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 2013-2014

Κανονική Εξέταση

Διάρκεια: 2 ώρες, 30 λεπτά

Η εξέταση πραγματοποιείται με ανοιχτά βιβλία και σημειώσεις.

Θέμα 1 (40%)

Συμμετέχετε σε ομάδα που σχεδιάζει και υλοποιεί το νέο μηχανισμό INTERCOM για διαδιεργασιακή επικοινωνία στο Linux. Ο μηχανισμός προσφέρει μοναδικό μονόδρομο κανάλι δεδομένων, για το οποίο ισχύουν τα εξής:

1. Το κανάλι είναι διαθέσιμο μέσω ειδικού αρχείου συσκευής χαρακτήρων `/dev/intercom`.
2. Το κανάλι έχει δύο άκρα, το άκρο ανάγνωσης (**R**) και το άκρο εγγραφής (**W**).
3. Μια διεργασία αποκτά πρόσβαση στο **R** ανοίγοντας το ειδικό αρχείο μόνο για ανάγνωση, πρόσβαση στο **W** ανοίγοντας το ειδικό αρχείο μόνο για εγγραφή, και πρόσβαση και στα δύο ανοίγοντας το ειδικό αρχείο για ανάγνωση και εγγραφή.
4. Δεδομένα που γράφονται στο **W**, μέσω της κλήσης συστήματος `write()`, παραμένουν στο κανάλι μέχρι να διαβαστούν από το **R**, μέσω της κλήσης συστήματος `read()`. Τα δεδομένα επιστρέφονται με τη σειρά που γράφτηκαν, το κανάλι είναι ένα ρεύμα χαρακτήρων.
5. Το άνοιγμα του ειδικού αρχείου για εγγραφή μπλοκάρει μέχρι να υπάρξουν αναγνώστες, και αντίστροφα.
6. Κλήσεις `read()` μπλοκάρουν, αν το κανάλι δεν περιέχει δεδομένα.
7. Κλήσεις `read()` επιστρέφουν `0` (EOF), αν δεν υπάρχουν πλέον εγγραφείς.
8. Κλήσεις `write()` μπλοκάρουν, αν το κανάλι δεν μπορεί να δεχτεί δεδομένα.

```
1   struct intercom_dev_struct {
2       uint128_t wcnt, rcnt; /* TOTAL number of bytes read/written, from/to the
3                                circular buffer. They are initialized to zero,
4                                and will never wrap. */
5   #define CIRC_BUF_SIZE (4096 * 1024)
6       char circ_buffer[CIRC_BUF_SIZE];
7       ...
8   } intercom;
9
10  int intercom_open(struct inode *inode, struct file *filp)
11  {
12          int m = filp->f_mode;
13          ...
14          filp->private_data = &intercom;
15  }
16
17  static ssize_t intercom_read(struct file *filp, char __user *usrbuf,
18      size_t cnt, loff_t *f_pos)
19  {
20      struct intercom_device *dev = filp->private_data;
21      ...
22      return ret;
23  }
```

Σας δίνεται ο σκελετός του οδηγού που υλοποιεί το `/dev/intercom`. Κάντε όποιες επεμβάσεις επιθυμείτε, αρκεί να τις περιγράψετε με ακρίβεια.

Ζητούνται εξής:

i. (3%) Τι χρειάζεται να κάνουν δύο διεργασίες που εκτελούνται ήδη στο σύστημα για να μπορέσουν να επικοινωνήσουν; Τι σχέση πρέπει να έχουν μεταξύ τους; Πώς γίνεται να δοθεί αποκλειστική πρόσβαση στο κανάλι επικοινωνίας σε συγκεκριμένο χρήστη;
ii. (2%) Δύο διεργασίες μεταφέρουν 1GB δεδομένων πάνω από το κανάλι επικοινωνίας. Ποιες είναι οι απαιτήσεις σε μνήμη RAM ή/και χώρο στο δίσκο από το σύστημα;
iii. (3%) Με ποιον υπάρχοντα μηχανισμό διαδιεργασιακής επικοινωνίας του Linux μοιάζει ο μηχανισμός INTERCOM; Πού διαφέρει ο INTERCOM από τον υπάρχοντα μηχανισμό του Linux; *Υπόδειξη:* Μια διεργασία επιχειρεί να γράψει 32 bytes. Ποιες είναι οι δυνατές τιμές επιστροφής της `write()` στη μία και στην άλλη περίπτωση;
iv. (7%) Υλοποιήστε την `intercom_open()`. Πώς καλύπτονται οι προδιαγραφές 3, 5;
v. (8%) Υλοποιήστε τη `read()`. Περιγράψτε *πολύ* συνοπτικά τη λειτουργία της `write()`, αντίστοιχα με τη `read()` σας, χωρίς να δώσετε κώδικα ή ψευδοκώδικα. Πώς καλύπτει ο οδηγός σας τις προδιαγραφές 4, 6, 7;
vi. (2%) Ποιες οντότητες επιχειρούν ταυτόχρονη πρόσβαση σε δομές του οδηγού και πώς εξασφαλίζεται η ορθή λειτουργία του;

Σημείωση: Θεωρήστε ότι η δομή `struct file` έχει πεδίο `f_mode`, το οποίο παίρνει τιμές που είναι συνδυασμός των bits `FMODE_READ`, `FMODE_WRITE`.

```
1  #define TIMERHZ   10       /* timer() gets called 10 times/sec */
2  #define TOKENMAX  2097152
3  #define RATE      10485760 /* 10 MB/s */
4
5  struct intercom_dev_struct {
6      ...
7      int tokens;     /* Initialized to TOKENMAX */
8  };
9
10 static void timer(void)
11 {
12         struct intercom_dev_struct *dev = &intercom;
13
14         ...
15         dev->tokens += RATE / TIMERHZ;
16         if (dev->tokens >= TOKENMAX)
17             dev->tokens = TOKENMAX;
18         ...
19 }
```

Σε δεύτερη φάση, σας ζητείται η επέκταση του INTERCOM ώστε να έχει τη δυνατότητα *ελέγχου* του ρυθμού με τον οποίο περνάνε δεδομένα από μέσα του σε μόνιμη κατάσταση (rate limiting) ώστε να έχει δεδομένη τιμή, π.χ. `RATE` = 10MB/s. Επιθυμούμε να κάνουμε αλλαγές στο μέρος `read()` του οδηγού, έτσι ώστε να μην επιτρέπει την ανάκτηση δεδομένων από το κανάλι με ρυθμό μεγαλύτερο του `RATE`.

Σας δίνεται η επέκταση στο σκελετό του οδηγού που ακολουθεί. Διακοπές χρονιστή προκαλούν την εκτέλεση της συνάρτησης `timer()` με συχνότητα `TIMERHZ` φορές/sec.

i. (5%) Προτείνετε τρόπο με τον οποίο τα διάφορα μέρη του οδηγού μπορούν να συνεργαστούν ώστε να επιτευχθεί ο ζητούμενος έλεγχος του ρυθμού μεταφοράς. Περιγράψτε την τροποποιημένη συμπεριφορά της `read()`.
ii. (4%) Υλοποιήστε τη συνάρτηση `read()`.
iii. (3%) Υλοποιήστε τη συνάρτηση `timer()`. Ποιες οντότητες επιχειρούν ταυτόχρονη πρόσβαση σε δομές του οδηγού και πώς εξασφαλίζεται η ορθή λειτουργία του;
iv. (3%) Αρκεί η αλλαγή του κώδικα μόνο της `read()` για την επίτευξη του στόχου; Πώς θα φαίνεται το κανάλι από την πλευρά μιας διεργασίας που εκτελεί `write()`;

Θέμα 2 (35%)

Εργάζεστε σε έναν πάροχο υπηρεσιών cloud, προσφέροντας στους πελάτες σας εικονικές μηχανές QEMU. Για λόγους ασφαλείας, οι πελάτες σας έχουν πλήρη διαχειριστική πρόσβαση μέσα στις εικονικές μηχανές, αλλά καμία πρόσβαση στους φυσικούς κόμβους στους οποίους εκτελούνται οι μηχανές. Οι πελάτες σας χρησιμοποιούν τις εικονικές μηχανές για να προσφέρουν με τη σειρά τους δικτυακές υπηρεσίες, π.χ. HTTPS και SSH servers. Οι υπηρεσίες αυτές απαιτούν για τη λειτουργία τους πρόσβαση σε αξιόπιστη ακολουθία τυχαίων αριθμών.

Στο Linux για το σκοπό αυτό υπάρχει το ειδικό αρχείο `/dev/random`, το οποίο μια διεργασία μπορεί να ανοίξει και να διαβάσει τυχαία bytes. Το `/dev/random` συλλέγει εντροπία από

το σύστημα, παρατηρώντας εξωτερικά γεγονότα όπως την κίνηση του ποντικιού και τα χρονικά διαστήματα ανάμεσα σε πατήματα πλήκτρων. Επειδή ο ρυθμός με τον οποίο παράγονται τυχαίοι αριθμοί είναι πολύ μικρός, το πολύ μερικά Mbit/s, συνεχόμενες κλήσεις `read()` στη συσκευή `/dev/random` *μπλοκάρουν* έως ότου το σύστημα έχει συλλέξει αρκετή εντροπία ώστε να μπορεί να συνεχίσει.

Το πρόβλημα με την παραγωγή τυχαίων αριθμών σε εικονικές μηχανές είναι η έλλειψη καλών πηγών εντροπίας, γιατί δεν υπάρχουν οι αντίστοιχες φυσικές συσκευές. Η συσκευή `/dev/random` μπορεί να μιλά και με γεννήτριες τυχαίων αριθμών στο υλικό ("hardware random number generators"). Οι συσκευές αυτές παρακολουθούν φυσικά φαινόμενα (π.χ. θερμικό θόρυβο) για να εξάγουν μια ακολουθία τυχαίων αριθμών. Ως cloud provider, έχετε προνοήσει να τοποθετήσετε σε κάθε φυσικό server από μία τέτοια συσκευή παραγωγής τυχαίων αριθμών σε υλικό.

Για να κάνει διαθέσιμες αυτές τις συσκευές στις εικονικές μηχανές, η ομάδα σας έφτιαξε έναν οδηγό με χρήση του VirtIO split-driver model (frontend/backend). Ο οδηγός αυτός επιτρέπει στις εικονικές μηχανές να χρησιμοποιούν τις συσκευές που είναι συνδεδεμένες στους φυσικούς εξυπηρετητές για να αποκτούν τυχαίους αριθμούς. Το μόνο που χρειάζεται να κάνει μια διεργασία μέσα στην εικονική μηχανή είναι να ανοίξει το ειδικό αρχείο χαρακτήρων `/dev/virtio-random` και να διαβάσει από αυτό. Στο backend, ο οδηγός σας χρησιμοποιεί το αρχείο `/dev/random` του host για να λάβει δεδομένα από τη γεννήτρια τυχαίων αριθμών.

Σας δίνεται σκελετός για το frontend του οδηγού, το οποίο υλοποιεί έναν οδηγό χαρακτήρων στο ΛΣ Linux.

Ζητούνται τα εξής:

i. (2%) Για κάθε μία από τις συναρτήσεις που σας δίνονται, αναφέρετε αν εκτελούνται σε process ή interrupt context.
ii. (5%) Γιατί ο οδηγός χρησιμοποιεί την `copy_to_user()` και όχι απευθείας τη `memcpy()`; Περιγράψτε με ακρίβεια ένα σενάριο όπου ένας κακόβουλος χρήστης εικονικής μηχανής `VM1` θα μπορούσε να εκμεταλλευτεί ενδεχόμενη χρήση της `memcpy()` αντί της `copy_to_user()`, και τις συνέπειες του. Πώς θα επηρεαζόταν η `VM1` και πώς οι υπόλοιπες εικονικές μηχανές στον ίδιο φυσικό κόμβο;
iii. (4%) Πότε χρησιμοποιούνται `spinlocks` και πότε `semaphores` για την προστασία δομών στη μνήμη; Ο οδηγός που σας δίνεται χρησιμοποιεί τους σωστούς μηχανισμούς κλειδώματος; Εξηγήστε την απάντησή σας.
iv. (5%) Σε τι κατάσταση (process state) βρίσκεται μια διεργασία ενώ περιμένει την παραλαβή δεδομένων από την virtqueue; Τι επιπτώσεις έχει αυτό στην απόδοση του εικονικού συστήματος;
v. (10%) Σκιαγραφήστε σχεδίαση του οδηγού στην οποία οι διεργασίες περνάνε σε κατάσταση αναμονής έως ότου έρθουν τα δεδομένα που χρειάζονται. Ποια μέρη του οδηγού επηρεάζονται και με ποιον τρόπο;
vi. (4%) Υπάρχει περίπτωση μια κακόβουλη διεργασία στη `VM1` που διαβάζει συνεχώς από το `/dev/virtio-random` να προκαλέσει πρόβλημα στο υπόλοιπο σύστημα; Πώς θα επηρεαζόντουσαν διεργασίες στη `VM1` και πώς στις υπόλοιπες εικονικές μηχανές;
vii. (5%) Για να προστατευτείτε από αυτό το ενδεχόμενο, επεκτείνετε τον οδηγό ώστε το

frontend να *περιορίζει* το ρυθμό με τον οποίο ζητά τυχαίους αριθμούς από το backend. Αρκεί αυτό ώστε να εξασφαλίσετε τη δίκαιη χρήση των φυσικών γεννητριών τυχαίων αριθμών; Αν όχι, τι πρότεινετε;

```
1  struct rng_device {
2      struct list_head list;      /* Next rng device in the list,
3                                     head is in the rngdrvdata struct */
4      struct virtio_device *vdev; /* The virtio device we are
5                                     associated with */
6      struct virtqueue *vq;
7      struct semaphore vq_sem;    /* Semaphore to protect the virtqueue */
8      unsigned int minor;         /* The minor number of the device */
9  };
10
11 static ssize_t rng_chrdev_read(struct file *filp, char __user *usrbuf,
12     size_t cnt, loff_t *f_pos)
13 {
14     int ret, err, len;
15     struct rng_open_file *rngof = filp->private_data;
16     struct rng_device *rngdev = rngof->rngdev;
17     struct scatterlist syscall_type_sg, read_cnt_sg, buffer_sg, *sgs[3];
18     unsigned int syscall_type = VIRTIO_RNG_SYSCALL_READ;
19     char *kern_buffer;
20     size_t read_cnt;
21
22     if (cnt == 0)
23         return 0;
24 #define MAX_CNT_PER_READ 65536
25     if (cnt > MAX_CNT_PER_READ)
26         cnt = MAX_CNT_PER_READ;
27     read_cnt = cnt;
28
29     kern_buffer = kzalloc(cnt, GFP_KERNEL);
30     if (!kern_buffer) {
31         ret = -ENOMEM;
32         goto out;
33     }
34
35     sg_init_one(&syscall_type_sg, &syscall_type, sizeof(syscall_type));
36     sgs[0] = &syscall_type_sg;
37     sg_init_one(&read_cnt_sg, &read_cnt, sizeof(read_cnt));
38     sgs[1] = &read_cnt_sg;
39     sg_init_one(&buffer_sg, kern_buffer, cnt * sizeof(*kern_buffer));
40     sgs[2] = &buffer_sg;
41
42     down(&rngdev->vq_sem);
43     err = virtqueue_add_sgs(rngdev->vq, sgs, 1, 2,
44         &syscall_type_sg, GFP_ATOMIC);
45     virtqueue_kick(rngdev->vq);
46     while (virtqueue_get_buf(rngdev->vq, &len) == NULL)
47         /* do nothing */;
48     up(&rngdev->vq_sem);
49
50     ret = copy_to_user(usrbuf, kern_buffer, read_cnt);
51     if (ret) {
52         ret = -EFAULT;
53         goto out_with_buf;
54     }
55
```

```
56      ret = read_cnt;
57
58  out_with_buf:
59      kfree(kern_buffer);
60  out:
61      return ret;
62  }
63
64  static void vq_has_data(struct virtqueue *vq)
65  {
66  }
67
68  static struct virtqueue *find_vq(struct virtio_device *vdev)
69  {
70      int err;
71      struct virtqueue *vq;
72
73      vq = virtio_find_single_vq(vdev, vq_has_data, "rng-vq");
74      if (IS_ERR(vq)) {
75          debug("Could not find vq");
76          vq = NULL;
77      }
78
79      return vq;
80  }
81
82  static int virtcons_probe(struct virtio_device *vdev)
83  {
84      int ret = 0;
85      struct rng_device *rngdev;
86
87      rngdev = kzalloc(sizeof(*rngdev), GFP_KERNEL);
88      if (!rngdev) {
89          ret = -ENOMEM;
90          goto out;
91      }
92
93      rngdev->vdev = vdev;
94      vdev->priv = rngdev;
95
96      rngdev->vq = find_vq(vdev);
97      if (!(rngdev->vq)) {
98          ret = -ENXIO;
99          goto out;
100     }
101
102     sema_init(&rngdev->vq_sem, 1);
103
104     /* Grab the next minor number and put the device in the driver's list */
105     spin_lock_irq(&rngdrvdata.lock);
106     rngdev->minor = rngdrvdata.next_minor++;
107     list_add_tail(&rngdev->list, &rngdrvdata.devs);
108     spin_unlock_irq(&rngdrvdata.lock);
109     debug("Got minor = %u", rngdev->minor);
110
111 out:
112     return ret;
113 }
```

Θέμα 3 (25%)

Θεωρήστε έναν περιηγητή ιστού στο Linux, όπως ο Firefox, ο οποίος υποστηρίζει *plugins* γραμμένα από τρίτους για την υποστήριξη διαφορετικών μορφών αρχείου ήχου, π.χ. `mp3`. Θεωρήστε ότι τα plugins διανέμονται ως μοιραζόμενες βιβλιοθήκες και υλοποιούν συναρτήσεις

- `int encode_snd(char *in, int ilen, char *out, int *olen);`
- `int decode_snd(char *in, int ilen, char *out, int *olen);`

Έστω ότι ο χρήστης `user` εγκαθιστά το plugin `p1` για αναπαραγωγή αρχείων ήχου `mp5` στον Firefox. Απαντήστε στα εξής:

i. (2%) Μέσα σε ποια διεργασία εκτελείται ο κώδικας της `decode_snd()` και με τα δικαιώματα ποιου χρήστη;
ii. (2%) Τι θα συμβεί στον Firefox αν ο κώδικας του `p1` έχει κάποιο προγραμματιστικό σφάλμα, π.χ. λόγω bug κάνει προσπέλαση εκτός ορίων κάποιου πίνακα;
iii. (3%) Περιγράψτε πολύ συνοπτικά πώς θα μπορούσε ο Firefox να προστατευτεί από τέτοια δυσλειτουργία.
iv. (3%) Έστω ότι ο κώδικας του `p1` είναι *κακόβουλος*. Περιγράψτε την αλληλουχία κλήσεων συστήματος που μπορεί να εκτελέσει ώστε να αποστείλει δικτυακά το περιεχόμενο του αρχείου `/home/user/secret_passwd` σε επιτιθέμενο.

Θεωρήστε ότι ο πυρήνας του Linux επεκτείνεται ώστε να περιέχει κλήση συστήματος `start_secure()`. Από τη στιγμή που μια διεργασία εκτελέσει αυτή την κλήση, *κάθε* επόμενη κλήση της εκτός των `read()`, `write()`, `exit()` αποτυγχάνει με κωδικό σφάλματος `EPERM` (Permission Denied).

v. (6%) Περιγράψτε αναλυτικά επέκταση του μηχανισμού εκτέλεσης των plugins με χρήση της `start_secure()` έτσι ώστε ο χρήστης να προστατεύεται και από κακόβουλο κώδικα της περίπτωσης (iv). Δώστε ψευδοκώδικα σε C για το βελτιωμένο πλαίσιο εκτέλεσης του plugin. Θεωρήστε ήδη διαθέσιμα τα σύμβολα `encode_snd()`, `decode_snd()`.
vi. (4%) Έστω ένα νήμα του Firefox που πριν καλούσε απευθείας την `decode_snd()`. Τώρα, στο νέο πλαίσιο που ορίσατε, τι πρέπει να κάνει; Πώς μπορεί στο συγκεκριμένο πλαίσιο ο Firefox να αποκωδικοποιεί δύο αρχεία ήχου ταυτόχρονα;
vii. (5%) Έστω ότι ο Firefox έχει ήδη στη μνήμη του πίνακα `mp5data[LEN]` που περιέχει ηχητικά δεδομένα προς αναπαραγωγή. Περιγράψτε βήμα προς βήμα τον τρόπο με τον οποίο ένα νήμα θα χρησιμοποιήσει το plugin στο σενάριό σας, ώστε να μπορέσει να τα αποσυμπιέσει και να τα αναπαραγάγει από τα ηχεία. Θεωρήστε ότι ο Firefox έχει ήδη ανοιχτό περιγραφητή αρχείου `audiofd`. Εκεί, μπορεί να κάνει `write()` ασυμπίεστα δείγματα ήχου PCM για να ακουστούν από τα ηχεία. Οι κλήσεις `write()` μπορεί να μπλοκάρουν.

Μπορείτε να κάνετε οποιαδήποτε λογική παραδοχή για τη λειτουργία των `encode_snd()`, `decode_snd()` χρειάζεστε, αρκεί να την περιγράψετε συγκεκριμένα.