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 2012-2013

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 (25%)

Δίνεται το πλήρες 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. (5%) Απαντήστε *συνοπτικά*, όχι πάνω από δύο γραμμές, στα ακόλουθα:

i. Η κλήση `getppid()` επιστρέφει το PID του γονέα της διεργασίας που την καλεί. Υπάρχει περίπτωση να αλλάξει η τιμή επιστροφής της από κλήση σε κλήση για μια διεργασία;
ii. Όταν επιτυγχάνει κλήση `fork()`, τι σχέση έχει η μνήμη που βλέπει η διεργασία-παιδί με τη μνήμη της διεργασίας-πατέρα;
iii. Τι κάνουν οι κλήσεις `kill(pid, SIGINT)` και `signal(SIGINT, handler)` στο UNIX;
iv. Έστω διεργασίες `P0`, `P1` που εκτελούνται ταυτόχρονα. Η `P0` υπολογίζει την ψευδοτυχαία τιμή `rndval` και κάνει `*p = rndval`, όπου `p` δείκτης. Με ποιον μηχανισμό μπορεί η `P1` να διαβάσει αυτή την τιμή ως `*q`, όπου `q` δείκτης στη δική της μνήμη;

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

γ. (4%) Για κάθε κόμβο του δέντρου διεργασιών, γράψτε: (i) την κλήση συστήματος ή συνάρτηση μέσα στην οποία βρίσκεται ο PC 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.

δ. (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  #include <...> /* Θεωρήστε όλες τις απαραίτητες οδηγίες #include */
2
3  void Fn(...); /* Θεωρήστε δεδομένο τον ορισμό της */
4
5  int pg[2];
6  int glob = 1;
7
8  void handler(int signum)
9  { write(pg[1], &glob, sizeof(glob)); Fn(0, -5); }
10
11 int main(void)
12 {
13     int n = 5, i;
14     pid_t p, a;
15
16     glob = 2;
17     pipe(pg);
18     signal(SIGUSR1, handler);
19
20     a = getpid();
21     p = fork();
22     p = fork();
23
24     if (a == getpid()) {
25         while (1)
26             ;
27         glob = 4; write(pg[1], &glob, sizeof(glob));
28         Fn(0, -1);
29     }
30
31     if (p == 0) {
32         if (a == getppid()) {
33             Fn(1, -1);
34             exit(3);
35         }
36         sleep(5);
37         glob = 3;
38         kill(a, SIGUSR1);
39         read(pg[0], &n, sizeof(n));
40
41         for (i = 0; i < n; i++)
42                 if (fork() == 0)
43                     Fn(3, i);
44         Fn(2, -1);
45     }
46     wait(NULL);
47 }
```

Θέμα 2 (25%)

α. (5%) Τι είναι σημαφόρος και ποια είναι η χρησιμότητά του; Αναφέρετε πρόβλημα συγχρονισμού που μπορεί να λύσει ο σημαφόρος και δεν μπορεί το απλό κλείδωμα (lock).

β. (5%) Δώστε τον ορισμό του σημαφόρου κατά Dijkstra. Η υλοποίηση με βάση τον ορισμό αυτό λέμε ότι οδηγεί στο πρόβλημα της "ενεργού αναμονής". Τι είναι το πρόβλημα αυτό και γιατί είναι ανεπιθύμητο;

γ. (5%) Δώστε υλοποίηση του σημαφόρου που αποφεύγει την ενεργό αναμονή. Επισημάνετε σε ποια σημεία έχει περιοριστεί πλέον η αναμονή.

δ. (10%) Υλοποιήστε σχήμα συγχρονισμού που θα προσομοιώνει τη συμπεριφορά πελατών μιας τράπεζας ως εξής: Η τράπεζα έχει K καθίσματα στην αίθουσα αναμονής και ένα γκισέ εξυπηρέτησης. Οι πελάτες της τράπεζας μπορούν να δουν από το παράθυρο αν υπάρχουν ελεύθερα καθίσματα. Αν δεν υπάρχουν, πηγαίνουν έναν περίπατο (`take_a_walk()`) και ξαναπροσπαθούν αργότερα. Αν υπάρχουν, εισέρχονται στην αίθουσα αναμονής και προσπαθούν να εξυπηρετηθούν – ένας κάθε φορά – στο γκισέ. Ο πελάτης εξυπηρετείται καλώντας την `make_transaction()`. Χρησιμοποιήστε μοιραζόμενες μεταβλητές και σημαφόρους για τη λύση σας. Μπορείτε να δουλέψετε κάνοντας όποιες αλλαγές θεωρείτε αναγκαίες στα σημεία που υποδηλώνονται με `. . .` στο τμήμα κώδικα που ακολουθεί:

```
. . .
void bank_client()
{
    while (1) {
        . . .
        if (. . .) { /* if seats available */
            . . .
            make_transaction();
            . . .
            break;
        }
        else {
            . . .
            take_a_walk();
        }
    }
    return_home();
}
```

Θέμα 3 (25%)

α. (5%) Σχεδιάστε ενδεικτικό διάγραμμα καταστάσεων διεργασίας, με τουλάχιστον τις καταστάσεις NEW, RUNNING, WAITING, READY, TERMINATED.

β. (10%) Αναφέρετε μία αιτία για κάθε μετάβαση. Για κάθε μετάβαση, δώστε ενδεικτικό παράδειγμα/τμήμα κώδικα C σε UNIX στο οποίο μπορεί να συμβεί.

γ. (5%) Περιγράψτε τους αλγορίθμους χρονοδρομολόγησης FCFS και SJF. Αναφέρετε πλεονεκτήματα και μειονεκτήματα καθενός από τους δύο.

δ. (5%) Σε τι διαφέρει η διακοπτή από τη μη-διακοπτή χρονοδρομολόγηση; Αναφέρετε ένα πλεονέκτημα κι ένα μειονέκτημα για κάθε μία από τις δύο στρατηγικές. Έστω ένας υπολογιστικός κόμβος, στον οποίο χρήστες υποβάλλουν μαζικά εργασίες προς εκτέλεση. Ποια από τις δύο θα διαλέγατε για το ΛΣ που εκτελείται εκεί;

Θέμα 4 (25%)

α. (10%) Έστω ΛΣ εικονικής μνήμης με σελιδοποίηση που ακολουθεί το μοντέλο `fork()`/`exec()` με Copy-On-Write (COW) για τη δημιουργία νέων διεργασιών. Το μέγεθος σελίδας είναι 4096

bytes. Η λίστα των ελεύθερων πλαισίων είναι `303`, `127`, `309`, `310`, `311`, `312`, `313`, `314`, `...`
Δίνεται ο πίνακας σελίδων της υπό εκτέλεση διεργασίας `P0`:

page	valid	perms	frame
0	i		
1	v	rx	128
2	v	r	301
3	v	rw	129
4	v	rw	126

i. Τι μέγεθος έχει ο εκτελέσιμος κώδικας της διεργασίας, και γιατί;
ii. Ποιες σελίδες αποθηκεύουν *σταθερές*, π.χ. strings, για τη διεργασία; Τι θα γίνει αν επιχειρήσει να γράψει σε αυτές;
iii. Τι page faults θα συμβούν, ποιοι θα είναι οι πίνακες σελίδων για τις `P0`, `P1`, `P2` και τι θα κάνει το ΛΣ όταν οι διεργασίες επιχειρήσουν τα εξής, το ένα μετά το άλλο; Απαντήστε για κάθε βήμα χωριστά.

   1. Η `P0` εκτελεί `fork()` και γεννιέται η `P1`.
   2. Η `P1` διαβάζει από τη διεύθυνση `8512`.
   3. Η `P0` γράφει στη διεύθυνση `12800`.
   4. Η `P1` γράφει στη διεύθυνση `12928`.
   5. Η `P1` εκτελεί `fork()` και γεννιέται η `P2`.
   6. Η `P2` γράφει στη διεύθυνση `4224`.

Περιγράψτε επαρκώς όποιες παραδοχές κάνετε.

β. (5%)

i. Τι είναι ο Τρέχων Κατάλογος (current working directory), για μια διεργασία; Πού κρατάει το ΛΣ την πληροφορία για τον τρέχοντα κατάλογο μιας διεργασίας;
ii. Έστω δύο έργα/tasks `T0`, `T1` στο Linux που εκτελούνται με τρέχοντα κατάλογο τον ριζικό ("`/`"), ο οποίος περιέχει αρχείο `/file1` και άδειο κατάλογο `/dir1`. Το `T0` αλλάζει τον τρέχοντα κατάλογο εκτελώντας `chdir("/dir1")`. Έπειτα, το `T1` εκτελεί `open("file1", ...)`. Τι θα συμβεί αν τα `T0`, `T1` είναι (α) χωριστές διεργασίες, (β) νήματα της ίδιας διεργασίας, (γ) διεργασίες πατέρας-παιδί;

γ. (10%) Σε σύστημα αρχείων το FCB (i-node) περιέχει μετρητή (ακέραιο αριθμό) για την υλοποίηση *μη συμβολικών* συνδέσμων (hard links).
Απαντήστε *συνοπτικά* στα εξής:

i. Τι συμβαίνει στα ανοιχτά αρχεία μιας διεργασίας όταν αυτή εκτελέσει `exit()`, και τι όταν σκοτωθεί από σήμα;
ii. Ο ελεύθερος χώρος στο σύστημα αρχείων που είναι προσαρτημένο στη θέση `/mnt` είναι 3GB. Το `/mnt/data1`, `/mnt/data2` είναι hard links προς το ίδιο αρχείο, το οποίο περιέχει 2GB μη μηδενικά δεδομένα.
   Εξηγήστε πώς μπορεί να συμβεί το εξής σενάριο:

   1. Εκτελούμε `rm /mnt/data1`. Ο ελεύθερος χώρος στο `/mnt` παραμένει αμετάβλητος.
   2. Εκτελούμε `rm /mnt/data2`. Ο ελεύθερος χώρος στο `/mnt` παραμένει αμετάβλητος.
   3. Εκτελούμε κατάλληλη εντολή `kill`. Ο ελεύθερος χώρος στο `/mnt` γίνεται 5GB. Τι συνέβη;

iii. Τι θα συνέβαινε στο παραπάνω σενάριο αν υπήρχε ο συμβολικός δεσμός (soft link) `/mnt/data3` → `/mnt/data2`;