέταση επιστρέφοντας την τιμή -EAGAIN, ώστε να σημανθεί στην lunix_chrdev_read ότι δεν υπάρχουν ακόμα νέα δεδομένα και πρέπει να μεταχειριστεί την διεργασία αναλόγως.

Η χρήση spinlock για την δομή lunix_sensor_struct οφείλεται στους εξής παράγοντες:
1) Το αντίστοιχο API προσφέρει συναρτήσεις όπως την spin_lock_irqsave, μέσω των οποίων μπορούμε προσωρινά να απενεργοποιήσουμε τις διακοπές μέχρι να απελευθερώσουμε το κλείδωμα. Αυτό είναι απαραίτητο γιατί το κλείδωμα μπορεί να διεκδικηθεί και σε interrupt context (αν ληφθούν δεδομένα από τον σταθμό βάσης, θα ειδοποιηθούμε με διακοπή, και θα κληθούν οι ακόλουθες συναρτήσεις με την σειρά που παρουσιάζεται lunix_ldisc_receive() → lunix_protocol_received_buf() → lunix_protocol_update_sensors() → lunix_sensor_update(), με την τελευταία να διεκδικεί το κλείδωμα), οπότε αν έχουμε το κλείδωμα και προκύψει interrupt, η διαδικασία εξυπηρέτησης της διακοπής δεν θα μπορέσει να τελειώσει.
2) Δεν μπορούμε να χρησιμοποιήσουμε σημαφόρους, γιατί μέσω των σημαφόρων μια διεργασία μπορεί να κοιμηθεί, το οποίο δεν είναι δυνατό σε interrupt context στο οποίο η εξυπηρέτηση πρέπει να γίνει αδιάκοπα. Μέσω των spins δεν σταματάει η εκτέλεση και επιτυγχάνεται αυτό που θέλουμε.

Επίσης, είναι απαραίτητο να κρατήσουμε το spinlock όσο λιγότερο γίνεται, καθώς υπάρχει ενεργή απασχόληση του επεξεργαστή για τις διεργασίες που αναμένουν το προαναφερθέν κλείδωμα (οι οποίες δεν κοιμούνται αλλά περιμένουν την αποδέσμευση του), και αν δεν υπάρξει μέριμνα για να κρατηθεί το λιγότερο δυνατόν, αναιρείται όλη η προσπάθεια μείωσης του kernel latency από το ίδιο το λειτουργικό. Στην συνέχεια παρουσιάζεται η υλοποίηση της συνάρτησης υπό εξέταση:

```
static int lunix_chrdev_state_update(struct lunix_chrdev_state_struct *state)
{
        struct lunix_sensor_struct *sensor;
        /* ! */
        uint16_t value;
        long int temp = 0;
        unsigned long flags;

        debug("leaving\n");

        /*
         * Grab the raw data quickly, hold the
         * spinlock for as little as possible.
         */

        /* ! */
        WARN_ON ( !(sensor = state->sensor) );
        /* Acquire the sensor lock */
        /* NOTE: We use spin_lock_irqsave() beacuse it is possible that new data
         * may be sent from the same sensor, causing an interrupt and putting this
         * function on hold while we call the sequence: lunix_ldisc_receive() ->
         * lunix_protocol_received_buf() -> lunix_protocol_update_sensors() ->
         * lunix_sensor_update() and thus trying to lock again the sensor
         * spinlock. */
        spin_lock_irqsave(&sensor->lock, flags);

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

        /*
         * Any new data available?
         */

        /* ! */
        /* We examine the result of lunix_chrdev_state_needs_refresh() */
        if (lunix_chrdev_state_needs_refresh(state)) {
                /* Grab the value for the specific sensor */
                value = sensor->msr_data[state->type]->values[0];
                /* Update the timestamp (while locked) */
                state->buf_timestamp = sensor->msr_data[state->type]->last_update;
        } else {
                /* We must not forget to unlock the spinlock */
                spin_unlock_irqrestore(&sensor->lock, flags);
                /* We return the value indicated by lunix_chrdev_read()
                 * EAGAIN = "there is no data available right now, try
                 * again later" */
                return -EAGAIN;
        }

        /* Restore the sensor lock */
        spin_unlock_irqrestore(&sensor->lock, flags);

        /*
         * Now we can take our time to format them,
         * holding only the private state semaphore
         */

        /* ! */
        /* Look at the lookup table in lunix-lookup.h, for the
         * measurement specified by state->type */

        if (state->switch_raw) {
                temp = value;
                goto out;
        }

        switch (state->type) {
                case BATT  : temp = lookup_voltage[value];      break;
                case TEMP  : temp = lookup_temperature[value];  break;
```

```
                case LIGHT : temp = lookup_light[value];        break;
                case N_LUNIX_MSR : /*This case is unlikely to occur */ ;
     /* default    : [IT MAY BE BETTER TO RETURN WITH AN ERROR] */
        }

out:    /* Format the value acquired from the lookup table. If it is a LIGHT
         * value, no decimal point is required */
        format_value(state, temp, state->type == LIGHT || state->switch_raw ? 0 :
DEFAULT_DOT_POS);
        debug("leaving\n");
        return 0;
}
```

Έχοντας την νέα τιμή της μέτρησης, λαμβάνουμε την μορφοποιημένη τιμή αυτής μέσω του αντίστοιχου lookup table στην επικεφαλίδα lunix-lookup.h, αναλόγως με τον τύπο της μέτρησης του αντίστοιχου state, και στην συνέχεια την μορφοποιούμε περαιτέρω μέσω της συνάρτησης format_value που προστέθηκε με σκοπό την καλύτερη αναγνωσιμότητα του κώδικα που συμπληρώσαμε, και την υλοποίηση της οποίας παρουσιάζουμε στην συνέχεια. Η λειτουργία της στην ουσία είναι η αντιγραφή της μορφοποιημένης τιμής ως πίνακα χαρακτήρων στον buffer του state, η προσθήκη της υποδιαστολής σε περίπτωση που χρειάζεται, καθώς επίσης και ενός χαρακτήρα αλλαγής γραμμής στο τέλος της συμβολοσειράς. Όλα τα προαναφερθέντα πραγματοποιούνται με ασφάλεια, εφόσον η εκτέλεση της lunix_chrdev_state_update γίνεται μέσω της lunix_chrdev_read, έχοντας πρώτα κλειδώσει τον σημαφόρο για το ανάλογο state.

```
static void format_value (
        /* The state struct to update */ struct lunix_chrdev_state_struct *state,
        /* Value to convert to string and format */ long int value,
        /* Dot position, starting from the most right digit */  int dot
        ) {
        int i, s;
        /* Copy the value from the lookup table into the buffer */
        s = snprintf(state->buf_data, LUNIX_CHRDEV_BUFSZ, "%ld", value);
        /* Position the decimal point if needed */
        if (dot != 0) {
                for (i = 0; i < dot; i++)
                        state->buf_data[s - i] = state->buf_data[s - i - 1];
                state->buf_data[s - i] = '.';
        }
        /* Add the new line character at the end */
        state->buf_data[s + (dot != 0)] = '\n';
        /* Update the buffer limit to be ready for copy_to_user() */
        state->buf_lim = s + 1 + (dot != 0);
}
```

Συνάρτηση lunix_chrdev_open

Στην συνέχεια εξετάζουμε τις υλοποιήσεις των συναρτήσεων που σχετίζονται με την δομή file_operations, ξεκινώντας από αυτήν που εκτελείται πρώτη κατά την λειτουργία του οδηγού, δηλαδή την lunix_chrdev_open. Στην open βασικός μας σκοπός είναι η αρχικοποίηση ενός αντικειμένου lunix_chrdev_state_struct με κατάλληλο τρόπο ώστε αυτό να ανατεθεί στο πεδίο private_data του αντίστοιχου file pointer και να μπορεί να χρησιμοποιηθεί στις υπόλοιπες συναρτήσεις της δομής file_operations. Για την δέσμευση του χώρου που χρειάζεται το προαναφερθέν αντικείμενο αξιοποιούμε την kmalloc, και για την αρχικοποίηση του βασιζόμαστε στην σημασιολογία του κάθε πεδίου και τις προδιαγραφές του οδηγού. Συγκεκριμένα:

1) Το πεδίο type που δείχνει τον τύπο της μέτρησης θα είναι ίσο με τα τρία τελευταία bits του minor number του αρχείου που προσπαθούμε να ανοίξουμε.

2) Το πεδίο sensor τίθεται ίσο με την διεύθυνση του στοιχείου του πίνακα lunix_sensors (ο οποίος έχει αρχικοποιηθεί στο αρχείο lunix-module.c) που αναλογεί στον αισθητήρα του αρχείου που εξετάζουμε. Ο ζητούμενος δείκτης προκύπτει από τον minor number αγνοώντας τα τρία πρώτα bit που δείχνουν την μέτρησης.
3) Το πεδίο buf_lim στην πραγματικότητα δεν χρειάζεται αρχικοποίηση, καθώς η τιμή της θα τεθεί την πρώτη φορά όταν κληθεί η format_value. Ωστόσο αρχικοποιείται στην υλοποίηση μας για πληρότητα της παρουσίασης.
4) Το πεδίο lock πρέπει να αρχικοποιηθεί μέσω της sema_init στην τιμή 1, εφόσον θέλουμε αμοιβαίο αποκλεισμό μεταξύ διεργασιών.
5) Το πεδίο buf_timestamp είναι σημαντικό να αρχικοποιηθεί στην τιμή 0 καθώς η σελίδα που αντιστοιχεί στα δεδομένα του ανάλογου αισθητήρα είναι αρχικοποιημένη ως μηδενική. Έτσι, αν αυτός ο αισθητήρας είναι ανενεργός και δεν έχει ενημερωθεί η αντίστοιχη σελίδα, θέλουμε ο έλεγχος lunix_chrdev_state_needs_refresh να αποτύχει και να μην εμφανιστεί κάτι αν προσπαθήσουμε να διαβάσουμε το αρχείο που αντιστοιχεί στο υπό εξέταση state.

```
static int lunix_chrdev_open(struct inode *inode, struct file *filp)
{
        /* Declarations */
        /* ! */
        struct lunix_chrdev_state_struct *state;
        int ret;

        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>]
         */

        /* Allocate a new Lunix character device private state structure */
        /* ! */
        /* We perform the allocation using kmalloc */
        state = kmalloc(sizeof(struct lunix_chrdev_state_struct),
                        GFP_KERNEL);
        if (!state) {
                /* If the allocation failed, return the suitable
                 * errno value */
                ret = -ENOMEM;
                goto out;
        }

        /* TYPE: We acquire the type of measurement from the
         * last 3 bits of the minor number */
        state->type = iminor(inode) & 0b111;
        /* SENSOR: Associate the file with the corresponding
         * sensor struct defined in lunix.h */
        state->sensor = &lunix_sensors[iminor(inode) >> 3];
        /* BUF_LIM: Initially no measurement is cahced.
         * This is not really necessary */
        state->buf_lim = 0;
        /* LOCK: Initialize the semaphore with value equal
         * to 1. This is the same as init_MUTEX */
        sema_init(&state->lock, 1);
        /* BUF_TIMESTAMP: It is important to be initialized
         * to 0, because the sensor MSR DATA is initialized
         * as a zeroed page */
        state->buf_timestamp = 0;
        state->switch_raw = 0;

        /* Assign the state structure to the private_data field
```

```
         * of the given file pointer */
        filp->private_data = state;
out:
        debug("leaving, with ret = %d\n", ret);
        return ret;
}
```

Έχοντας δει και την προηγούμενη υλοποίηση, είναι σημαντικό να εξετάσουμε την συμπεριφορά των spinlocks των αισθητήρων. Συγκεκριμένα, τα spinlock για τον κάθε sensor είναι κοινά για όλες τις διεργασίες, επειδή η κατασκευή του πίνακα lunix_sensors γίνεται εντός του αρχείου lunix-module.c, οπότε κατασκευάζεται στον χώρο διευθύνσεων του πυρήνα, και κάθε state struct συμβουλεύεται αυτόν τον χώρο διευθύνσεων που δεν αλλάζει από διεργασία σε διεργασία. Θα αντιπαραβάλουμε αυτήν την συμπεριφορά με εκείνην των σημαφόρων των αντικειμένων lunix_chrdev_state_struct, όταν προβούμε στην ανάλυση της συνάρτησης lunix_chrdev_read.

Συνάρτηση lunix_chrdev_release

Η υλοποίηση της συνάρτησης release είναι αρκετά απλή, εφόσον το μόνο για το οποίο πρέπει να μεριμνήσουμε είναι η απελευθέρωση του χώρου που κατέλαβε το αντικείμενο τύπου lunix_chrdev_state_struct που προσαρτήσαμε στο πεδίο private_data του δείκτη του αρχείου.

```
static int lunix_chrdev_release(struct inode *inode, struct file *filp)
{
        /* ! */
        /* Deallocate the memory used for the private_data structure */
        kfree(filp->private_data);
        return 0;
}
```

Συνάρτηση lunix_chrdev_read

Έχοντας υλοποιήσει την συνάρτηση lunix_chrdev_state_update είμαστε σε θέση να συμπληρώσουμε την συνάρτηση lunix_chrdev_read, που αποτελεί και την ουσία της υλοποίησης της συσκευής χαρακτήρων. Εφόσον έχουν γίνει οι δοθείσες αρχικοποιήσεις για τις μεταβλητές sensor και state, θέλουμε αμέσως μετά να διεκδικήσουμε τον σημαφόρο του state. Έχοντας εξασφαλίσει αυτό, εξετάζουμε την τιμή του f_pos. Αν η τιμή του είναι ίση με 0, που σημαίνει ότι αναμένεται η ανάγνωση μιας καινούργιας μέτρησης, εξετάζουμε αν όντως υπάρχει μια τέτοια μέσω της lunix_chrdev_state_update. Αν δεν υπάρχει (δηλαδή η προαναφερθείσα συνάρτηση επιστρέφει –EAGAIN), τότε απελευθερώνουμε τον σημαφόρο και προσθέτουμε την τρέχουσα διεργασία στην ουρά αναμονής του αντίστοιχου αισθητήρα (sensor→wq) μέσω της wait_event_interruptible και με συνθήκη την lunix_chrdev_state_needs_refresh(state) == 1. Σημειώνεται ότι χρησιμοποιείται η συνάρτηση wait_event_interruptible γιατί είναι επιθυμητό να μπορούμε να τερματίσουμε μια διεργασία με την αποστολή ανάλογου σήματος (Ctrl+C από το τερματικό) ακόμα και αν αυτή βρίσκεται σε κατάσταση αναμονής για νέα μέτρηση. Όλες οι διεργασίες στην προαναφερθείσα ουρά αναμονής ξυπνάνε όταν έρθει μια νέα μέτρηση (οπότε ελέγχεται και η προηγούμενη συνθήκη η οποία υπό κανονικές συνθήκες επιτυγχάνει) και αφότου εκτελεστεί εντός της lunix_sensor_update (που ορίζεται στο αρχείο lunix-sensors.c) η εντολή wake_up_interruptible(&s→wq), ανταγωνίζονται για τον σημαφόρο (όπως φαίνεται στις εντολές κάτω από το σχόλιο /* Re-acquire the lock before continuing */), τον οποίον λαμβάνει μια διεργασία, και οι υπόλοιπες επιστρέφουν στην ουρά αναμονής.

Είναι σημαντικό εδώ να γίνει αναφορά στην ανάγκη χρήσης των σημαφόρων, έναντι κάποιας άλλης δομής κλειδώματος όπως πχ τα spinlocks. Συγκεκριμένα, οι σημαφόροι εξυπηρετούν στην περίπτωση του state γιατί είναι επιθυμητό οι διεργασίες να κοιμούνται σε περίπτωση που βρίσκουν το critical path κατειλημμένο. Ο χρόνος μορφοποίησης των τιμών και αποθήκευσης στον buffer είναι σημαντικός για να υπάρξει ενεργή αναμονή των διεργασιών. Όσον αφορά το υπό ποιες συνθήκες ο ίδιος σημαφόρος διεκδικείται από πολλές διεργσαίες, παρατηρούμε ότι ο σημαφόρος lock ανήκει στο state struct, το οποίο αρχικοποιείται με την open. Η open πρέπει να είναι κοινή για τις διεργασίες ώστε να υπάρξει ανταγωνισμός, δηλαδή μόνο σε περίπτωση πατέρα-παιδιών μέσω fork μπορούμε να έχουμε ανταγωνισμό για τον ίδιο σημαφόρο. Αν δηλαδή έχουμε δύο ξεχωριστές διεργασίες οι οποίες έχουν κάνει η κάθε μια το δικό της open, πρόκειται για διαφορετικούς σημαφόρους, οπότε δεν υπάρχει ανταγωνισμός σε αυτό το επίπεδο.
Αν τώρα η διεργασία ξυπνήσει λόγω νέας μέτρησης και λάβει τον σημαφόρο, ή αν ακόμα το f_pos είχε μη μηδενική τιμή, τότε ελέγχουμε αρχικά αν η τιμή f_pos έχει ξεπεράσει για τον οποιοδήποτε λόγο το όριο που έχουμε θέσει στην format_value μέσω του πεδίου buf_lim του state, περίπτωση στην οποία το επαναφέρουμε σε 0 για να είναι έτοιμο για την επόμενη ανάγνωση. Ύστερα προσαρμόζουμε την τιμή cnt (που αντιστοιχεί στο σε πόσα bytes θα αντιγραφτούν στον χρήστη), ώστε να είναι εντός των ορίων που έχουν τεθεί με το buf_lim, σε περίπτωση που ζητούνται περισσότερα bytes από αυτά που έχουμε διαθέσιμα προς αντιγραφή, και τελικά επιστρέφουμε στον buffer του χρήστη (usrbuf) το κομμάτι του state->buf_data που εν τέλει προσδιορίστηκε μέσω του cnt και της f_pos. Είναι σημαντικό να χρησιμοποιηθεί η copy_to_user και να μην επιχειρήσουμε για παράδειγμα την απευθείας επεξεργασία του usrbuf, καθώς η copy_to_user προσφέρει ασφάλεια κατά την αντιγραφή δεδομένων προς τον χρήστη, γιατί δεν είναι πάντα σίγουρο ότι ο χρήστης θα δώσει μια αποδεχτή διεύθυνση ή ότι γενικότερα δεν θα επιχειρήσει να δώσει τέτοιες παραμέτρους με τις οποίες θα προκύψουν μη επιθυμητά αποτελέσματα και θα οδηγήσουν σε προβλήματα ασφαλείας. Στο τέλος της συνάρτησης φροντίζουμε να ξεκλειδώσουμε τον σημαφόρο και να επιστρέψουμε την τιμή των bytes που αντιγράφτηκαν (μεταβλητή ret) όπως αυτή διαμορφώθηκε στην πορεία της εκτέλεσης.

```
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;

        state = filp->private_data;
        WARN_ON(!state);

        sensor = state->sensor;
        WARN_ON(!sensor);

        /* ! */
        /* Lock? */
        /* We try to acquire the semaphore (used in case many processes
         * with father-child relationship try to read the same sensor
         * measurement) */
        if (down_interruptible(&state->lock))
                return -ERESTARTSYS;

        /*
         * 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_update(state) == -EAGAIN) {
                        /* ! */
```

```
                        /* "Release" the semaphore before sleeping */
                        up(&state->lock);
                        /* If non-blocking is requested by the process,
                         * we return a -EAGAIN errno value and avoid
                         * putting the process to sleep */
                        if (filp->f_flags & O_NONBLOCK)
                                return -EAGAIN;

                        /* Add the process to the waiting queue while no
                         * new data is available */
                        if (wait_event_interruptible(sensor->wq,
                            lunix_chrdev_state_needs_refresh(state)))
                                return -ERESTARTSYS;

                        /* Re-acquire the lock before continuing */
                        if (down_interruptible(&state->lock))
                                return -ERESTARTSYS;
                        /* The process needs to sleep */
                        /* See LDD3, page 153 for a hint */
                }
        }
        /* End of file */
        /* ! */
        /* We initially assume that zero bytes are going to be read */
        ret = 0;
        /* Unlikely: If the f_pos value exceeds the buf_lim (what we
         * actually have to copy), reset it and return */
        if (*f_pos > state->buf_lim) {
                *f_pos = 0;
                goto out;
        }

        /* Determine the number of cached bytes to copy to userspace */
        /* ! */
        /* If more bytes than those available are requested, redefine
         * the cnt value */
        if (*f_pos + cnt > state->buf_lim)
                cnt = state->buf_lim - *f_pos;

        /* cnt indicates how many bytes will be transfered to the user */
        ret = cnt;

        if (copy_to_user(usrbuf, state->buf_data + *f_pos, cnt)) {
                /* Indicate that a bad address was given */
                ret = -EFAULT;
                goto out;
        }

        /* Increase the f_pos by cnt for it to be ready at the next call */
        *f_pos += cnt;

        /* Auto-rewind on EOF mode? */
        /* ! */
        /* If we reached at the end of the buffer, reset f_pos */
        if (*f_pos == state->buf_lim)
                *f_pos = 0;
out:
        /* ! */
        /* Unlock? */
        /* Release the seamphore to be used by the next process
         * probably waiting */
        up(&state->lock);
        return ret;
}
```

Δοκιμές σωστής λειτουργίας

Πρώτη Δοκιμή

Η πρώτη βασική δοκιμή είναι να ελέγξουμε την συμπεριφορά του οδηγού μας διαβάζοντας κάποιο αρχείο του μέσω της εντολής cat. Από την ανάγνωση των αρχείων των πρώτων 2 αισθητήρων και την επιβεβαίωση της αναμενόμενης συμπεριφοράς για τους ανενεργούς αισθητήρες (μη εμφάνιση κάποιας τιμής – μόνιμη αναμονή), συμπεραίνουμε ότι ο οδηγός λειτουργεί με τον επιθυμητό τρόπο και σύμφωνα με τις προδιαγραφές της εκφώνησης. Επίσης χρησιμοποιήθηκε και η εντολή dd για να δοκιμάσουμε να διαβάσουμε λιγότερα bytes από αυτά που αντιστοιχούν σε μια μέτρηση, λαμβάνοντας επιτυχώς πρόθεμα της πιο πρόσφατης τιμής.

Δεύτερη Δοκιμή

Στην δοκιμή αυτή που πραγματοποιείται με το πρόγραμμα που παρουσιάζεται στην συνέχεια, ελέγχουμε την σωστή διαχείριση της παραμέτρου f_pos, διαβάζοντας μέγεθος μικρότερο από αυτό ολόκληρης της τιμής (συγκεκριμένα διαβάζοντας 4 ή 3 bytes).

```c
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

#include <sys/ioctl.h>
#define SIZE 100

int main (int argc, char **argv) {
        int fd, sz, i, p;
        char *c = (char *) calloc(SIZE, sizeof(char));

        fd = open(argv[1], O_RDONLY);

        if (fd < 0) {
                perror("OPEN");
                exit(1);
        }

        i = 0;
        while(1) {
                i++;
                sz = read(fd,c, 3 + i % 2);
                c[sz] = '\0';
                printf("READ VALUE [%d BYTES]:: %s \n", sz, c);
        }
}
```

Για παράδειγμα, με εκτέλεση της εντολής `./test /dev/lunix0-temp` (όπου test θεωρούμε το εκτελέσιμο που παράγεται από τον κώδικα της κάθε δοκιμής) λαμβάνουμε την ακόλουθη έξοδο:

```
READ VALUE [4 BYTES]:: 25.3
READ VALUE [3 BYTES]:: 56

READ VALUE [4 BYTES]:: 25.3
READ VALUE [3 BYTES]:: 56

READ VALUE [4 BYTES]:: 25.2
```

```
READ VALUE [3 BYTES]:: 59
...
```

η οποία επαληθεύει το ζητούμενο εφόσον παρατηρούμε ότι η ανάγνωση της τιμής στην επόμενη επανάληψη συνεχίζει από το σημείο που σταματήσαμε στην προηγούμενη, και αν είχε ολοκληρωθεί λαμβάνουμε καινούργια τιμή.

Τρίτη Δοκιμή

Σε αυτήν την δοκιμή κάνουμε μια fork() προκειμένου να εξετάσουμε την σωστή λειτουργία των σημαφόρων για την απλή περίπτωση ενός ζεύγους πατέρα – παιδιού, που όμως συναγωνίζονται διαρκώς για την ανάγνωση της τιμής.

```c
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

#define SIZE 100

int main (int argc, char** argv) {
        int fd, sz, i;
        pid_t pid;
        char *c = (char *) calloc(SIZE, sizeof(char));

        fd = open(argv[1], O_RDONLY);
        if (fd < 0) {
                perror("OPEN");
                exit(1);
        }

        pid = fork();

        while(1) {
                sz = read(fd,c, 1000);
                c[sz] = '\0';
                printf("[%s] [%d BYTES] READ :: %s", pid? "PARENT":"CHILD", sz, c);
        }
}
```

Πράγματι, από εκτέλεση της εντολής `./test /dev/lunix1-batt` λαμβάνουμε μια έξοδο της μορφής:

```
[PARENT] [6 BYTES] READ :: 3.292
[CHILD] [6 BYTES] READ :: 3.301
[CHILD] [6 BYTES] READ :: 3.301
[CHILD] [6 BYTES] READ :: 3.301
[PARENT] [6 BYTES] READ :: 3.301
[CHILD] [6 BYTES] READ :: 3.301
[PARENT] [6 BYTES] READ :: 3.301
[CHILD] [6 BYTES] READ :: 3.301
[PARENT] [6 BYTES] READ :: 3.301
[CHILD] [6 BYTES] READ :: 3.301
...
```

από την οποία παρατηρούμε η ανάγνωση του ίδιου state struct από δύο διαφορετικές διεργασίες γίνεται επιτυχώς, οπότε εκτιμάται ότι οι σημαφόροι λειτουργούν με τον επιθυμητό τρόπο για αυτήν την περίπτωση.

Υλοποίηση των Επεκτάσεων

Υποστήριξη κλήσεων ioctl() για την μεταβολή της συμπεριφοράς του οδηγού

Για την υποστήριξη αυτής της λειτουργίας προστέθηκε στον ορισμό του lunix_chrdev_state_struct το πεδίο switch_raw τύπου int, το οποίο αξιοποιούμε ως τιμή αληθείας. Επίσης, εκτός του προαναφερθέντος, στο αρχείο lunix-chrdev.h προσθέτουμε κάτω από τον ορισμό του LUNIX_IOC_MAGIC την ακόλουθη γραμμή:

```
#define LUNIX_IOC_SWITCH                _IO(LUNIX_IOC_MAGIC, 0)
```

Στις αρχικοποιήσεις της lunix_chrdev_open αυτό το πεδίο τέθηκε ίσο με 0, δηλαδή θεωρούμε ότι εξ ορισμού οι τιμές επιδέχονται επεξεργασία πριν διαβαστούν. Στην συνάρτηση lunix_chrdev_state_update ελέγχουμε την τιμή αυτού του πεδίου, και αν είναι διάφορη του μηδενός (δηλαδή δεν είναι επιθυμητή η μορφοποίηση των τιμών), τότε παραλείπουμε την αντιστοίχηση μέσω των lookup tables και καλούμε την format_value χωρίς την προσθήκη υποδιαστολής. Όσον αφορά την ίδια την κλήση ioctl, η υλοποίηση είναι αυτή που φαίνεται στην συνέχεια, στην οποία θεωρούμε ότι μόνο ο διαχειριστής του λειτουργικού μπορεί να κάνει την αλλαγή της επιλογής μορφοποίησης ή μη (το οποίο αποτελεί απλά μια παραδοχή που έγινε για περαιτέρω ανάπτυξη της υλοποίησης). Σε περίπτωση που ο χρήστης έχει τα ανάλογα δικαιώματα, τότε διεκδικούμε τον σημαφόρο για να θέσουμε την τιμή της switch_raw στην συμπληρωματική αυτής, και να αλλάξουμε κατάσταση.

```
static long lunix_chrdev_ioctl(struct file *filp, unsigned int cmd, unsigned long
arg)
{
        int retval = 0;
        struct lunix_chrdev_state_struct *state;
        state = filp->private_data;

        switch(cmd) {
                case LUNIX_IOC_SWITCH:
                        if (!capable(CAP_SYS_ADMIN))
                                return -EPERM;
                        if (down_interruptible(&state->lock))
                                return -ERESTARTSYS;
                        state->switch_raw = !state->switch_raw;
                        up(&state->lock);
                        break;
                default:
                        return -ENOTTY;
        }

        return retval;
}
```

Δοκιμή

Η εξέταση αυτής της λειτουργίας γίνεται με το ακόλουθο απλό πρόγραμμα, στο οποίο για κάθε δεύτερη τιμή που διαβάζεται αλλάζει την παρουσίαση μέσω της ανάλογης κλήσης ioctl (για ευκολία θεωρούμε ότι το εκτελέσιμο βρίσκεται στον ίδιο κατάλογο με τον lunix-tng-helpcode-*):

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

#include <sys/ioctl.h>
#define SIZE 100
#include "lunix-tng-helpcode-20201029/lunix-chrdev.h"

int main (int argc, char** argv) {
        int fd, sz, i, p;
        char *c = (char *) calloc(SIZE, sizeof(char));

        fd = open(argv[1], O_RDONLY);

        if (fd < 0) {
                perror("OPEN");
                exit(1);
        }

        i = 0;
        while(1) {
                i++;
                sz = read(fd,c, 1000);
                c[sz] = '\0';
                printf("READ VALUE [%d BYTES] :: %s", sz, c);
                if (i % 2 == 0) {
                        ioctl(fd, LUNIX_IOC_SWITCH);
                }
        }
}
```

Από την εκτέλεση αυτού του προγράμματος (ως root), με την εντολή `./test /dev/lunix0-batt` λαμβάνουμε την ακόλουθη έξοδο:

```
READ VALUE [6 BYTES] :: 3.354
READ VALUE [6 BYTES] :: 3.354
READ VALUE [4 BYTES] :: 373
READ VALUE [4 BYTES] :: 373
READ VALUE [6 BYTES] :: 3.354
READ VALUE [6 BYTES] :: 3.354
...
```

Υποστήριξη επικοινωνίας χώρων πυρήνα – χρήστη χωρίς κλήση συστήματος read(), με memory-mapped I/O

Στην συνέχεια παρουσιάζεται μια απλή υλοποίηση της συνάρτησης mmap, με την οποία απεικονίζουμε την σελίδα που αφορά τον συγκεκριμένο αισθητήρα και την μέτρηση στον χώρο χρήστη αξιοποιώντας την remap_pfn_range. Επειδή το πεδίο vm_pgoff δεν έχει νόημα σε αυτήν την περίπτωση, εφόσον το περιεχόμενο της σελίδας που μας ενδιαφέρει υπερκαλύπτεται από το μέγεθος της, επιλέξαμε να επιστρέφουμε σφάλμα EINVAL σε περίπτωση που δοθεί μη μηδενική τιμή του offset. Στην ουσία το σημαντικότερο μέρος της ακόλουθης υλοποίησης είναι η εφαρμογή

της συνάρτησης virt_to_phys() στην διεύθυνση της ανάλογης σελίδας για να λάβουμε την φυσική διεύθυνση αυτής, και να μετατοπίσουμε την προαναφερθείσα κατά PAGE_SHIFT ώστε να λάβουμε τον αντίστοιχο αριθμό σελίδας που θα ληφθεί ως όρισμα στην remap_pfn_range.

```
static int lunix_chrdev_mmap(struct file *filp, struct vm_area_struct *vma)

{
        struct lunix_chrdev_state_struct *state;
        unsigned long page;
        state = filp->private_data;

        if (vma->vm_pgoff)
                return -EINVAL;

        page = virt_to_phys((unsigned long *)
               state->sensor->msr_data[state->type]) >> PAGE_SHIFT;

        if (remap_pfn_range(vma, vma->vm_start, page,
                            vma->vm_end - vma->vm_start,
                            vma->vm_page_prot))
                return -EAGAIN;

        return 0;
}
```

Δοκιμή

Με το ακόλουθο δοκιμαστικό πρόγραμμα απεικονίζουμε το αρχείο που δίνεται ως όρισμα στον χώρο μνήμης της διεργασίας που δημιουργείται, και ανά 1 δευτερόλεπτο διαβάζουμε την θέση της σελίδας στην οποία αποθηκεύεται το μετρούμενο μέγεθος.

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/io.h>
#include <sys/mman.h>
#include <time.h>

#define PAGE_SIZE 4096

int main(int argc, char const *argv[])
{
    unsigned long int *f;
    int size;
    struct stat s;
    const char * file_name = argv[1];
    int fd = open (argv[1], O_RDONLY);

    f = (unsigned long int *) mmap (0, PAGE_SIZE, PROT_READ, MAP_SHARED, fd, 0);

    if (f == MAP_FAILED) {
        perror("Unable to mmap the given file");
        close(fd);
        return 1;
    }
```

```
    while(1){
        printf("%d\n", f[1]);
        sleep(1);
    }

    return 0;
}
```

Από την εκτέλεση αυτού διαπιστώνουμε ότι πράγματι λαμβάνονται μη επεξεργασμένες τιμές οι οποίες αλλάζουν όταν υπάρξει μεταβολή στην μέτρηση.