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 2011-2012

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:

i. Τι κάνουν οι κλήσεις `kill(pid, SIGSTOP)` και `signal(SIGINT, handler)` στο UNIX;
ii. 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()`;
iii. Τι κάνει η κλήση `pid = wait(&status)`; Έστω ότι μετά την κλήση της από τη διεργασία με PID `1000` ισχύει `pid == 1002`, `WIFEXITED(status) == 1`, `WEXITSTATUS(status) == 101`. Τι σχέση είχαν οι διεργασίες `1000`, `1002` και ποια ήταν η τελευταία κλήση συστήματος της `1002`;
iv. Τι παθαίνει 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.

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  int pg[2];
2
3  void handler(int signum)
4  { exit(4); Fn(0, -1); }
5
6  int main(void)
7  {
8      int i, j, n;
9      char c[4];
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                 if (i == 1) sleep(5);
19                 n = read(pg[0], c, 2);
20                 for (j = 0; j < i; j++) {
21                     pid[i] = fork();
22                     if (pid[i] == 0)
23                         Fn(getppid(), j);
24                 }
25                 for (j = 0; j < i; j++)
26                     wait(NULL);
27                 exit(2);
28         }
29     }
30
31     kill(pid[0], SIGUSR1);
32     for (i = 0; i < 4; i++) {
33         wait(&n);
34         write(pg[1], c, WEXITSTATUS(n));
35     }
36     Fn(i, -1);
37     return 0;
38 }
```

Θέμα 2 (25%)

α. (10%) Περιγράψτε το πρόβλημα του κρίσιμου τμήματος (critical section) και το πρόβλημα της σειριοποίησης (ordering). Δώστε από ένα παράδειγμα κώδικα στο οποίο εμφανίζονται. Αναφέρετε ένα μηχανισμό που επιλύει το καθένα από αυτά.

β. (5%) Υλοποιήστε σχήμα συγχρονισμού που εξασφαλίζει ότι το πολύ N διεργασίες μπορούν να εκτελούν ένα συγκεκριμένο κομμάτι κώδικα ταυτόχρονα. Μπορείτε να χρησιμοποιήσετε λειτουργίες `signal()` και `wait()` σε κατάλληλα αρχικοποιημένους σημαφόρους κι όποιες μεταβλητές

μοιραζόμενες ανάμεσα στις διεργασίες χρειάζεστε. Δουλέψτε κάνοντας όποιες αλλαγές θεωρείτε αναγκαίες στα σημεία που υποδηλώνονται με . . . στο τμήμα κώδικα που ακολουθεί:

```
. . .
void work()
{
    before_work();
    . . .
    only_up_to_N_processes_do_this_work();
    . . .
    after_work();
}
```

γ. (10%) Υλοποιήστε σχήμα συγχρονισμού που θα προσομοιώνει τη διαδικασία επιβίβασης ταξιδιωτών σε αεροπλάνα, μέσω μιας αίθουσας αναμονής N θέσεων. Συνεχώς καταφτάνουν ταξιδιώτες, καθένας από τους οποίους εκτελεί τη συνάρτηση `traveler()`. Στην αίθουσα αναμονής μπορούν να παραβρίσκονται μέχρι N ταξιδιώτες. Κάθε ταξιδιώτης που μπαίνει στην αίθουσα αναμονής δίνει το διαβατήριό του για έλεγχο (συνάρτηση `pass_passport_check()`). Όταν συμπληρωθούν N ταξιδιώτες που έχουν περάσει τον έλεγχο διαβατηρίων, τότε όλοι μαζί εγκαταλείπουν την αίθουσα και επιβιβάζονται σε λεωφορείο (συνάρτηση `take_bus_to_plane()`), αφήνοντας την αίθουσα άδεια, διαθέσιμη για την επόμενη ομάδα ταξιδιωτών. Μπορείτε να δουλέψετε κάνοντας όποιες αλλαγές θεωρείτε αναγκαίες στα σημεία που υποδηλώνονται με . . . στο τμήμα κώδικα που ακολουθεί:

```
. . .
void traveler()
{
   . . .
   pass_passport_check();
   . . .
   take_bus_to_plane();
}
```

Θέμα 3 (25%)

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

β. (5%) Αναφέρετε μια αιτία για κάθε μετάβαση.

γ. (5%) Η τεχνολογική τάση οδηγεί σε αλγορίθμους διακοπτής χρονοδρομολόγησης που θα είναι σε θέση να διακόπτουν μια διεργασία είτε αυτή είναι σε χώρο χρήστη είτε είναι σε χώρο πυρήνα. Γιατί πιστεύετε ότι συμβαίνει αυτό; Πώς επηρεάζει η παραπάνω τάση το σχεδιασμό του συγχρονισμού;

δ. (10%) Δίνεται το τμήμα προγράμματος C που ακολουθεί μετά το ζητούμενο.
Περιγράψτε ένα σενάριο μετάβασης καταστάσεων της διεργασίας που τρέχει αυτόν τον κώδικα σε ΛΣ εικονικής μνήμης με σελιδοποίηση κατ'απαίτηση που εφαρμόζει (i) διακοπτή (ii) μη-διακοπτή πολιτική χρονοδρομολόγησης, με αναφορά σε συγκεκριμένες γραμμές. Θεωρήστε ότι ο χρήστης έχει δώσει `N` τέτοιο ώστε τα υπολογιστικά μέρη του κώδικα να είναι της τάξης των sec και άνω.

```
1  int main(int argc, char *argv[])
2  {
3      double *array;
4      int N, i, j;
5
6      printf("What is the size of the array?\n");
```

```
7       scanf("%d", &N);
8
9       array = malloc(N * sizeof(double));
10
11      for (i = 0; i < N; i++)
12          array[i] = i;
13
14      for (i = 0; i < N; i++)
15          for (j = 0; j < i; j++)
16              array[i] += sqrt(array[j]);
17
18      return 0;
19  }
```

Θέμα 4 (25%)

α. (5%) Σε σύστημα διαχείρισης της φυσικής μνήμης, περιγράψτε το πρόβλημα του (i) εσωτερικού και του (ii) εξωτερικού κατακερματισμού. Ποιο από τα δύο επιλύει ο μηχανισμός της μετάφρασης με σελιδοποίηση και γιατί;

β. (10%) 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.

i. 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()`.
ii. Αν `fptr = &var` η διεύθυνση της `var` στον πατέρα, `cptr = &var` στο παιδί, τότε `fptr == cptr`.
iii. Αμέσως μετά το `fork()`, κάθε ανάγνωση της `var` από τον πατέρα καταλήγει στην ίδια φυσική διεύθυνση με κάθε ανάγνωση της `var` από το παιδί.
iv. Η εντολή `var = 3` στον πατέρα προκαλεί page fault.
v. Η εντολή `var = 3` στο παιδί προκαλεί page fault.
vi. Όταν μια διεργασία επιχειρεί να γράψει σε διεύθυνση για την οποία η αντίστοιχη εγγραφή στον πίνακα σελίδων της έχει το permission bit WRITE σβηστό, τερματίζεται πάντα με Segmentation Fault.
vii. Όταν μια διεργασία επιχειρεί να γράψει σε διεύθυνση για την οποία δεν υπάρχει έγκυρη εγγραφή στον πίνακα σελίδων της, τερματίζεται πάντα με Segmentation Fault.
viii. Όταν μια διεργασία επιχειρεί να γράψει σε διεύθυνση για την οποία δεν υπάρχει έγκυρη περιοχή στον χάρτη μνήμης της, τερματίζεται πάντα με Segmentation Fault.

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

i. Τι συνεπάγεται η εκτέλεση των παρακάτω λειτουργιών για τις δομές i-node που υπάρχουν στο σύστημα αρχείων και τους μετρητές που περιέχουν;
   - Δημιουργία νέου αρχείου
   - Δημιουργία νέου hard link προς υπάρχον αρχείο
   - Δημιουργία νέου soft link προς υπάρχον αρχείο
ii. Ο ελεύθερος χώρος είναι F. Ο χρήστης εκτελεί `rm data1`, το `data1` περιέχει 2GB δεδομένων. Αν ο ελεύθερος χώρος μετά είναι F', περιγράψτε ένα σενάριο όπου $F' \approx F$, ακόμη και μετά τη διαγραφή.
iii. Δύο συστήματα αρχείων είναι προσαρτημένα (mounted) στα σημεία `/mnt1`, `/mnt2`. Με την εντολή `ls -i` βλέπουμε ότι τα αρχεία `/mnt1/a`, `/mnt2/b` έχουν το ίδιο i-node number. Αυτό σημαίνει ότι είναι hard links στο ίδιο αρχείο. Αληθές ή ψευδές;