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 2012-2013

Επαναληπτική Εξέταση

Διάρκεια: 2 ώρες, 30 λεπτά

Η εξέταση πραγματοποιείται με ανοιχτά βιβλία και σημειώσεις.

Θέμα 1 (40%)

Συμμετέχετε σε ομάδα που σχεδιάζει και υλοποιεί ενσωματωμένο σύστημα Linux το οποίο ελέγχει την αναπαραγωγή μουσικής στο ηχοσύστημα MEGA BLASTER. Ο ήχος αναπαριστάται ως μονοφωνικό ρεύμα δειγμάτων PCM, 8-bit, με ρυθμό δειγματοληψίας 22kHz.

Θεωρήστε τον οδηγό συσκευής για το υλικό αναπαραγωγής ήχου ("κάρτα ήχου") σε ΛΣ Linux. Από την πλευρά των διεργασιών ισχύουν τα εξής:

1. Μια διεργασία μπορεί να παίξει μουσική ανοίγοντας ειδικό αρχείο συσκευής χαρακτήρων `/dev/audio` και γράφοντας σε αυτό.
2. Δεν επιτρέπονται περισσότερα του ενός ανοιχτά αρχεία από το `/dev/audio`. Προσπάθεια για δεύτερο άνοιγμά του πρέπει να επιστρέφει κωδικό λάθους `EBUSY` (Device or Resource Busy).

Το υλικό προσφέρει μόνο δύο καταχωρητές προσβάσιμους από τον οδηγό:

- Καταχωρητής `SND_HW_READY`, πλάτους `char`: Έχει τιμή `1` όταν η κάρτα ήχου μπορεί να δεχτεί δείγμα PCM προς αναπαραγωγή. Γίνεται `0` για όσο διάστημα η κάρτα ήχου είναι απασχολημένη.
- Καταχωρητής `SND_HW_PCM`, πλάτους `char`: Ο οδηγός γράφει εδώ ένα 8-bit δείγμα PCM προς αναπαραγωγή, αρκεί να ισχύει `SND_HW_READY == 1`.

```
1  struct sound_dev_struct {
2      struct semaphore lock;
3      ...
4  } sound;
5
6  void hw_output_pcm(char val)
7  {
8      while ( !inb(SND_HW_READY))
9          ;
10     outb(val, SND_HW_PCM);
11 }
12
13 void hw_output_buffer(const char *buf, int len)
14 {
15     int i;
16
17     for (i = 0; i < len; i++)
18         hw_output_pcm(buf[i]);
19 }
20
21 int snd_chrdev_open(struct inode *inode, struct file *filp)
22 {
23         filp->private_data = &sound;
24         ...
25 }
26
27 static ssize_t snd_chrdev_write(struct file *filp, const char __user  *usrbuf,
28     size_t cnt, loff_t *f_pos)
29 {
30 #define WRITE_BUF_SZ 3072
31     char tmpbuf[WRITE_BUF_SZ];
32     struct sound_dev_struct *sd = filp->private_data;
33     ...
34     if (down_interruptible(&sd->lock))
35         return -ERESTARTSYS;
36     ...
37     hw_output_buffer(tmpbuf, n);
38     ...
39     up(&sd->lock);
40     ...
41     return cnt;
42 }
```

Σας δίνεται σκελετός αρχικής υλοποίησης του οδηγού, ο οποίος περιέχει:

- `inb(port)`, `outb(val, port)`:
  Μακροεντολές που χρησιμοποιούνται για πρόσβαση στους καταχωρητές υλικού. Κάνουν Ε/Ε από και προς το χώρο διευθύνσεων Ε/Ε ("ports") εκτελώντας απευθείας τις αντίστοιχες ειδικές εντολές του επεξεργαστή, π.χ. `IN`, `OUT` για τον x86.
- `hw_output_pcm(val)`, `hw_output_buffer(buf, len)`:
  Συναρτήσεις οι οποίες χρησιμοποιούν τις `inb()`, `outb()` για να στείλουν στην κάρτα ήχου ένα δείγμα PCM κι έναν ολόκληρο απομονωτή με δείγματα PCM, αντίστοιχα.
- `snd_chrdev_write(...)`:
  Η μέθοδος `write()` του οδηγού συσκευής, η οποία βασίζεται στην `hw_output_buffer()`.

Θεωρήστε ότι ο οδηγός εκτελείται σε σύστημα Linux όπου δεν θα υποστεί ποτέ context switch μια διεργασία όταν βρίσκεται σε κατάσταση πυρήνα (non-preemptible kernel).

Κάντε όποιες επεμβάσεις επιθυμείτε στο σκελετό, αρκεί να τις περιγράψετε με ακρίβεια. Ζητούνται τα εξής:

i. (3%) Υλοποιήστε την `snd_chrdev_open()`. Πώς καλύπτει ο οδηγός σας την προδιαγραφή 2;
ii. (5%) Περιγράψτε σενάριο που οδηγεί στο να έχουν δύο διαφορετικές διεργασίες πρόσβαση σε ανοιχτό αρχείο από το `/dev/audio`, ταυτόχρονα. Πώς εξασφαλίζεται η ασφαλής χρήση του οδηγού σας αν κι οι δύο επιχειρήσουν `write()` και τι θα ακουστεί από τα ηχεία; Γιατί το πεδίο `lock` είναι τύπου `struct semaphore`;
iii. (3%) Σε τι κατάσταση βρίσκεται μια διεργασία ακριβώς όταν εκτελεί κλήση συστήματος `write()` στον οδηγό σας και από ποιες καταστάσεις περνάει μέχρι την επιστροφή της `write()`;
iv. (4%) Πόσο χρόνο χρειάζεται η `hw_output_buffer()` για να στείλει στην κάρτα ήχου έναν απομονωτή μήκους 33kB; Η υλοποίηση της `snd_chrdev_write()` με χρήση της `hw_output_buffer()` είναι ακατάλληλη για χρήση σε πραγματικό σύστημα, για να υποστηρίξει π.χ. διεργασία `mp3player`. Γιατί;

Λόγω ακαταλληλότητας της υλοποίησης, η ομάδα σας αποφασίζει να προχωρήσει σε επέκταση του υλικού ώστε να υποστηρίζει *διακοπές*. Η κάρτα ήχου προκαλεί διακοπή υλικού κάθε φορά που ο καταχωρητής `SND_HW_READY` μεταβαίνει `0` $\longrightarrow$ `1`. Επιπλέον, η σχεδίαση του οδηγού αλλάζει ώστε η υλοποίηση της μεθόδου `write()` να εκτελεί ενδιάμεση αποθήκευση των δεδομένων σε κυκλικό απομονωτή. Τα δεδομένα προωθούνται από τον κυκλικό απομονωτή στην κάρτα ήχου όταν χρειάζεται.

```
1   struct sound_dev_struct {
2       struct semaphore lock;
3       uint128_t wcnt, rcnt;  /* TOTAL number of bytes read/written, from/to the
4                                 circular buffer. They are initialized to zero,
5                                 and will never wrap. */
6   #define CIRC_BUF_SIZE (1024 * 1024)
7       char circ_buffer[CIRC_BUF_SIZE];
8       ...
9   } sound;
10
11  static void snd_hw_intr(void)
12  {
13      struct sound_dev_struct *sd = &sound;
14      ...
15      if (...the circular buffer contains data...)
16          if (inb(SND_HW_READY))
17              hw_output_pcm(...)
18      ...
19  }
20
21  static ssize_t snd_chrdev_write(struct file *filp, const char __user  *usrbuf,
22      size_t cnt, loff_t *f_pos)
23  {
24      ...
25  }
26
```

Σας δίνεται βελτιωμένος σκελετός υλοποίησης του οδηγού, ο οποίος περιέχει συνάρτηση χειρισμού διακοπών υλικού `snd_hw_intr()` και κυκλικό απομονωτή (circular buffer) με χρήση πεδίων `rcnt`, `wcnt`, `circ_buf`.

Ζητούνται τα εξής:

v. (5%) Υλοποιήστε την συνάρτηση χειρισμού διακοπών υλικού `snd_hw_intr()`. Κάντε όσες αλλαγές/προσθήκες χρειάζονται στη δομή `sound_dev_struct` ώστε να καλύπτεται η νέα σχεδίαση.
vi. (10%) Υλοποιήστε πλήρως τη μέθοδο `snd_chrdev_write()`. Στον οδηγό σας, όταν η κάρτα ήχου είναι ανενεργή, ποιο μέρος του προκαλεί την έναρξη της αναπαραγωγής ήχου;
vii. (5%) Σε τι κατάσταση βρίσκεται μια διεργασία ακριβώς όταν εκτελεί κλήση συστήματος `write()` στον οδηγό σας και από ποιες καταστάσεις περνάει μέχρι την επιστροφή της `write()`;
viii. (5%) Με ποιον τρόπο βοηθάει η υποστήριξη διακοπών την καλύτερη λειτουργία του οδηγού σας; Με τι ρυθμό γίνονται διακοπές υλικού; *Πολύ συνοπτικά*, προτείνετε πιθανή αλλαγή στο υλικό και στον οδηγό ώστε να βελτιωθεί κι άλλο η λειτουργία του συστήματος.

Θέμα 2 (35%)

Συμμετέχετε σε ομάδα που σχεδιάζει και υλοποιεί ένα εξειδικευμένο υποσύστημα αποδοτικής επικοινωνίας ανάμεσα σε διεργασίες που εκτελούνται σε διαφορετικές εικονικές μηχανές (VMs), επάνω στο ίδιο φυσικό μηχάνημα. Οι απαιτήσεις αναφέρουν ρητά ότι το μοναδικό σενάριο χρήσης του συστήματος αφορά ακριβώς δύο εικονικές μηχανές (στο ίδιο φυσικό μηχάνημα) και ακριβώς δύο διεργασίες (μία διεργασία σε κάθε εικονικό μηχάνημα), οι οποίες επικοινωνούν μέσω αμφίδρομου, αξιόπιστου ρεύματος χαρακτήρων που παρέχεται από τον μηχανισμό σας.

Η πλατφόρμα εικονικοποίησης που θα χρησιμοποιήσετε είναι QEMU/KVM σε ΛΣ Linux. Στο σύστημα Εισόδου/Εξόδου θα χρησιμοποιήσετε το VirtIO split-driver model (frontend-backend). Οι διεργασίες μέσα στα VMs χρησιμοποιούν κλήσεις `open()`, `read()`, `write()`, `close()` για τη χρήση του μηχανισμού. Οι εικονικές μηχανές QEMU χρησιμοποιούν UNIX domain sockets για διαδιεργασιακή επικοινωνία.

Απαντήστε στα ακόλουθα:

i. (10%) Περιγράψτε συνοπτικά την αρχιτεκτονική που προτείνετε. Σχεδιάστε ένα σχήμα στο οποίο θα φαίνονται τα διαφορετικά κομμάτια του συστήματός σας και η κατανομή τους στο λογισμικό και στο υλικό του φυσικού και των εικονικών μηχανημάτων. Δείξτε την επικοινωνία μεταξύ τους.
ii. (10%) Περιγράψτε με ακρίβεια ένα παράδειγμα χρήσης του συστήματός σας, για την αποστολή ενός μηνύματος (π.χ., "ping") από τη μία διεργασία στην άλλη. Περιγράψτε την αλληλουχία κλήσεων, όλα τα διαφορετικά στρώματα από τα οποία περνάει το μήνυμα και τον τρόπο επικοινωνίας ανάμεσά τους, στο εικονικό και στο

φυσικό μηχάνημα. Πώς ενεργοποιείται το επόμενο τμήμα του μονοπατιού σε κάθε περίπτωση;

iii. (5%) Έστω ότι η διεργασία Α του εικονικού μηχανήματος VM1 στέλνει ένα block δεδομένων στη διεργασία Β του εικονικού μηχανήματος VM2. Πόσες αντιγραφές αυτού του block θα γίνουν σε ολόκληρο το μονοπάτι Α⟶Β και από ποιο μέρος σε ποιο, κάθε φορά;

iv. (5%) Ενώ βρίσκεστε στη φάση παράδοσης και έχετε ήδη υλοποιήσει το σύστημα και τις διεργασίες ελέγχου (test cases), σας ειδοποιούν επειγόντως ότι άλλαξε μια βασική απαίτηση του συστήματος: οι εικονικές μηχανές θα βρίσκονται σε δύο διαφορετικά φυσικά μηχανήματα, τα οποία συνδέονται δικτυακά μεταξύ τους. Χρειάζεται να κάνετε αλλαγές στην αρχιτεκτονική του συστήματός σας και γιατί; Αν ναι, ποιες είναι αυτές και ποια μέρη του συστήματος θα μείνουν ανεπηρέαστα; Τα test cases χρειάζονται αλλαγή και γιατί;

v. (5%) Στην περίπτωση όπου τα VMs είναι σε διαφορετικά φυσικά μηχανήματα και ανταλλάσσονται ευαίσθητα δεδομένα ανάμεσα στις διεργασίες, προκύπτει η ανάγκη για κρυπτογράφησή τους. Σε ποιο/α μέρος/η του συστήματος θα προσθέτατε τη λειτουργικότητα αυτή και πώς επηρεάζονται τα test cases;

Θέμα 3 (25%)

α. (15%) Ένας φίλος σας, σας τηλεφωνεί γιατί το παιχνίδι `pongman` στο σύστημα του έχει "κρεμάσει" (hang), δηλαδή δεν αποκρίνεται πλέον σε είσοδο από το χρήστη (ποντίκι, πληκτρολόγιο). Το σύστημα έχει έναν επεξεργαστή και τρέχει Linux.

Απαντήστε στα εξής:

i. (3%) Με την καθοδήγησή σας, εκτελεί την εντολή `strace -p 1234`, όπου `1234` το PID του `pongman`. Η εντολή δεν παράγει *τίποτε* στην έξοδό της. Τι σημαίνει αυτό για τη διεργασία `1234`; *Υπόδειξη:* Θεωρήστε ότι κατά την εκκίνησή της η `strace` εκτυπώνει την κλήση συστήματος μέσα στην οποία μπορεί να βρίσκεται η διεργασία.

ii. (3%) Σε τι κατάσταση (process state) βρίσκεται η διεργασία `1234`; Πόσο είναι το CPU load του μηχανήματος;

iii. (2%) Δώστε *σύντομο* ενδεικτικό κώδικα σε Assembly που θα μπορούσε να έχει συμπεριφορά ανάλογη με αυτή του `pongman` εκείνη τη στιγμή. Θα μπορούσε η δυσλειτουργία της διεργασίας `1234` να κάνει μη-αποκρίσιμο ολόκληρο το σύστημα; Γιατί;

Μια άλλη μέρα:

iv. (2%) Ο φίλος σας παρατηρεί ότι το `pongman`, ενώ λειτουργεί κανονικά, εμφανίζει μικρές καθυστερήσεις κατά την εκτέλεσή του (lag), γιατί παράλληλα τρέχει ένα πρόγραμμα συμπίεσης σε MP3, μια υπολογιστικά απαιτητική διαδικασία. Γιατί έχει lag το `pongman`;

v. (5%) Για να διορθώσει την κατάσταση, ο φίλος σας αναθέτει τη μέγιστη δυνατή απόλυτη προτεραιότητα χρονοδρομολόγησης στη διεργασία του `pongman`. Τι θα συμβεί αν η διεργασία δυσλειτουργήσει όπως στα (i) – (iii); Τι πρέπει να έχει κάνει ο φίλος σας για να μπορεί να αντιμετωπίσει αυτό το ενδεχόμενο;

β. (10%) Εκτελούμε την εντολή
`ls -l | grep string > file1`
στο φλοιό του UNIX.

Απαντήστε στα εξής:

i. (2%) Σε ποιον περιγραφητή αρχείου γράφει η διεργασία του `ls`; Από ποιον περιγραφητή αρχείου διαβάζει η διεργασία του `grep`;
ii. (3%) Ποια διεργασία ανοίγει το αρχείο `file1` και με ποιες παραμέτρους;
iii. (5%) Μια δική σας διεργασία επιθυμεί να εκτελέσει την εντολή `ls` και να αποκτήσει τη συνολική έξοδό της στη μνήμη της, π.χ., σε απομονωτή `buf[]`. Περιγράψτε την αλληλουχία κλήσεων συστήματος για να το επιτύχει αυτό, χωρίς κανενός είδους ενδιάμεση αποθήκευση στο σύστημα αρχείων. Αν σας εξυπηρετεί, δώστε ψευδοκώδικα.