2η Εργαστηριακή Άσκηση

Διαμάντη Ιωάννα el15035
Ντούνης Πέτρος el15091

int lunix_chrdev_init (void):

Η συνάρτηση αυτή εκτελείται μία φορά κατά την εισαγωγή του driver στον πυρήνα (εντολή insmod). Με την εντολή cdev_init(&lunix_chrdev_cdev, &lunix_chrdev_fops) αρχικοποιούμε την συσκευή χαρακτήρων και τη συνδέσουμε με τη δομή file_operations lunix_chrdev_fops, στην οποία δηλώνεται ποιες συναρτήσεις υλοποιούν τις λειτουργίες που υποστηρίζει η συσκευή. Με την εντολή dev_no = MKDEV (LUNIX_CHRDEV_ MAJOR, 0) φτιάχνουμε έναν device number ο οποίος θα έχει major =60 και minor =0. Στη συνέχεια καλούμε την register_chrdev_region(dev_no,128,"lunix") ,η οποία αρχίζοντας από τον dev_no δεσμεύει 128 device numbers για τον driver μας. Τέλος καλώντας την cdev_add (&lunix_chrdev_cdev, dev_no, 128) προσθέτουμε στον πυρήνα 128 device numbers ,ξεκινώντας από το dev_no ,τα οποία αντιστοιχούν στην συσκευή χαρακτήρων lunix_chrdev_cdev.

static int lunix_chrdev_open(struct inode *inode, struct file *filp) :

Η συνάρτηση αυτή καλείται κάθε φορά που μία διεργασία ανοίγει ένα ειδικό αρχείο (struct inode) της συσκευής χαρακτήρων. Με την κλήση της nonseekable_open(inode,filp) δημιουργείται αυτόματα μία νέα δομή ανοιχτού αρχείου (struct file),η οποία χαρακτηρίζεται ως «μη ανιχνεύσιμη». Αυτό σημαίνει ότι ο χρήστης δεν έχει τη δυνατότητα να αλλάξει την θέση του file descriptor του ανοιχτού αρχείου (lseek(),pread(), pwrite() system calls). Στη συνέχεια δημιουργούμε μία νέα δομή lunix_chrev_state_struct και αρχικοποιούμε τα μέλη της ως εξής:
Καλώντας την συνάρτηση iminor(inode) βρίσκουμε τον minor number του ειδικού αρχείου που ανοίξαμε. Με βάση την εκφώνηση της άσκησης γνωρίζουμε ότι ο minor number ενός ειδικού αρχείου δίνεται από τον τύπο minor = αισθητήρας*8 + μέτρηση.
Έτσι θέτουμε state->type = minor%8, state->sensor = &lunix_sensors[minor/8]. Επίσης αρχικοποιούμε στο 0 τις μεταβλητές μέλη : buf_lim και buf_timestamp, οι οποίες αντιπροσωπεύουν το μέγεθος και το πόσο πρόσφατη είναι η μέτρηση αντίστοιχα που περιέχει το ανοιχτό αρχείο σε κάθε στιγμή. Τέλος αρχικοποιούμε τον σημαφόρο της δομής state(δηλαδή του ανοιχτού αρχείου) στην τιμή 1 (ξεκλείδωτος) και αποθηκεύουμε την δομή state στα private_data της δομής file του ανοιχτού αρχείου, ώστε να έχουμε εύκολη πρόσβαση σε αυτήν σε κάθε λειτουργία πάνω στο ανοιχτό αρχείο.

static int lunix_chrdev_state_needs_refresh(struct lunix_chrdev_state_struct *state):

Η συνάρτηση αυτή χρησιμοποιείται για να ελέγξουμε αν υπάρχει νέα διαθέσιμη μέτρηση για ένα ανοιχτό αρχείο που περιμένει νέα δεδομένα από κάποιο συγκεκριμένο αισθητήρα. Συγκεκριμένα ελέγχουμε αν ισχύει η συνθήκη : state->buf_timestamp != sensor-> msr_data[state->type]->last_update. Συγκρίνουμε δηλαδή την τιμή της μεταβλητής buf_timestamp του ανοιχτού αρχείου με την τιμή της μεταβλητής last_update του είδους μέτρησης του αισθητήρα που επιθυμούμε. Αν αυτές οι δύο τιμές είναι ίσες τότε η τελευταία μέτρηση του αισθητήρα έχει ήδη ληφθεί και δεν υπάρχει ακόμα νέα διαθέσιμη μέτρηση. Διαφορετικά έχει γίνει λήψη νέας μέτρησης από τον επιθυμητό αισθητήρα.

static int lunix_chrdev_state_update(struct lunix_chrdev_state_struct *state):

Η συνάρτηση αυτή καλείται κάθε φορά που μια διεργασία θέλει να διαβάσει από ένα ανοιχτό αρχείο, αλλά δεν υπάρχουν καθόλου διαθέσιμα δεδομένα (ούτε 1 byte διαθέσιμο προς ανάγνωση). Αρχικά κλειδώνουμε το spinlock του αντίστοιχου αισθητήρα με την εντολή spin_lock_irq(&sensor->lock), η οποία πριν το κλειδώσει απενεργοποιεί τα interrupts στη συγκεκριμένη CPU. Αυτό είναι απαραίτητο σε περίπτωση που ο driver εκτελείται σε ένα uniprocessor σύστημα με μη-διακοπτή χρονοδρομολόγηση

(non-preemptive scheduling). Στην περίπτωση αυτή υπάρχει το ενδεχόμενο μία διεργασία να κλειδώσει το spinlock ενός αισθητήρα και κατά την διάρκεια εκτέλεσης του κρίσιμου τμήματος (πριν απελευθερώσει το spinlock) να συμβεί κάποιο interrupt από τον συγκεκριμένο αισθητήρα (επειδή πχ ήρθε νέα μέτρηση). Τότε η διεργασία φεύγει από τη μοναδική CPU του συστήματος για να εκτελεστεί ο interrupt handler. Στη δική μας περίπτωση θα πρέπει να εκτελεστεί η συνάρτηση: void lunix_sensor_update(struct lunix_sensor_ struct *s, uint16_t batt, uint16_t temp, uint16_t light) , η οποία ανανεώνει τα δεδομένα του αισθητήρα. Για να συμβεί όμως αυτό η συνάρτηση πρέπει να κλειδώσει το spinlock του συγκεκριμένου αισθητήρα, το οποίο είναι ήδη κλειδωμένο όπως είδαμε από την lunix_chrdev_state_update. Επειδή έχουμε uniprocessor σύστημα, μη-διακοπτή χρονοδρομολόγηση και χρήση spinlock , ο interrupt handler δεν θα αφήσει ποτέ την CPU μέχρι να μπορέσει να κλειδώσει το spinlock και η συνάρτηση lunix_chrdev_state_ update δεν θα μπορέσει ποτέ να μπει στην CPU ώστε να ξεκλειδώσει το spinlock, καθώς αυτή είναι μονίμως κατειλημμένη. Έτσι καταλήγουμε σε deadlock του συστήματος. Συνεχίζοντας με την λειτουργία της lunix_chrdev_state_update μετά το κλείδωμα του spinlock, ελέγχουμε αν υπάρχει νέα μέτρηση με κλήση της lunix_chrdev_state_ needs_refresh. Αν δεν υπάρχει τότε απελευθερώνουμε το spinlock του αισθητήρα με την εντολή spin_unlock_irq(&sensor->lock) και η lunix_chrdev_ state_update επιστρέφει -EAGAIN (try again-nonblocking λειτουργία). Διαφορετικά παίρνω τα raw data από την μεταβλητή values της επιθυμητής μέτρησης (sensor->msr_data[state->type]->values[0]), ανανεώνω τη μεταβλητή buf_timestamp του ανοιχτού αρχείου με την τιμή της last_update (state->buf_timestamp = sensor->msr_data[state->type]->last_update) και ξεκλειδώνω το spinlock του αισθητήρα με την εντολή spin_unlock_irq(&sensor->lock). Στη συνέχεια με βάση την τιμή της μεταβλητής type της δομής state βρίσκω σε τι είδους μέτρηση αναφέρομαι (batt -0,temp-1,light-2) και περνάω τα raw data μέσα από το κατάλληλο lookup table, ώστε να λάβω μία προσημασμένη δεκαδική τιμή. Γνωρίζοντας ότι μία τιμή ±xx.yyy είναι αποθηκευμένη στο lookup table ως ±xxyyy, αρκεί μία διαίρεση με το 1000 ώστε να βρεθεί το ακέραιο και δεκαδικό μέρος της μέτρησης. Έτσι αν m = |xxyyy|%1000 = yyy , d = ±xxyyy/1000 = ±xx , η συνάρτηση sprintf(state->buf_data, "%ld.%ld",d,m) αντιγράφει την μέτρηση στην τελική αναγνώσιμη μορφή της στον πίνακα buf_data της δομής state, δηλαδή πρακτικά στο ανοιχτό μας αρχείο. Η συνάρτηση αυτή έχει ως τιμή επιστροφής τον αριθμό των byte που έγραψε στα buf_data, δηλαδή το μέγεθος της μέτρησης. Έτσι η τιμή της μεταβλητής buf_lim , που αντιπροσωπεύει το μέγεθος της εκάστοτε μέτρησης που υπάρχει στο ανοιχτό αρχείο τίθεται ίση με την τιμή επιστροφής της sprintf.

static ssize_t lunix_chrdev_read(struct file *filp, char __user *usrbuf, size_t cnt, loff_t *f_pos):

Η συνάρτηση αυτή καλείται κάθε φορά που μία διεργασία θέλει να διαβάσει κάποια bytes από το περιεχόμενο ενός ανοιχτού αρχείου. Αρχικά μέσω του δείκτη private_data παίρνουμε τις πληροφορίες που θέλουμε για το ανοιχτό αρχείο (state->lock,state->sensor κλπ). Κατόπιν με την κλήση της down_interruptible(&state->lock) κλειδώνουμε τον σημαφόρο της δομής state (του ανοιχτού αρχείου), ώστε να μην μπορεί κάποια άλλη διεργασία (αν πχ είχαμε κάνει προηγουμένως fork) να διαβάσει την ίδια στιγμή από το ίδιο ανοιχτό αρχείο. Στη συνέχεια ελέγχουμε αν η τιμή του δείκτη f_pos είναι 0. Μηδενική τιμή του δείκτη σημαίνει ότι δεν υπάρχουν διαθέσιμα δεδομένα στο ανοιχτό αρχείο. Σε αυτή την περίπτωση η διεργασία πρέπει να περιμένει την λήψη νέας μέτρησης. Καλείται λοιπόν η συνάρτηση lunix_chrdev_state_update(state). Αν αυτή επιστρέψει ότι δεν υπάρχουν ακόμα νέα δεδομένα ,ξεκλειδώνουμε τον σημαφόρο του ανοιχτού αρχείου και με την κλήση της wait_event_interruptible (sensor->wq, lunix_chrdev_state_needs_refresh(state)) κοιμίζουμε την διεργασία τοποθετώντας την στην ουρά αναμονής του αισθητήρα από τον οποίο περιμένει μέτρηση. Η διεργασία ξυπνάει κάποια στιγμή από τον αντίστοιχο αισθητήρα με την κλήση της wake_up_interruptible(&s->wq) μόλις ληφθούν νέες μετρήσεις και ελέγχεται η συνθήκη lunix_chrdev_state_needs_refresh(state). Κατόπιν κλειδώνεται πάλι ο σημαφόρος του ανοιχτού αρχείο για τους λόγους που περιγράψαμε παραπάνω και καλείται ξανά η lunix_chrdev_state_update(state). Η παραπάνω διαδικασία επαναλαμβάνεται για κάθε διεργασία μέχρι η συνάρτηση lunix_chrdev_state_update(state) να επιστρέψει ότι υπάρχουν νέα δεδομένα προς ανάγνωση.

Μόλις αυτό συμβεί, ελέγχουμε αν μπορούμε να δώσουμε στον χρήστη όσα byte ζήτησε. Σε περίπτωση που η τρέχουσα θέση στο ανοιχτό αρχείο(f_pos) + τον αριθμό των byte που ζητήθηκαν από τον χρήστη (cnt) υπερβαίνει το μέγεθος της μέτρησης (buf_lim), του επιστρέφουμε όσα byte απομένουν μέχρι το τέλος της μέτρησης (buf_lim-*f_pos) με χρήση της copy_to_user(usrbuf,&state->buf_data[*f_pos] ,cnt). Η συνάρτηση αυτή χρησιμοποιείται για την ασφαλή μεταφορά δεδομένων στο χώρο χρήστη, καθώς ο πυρήνας έχει πρόσβαση σε όλη τη μνήμη και χωρίς αυτήν μία αναφορά σε invalid διεύθυνση δεν θα γινόταν αντιληπτή (δεν θα προέκυπτε segmentation fault). Στη συνέχεια αυξάνουμε την τιμή του f_pos κατά τον αριθμό των byte που μπορέσαμε να δώσουμε στον χρήστη. Αν φτάσαμε στο τέλος της μέτρησης (δηλαδή αν *f_pos == state->buf_lim) το f_pos μηδενίζεται ,υποδεικνύοντας την έλλειψη δεδομένων προς ανάγνωση και την ανάγκη για update. Τέλος ξεκλειδώνουμε τον σημαφόρο του ανοιχτού αρχείου ώστε να επιστρέψουμε την πρόσβαση σε αυτό από άλλες διεργασίες και επιστρέφουμε στον χρήστη τον αριθμό των byte που μπορέσαμε να του δώσουμε.

static int lunix_chrdev_release(struct inode *inode, struct file *filp):

Σε ένα ανοιχτό αρχείο μπορεί να έχουν πρόσβαση παραπάνω από μία διεργασίες (όταν πχ κάνουμε fork και διαβάζουμε από ένα αρχείο). Η συνάρτηση lunix_chrdev_release καλείται κάθε φορά που ένα ανοιχτό αρχείο κλείνει από την τελευταία διεργασία που έχει πρόσβαση σε αυτό ή όταν αυτή τερματίζει την εκτέλεση της. Στην περίπτωση αυτή τα δεδομένα και η κατάσταση του ανοιχτού αρχείου δεν μας ενδιαφέρουν πλέον. Το στιγμιότυπο της δομής file καταστρέφεται αυτόματα ενώ εμείς απελευθερώνουμε τον χώρο στη μνήμη που είχαμε δεσμεύσει για τα ειδικά δεδομένα του συγκεκριμένου αρχείου κατά το άνοιγμα του με την εντολή kfree(filp->private_data).

void lunix_chrdev_destroy(void):

Η συνάρτηση αυτή εκτελείται κάθε φορά που γίνεται αφαίρεση του driver από τον πυρήνα (εντολή rmmod). Με την εντολή cdev_del(&lunix_chrdev_cdev) διαγράφει τη συσκευή χαρακτήρων από τον πυρήνα, ενώ με τις εντολές dev_no = MKDEV(LUNIX_CHRDEV_MAJOR, 0) και unregister_chrdev_region (dev_no, lunix_minor_cnt) απελευθερώνει τους device numbers που είχε δεσμεύσει η συσκευή χαρακτήρων κατά την αρχικοποίηση της.