ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
http://www.cslab.ece.ntua.gr

Λειτουργικά Συστήματα

7ο εξάμηνο, Ακαδημαϊκό Έτος 2013-2014

Κανονική Εξέταση

Διάρκεια: 2.5 ώρες

Οι εξετάσεις θα πραγματοποιηθούν ΧΩΡΙΣ την παρουσία βιβλίων, βοηθημάτων ή άλλου είδους σημειώσεων. Το μόνο που επιτρέπεται να έχετε μαζί σας είναι ένα φύλλο Α4, στο οποίο μπορείτε να έχετε γράψει ό,τι έχετε κρίνει πιο σημαντικό για το μάθημα και θέλετε να το έχετε ως βοήθημά σας. Απαγορεύεται η ανταλλαγή οποιουδήποτε αντικειμένου κατά την ώρα της εξέτασης, ούτε και των φύλλων Α4 που είναι ατομικά.

Θέμα 1 (15%)

Αντιμετωπίζουμε το πρόβλημα συγχρονισμού του νηπιαγωγείου ("Kindergarten"), όπου οι κανονισμοί επιβάλλουν να είναι πάντα παρών ένας δάσκαλος ανά `R` παιδιά, όπου `R` ακέραιος αριθμός (π.χ. αναλογία παιδιών/δασκάλων 3:1), ώστε να εξασφαλίζεται η σωστή επίβλεψη των παιδιών. Επιπλέον, το σχήμα μας επιθυμούμε να έχει τα παρακάτω χαρακτηριστικά:

1. Αν ένας δάσκαλος αποπειραθεί να αποχωρήσει αλλά αυτό δεν είναι δυνατό, θα πρέπει να επιστρέφει στα καθήκοντά του (και να μην "κολλάει" αναμένοντας πότε θα ικανοποιηθεί η συνθήκη εξόδου).
2. Κάθε γονέας θα πρέπει να μπορεί να εισέρχεται στο χώρο του νηπιαγωγείου προκειμένου να ελέγξει αν τηρείται ο κανονισμός.

Δώστε σχήμα συγχρονισμού των διεργασιών `Teacher`, `Child` και `Parent` που ικανοποιεί τις παραπάνω απαιτήσεις ορίζοντας τις συναρτήσεις `*_enter()`, `*_exit()` και `verify_compliance()` που χρησιμοποιούνται στη συνέχεια, ή αντικαθιστώντας τις κλήσεις τους με κατάλληλο κώδικα εντός των `Teacher`, `Child` και `Parent`. Μπορείτε να χρησιμοποιήσετε κλειδώματα, σημαφόρους και μοιραζόμενες μεταβλητές (mutexes/locks, semaphores, shared variables).

```
void Teacher()
{
    for (;;) {
        teacher_enter();
        ...critical section...
        teach();
        teacher_exit();
        go_home();
    }
}
```

```
void Child()
{
    for (;;) {
        child_enter();
        ...critical section...
        learn();
        child_exit();
        go_home();
    }
}
```

```
void Parent()
{
    for (;;) {
        parent_enter();
        ...critical section...
        verify_compliance();
        parent_exit();
        go_home();
    }
}
```

Θέμα 2 (20%)

Δίνεται το πρόγραμμα C σε περιβάλλον UNIX που ακολουθεί μετά τα ζητούμενα.
Θεωρήστε ότι οι κλήσεις συστήματος δεν αποτυγχάνουν, η `Fn()` δεν επιστρέφει ποτέ, η `Fn()` δεν εκτελεί κλήσεις συστήματος, κάθε νέα διεργασία κληρονομεί ακριβώς τον τρόπο χειρισμού σημάτων του πατέρα της και τέλος ότι οι κλήσεις συστήματος διακόπτονται από εισερχόμενα σήματα.
Δεδομένου ότι η `Fn()` δεν επιστρέφει ποτέ, οι διεργασίες που δημιουργεί το πρόγραμμα έρχονται σε *μόνιμη* κατάσταση: το δέντρο διεργασιών μένει σταθερό για πάντα και κάθε διεργασία είναι μέσα σε συγκεκριμένη συνάρτηση ή κλήση συστήματος.

α. (12%) Σχεδιάστε το δέντρο διεργασιών στην *τελική* του μορφή, όταν δηλαδή όλες οι διεργασίες έχουν φτάσει σε *μόνιμη* κατάσταση. Εξηγήστε *συνοπτικά* πώς προκύπτει.

β. (4%) Σε κάθε κόμβο του δέντρου διεργασιών επισημάνετε: (i) την κλήση συστήματος ή συνάρτηση μέσα στην οποία βρίσκεται ο Program Counter της αντίστοιχης διεργασίας, (ii) τα ορίσματα με τα οποία αυτή έχει κληθεί, (iii) τη γραμμή του προγράμματος απ' όπου έγινε η κλήση της.
Για τα ορίσματα, κάντε οποιαδήποτε υπόθεση χρειάζεστε για νούμερα που δεν γνωρίζετε, π.χ. PIDs που ανατίθενται από το σύστημα στις νέες διεργασίες.

γ. (4%) Συμπληρώστε το δέντρο διεργασιών ώστε να φαίνεται η διαδιεργασιακή επικοινωνία: για κάθε μεταφορά δεδομένων ή αποστολή σήματος που συνέβη, σχεδιάστε ένα διακεκομμένο βέλος από τη διεργασία-αποστολέα στη διεργασία-παραλήπτη. Πάνω στο βέλος γράψτε την τιμή που μεταφέρεται κάθε φορά.

```
1  int n, pd[2];
2  char c;
3  pid_t p;
4
5  void h(int s)
6  { write(pd[1], &p, 2); read(pd[0], &p, 2); Fn(pd[0], pd[1]); }
7
8  int main(void)
9  {
10     pipe(pd);
11     signal(SIGUSR1, h);
12     n = pd[1];
13
14     p = fork();
15     if (p == 0) {
16         pipe(pd);
17         pd[1] = n;
18         p = fork();
19     } else
20         p = 0;
21
22     if (p)
23         kill(p, SIGUSR1);
24
25     read(pd[0], &c, 1);
26     Fn(p, pd[0]);
27     wait(&status);
28     Fn(4, 2);
29     return 0;
30 }
```

Θέμα 3 (40%)

α. (20%) Ο κώδικας που σας δίνεται στη συνέχεια βρίσκει τους πρώτους αριθμούς σε δεδομένο διάστημα `[low, high]`. Το πρόγραμμα αυτό εκτελείται από τη διεργασία Α σε σύστημα με έναν επεξεργαστή και ΛΣ που υποστηρίζει εικονική μνήμη με σελιδοποίηση κατ' απαίτηση (demand paging).

```
1   int main(int argc, char *argv[])
2   {
3       long i, j, prime, low, high, counter = 0;
4       ...
5       low = atol(argv[1]);
6       high = atol(argv[2]);
7       ...
8       for (i = low; i <= high; i += 2) {
9           prime = 1;
10          for (j = 3; j < sqrt(i) + 1; j += 2)
11              if (i % j == 0) {
12                  prime = 0;
13                  break;
14          }
15          if (prime) {
16              counter++;
17              printf("%ld is a prime number\n", i);
18          }
19      }
20
21      printf("%ld prime numbers were found in interval [%ld, %ld]\n",
22          counter, low, high);
23
24      return 0;
25  }
```

i. Κατά την εκτέλεση του κυρίως αλγορίθμου (γραμμές 8 – 19) επισημάνετε τα σημεία στον κώδικα στα οποία μπορεί να κληθεί ο χρονοδρομολογητής αν έχουμε διακοπτή (preemptive) ή μη-διακοπτή (non-preemptive) πολιτική χρονοδρομολόγησης. (3%)

ii. Προτείνετε αλλαγές στον κώδικα του προγράμματος ώστε κατά την εκτέλεση των συγκεκριμένων γραμμών σε σύστημα με μη-διακοπτή χρονοδρομολόγηση, το πρόγραμμά σας να μονοπωλήσει τον επεξεργαστή, δηλαδή να ολοκληρωθεί η εκτέλεση των γραμμών χωρίς το πρόγραμμα να αφήσει τον επεξεργαστή. Η έξοδος του προγράμματος θα πρέπει να είναι ακριβώς η ίδια μετά τις αλλαγές που θα κάνετε. (5%)

iii. Πώς αναμένετε να αλλάξει η ταχύτητα εκτέλεσης αν επιτύχετε τη μονοπώληση και πού θα οφείλονται τυχόν αλλαγές; (4%)

iv. Είναι δυνατόν να επιτευχθεί η μονοπώληση για οποιοδήποτε εύρος του διαστήματος `[low, high]`; (4%)

v. Θεωρήστε ότι στο σύστημα υπάρχει άλλη μία διεργασία (έστω Β) έτοιμη προς εκτέλεση. Αν επιτύχετε την παραπάνω μονοπώληση σχολιάστε πώς θα μεταβληθούν για το συγκεκριμένο χρονικό διάστημα της μονοπώλησης οι μετρικές: ρυθμός διεκπεραίωσης (throughput) και χρόνος αναμονής (waiting time) για τη διεργασία Α, τη διεργασία Β και το σύστημα συνολικά. Δικαιολογήστε την απάντησή σας. (4%)

β. (20%) Αγοράσατε σύστημα με πολυπύρηνο επεξεργαστή και θέλετε να εκτελέσετε το πρόγραμμα υπολογισμού των πρώτων αριθμών του προηγούμενου ερωτήματος παράλληλα. Καταστρώστε ενδεικτικό παράλληλο πρόγραμμα:

i. Με πολλαπλές διεργασίες στο μοντέλο του UNIX (`fork()` / `wait()`).
ii. Με πολλαπλά νήματα ακολουθώντας τη λογική των POSIX threads.

Η έξοδος και των δύο εκδόσεων του παράλληλου προγράμματος θα πρέπει να συμφωνεί με αυτή του σειριακού προγράμματος ως προς τα αποτελέσματα που προκύπτουν και όχι απαραίτητα ως προς της σειρά εμφάνισής τους.
ΣΗΜΕΙΩΣΗ: Δεν καλείστε να αναπαράγετε ακριβώς τις κλήσεις συναρτήσεων σε κάθε περίπτωση, αλλά να χρησιμοποιήσετε τους αντίστοιχους μηχανισμούς που σας παρέχονται.

Θέμα 4 (25%)

α. (10%) Έστω ΛΣ με υποστήριξη σελιδοποίησης κατ'απαίτηση (demand paging), μοιραζόμενης μνήμης για διαδιεργασιακή επικοινωνία και CoW για τη δημιουργία νέων διεργασιών.

i. Μια διεργασία εκτελεί `fork()`. Τι παθαίνουν οι εγγραφές του πίνακα σελίδων της;
ii. Περιγράψτε ένα σενάριο όπου αμέσως μετά το `fork()` υπάρχουν ακόμη σελίδες με το bit WRITE αναμμένο στον πίνακα σελίδων της γονικής διεργασίας. Γιατί συμβαίνει αυτό;
iii. Περιγράψτε ένα σενάριο όπου μια διεργασία έχει δικαίωμα εγγραφής σε μεταβλητή `var` χωρίς να είναι αναμμένο το bit WRITE στην αντίστοιχη εγγραφή του πίνακα σελίδων.
iv. Υπάρχει περίπτωση μια διεργασία να έχει δικαίωμα εγγραφής σε μεταβλητή `var` και να μην έχει αναμμένο το bit WRITE στην αντίστοιχη εγγραφή του χάρτη μνήμης;
v. Υπάρχει περίπτωση μια διεργασία να εκτελέσει `open()` σε αρχείο στο δίσκο και να μην πάει σε WAITING κατά την εκτέλεση κλήσεων `read()` από αυτόν τον περιγραφητή;
vi. Υπάρχει περίπτωση μια διεργασία να πάει σε WAITING χωρίς να δεχτεί κάποιο σήμα και χωρίς να εκτελέσει κλήση συστήματος;

β. (5%) Έστω κατάλογος `/u` ιδιοκτησίας `user2` στον οποίο όλοι οι χρήστες έχουν δικαίωμα εγγραφής χωρίς περιορισμούς, αρχείο `/u/prv1` που ανήκει στο χρήστη `user1`, και διεργασία με PID `1234` του `user2`. Οι δύο χρήστες εκτελούν κατά σειρά τις εξής εντολές, με επιτυχία:

- (χρήστης `user1`) `chmod ugo=r /u/prv1`: Όλοι οι χρήστες έχουν πρόσβαση μόνο για ανάγνωση
- (χρήστης `user2`) `ln /u/prv1 /u/prv2`: Δημιουργία νέου hard link `prv2` στο `prv1`

Απαντήστε στα ακόλουθα, με σύντομη αιτιολόγηση:

i. Τι θα γίνει αν η διεργασία `1234` καλέσει `open("/u/prv1", O_WRONLY);`
ii. Τι θα γίνει αν η διεργασία `1234` καλέσει `open("/u/prv2", O_RDONLY);`
iii. Τι δικαιώματα έχει ο `user2` στο `prv1` και στο `prv2`;
iv. Τι θα γίνει αν ο `user2` επιχειρήσει να εκτελέσει `rm prv2`;
v. Τι θα γίνει αν ο `user2` επιχειρήσει να εκτελέσει `rm prv1`;

γ. (10%) Η διεργασία `1234` έχει ανοίξει το `/u/prv2` και εκτελεί συνεχόμενες κλήσεις `read()` για να διαβάσει ευαίσθητα δεδομένα του `user1` και να τα εμφανίσει στην οθόνη.
Ο `user1` επιθυμεί να σταματήσει την πρόσβαση της `1234` στο αρχείο. Ποιες από τις παρακάτω μεθόδους ή συνδυασμός τους θα δουλέψουν και γιατί; (5%)

i. Εκτέλεση της εντολής `chmod go-r prv1`, ώστε να έχει αποκλειστική πρόσβαση στο `prv1`.
ii. Εκτέλεση της εντολής `chmod go-r prv2`, ώστε να έχει αποκλειστική πρόσβαση στο `prv2`.
iii. Εκτέλεση της εντολής `kill -9 1234`.
iv. Διαγραφή των αρχείων `prv1`, `prv2`, με την εντολή `rm prv1 prv2`.

Περιγράψτε τρόπο με τον οποίο ο `user1` μπορεί σίγουρα να σταματήσει τις αναγνώσεις της `1234`. (5%)