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

Επαναληπτική Εξέταση ακ. έτους 2010-2011

Διάρκεια: 2:30 ώρες

Η εξέταση πραγματοποιείται με ανοιχτά βιβλία και σημειώσεις.

Θέμα 1 (30%)

α. (15%) Απαντήστε τα ακόλουθα ερωτήματα με *σύντομη* αιτιολόγηση, όχι πάνω από 2-3 γραμμές:

i. Εξηγήστε το μηχανισμό που εξασφαλίζει την κληρονομικότητα των κατάρων.
ii. Πώς εξασφαλίζεται ότι αρχικά καμία διεργασία δεν είναι σημειωμένη με οποιαδήποτε κατάρα;
iii. Πώς γίνεται μια διεργασία να είναι σημειωμένη από δημιουργίας της;
iv. Πώς είναι δυνατό μια διεργασία να είναι σημειωμένη αλλά τα παιδιά της να μην είναι;
v. Ποιος θεωρείται κανονικός τρόπος τερματισμού μιας διεργασίας, και ποιος απρόβλεπτος;
vi. Ποια η σημασιολογική διαφορά ενός προγράμματος από μία διεργασία;
vii. Πώς δημιουργείται και πώς τερματίζει μια διεργασία;
viii. Πώς αρχίζει και πότε τερματίζει την εκτέλεσή του ένα εκτελέσιμο πρόγραμμα;
ix. Τι περιβάλλον κληρονομεί μια νέα διεργασία;

β. (15%) Επιθυμούμε να σχεδιάσουμε μια κατάρα-επαναφορέα διεργασιών, που λειτουργεί ως εξής: Όταν μια διεργασία σημειωμένη από δημιουργίας τερματίζει απρόβλεπτα, τότε αυτή «επαναφέρεται» στο σημείο δημιουργίας της, στο ίδιο ακριβώς περιβάλλον.

Προτείνετε μια σχεδίαση για το μηχανισμό αυτό.

i. Σε ποιά σημεία του πυρήνα θα επέμβετε; Εξηγήστε.
ii. Πώς εξασφαλίζετε την επαναφορά της διεργασίας στο ίδιο περιβάλλον;
iii. Τί προβλέπετε στην περίπτωση που η επαναφερμένη διεργασία επίσης τερματίσει απρόβλεπτα;
iv. Τί πρόβλημα δημιουργείται αν μια διεργασία αποτυγχάνει αμέσως και διαρκώς και πώς θα μπορούσατε να το αντιμετωπίσετε;

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. Κώδικας που εκτελείται σε interrupt context μπορεί να καλέσει την `down_interruptible()`.
ii. Ο κώδικας του πυρήνα εκτελείται πάντα με απενεργοποιημένες τις διακοπές.
iii. Η `spin_lock_irqsave()` απενεργοποιεί τις διακοπές στον τοπικό επεξεργαστή.
iv. Μια υλοποίηση της `read()` μπορεί να εκτελέσει την `copy_to_user()` ενώ κρατάει έναν σημαφόρο.
v. Όταν συμβαίνει μια διακοπή υλικού, εκτελείται κώδικας πυρήνα του Λ.Σ.
vi. Όταν γίνεται `fork()`, οι file descriptors του παιδιού αντιστοιχίζονται στις ίδιες δομές `struct file` με τους file descriptors του πατέρα.
vii. Όταν ένα παιδί κάνει `lseek()` σε ανοιχτό αρχείο που έχει κληρονομήσει από το `fork()`, οι κλήσεις `read()` και `write()` του πατέρα του επηρεάζονται.
viii. Όταν ένα παιδί κλείσει έναν περιγραφητή αρχείου που έχει κληρονομήσει από το `fork()`, ο πατέρας δεν μπορεί πλέον να τον χρησιμοποιήσει ("Bad file descriptor").
ix. Όταν ένα νήμα μιας διεργασίας κλείσει ένα ανοιχτό αρχείο, η κλήση από άλλο νήμα της `read()` για τον ίδιο περιγραφητή επιστρέφει δεδομένα μέχρι κι αυτό να καλέσει την `close()`.

β. (25%) Θεωρήστε ένα σενάριο οδηγού συσκευής για εκτυπωτή Braille: Ο εκτυπωτής αποτυπώνει χαρακτήρες κώδικα Braille ως ανάγλυφα επάνω σε ειδική ταινία χαρτιού. Ο εκτυπωτής έχει δύο memory-mapped καταχωρητές:

- `brailleready`: είναι `1` όταν ο εκτυπωτής είναι έτοιμος να δεχτεί χαρακτήρα για εκτύπωση, και `0` όσο είναι απασχολημένος. Στη μετάβαση `0` $\longrightarrow$ `1` ο εκτυπωτής προκαλεί διακοπή υλικού.
- `braillechar`: περιέχει τον χαρακτήρα προς εκτύπωση κάθε φορά. Όταν ο οδηγός γράφει κάτι στον `braillechar`, ο `brailleready` γίνεται `0` κι ο εκτυπωτής ξεκινά να αποτυπώνει τον χαρακτήρα. Η εγγραφή στον `braillechar` κι η μετάβαση `1` $\longrightarrow$ `0` του `brailleready` γίνονται ατομικά.

Ο οδηγός του εκτυπωτή υλοποιεί μια συσκευή χαρακτήρων `/dev/braille`. Για να τυπώσει κάτι στην ταινία, μια διεργασία ανοίγει το ειδικό αρχείο και κάνει `write()`. Παρακάτω σας δίνεται ένας σκελετός του οδηγού συσκευής.

Ο οδηγός διαθέτει έναν κυκλικό απομονωτή, στον οποίο η `write()` αποθηκεύει δεδομένα προς εκτύπωση. Ο απομονωτής είναι πεπερασμένου μεγέθους· κάθε `write()` γράφει σε αυτόν όσα bytes χωράνε κι επιστρέφει ανάλογο αριθμό στον καλούντα. Αν δεν υπάρχει καθόλου χώρος, η `write()` πρέπει να μπλοκάρει, έως ότου ελευθερωθεί χώρος για να γράψει.

Η συνάρτηση `braille_print()` στέλνει έναν χαρακτήρα προς εκτύπωση. Όταν ολοκληρωθεί, ο εκτυπωτής προκαλεί διακοπή υλικού.

```
struct braille_dev {
        /* Memory-mapped device registers */
        volatile int *braillеready;
        volatile int *braillechar;

#define CIRC_BUF_SIZE (1024 * 1024)
#define BYTES_IN_CIRC_BUFFER(rp, wp) (((wp) - (rp) + CIRC_BUF_SIZE) % CIRC_BUF_SIZE)
/* Only write up to CIRC_BUF_SIZE - 1 bytes, to allow for detection of overflow */
#define FREE_IN_CIRC_BUFFER(rp, wp) (CIRC_BUF_SIZE - BYTES_IN_CIRC_BUFFER((rp), (wp)) - 1)

        int rp, wp; /* Αρχικοποιημένα και τα δύο στο 0 */
        char circ_buffer[CIRC_BUF_SIZE];

        ...locktype... lock;
        wait_queue_head_t wq;
} braille_dev;

void braille_print(struct braille_dev *bd)
{
        char c = bd->circ_buffer[bd->rp];
        bd->rp = (bd->rp + 1) % CIRC_BUF_SIZE;
        *bd->braillechar = c;
}

void intr(void)
{
        struct braille_dev *bd = &braille_dev;

        ...lock...
        /* The device completed this character, give it another one */
        if (BYTES_IN_CIRC_BUFFER(bd->rp, bd->wp))
                if (*bd->braillеready)
                        braille_print(bd);
        ...unlock...
        ...
}


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

static int braille_chrdev_open(struct inode *inode, struct file *filp)
{
        filp->private_data = &braille_dev;
        ...
}

static ssize_t braille_chrdev_write(struct file *filp, char __user *usrbuf,
              size_t cnt, loff_t *f_pos)
{
        struct braille_dev *bd = filp->private_data;
        ...
}
```

Ζητούνται τα εξής:

i. (2%) Ποιος ο ρόλος των πεδίων `rp`, `wp` της δομής `braille_dev`;
ii. (3%) Ποιος ο τύπος κι ο ρόλος του πεδίου `lock`;
iii. (2%) Ποιος ο ρόλος του πεδίου `wq`;
iv. (15%) Υλοποιήστε τις `intr()` και `write()`, συμπληρώνοντας τον σκελετό.
v. (3%) Στον οδηγό σας, όταν ο εκτυπωτής είναι ανενεργός, πώς ξεκινά η εκτύπωση;

Θέμα 3 (30%)

α. (10%) Αφού απαντήσετε *συνοπτικά* στα εξής

i. Τι συμβαίνει στα παιδιά μιας διεργασίας αν αυτή δεχτεί `SIGKILL`;
ii. Πότε μια διεργασία γίνεται zombie; Τι πρέπει να συμβεί για να εξαφανιστεί;

λύστε την εξής πρόκληση:

```
void challenge()
{
    pid_t parent;
    parent = getppid(); /* Get Parent PID */
    sleep(10);
    if (getppid() == parent) printf("FAIL");
    else printf("SUCCESS");
}
```

Γράψτε πρόγραμμα που να την καλεί, όπου κι όπως θέλετε, ώστε να τυπώνει "SUCCESS". Δεν επιτρέπεται να αλλάξετε τον κώδικα της `challenge()`, ή των κλήσεων που εκτελεί.

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

i. (5%) Με χρήση της εντολής `ls -l -i mydir1` διαπιστώνετε ότι το FCB για το αρχείο `file1` του `mydir1` είναι το i-node με αριθμό `1234`. Τρέχετε `rm file1`. Τι συμβαίνει στο `file1`, στο `mydir1`, και στο i-node `1234`;
ii. (2%) Αληθές ή ψευδές; Κάθε φορά που μια διεργασία εκτελεί την κλήση συστήματος `open()` στο ειδικό αρχείο `/dev/chardev`, καλείται η μέθοδος `open()` της δομής `struct file_operations` του αντίστοιχου οδηγού συσκευής.
iii. (3%) Αληθές ή ψευδές; κάθε φορά που μια διεργασία εκτελεί `close()` σε περιγραφητή αρχείου που έχει επιστραφεί από την `open()` του προηγούμενου ερωτήματος, καλείται η μέθοδος `release()` της δομής `file_operations`.

γ. (10%) Ένας φίλος σας, σας τηλεφωνεί πανικόβλητος: είχε μείνει από χώρο στο δίσκο κι η εντολή `df` έδειχνε μηδενικό ελεύθερο χώρο. Ζητώντας κατάλογο όλων των αρχείων του δίσκου (π.χ. με `ls`, ή `du`) βρήκε ότι το αρχείο `/var/log/kern.log` είχε φτάσει στα 2TB. Το έσβησε, κι ο δίσκος *συνεχίζει* να αναφέρει μηδενικό ελεύθερο χώρο, ενώ το συνολικό μέγεθος των αρχείων, όπως το βρίσκει η `du` *πράγματι* μειώθηκε κατά 2TB. Τι έχει συμβεί; πού πήγε ο χώρος στο δίσκο και τι πρέπει να κάνει για να τον ανακτήσει; Θεωρήστε ότι το `/var/log/kern.log` ήταν το μοναδικό hard link προς το i-node του.