ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολη ΗΜ&ΜΥ
Εργαστήριο Λειτουργικών Συστημάτων
Άσκηση 1
Ακ. Έτος 2012-13

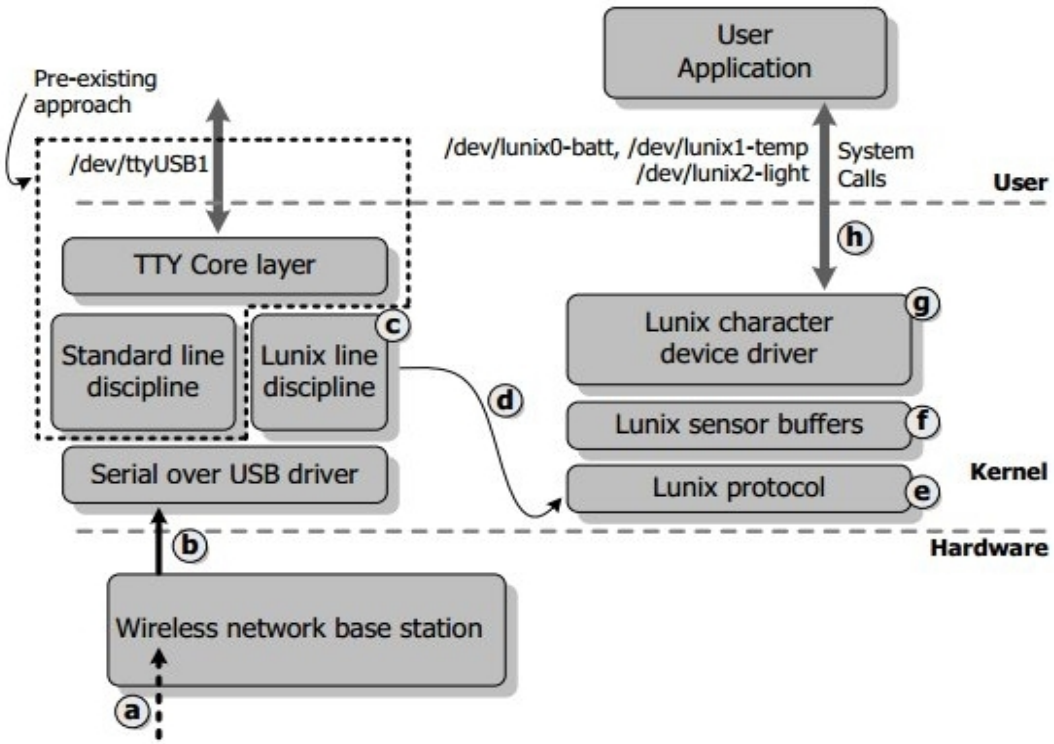
Παπαδημητρίου Κων/νος | ΑΜ: 03108769
Ημερομηνία: 06/07/2013

Εισαγωγή

Στην άσκηση αυτή μας ζητήθηκε να υλοποιήσουμε έναν οδηγό για ασύρματο δίκτυο αισθητήρων στο λειτουργικό σύστημα Linux. Το συγκεκριμένο δίκτυο διαθέτει έναν αριθμό από αισθητήρες και ένα σταθμό βάσης. Ο σταθμός βάσης συνδέεται μέσω USB με υπολογιστικό σύστημα Linux στο οποίο και εκτελείται ο οδηγός συσκευής. Οι αισθητήρες αναφέρουν περιοδικά το αποτέλεσμα τριών διαφορετικών μετρήσεων:
της τάσης της μπαταρίας που τους τροφοδοτεί, της θερμοκρασίας και της φωτεινότητας του χώρου όπου βρίσκονται. Τα δεδομένα μεταφέρονται μέσω ενός δικτύου mesh, από εναλλακτικές διαδρομές, έτσι ώστε το δίκτυο να προσαρμόζεται αυτόματα στην περίπτωση όπου ένας οι 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 μία εικονική σειριακή θύρα, /dev/ttyUSB1. Ζητούμενο ήταν η κατασκευή οδηγού συσκευής χαρακτήρων, ο οποίος λαμβάνει τα δεδομένα των μετρήσεων από το δίκτυο αισθητήρων και τα εξάγει στο χώρο χρήστη σε διαφορετικές συσκευές, ανάλογα με το είδος της μέτρησης και τον αισθητήρα απ' όπου προέρχεται.

Αρχιτεκτονική οδηγού συσκευής

Στο παρακάτω σχήμα φαίνεται αναλυτικά η αρχιτεκτονική την οποία ακολουθεί ο οδηγός συσκευής.



Σχήμα 1: Αρχιτεκτονική λογισμικού του υπό εξέταση συστήματος

Το σύστημα οργανώνεται σε δύο κύρια μέρη: Το πρώτο (τμήματα (c), (d), (e) του σχήματος) αναλαμβάνει τη συλλογή των δεδομένων από το σταθμό βάσης και την επεξεργασία τους με συγκεκριμένο πρωτόκολλο, έτσι ώστε να εξαχθούν οι τιμές των μετρούμενων μεγεθών, οι οποίες και κρατούνται σε κατάλληλες δομές προσωρινής

αποθήκευσης (buffers) στη μνήμη (τμήμα (f)). Το δεύτερο (τμήματα (g), (h)), αναλαμβάνει την εξαγωγή των δεδομένων στο χώρο χρήστη, υλοποιώντας μια σειρά από συσκευές χαρακτήρων. Τα δεδομένα προέρχονται από ανάγνωση των χώρων ενδιάμεσης αποθήκευσης. Το τελικό σύστημα λειτουργεί ως εξής: Τα δεδομένα των μετρήσεων αφού (a) λαμβάνονται από το σταθμό βάσης (b) προωθούνται μέσω USB στο υπολογιστικό σύστημα. Στη συνέχεια όμως δεν ακολουθούν τη συνήθη πορεία τους, προς το /dev/ttyUSB1 αλλά (c) παραλαμβάνονται από ένα φίλτρο, τη διάταξη γραμμής (line discipline) του Lunix, η οποία (d) τα προωθεί σε κατάλληλο στρώμα ερμηνείας του πρωτοκόλλου των αισθητήρων (e) Το τμήμα αυτό ερμηνεύει το περιεχόμενο των πακέτων και αποθηκεύει τα δεδομένα των μετρήσεων σε διαφορετικούς buffers ανά αισθητήρα (f). Η ζητούμενη συσκευή χαρακτήρων (g), αναλόγως με το ποιο ειδικό αρχείο χρησιμοποιεί η εφαρμογή χρήστη, θα επιτρέπει την ανάκτηση (h) από τους buffers των δεδομένων που αντιστοιχούν σε κάθε περίπτωση. Τα τμήματα (a), (b), (c), (d), (e) και (f) μας δίνονταν έτοιμα, οπότε απέμενε η υλοποίηση των (g) και (h). Έτσι ουσιαστικά έπρεπε να υλοποιηθούν τα file_operations lunix_chrdev_open(), lunix_chrdev_release() και lunix_chrdev_read().

Ανάπτυξη του οδηγού

Αυτή τη στιγμή υποστηρίζονται οι κλήσεις συστήματος open(), read() και close(). Έτσι, όταν ο χρήστης κάνει open το ειδικό αρχείο ενός αισθητήρα θα δημιουργούνται οι κατάλληλες δομές και όταν κάνει close θα ελευθερώνεται η μνήμη που είχε δεσμευθεί κατα την κλήση της open. Όταν καλεί τη read, θα πρέπει ο driver να βρίσκει τα τελευταία δεδομένα του αισθητήρα και να τα επιστρέφει.

Όταν ένας αισθητήρας έχει δεδομένα, καλείται η lunix_ldisc_receive(), η οποία με τη σειρά της καλεί την lunix_protocol_received_buf(), οπότε και τα δεδομένα στέλνονται στον αντίστοιχο sensor buffer. Στα επόμενα βήματα καλείται η lunix_sensor_update() ώστε να ανανεωθούν οι δομές του sensor. Όλα αυτά γίνονται σε interrupt context και οποιαδήποτε επικοινωνία υπάρχει με κώδικα που εκτελείται σε process context πρέπει να προστατεύεται με spinlocks (γίνεται αναφορά παρακάτω). Τα παραπάνω μας δίνονταν έτοιμα.

Τέλος, όταν ο χρήστης κάνει read, θα πρέπει ο driver να ελέγχει αν έχει έτοιμα δεδομένα να στείλει αλλιώς πρέπει να περιμένει (μπλοκάροντας τη διεργασία) μέχρι να έρθουν καινούρια δεδομένα από τον αντίστοιχο αισθητήρα.

Στη συνέχεια περιγράφονται τα file operations (καθώς και δύο συναρτήσεις τις οποίες χρησιμοποιούν) με τα οποία υλοποιήσαμε τα παραπάνω.

File operations

Η lunix_chrdev_open() δέχεται ορίσματα ένα δείκτη σε ένα struct inode καθώς και ένα δείκτη σε ένα struct file *filp. Αφού βρει σε ποιον αισθητήρα αναφέρεται το αρχείο κοιτάζοντας το minor number του, δημιουργεί και αρχικοποιεί ένα struct lunix_chrdev_state_struct, στο οποίο θα δείχνει το private_data της δομής file. Το lunix_chrdev_state_struct κρατάει πληροφορίες, όπως για παράδειγμα ένα δείκτη προς το αντίστοιχο struct του αισθητήρα στον οποίο αναφέρεται και τον τύπο του αισθητήρα (μπαταρία, φωτεινότητα ή θερμοκρασία).

Η lunix_chrdev_release() δέχεται τα ίδιο ορίσματα με τη lunix_chrdev_open() και το μόνο που κάνει είναι να ελευθερώνει το struct lunix_chrdev_state_struct.

Η lunix_chrdev_read() δέχεται σαν ορίσματα δείκτες σε ένα struct file, σε ένα buffer στον οποίο πρέπει να γράψει το αποτέλεσμα και στο f_pos του αρχείου, καθώς και τον αριθμό των bytes που ζήτησε ο χρήστης. Έτσι η read περιμένει μέχρι να έρθουν δεδομένα από τον αισθητήρα, κοιμίζοντας τη διεργασία. Για να δει αν υπάρχουν διαθέσιμα δεδομένα χρησιμοποιεί τη lunix_chrdev_state_update(), η οποίο εξετάζεται παρακάτω. Όταν τελικά πάρει τα δεδομένα, βρίσκει πόσα bytes θα επιστρέψει στον χρήστη και τα στέλνει (χρησιμοποιώντας τη copy_to_user). Τέλος πρέπει να αλάξει και το f_pos και αν έχει ξεπεράσει το όριο του buffer να το κάνει 0.

Βοηθητικές συναρτήσεις

Η lunix_chrdev_state_update(), χρησιμοποιεί τη βοηθητική συνάρτηση lunix_chrdev_state_needs_refresh() για να ελέγξει αν έχουν έρθει νέα δεδομένα στον αισθητήρα. Αν όντως έχουν έρθει, πρέπει να ανανεωθεί το lunix_chrdev_state_struct, οπότε η lunix_chrdev_state_update αντιγράφει τα δεδομένα από το αντίστοιχο buffer του αισθητήρα και ανανεώνει το timestamp του lunix_chrdev_state_struct το οποίο κρατάει την τελευταία φορά που ανανεώθηκε ο buffer. Τελικά αφού επεξεργαστεί τα δεδομένα, τα τοποθετεί στον buffer και επιστρέφει.

Η lunix_chrdev_state_needs_refresh(), απλώς κοιτάει την τελευταία φορά που ανανεώθηκε το struct του αισθητήρα και αν είναι διαφορετική από αυτή που έχει το timestamp του lunix_chrdev_state_struct επιστρέφει 1, διαφορετικά επιστρέφει 0.

Κλειδώματα

Μεγάλη προσοχή χρειάστηκε το θέμα των κλειδωμάτων. Υπάρχουν κρίσιμα τμήματα στον κώδικα του οδηγού, τα οποία πρέπει να εκτελούνται ατομικά.

Αν μία διεργασία ανοίξει κάποια συσκευή και στη συνέχεια αποκτήσει απογόνους, τότε αυτοί θα έχουν το ίδιο lunix_chrdev_state_struct το οποίο μπορούν να επεξεργαστούν. Οπότε όλες οι ενέργειες σε αυτό θα πρέπει να γίνονται ατομικά. Για τον λόγο αυτό, περιέχεται και ένας σεμαφόρος, τον οποίο πρέπει να κλειδώνουμε κάθε φορά που πειράζουμε το struct.

Ένα άλλο κρίσιμο τμήμα είναι αυτό του struct του αισθητήρα, γιατί γίνεται πρόσβαση σε αυτό είτε σε process context (δηλαδή όταν εκτελείται κώδικας εκ μέρους της διεργασίας) είτε σε interrupt context (όταν εκτελείται κώδικας προς ικανοποίηση ενός interrupt). Εδώ όμως πρέπει να χρησιμοποιηθεί spinlock αντί για semaphore, καθώς απαγορεύεται ο πυρήνας να κοιμηθεί όταν βρίσκεται σε interrupt context.