ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
http://www.cslab.ece.ntua.gr

Εργαστήριο Λειτουργικών Συστημάτων

8ο εξάμηνο, Ακαδημαϊκή περίοδος 2010-2011

Εξετάσεις Ιουνίου 2011

Διάρκεια: 2:30 ώρες

Η εξέταση πραγματοποιείται με ανοιχτά βιβλία και σημειώσεις.

Θέμα 1 (30%)

α. (10%)

i. Εξηγήστε το μηχανισμό που εξασφαλίζει την κληρονομικότητα των κατάρων.

ii. Πώς εξασφαλίζεται ότι αρχικά καμία διεργασία δεν είναι σημειωμένη με οποιαδήποτε κατάρα;

β. (10%) Επιθυμούμε να μεταβάλουμε το μηχανισμό των κατάρων ώστε όταν θέτουμε μια κατάρα για μια διεργασία, να εξασφαλίζεται ότι και όλοι οι υπάρχοντες απόγονοί της ανεξαιρέτως θα υπόκεινται στην κατάρα αυτή.

Προτείνετε μια σχεδίαση για το μηχανισμό αυτό.

γ. (10%) Επιθυμούμε να υλοποιήσουμε μια κατάρα-μνημοθέτη, ώστε οι καταραμένες διεργασίες να κληροδοτούν μια προκαθορισμένη περιοχή της εικονικής μνήμης τους σε όλους τους μέλλοντες απογόνους τους. Η κληρονομημένη αυτή εικονική μνήμη, παραμένει κοινή για όλους και είναι αδύνατο να καταστραφεί.

Σε ποιές κλήσεις συστήματος θα πρέπει να επέμβουμε και γιατί;

Θέμα 2 (40%)

α. (15%) Για κάθε μία από τις παρακάτω προτάσεις απαντήστε αν είναι αληθής ή ψευδής, με *σύντομη* αιτιολόγηση 2-3 γραμμών.

i. Όταν συμβαίνει μια διακοπή υλικού, εκτελείται κώδικας πυρήνα του Λ.Σ.

ii. Μια δομή η οποία προσπελάζεται ταυτόχρονα από process και interrupt context μπορεί να προστατευτεί με σημαφόρους.

iii. Μια κλήση συστήματος εκτελείται πάντοτε σε process context.

iv. Κώδικας που εκτελείται σε process context επιτρέπεται να κοιμηθεί.

v. Κώδικας που εκτελείται σε interrupt context μπορεί να καλέσει την `copy_to_user()`.

vi. Δεν υπάρχει τρόπος ανάγνωσης δεδομένων από αρχείο στο Linux χωρίς χρήση της κλήσης `read()`.

β. (25%) Θεωρήστε ένα σενάριο οδηγού συσκευής παρόμοιου με το Lunix:TNG. Στη Fukushima, ένα στρώμα συλλογής δεδομένων εκτελείται σε interrupt context και τοποθετεί εισερχόμενα δεδομένα (μετρήσεις ακτινοβολίας) σε δομή κυκλικού απομονωτή (circular buffer).

```
struct input_data {
        ...locktype... lock;
        wait_queue_head_t wq;
        uint32_t cnt;

#define CIRC_BUF_SIZE (1024 * sizeof(measurement_t))
        char circ_buffer[CIRC_BUF_SIZE];
} input_data;
```

Οι μετρήσεις είναι ποσότητες τύπου `measurement_t`, ενδεικτικού μεγέθους 128 bytes. Το πεδίο `cnt` αρχικοποιείται στην τιμή `0` κατά την εκκίνηση του ΛΣ. Ο κώδικας του interrupt context είναι:

```
void intr(void)
{
        struct input_data *inp = &input_data;
        ... lock input_data ...
        memcpy(&inp->circ_buffer[inp->cnt % CIRC_BUF_SIZE],
                device_memory, sizeof(measurement_t));
        inp->cnt += sizeof(measurement_t);
        ... unlock input_data ...
        ...
}
```

Ο οδηγός συσκευής υλοποιεί μια συσκευή χαρακτήρων. Κάθε διεργασία που ανοίγει το ειδικό αρχείο της συσκευής `/dev/nuclear` διαβάζει με διαδοχικές κλήσεις `read()` τις μετρήσεις που γράφτηκαν στον κυκλικό απομονωτή από τη στιγμή του `open()` και μετά, με τη σειρά που γράφτηκαν, ως ένα *ρεύμα* (stream) από bytes. Ενώ οι μετρήσεις έχουν σταθερό μέγεθος, η αντιγραφή τους από τον απομονωτή μέσω της `read()` γίνεται ελεύθερα, ανά byte. Ισχύουν τα εξής:

1. Το ρεύμα των μετρήσεων ξεκινά με την πρώτη μέτρηση που θα έρθει από τη συσκευή μετά την εκτέλεση της `open()` και συνεχίζεται καθώς φτάνουν νέες μετρήσεις. Όταν έχουν διαβάσει πλήρως όλες τις διαθέσιμες μετρήσεις, οι διεργασίες κοιμούνται, μέχρι να εγγραφεί νέα μέτρηση στον απομονωτή από το interrupt context.
2. Το interrupt context δεν περιμένει την ανάκτηση των μετρήσεων από τις διεργασίες, αλλά αντικαθιστά την παλαιότερη με την πιο νέα. Διαδοχικές κλήσεις `read()` επιστρέφουν πάντοτε έγκυρα και διαδοχικά δεδομένα μετρήσεων. Όμως, όταν αυτό δεν είναι δυνατό, επιστρέφεται EOF.

Ο οδηγός συσκευής έχει την εξής δομή – θεωρούμε ότι όλες οι κλήσεις για δέσμευση μνήμης πετυχαίνουν:

```
struct chrdev_state {
        ...locktype... lock;
        struct input_data *inp;
        /* Όποια άλλα πεδία χρειάζεστε εδώ ... */
};

#define wait_event_interruptible(waitqueue, condition) ...
unsigned long copy_to_user(void __user *dst, const void *src, unsigned long len);

static int chrdev_open(struct inode *inode, struct file *filp)
{
        struct chrdev_state *state = kmalloc(sizeof(*state), GFP_KERNEL);
        filp->private_data = state;

        /* Συμπληρώστε ... */
}

static ssize_t lunix_chrdev_read(struct file *filp, char __user *usrbuf,
        size_t cnt, loff_t *f_pos)
{
        struct chrdev_state *state = filp->private_data;
        struct input_data *inp = state->inp;

        /* Συμπληρώστε ... */
}
```

Ζητούνται τα εξής:

i. (5%) Τι μετράει το πεδίο `cnt` της δομής `input_data`; Ποιος ο ρόλος των πεδίων `lock` και `wq` της ίδιας δομής; Τι τύπου είναι τα κλειδώματα των δύο δομών, `state` και `input_data`;

ii. (20%) Υλοποιήστε τις συναρτήσεις `open()` και `read()` για τον οδηγό συσκευής.

Θέμα 3 (30%)

α. (10%) Ένα pipe, όπως αυτό δημιουργείται από την `pipe()`, χαρακτηρίζεται ως *ανώνυμο*. Γιατί; Τι περιορισμό εισάγει η ανωνυμία του όσον αφορά στην δυνατότητα πρόσβασης σε αυτό από διεργασίες; Υπάρχει αντίστοιχος μηχανισμός που δεν έχει αυτόν τον περιορισμό;

β. (10%) Περιγράψτε ένα μηχανισμό του Linux που επιτρέπει σε διαφορετικές διεργασίες να έχουν κοινή μνήμη.

γ. (10%) Σκιαγραφήστε την υλοποίηση προγράμματος που δέχεται ως είσοδο ένα εκτελέσιμο *P* και ένα αρχείο *F* κι εκτελεί το *P* με τέτοιο τρόπο ώστε η καθιερωμένη έξοδός του (standard output) να καταλήγει στο *F*.