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

Επαναληπτική Εξέταση

Διάρκεια: 2.5 ώρες

Η εξέταση πραγματοποιείται με ανοιχτά βιβλία και σημειώσεις.

Θέμα 1 (40%)

Μια επιστημονική αποστολή χρησιμοποιεί αυτόνομους αισθητήρες για να χαρτογραφήσει ένα ανεξερεύνητο υπόγειο σπήλαιο. Οι (εναέριοι) αισθητήρες κινούνται ανεξάρτητα και χρησιμοποιούν τεχνολογία LIDAR (Light Detection And Ranging) για να συλλέξουν τοπογραφικά δεδομένα για τον περιβάλλοντα χώρο τους: φωτίζουν με παλμούς laser διαφορετικού μήκους κύματος τον περιβάλλοντα χώρο, ώστε να εκτιμήσουν την απόσταση από τα γύρω τους αντικείμενα και άλλες ιδιότητές τους. Κάθε παλμός αντιστοιχεί σε διακριτή μέτρηση. Οι αισθητήρες σχηματίζουν ασύρματο δίκτυο κι αποστέλλουν τεράστιο όγκο δεδομένων σε σταθμό βάσης. Ο σταθμός συνδέεται με υπολογιστικό σύστημα που αναλαμβάνει την επεξεργασία των εισερχόμενων μετρήσεων: για κάθε μήκος κύματος κατασκευάζεται ένας τριδιάστατος χάρτης του σπηλαίου.

Θεωρήστε τον οδηγό συσκευής του σταθμού βάσης σε ΛΣ Linux. Κάθε μέτρηση χαρακτηρίζεται, ανάμεσα στα άλλα, από τον αριθμό του αισθητήρα που την κατέγραψε, το μήκος κύματος του laser που χρησιμοποιήθηκε και τις συντεταγμένες του αισθητήρα. Κάθε φορά που ο σταθμός βάσης παραλαμβάνει μια μέτρηση, προκαλείται διακοπή υλικού. Η συνάρτηση χειρισμού της διακοπής αποθηκεύει την εισερχόμενη μέτρηση σε κυκλικό απομονωτή στη μνήμη του οδηγού, αναλόγως με το μήκος κύματος στο οποίο αντιστοιχεί.

Τα δεδομένα των μετρήσεων γίνονται διαθέσιμα σε διεργασίες προς επεξεργασία μέσω ειδικών αρχείων, π.χ. `/dev/lidar/248nm`, `/dev/lidar/532nm`, `/dev/lidar/1064nm` που υλοποιεί ο οδηγός του σταθμού βάσης ως οδηγός συσκευής χαρακτήρων.

Από την πλευρά των διεργασιών ισχύουν τα εξής:

i. Κάθε `read()` επιστρέφει bytes για *ακέραιο* αριθμό μετρήσεων.
ii. Όσο δεν υπάρχουν νέα δεδομένα μετρήσεων για το συγκεκριμένο μήκος κύματος, η διεργασία που επιχειρεί `read()` μπλοκάρει.
iii. Λόγω του υπολογιστικού φόρτου είναι πιθανό οι διεργασίες να μην μπορούν να ανταπεξέλθουν στο ρυθμό των εισερχομένων μετρήσεων. Ο οδηγός εξασφαλίζει ότι ποτέ δεν θα επιστραφεί στη διεργασία παλαιότερη μέτρηση (χρόνος άφιξης) από μία που της έχει ήδη επιστραφεί προς επεξεργασία.

Δίνονται οι εξής συναρτήσεις:

- `lidar_base_intr()`:
  Συνάρτηση χειρισμού διακοπών του σταθμού βάσης. Αποθηκεύει την εισερχόμενη μέτρηση σε κατάλληλη δομή `lidar_buffer`.
- `get_hw_measurement(msr)`:
  Διαβάζει την εισερχόμενη μέτρηση από το hardware του σταθμού βάσης και την αποθηκεύσει στη δομή `msr`.
- `get_buf_from_wavelength(wavelength)`:
  Επιστρέφει δείκτη προς τη δομή τύπου `lidar_buffer` που αντιστοιχεί στο μήκος κύματος `wavelength`.
- `get_wavelength_from_minor(minor)`:
  Επιστρέφει το μήκος κύματος που αντιστοιχεί σε δεδομένο minor number ενός ειδικού αρχείου του οδηγού της συσκευής.

Ζητούνται τα εξής:

i. (3%) Ποιος ο τύπος κι ο ρόλος του πεδίου `lock` της δομής `lidar_buffer`;
ii. (2%) Ποιος ο ρόλος του πεδίου `wq` της δομής `lidar_buffer`;
iii. (5%) Πώς εξασφαλίζεται η ανεξάρτητη λειτουργία των διαφορετικών ρευμάτων δεδομένων; πώς εξασφαλίζεται ότι μία διεργασία που επεξεργάζεται δεδομένα μετρήσεων στα 1064nm δεν επηρεάζεται από εισερχόμενες μετρήσεις στα 248nm;
iv. (5%) Μια διεργασία εκτελεί `read(fd, ...)`. Από ποιο από τα εισερχόμενα ρεύματα δεδομένων θα προέρχονται οι μετρήσεις που θα διαβάσει; πώς καθορίζεται αυτό; Σχολιάστε τη λειτουργία των `get_wavelength_from_minor()`, `get_buf_from_wavelength()`.
v. (10%) Ποιος ο ρόλος των πεδίων `rcnt`, `wcnt` της δομής `lidar_buffer`; Υλοποιήστε την `lidar_base_intr()`, συμπληρώνοντας τον σκελετό. Πώς ικανοποιείται η προδιαγραφή (iii) από τον κώδικά σας;
vi. (5%) Υλοποιήστε την `lidar_chrdev_open()`, συμπληρώνοντας τον σκελετό.
vii. (10%) Υλοποιήστε την `lidar_chrdev_read()`, συμπληρώνοντας τον σκελετό.

Ο οδηγός στην τελική μορφή του, όπως προκύπτει μετά την υλοποίηση των συναρτήσεων που ζητούνται θα πρέπει να ικανοποιεί τις προδιαγραφές που αναφέρονται παραπάνω.

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

```
1  struct measurement {
2      uint16_t wavelength;
3      ...
4  };
5
6
7  struct lidar_buffer {
8      ..locktype.. lock;
9      uint16_t wavelength;  /* The wavelength corresponding to this buffer */
10     wait_queue_head_t wq;
11     uint128_t wcnt, rcnt; /* These are initialized to zero, and
12                            * will never wrap. */
13 #define CIRC_BUF_SIZE (1024 * 1024)
14     struct measurement circ_buffer[CIRC_BUF_SIZE];
15 };
16
17 void get_hw_measurement(struct measurement *msr);
18 struct lidar_buffer *get_buf_from_wavelength(uint16_t wavelength);
19 uint16_t get_wavelength_from_minor(unsigned int minor);
20
21 void lidar_base_intr(void)
22 {
23     struct lidar_buffer *buf;
24     struct measurement msr;
25
26     get_hw_measurement(&msr);
27     buf = get_buf_from_wavelength(msr->wavelength);
28     ... lock buf ...
29     memcpy(&buf->circ_buffer[buf->wcnt % CIRC_BUF_SIZE],
30         &msr, sizeof(struct measurement));
31     ++buf->wcnt;
32     . . .
33     ...unlock buf...
34     . . .
35 }
36
37 static int lidar_chrdev_open(struct inode *inode, struct file *filp)
38 {
39     unsigned int minor = iminor(inode);
40     . . .
41 }
42
43 ssize_t lidar_chrdev_read(struct file *filp, char __user *usrbuf,
44     size_t cnt, loff_t *f_pos)
45 {
46     struct lidar_buffer *buf = filp->private_data;
47
48     /* Only return whole measurements */
49     . . .
50
51     /* Retrieve measurements, block if no data available */
52     . . .
53
54     /* Eventually, i measurements are being returned */
55     return i * sizeof(msr);
56 }
```

Θέμα 2 (30%)

Μια κρίσιμη εφαρμογή εξυπηρετητή παρουσιάζει απρόβλεπτα προβλήματα στη λειτουργία της. Θα θέλαμε, κατά βούληση, να καταγράφουμε ένα στιγμιότυπο ολόκληρης της εικονικής μνήμης της διεργασίας, ώστε να μπορέσουμε να το μελετήσουμε προς αναζήτηση της αιτίας των προβλημάτων.

Λόγω της κρισιμότητας της εφαρμογής και της φύσης του προβλήματος έχουμε τους εξής περιορισμούς:

i. Η λειτουργία της εφαρμογής δεν πρέπει να σταματήσει.
ii. Η επίδοση της εφαρμογής δεν πρέπει να επιβαρυνθεί σημαντικά (όπως όταν συνδέουμε πάνω της κάποιο εργαλείο παρακολούθησης με `ptrace()`).
iii. Το στιγμιότυπο θέλουμε να είναι όσο το δυνατόν συνεπές, και όχι συνοθύλευμα καταστάσεων διάσπαρτων στο χρόνο.

θα ενσωματώσουμε τη λύση μας σε μία κατάρα Μνημογράφο.

α. (10%) Με ποιον τρόπο ζητάμε την καταγραφή της μνήμης μιας διεργασίας; με ποιον τρόπο γίνεται η καταγραφή και πώς αποκτούμε πρόσβαση στην καταγεγραμμένη μνήμη;

β. (15%) Περιγράψτε τι δομές δεδομένων στον πυρήνα χρησιμοποιεί η κατάρα Μνημογράφος. σε ποια σημεία επεμβαίνει και πώς;

γ. (5%) Περιγράψτε πώς χειρίζεστε την υπόλοιπη κατάσταση της διεργασίας (καταχωρητές, περιγραφητές αρχείων, διαχείριση σημάτων, πολυνηματισμός, μοιραζόμενη μνήμη κ.ά.)

θέμα 3 (30%)

α. (15%) Απαντήστε *συνοπτικά*, δικαιολογήστε τις απαντήσεις σας:

i. (10%) 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. (5%) Αληθές ή ψευδές; Απαντήστε με *σύντομη* αιτιολόγηση.
   - Για την ίδια διεργασία, δύο διαφορετικοί file descriptors αντιστοιχίζονται πάντα σε διαφορετικές δομές `struct file` του πυρήνα.
   - Μια διεργασία σε κατάσταση WAITING εγγυημένα έχει εκτελέσει κλήση συστήματος και βρίσκεται μέσα στον πυρήνα.

β. (15%) Δίνεται το παρακάτω πρόγραμμα.

```
1  #include <...>
2
3  void child(void)
4  {
5      char *newargv[] = { "executable", NULL };
6      char *newenviron[] = { NULL };
7      execve(newargv[0], newargv, newenviron);
8      exit(1);
9  }
10
11 volatile int flag = 0;
12 void hndlr(int signum)
13 { flag = 1; }
14
15 int main(void)
16 {
17     pid_t p;
18
19     signal(SIGCHLD, hndlr);
20
21     p = fork();
22     if (p == 0)
23         child();
24
25     sleep(2);
26     kill(p, SIGTERM);
27     while (!flag)
28         ;
29     if (kill(p, SIGKILL) == -1) printf("SUCCESS!\n"); else printf("FAIL!\n");
30     return 0;
31 }
```

Θεωρήστε ότι οι κλήσεις συστήματος εκτός της `kill()` δεν αποτυγχάνουν.
Απαντήστε *συνοπτικά* στα εξής:

i. (2%) Τι κάνει η κλήση συστήματος `signal()`;
ii. (3%) Υπάρχει περίπτωση το πρόγραμμα να μην τερματίσει ποτέ; Αν ναι, περιγράψτε ένα τέτοιο σενάριο, κάνοντας ό,τι υπόθεση χρειάζεται για τη συμπεριφορά του `executable`.
iii. (5%) Υπάρχει περίπτωση το πρόγραμμα να τερματίσει εκτυπώνοντας `FAIL`; Αν ναι, περιγράψτε ένα τέτοιο σενάριο, κάνοντας ό,τι υπόθεση χρειάζεται για τη συμπεριφορά του `executable`.
iv. (5%) Τι δεν κάνει σωστά αυτό το πρόγραμμα; Ποιες αλλαγές θα κάνατε στον παραπάνω κώδικα ώστε *ανεξάρτητα* από τη συμπεριφορά του `executable` το ισοδύναμο πρόγραμμα να τερματίζει εκτυπώνοντας `SUCCESS`; Δεν μπορείτε να κάνετε *καμία* παραδοχή για το εκτελέσιμο `executable`.