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

Επαναληπτική 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 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:

1. Τι κάνουν οι κλήσεις `kill(pid, SIGSTOP)` και `signal(SIGINT, handler)` στο UNIX;
2. Τι σημαίνει στην κλήση `n = read(fd, buf, cnt)` το όρισμα `cnt`; Τι τιμές μπορεί να έχει η μεταβλητή `n` μετά την επιστροφή της `read()`;
3. Τι κάνει η κλήση `wp = wait(&status)`; Έστω ότι μετά την κλήση της από τη διεργασία με PID `1000` ισχύει `wp == 1011`, `WIFSIGNALED(status) == 1`, `WTERMSIG(status) == 3`. Έστω ότι η `1000` καθόρισε την τύχη της `1011`. Τι συνέβη στη διεργασία `1011` και με ποια κλήση (έστω σε γλώσσα C) την επηρέασε η `1000`;
4. Τι συμβαίνει σε μια διεργασία που καλεί την `read()` σε άδειο pipe όταν είναι ανοιχτό το άκρο εγγραφής;

β. (12%) Σχεδιάστε το δέντρο διεργασιών στην *τελική* του μορφή, όταν δηλαδή όλες οι διεργασίες έχουν φτάσει σε *μόνιμη* κατάσταση. Εξηγήστε *συνοπτικά* πώς προκύπτει. Λύστε πιθανή κατάσταση συναγωνισμού (race) στη γραμμή 18, θεωρώντας ότι *τουλάχιστον* τρεις διεργασίες φτάνουν στο σημείο που σημειώνεται ως `/* A */`, περνώντας το `read()` με τη σειρά γέννησής τους.

γ. (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.

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   int pg[2];
2
3   void handler(int signum)
4   { exit(6); Fn(0, -1); }
5
6   int main(void)
7   {
8       int i, j, n;
9       char c[10];
10      pid_t pid[4];
11
12      signal(SIGUSR1, handler);
13
14      pipe(pg);
15      for (i = 0; i < 4; i++) {
16          pid[i] = fork();
17          if (pid[i] == 0) {
18                  n = read(pg[0], c, i + 2);
19                  /* A */
20                  for (j = 0; j < n; j++) {
21                      pid[i] = fork();
22                      if (pid[i] == 0)
23                          Fn(getpid(), j);
24                  }
25                  for (j = 0; j < i; j++)
26                      wait(NULL);
27                  Fn(i, 12);
28                  exit(2);
29          }
30      }
31
32      kill(pid[i - 2], SIGUSR1);
33      for (i = 0; i < 4; i++) {
34          wait(&n);
35          write(pg[1], c, WEXITSTATUS(n));
36      }
37      Fn(i, -1);
38      return 0;
39  }
```

Θέμα 2 (25%)

α. (5%) Παρακάτω δίνεται η λύση του Peterson για την υλοποίηση κλειδώματος (lock) για δύο διεργασίες στο λογισμικό. Εξηγήστε τι κάνει το συγκεκριμένο τμήμα κώδικα και γιατί η λύση αυτή δεν λειτουργεί στα συστήματα του πραγματικού κόσμου.

```
1  do {
2     flag[me] = TRUE;
3     turn = other;
4     while (flag[other] && turn == other)
5        ;
6        //critical section
7     flag[me] = FALSE;
8     //remainder section
9  } while (TRUE);
```

β. (10%) Σε πολλές περιπτώσεις κατά την εκτέλεση ταυτόχρονων ή παράλληλων προγραμμάτων με πολλές διεργασίες υπάρχει η ανάγκη όλες οι διεργασίες να ξεκινήσουν την εκτέλεση κάποιου τμήματος κώδικα ταυτόχρονα: οι διεργασίες που καταφθάνουν πρώτες σε ένα σημείο του κώδικα περιμένουν όλες τις υπόλοιπες. Όταν όλες έχουν έρθει σε αυτό το σημείο, τότε η εκτέλεση της κάθε μιας συνεχίζεται από την επόμενη εντολή. Το σχήμα συγχρονισμού που υλοποιεί αυτή την απαίτηση ονομάζεται "φράγμα συγχρονισμού" (barrier). Δώστε κώδικα που υλοποιεί το barrier για N διεργασίες, χρησιμοποιώντας σημαφόρους και μία κοινή μεταβλητή.

γ. (10%) Περιγράψτε το πρόβλημα των αναγνωστών-εγγραφέων (readers-writers problem). Δώστε λύση που να αποφεύγει τη λιμοκτονία για τους εγγραφείς.

Θέμα 3 (30%)

α. (5%) Σχεδιάστε ενδεικτικό διάγραμμα καταστάσεων διεργασίας, με τουλάχιστον τις καταστάσεις RUNNING, WAITING, READY, TERMINATED. Αναφέρετε μια αιτία για κάθε μετάβαση.

β. (10%) Σε έναν υπολογιστικό κόμβο οι χρήστες υποβάλουν εργασίες προς εκτέλεση, δεν αλληλεπιδρούν καθόλου με αυτές και στο τέλος συλλέγουν τα αποτελέσματα. Οι διαχειριστές του συγκεκριμενου συστήματος επιθυμούν να ελαχιστοποιήσουν το μέσο χρόνο αναμονής των χρηστών. Προτείνετε αλγόριθμο δρομολόγησης που να επιτυγχάνει την παραπάνω απαίτηση. Τι πληροφορία θα πρέπει να δίνουν οι χρήστες κατά την υποβολή των εργασιών τους ώστε να μπορεί να εκτελεστεί ο αλγόριθμος; Περιγράψτε επέκταση του παραπάνω αλγορίθμου που να αποφεύγει το φαινόμενο της λιμοκτονίας. Δώστε τον αλγόριθμο ή σε ψευδοκώδικα με σχόλια/επεξηγήσεις (κατά προτίμηση) ή σε φυσική γλώσσα.

γ. (15%) Θεωρήστε ένα πρόγραμμα αναπαραγωγής μουσικής από αρχεία MP3 που εκτελείται στο κινητό σας τηλέφωνο. Το πρόγραμμα εκτελείται σε δύο διεργασίες: η P0 αποκωδικοποιεί το αρχείο MP3 σε πακέτα ήχου PCM (συνάρτηση `decode()`), τα οποία αποστέλλει μέσω σωλήνωσης (pipe `pd`) στην P1. Η P1 αναλαμβάνει να στείλει τα πακέτα στην κάρτα ήχου (συνάρτηση `play_sound()`), οπότε ακούγεται μουσική από το ηχείο. Θεωρήστε τα εξής:

- Κάθε τελικό πακέτο ήχου PCM είναι μήκους `L` bytes.
- Η `P1` μπλοκάρει κατά την εκτέλεση της `play_sound()`. Μένει σε αυτή την κατάσταση έως ότου η κάρτα ήχου έχει παίξει ολόκληρο το πακέτο (υποθέτουμε ότι η κάρτα ήχου υποστηρίζει DMA ώστε να μην εμπλέκει τη CPU).
- Ο buffer που χρησιμοποιεί εσωτερικά το ΛΣ για την υλοποίηση του pipe χωράει ακριβώς 3 πακέτα ήχου.
- Η κλήση συστήματος `write()` σε γεμάτο pipe μπλοκάρει.
- Η αποκωδικοποίηση απαιτεί χρόνο C για την παραγωγή ενός πακέτου μήκους L, ενώ η αναπαραγωγή του από το ηχείο απαιτεί χρόνο $P = 5 \times C$.
- Το τηλέφωνο διαθέτει μία CPU και τρέχει μόνο τις δύο διεργασίες. Οι `read()`, `write()` έχουν αμελητέα ανάγκη υπολογισμού και το context switch έχει αμελητέα επιβάρυνση. Το αρχείο MP3 θεωρούμε ότι είναι ήδη φορτωμένο στη μνήμη.

Σας δίνεται ο κώδικας που εκτελούν οι δύο διεργασίες:

```
void P0(int pd[])
{
    int i; char buf[L];

    for (i = 0; i < PACKET_COUNT; i++) {
        decode(FILE, i, buf);
        write(pd[1], buf, L);
    }
}
```

```
void P1(int pd[])
{
    int i; char buf[L];

    for (i = 0; i < PACKET_COUNT; i++) {
        read(pd[0], buf, L);
        play_sound(buf, L);
    }
}
```

Ζητούνται τα εξής:

1. Σε ποια κατάσταση είναι κυρίως η P1; Σε ποια γραμμή βρίσκεται όταν είναι σε αυτή την κατάσταση και πόσο χρόνο μένει εκεί κάθε φορά; Τι συμβαίνει για να φύγει από εκεί;
2. Ποιες καταστάσεις περνά η P0; Γιατί δεν καταλαμβάνει το 100% της CPU;
3. Σχεδιάστε διάγραμμα ως προς το χρόνο όπου να φαίνεται για τις δύο διεργασίες χωριστά σε ποια φάση της εκτέλεσής τους βρίσκονται ανά πάσα στιγμή. Συμβολίστε `DEC(i)`, `PLAY(i)` τις φάσεις αποκωδικοποίησης και αναπαραγωγής του `i`-οστού πακέτου, αντίστοιχα. Χρησιμοποιήστε βέλη για να δείξετε τις εξαρτήσεις ανάμεσα στις δύο φάσεις.
4. Ποιος είναι ο βαθμός χρησιμοποίησης της CPU στη μόνιμη κατάσταση;
5. Για λόγους εξοικονόμησης ενέργειας το κινητό μειώνει στο μισό το ρολόι της CPU, θεωρήστε ότι αυτό μειώνει στο μισό την ταχύτητά της. Ποιος είναι ο νέος βαθμός χρησιμοποίησης της CPU;
6. Στη μέση του κομματιού στέλνουμε `SIGSTOP` στην P0. Θα διακοπεί η αναπαραγωγή; Αν ναι, πότε; Σε ποια γραμμή και σε ποια κατάσταση είναι οι P0, P1 τότε;

Θέμα 4 (20%)

α. (8%) Απαντήστε *συνοπτικά*, όχι πάνω από δύο-τρεις γραμμές:

i. Με ποιον τρόπο βοηθά η υποστήριξη modify bit από το υλικό την υλοποίηση αποδοτικού μηχανισμού για σελιδοποίηση κατ'απαίτηση από το ΛΣ;

ii. Τι θα γινόταν αν μπορούσε μια διεργασία χρήστη να ελέγξει τον χρονιστή (timer) του συστήματος; Πώς αποτρέπεται αυτό το ενδεχόμενο;

β. (12%) Έστω ΛΣ εικονικής μνήμης με σελιδοποίηση που ακολουθεί το μοντέλο `fork()`/`exec()` με COW για τη δημιουργία νέων διεργασιών. Απαντήστε αν οι ακόλουθες προτάσεις είναι αληθείς ή ψευδείς, με *σύντομη* αιτιολόγηση. Περιγράψτε επαρκώς όποιες παραδοχές κάνετε.

1. Μια διεργασία θέτει `var = 5` και κάνει `fork()`. Το παιδί διαβάζει τη μεταβλητή `var` και τη βρίσκει να έχει τιμή `5`, αρκεί ο πατέρας να μην έχει προλάβει να την αλλάξει μετά το `fork()`.
2. Η μεταβλητή `var` βρίσκεται σε άλλη διεύθυνση `&var` στον πατέρα απ'ότι στο παιδί.
3. Η διεύθυνση `&var` μιας μεταβλητής είναι φυσική διεύθυνση.
4. Αμέσως μετά το `fork()`, κάθε εγγραφή της `var` από το παιδί καταλήγει στην ίδια φυσική διεύθυνση με κάθε εγγραφή της `var` από τον πατέρα.
5. Η εντολή `var = 3` στον πατέρα προκαλεί page fault.
6. Η εντολή `var = 3` στο παιδί προκαλεί page fault.
7. Όταν ο Program Counter μιας διεργασίας βρεθεί σε διεύθυνση για την οποία η αντίστοιχη εγγραφή στον πίνακα σελίδων της έχει το permission bit EXECUTE σβηστό, τερματίζεται πάντα με Segmentation Fault.
8. Όταν μια διεργασία επιχειρεί να γράψει σε διεύθυνση για την οποία δεν υπάρχει έγκυρη εγγραφή στον πίνακα σελίδων της, τερματίζεται πάντα με Segmentation Fault.
9. Όταν μια διεργασία επιχειρεί να γράψει σε διεύθυνση για την οποία δεν υπάρχει έγκυρη περιοχή στον χάρτη μνήμης της, τερματίζεται πάντα με Segmentation Fault.