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 2011-2012

Κανονική Εξέταση ακ. έτους 2011-2012

Διάρκεια: 2.5 ώρες

Η εξέταση πραγματοποιείται με ανοιχτά βιβλία και σημειώσεις.

Θέμα 1 (40%)

Το πείραμα ATLAS είναι ένα από τα πειράματα που εκτελούνται στον Μεγάλο Επιταχυντή Αδρονίων, LHC. Καθώς δέσμες σωματιδίων συγκρούονται, ανιχνευτές παράγουν τεράστιο όγκο δεδομένων.

Θεωρήστε πειραματική διάταξη υλικού και αντίστοιχου οδηγού συσκευής στο Linux. Το υλικό αποτελείται από τους ανιχνευτές (`0` έως `detector_cnt - 1`) και μια διάταξη περιοδικής παραγωγής διακοπών υλικού (διακοπές `SAMPLER`). Με κάθε διακοπή `SAMPLER`, ο οδηγός συλλέγει τις μετρήσεις όλων των ανιχνευτών και τις καταγράφει σε δικό του κυκλικό απομονωτή στη μνήμη. Διεργασίες χώρου χρήστη απορροφούν τα δεδομένα αυτά μέσω συσκευής χαρακτήρων `/dev/atlas-device` για περαιτέρω ανάλυση και αρχειοθέτηση.

Μας ενδιαφέρουν μετρήσεις μόνο κατά τη διάρκεια επιλεγμένων *γεγονότων* (events), π.χ. συγκρούσεων με συγκεκριμένα χαρακτηριστικά, οπότε είναι πιθανό να έχει δημιουργηθεί ένα ως τώρα άγνωστο σωματίδιο. Επομένως, ξεκινάμε και σταματάμε την καταγραφή και απορρόφηση των δεδομένων ώστε να συμπίπτουν με τα γεγονότα.

Ένας από τους ανιχνευτές (`trigger_detector_id`) έχει ειδικό ρόλο: κάθε φορά που ανανεώνεται η τιμή του, το υλικό προκαλεί διακοπή `TRIGGER`. Όταν η τιμή αυτή ξεπεράσει κάποιο όριο, ο οδηγός θεωρεί ότι γεγονός είναι σε εξέλιξη, και ξεκινά την καταγραφή κι απορρόφηση. Η καταγραφή συνεχίζεται μέχρι το τέλος του γεγονότος ή την πλήρωση του κυκλικού απομονωτή γιατί οι διεργασίες δεν τον πρόλαβαν.

Από την πλευρά των διεργασιών, κάθε `read()` επιστρέφει bytes για *ακέραιο* αριθμό (όχι κατ'ανάγκη διαδοχικών) μετρήσεων τύπου `struct measurement` από το ρεύμα των εισερχόμενων δεδομένων. Αν δεν υπάρχουν δεδομένα, η `read()` μπλοκάρει.

Το ρεύμα τερματίζεται όταν ο οδηγός σταματήσει την καταγραφή κι ο απομονωτής αδειάσει. Στην περίπτωση αυτή, το αμέσως επόμενο `read()` από οποιονδήποτε επιστρέφει `EOF` και τα υπόλοιπα μπλοκάρουν έως την έναρξη του επόμενου γεγονότος.

Δίνονται τα εξής:

- `get_hw_measurement(dev, detectorid, msr)`:
  Διαβάζει τη μέτρηση του ανιχνευτή `detectorid` της συσκευής `dev` στη δομή `msr`.
- `event_in_progress(triggermsr)`:
  Επιστρέφει αληθές αν η μέτρηση `triggermsr` του ανιχνευτή `trigger_detector_id` δείχνει ότι γεγονός είναι σε εξέλιξη.
- `start_recording(dev)`:
  Ενεργοποιεί διακοπές `SAMPLER` και περιοδική καταγραφή εισερχομένων μετρήσεων στον κυκλικό απομονωτή για τη συσκευή.
- `stop_recording(dev)`:
  Απενεργοποιεί διακοπές `SAMPLER` και σταματά την καταγραφή μετρήσεων.
- `is_recording(dev)`:
  Επιστρέφει αληθές αν γίνεται καταγραφή δεδομένων.
- `store_measurement(dev, msr)`:
  *Αποθηκεύει τη μέτρηση `msr` στον κυκλικό απομονωτή για τη συσκευή `dev`. Επιστρέφει αληθές αν ο απομονωτής έχει γεμίσει (*overflow*).*
- `retrieve_measurement(dev, msr)`:
  Διαβάζει από τον κυκλικό απομονωτή της συσκευής `dev` μια νέα μέτρηση `msr`, επιστρέφει ψευδές αν δεν υπάρχουν έγκυρα δεδομένα.
- `intr(interrupt_mask)`:
  Εξυπηρετεί τις διακοπές υλικού της συσκευής. Στο όρισμα `interrupt_mask` δίνονται οι διακοπές οι οποίες συνέβησαν με τη μορφή αληθών bit.

Όλες οι συναρτήσεις που δέχονται όρισμα τον περιγραφητή της συσκευής `dev`, την κλειδώνουν για να εκτελέσουν τη λειτουργία τους και την απελευθερώνουν.

```
1  struct measurement { ...  }; /* 128 bytes */
2
3  struct atlas_device {
4      ..locktype.. lock;
5      int detector_cnt, detector_trigger_id;
6      int recording;
7      wait_queue_head_t wq;
8      uint128_t wcnt, rcnt; /* These are initialized to zero, and
9                             * will never wrap. */
10     . . .
11 } atlas;
12
13 int store_measurement(struct atlas_device *dev, struct measurement *msr);
14 int retrieve_measurement(struct atlas_device *dev, struct measurement *msr)
15 void get_hw_measurement(struct atlas_device *dev, uint32_t detectorid,
16                         struct measurement *msr);
17 unsigned int get_hw_interrupt_mask(struct atlas_device *dev);
```

```
18  void set_hw_interrupt_mask(struct atlas_device *dev, unsigned int mask);
19  int event_in_progress(struct measurement *triggermsr);
20  int is_recording(struct atlas_device *dev);
21  void start_recording(struct atlas_device *dev);
22
23  void stop_recording(struct atlas_device *dev)
24  {
25      unsigned long flags;
26      unsigned int mask;
27
28      ... lock dev ...
29      dev->recording = 0;
30      . . .
31
32      /* Disable periodic sampling interrupts */
33      mask = get_hw_interrupt_mask(dev);
34      set_hw_interrupt_mask(dev, mask & ~(1 << INTR_SAMPLER_SHIFT));
35      wake_up_interruptible(&dev->wq);
36      ... unlock dev ...
37  }
38
39  void intr(unsigned int mask)
40  {
41      int i, overflow;
42      struct measurement msr;
43      struct atlas_device *dev = &atlas;
44
45      if (mask & (1 << INTR_TRIGGER_SHIFT)) {
46          get_hw_measurement(dev, dev->detector_trigger_id, &msr);
47          if (event_in_progress(&msr) && !is_recording(dev))
48              start_recording(dev);
49          if (!event_in_progress(&msr) && recording(dev))
50              stop_recording(dev);
51      }
52
53      if (mask & (1 << INTR_SAMPLER_SHIFT)) {
54          for (i = 0; i < dev->detector_cnt; i++) {
55              get_hw_measurement(dev, i, &msr);
56              overflow = store_measurement(dev, &msr);
57              if (overflow) {
58                  stop_recording(dev);
59                  return;
60              }
61          }
62          wake_up_interruptible(&dev->wq);
63      }
64  }
65
66  ssize_t atlas_chrdev_read(struct file *filp, char __user *usrbuf,
67      size_t cnt, loff_t *f_pos)
68  {
69      struct atlas_device *dev = filp->private_data;
70
71      /* Only return whole measurements */
72      . . .
73
74      /* Retrieve measurements, block if no data available,
75       * return EOF on event termination or buffer overflow. */
76
77      /* Eventually, i measurements are being returned */
78      return i * sizeof(msr);
79  }
```

Ζητούνται τα εξής:

i. (3%) Ποιος ο τύπος κι ο ρόλος του πεδίου `lock`;
ii. (2%) Ποιος ο ρόλος του πεδίου `wq`;
iii. (25%) Υλοποιήστε την `atlas_chrdev_read()`, συμπληρώνοντας τον σκελετό.
iv. (10%) Υλοποιήστε την `start_recording()`, συμπληρώνοντας τον σκελετό.

Αν το χρειαστείτε, μπορείτε να προσθέσετε νέα πεδία σε δομές, ή νέες συναρτήσεις στον κώδικα, αρκεί να περιγράψετε με ακρίβεια τη λειτουργία τους.

Θέμα 2 (30%)

Θέλουμε να υλοποιήσουμε μια κατάρα Αναγνωριστή [Α], η οποία παρακολουθεί την αλληλουχία των κλήσεων συστήματος μιας διεργασίας. Εάν η αλληλουχία αυτή ανταποκρίνεται σε μία προκαθορισμένη πρότυπη ακολουθία [Π], ενεργοποιεί ("οπλίζει") μια άλλη κατάρα [Κ] για τη διεργασία.

Σε κάθε κλήση συστήματος αντιστοιχίζεται ένα αναγνωριστικό (πχ `read`), και η πρότυπη ακολουθία [Π] σχηματίζεται από τέτοια αναγνωριστικά.

Η κατάρα [Κ] και το πρότυπο [Π] επιλέγονται από το διαχειριστή την ώρα που ενεργοποιεί την κατάρα Αναγνωριστή.

α. (10%) Προτείνετε ένα σχεδιασμό για την κατάρα Α και απαντήστε:

i. Ποιές δομές στην υποδομή κατάρων και στον πυρήνα χρησιμοποιείτε για την κατάρα [Κ] και το πρότυπο [Π];
ii. Πώς επεμβαίνετε στις κλήσεις συστήματος, πώς είναι και πώς καλείται το checkpoint σας;

β. (10%) Θέλουμε να αποκτήσουμε με τη βοήθεια της κατάρας Αναγνωριστή, ένα φλοιό από τον οποίο όλα τα προγράμματα που εκτελούμε δεν θα μπορούν να εκτελούν άλλα προγράμματα.

Περιγράψτε πώς θα χρησιμοποιήσετε την κατάρα Αναγνωριστή [Α], την ακολουθία [Π], και τη λειτουργικότητα της κατάρας [Κ].

γ. (10%) Έστω ένα δικτυακό πρόγραμμα εξυπηρετητή ο οποίος συγχρονίζει την ώρα των πελατών οι οποίοι συνδέονται σε αυτόν με πρωτόκολλο TCP. Το πρόγραμμα αυτό, στην αρχή της εκτέλεσής του ανοίγει μόνο ένα αρχείο με `open()`, αυτό με τη ρύθμιση του σε ποια πόρτα TCP να ακούει. Για λόγους ασφαλείας, θέλουμε να απαγορέψουμε στο πρόγραμμα να ανοίξει οποιοδήποτε άλλο αρχείο μέσω της `open()`.

Περιγράψτε πώς θα χρησιμοποιήσετε την κατάρα Αναγνωριστή [Α], την ακολουθία [Π], και τη λειτουργικότητα της κατάρας [Κ].

Θέμα 3 (30%)

α. (20%) Απαντήστε *συνοπτικά* στα εξής:

i. Ποιος περιγραφητής αρχείου είναι η καθιερωμένη είσοδος για μια διεργασία;
ii. Υπάρχει περίπτωση μια διεργασία να κάνει `read()`, `write()` σε περιγραφητή αρχείου από το δίσκο, χωρίς να έχει κάνει η ίδια το αντίστοιχο `open()`;
iii. Αληθές ή ψευδές; Όταν γίνεται `fork()`, οι file descriptors του παιδιού αντιστοιχίζονται στις ίδιες δομές `struct file` με τους file descriptors του πατέρα.
iv. Τι συμβαίνει σε μια διεργασία όταν εκτελέσει επιτυχώς `execve()`;
v. Τι συμβαίνει στα ανοιχτά αρχεία μιας διεργασίας όταν εκτελέσει επιτυχώς `execve()`;
vi. Λύστε την εξής πρόκληση (εκτελέσιμο `challenge`):

```
1       int main(void)
2       {
3           off_t a, b;
4
5           a = lseek(42, 0, SEEK_CUR); /* Find out current file position */
6           sleep(10);
7           b = lseek(42, 0, SEEK_CUR);
8           if (a == b) FAIL(); else SUCCESS();
9           return 0;
10      }
11
```

Γράψτε πρόγραμμα που να εκτελεί το `challenge`, όπου κι όπως θέλετε, ώστε να επιτυγχάνει. Κάθε κλήση συστήματος του εκτελέσιμου πρέπει να είναι επιτυχής.

β. (10%) Απαντήστε *συνοπτικά*, δικαιολογήστε τις απαντήσεις σας:

i. Σε ποιες από τις παρακάτω δομές εφαρμόζεται τεχνική Copy-on-Write και γιατί;
   - Σώμα Ελέγχου Διεργασίας (`struct task_struct`)
   - Διαπιστευτήρια χρήστη (`struct cred`)
   - Σελίδες που αποτελούν το σωρό (heap) μιας διεργασίας
ii. Θεωρήστε το παρακάτω πρόγραμμα:

```
1           unsigned long t, t0;
2           ...
3
4           c = connect(s, (struct sockaddr *)&c_addr, &c_addr_len);
5           t0 = 0;
6           for (;;) {
7                   t = time(NULL);
8                   if (t > t0) {
9                           write(c, &t, sizeof(t));
10                          t0 = t;
11                  }
12          }
13
```

Τι λειτουργία κάνει το πρόγραμμα; τι προβλήματα βλέπετε στην παραπάνω υλοποίησή της, και πώς θα τα διορθώνατε;
Υπόδειξη: Η `time(NULL)` επιστρέφει τον αριθμό των δευτερολέπτων που έχουν περάσει από τη χρονική στιγμή Epoch, 1970-01-01 00:00:00 +0000 (UTC).