ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Β′ Περιόδου 2006

1. (α′) Κάτω από ποιες προϋποθέσεις μπορούμε να δημιουργήσουμε ένα αρχείο/σκληρό σύνδεσμο σ′ ένα αρχείο που ανήκει σε άλλο ιδιοκτήτη; Όταν το κάνουμε αυτό, το αρχείο/σκληρός σύνδεσμος που δημιουργήσαμε έχει σαν ιδιοκτήτη εμάς, ή τον ιδιοκτήτη του αρχικού αρχείου, και γιατί; Πότε μπορεί, αν μπορεί, ο άλλος ιδιοκτήτης να σβήσει το σκληρό σύνδεσμο που δημιουργήσαμε; Αν σβήσει ο άλλος ιδιοκτήτης το αρχικό αρχείο, τότε σβήνει αυτόματα και ο σκληρός σύνδεσμος που δημιουργήσαμε, ή όχι, και γιατί;

   (β′) "*Όταν τερματίσει μία διεργασία, έχει τη δυνατότητα να επιστρέψει έναν ακέραιο κωδικό εξόδου στη διεργασία που την δημιούργησε, και μόνο σ′ αυτήν. Το ίδιο ακριβώς ισχύει και για τα νήματα, δηλαδή όταν τερματίζει ένα νήμα, μπορεί να επιστρέψει έναν ακέραιο κωδικό εξόδου στο νήμα που το δημιούργησε, και μόνο σ′ αυτό. Η μόνη διαφορά είναι ότι πρέπει να επιστραφεί ο ακέραιος αυτός κωδικός προσαρμοσμένος σε δείκτη σε* `void`*. Επίσης, όταν δημιουργείται ένα νήμα, υπάρχει η δυνατότητα να περάσουμε στη συνάρτηση που θα αρχίσει να εκτελεί το νήμα μία παράμετρο. Αυτό όμως είναι αρκετά περιοριστικό, γιατί κάποιες φορές θα θέλαμε να περάσουμε στη συνάρτηση περισσότερες παραμέτρους, οπότε σ′ αυτές τις περιπτώσεις η μόνη μας λύση είναι η χρήση εξωτερικών μεταβλητών.*" Σχολιάστε τα προηγούμενα σε **20 γραμμές το πολύ**.

2. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`rmperc`") το οποίο να διαβάζει από την προκαθορισμένη είσοδο (stdin) γραμμές κειμένου και τις οποίες να μεταφέρει στην προκαθορισμένη έξοδο (stdout), έχοντας κάνει τις εξής τροποποιήσεις. Αν σε κάποια γραμμή της εισόδου υπάρχει ο χαρακτήρας `'%'`, στην έξοδο να μεταφέρεται η γραμμή της εισόδου, έχοντας απαλείψει ό,τι υπάρχει στη γραμμή από τον χαρακτήρα αυτό, συμπεριλαμβανομένου, μέχρι το τέλος της γραμμής. Αν υπάρχουν περισσότερες της μίας εμφανίσεις του `'%'` στη γραμμή, το προηγούμενο εφαρμόζεται για την πρώτη εμφάνισή του, δηλαδή δεν μεταφέρεται στην έξοδο όλο το τμήμα της γραμμής από αυτή την πρώτη εμφάνιση μέχρι το τέλος της. Αν κάνοντας αυτή τη διαγραφή, αυτό που απομένει στη γραμμή είναι μόνο κενοί χαρακτήρες, τότε να μην μεταφερθεί καθόλου η γραμμή στην έξοδο. Όμως, αν στην είσοδο υπάρχουν κενές γραμμές, χωρίς `'%'`, αυτές να μεταφερθούν αυτούσιες στην έξοδο, όπως άλλωστε θα πρέπει να συμβαίνει και για οποιαδήποτε γραμμή της εισόδου δεν περιέχει `'%'`. Πληροφοριακά, ο χαρακτήρας `'%'` χρησιμοποιείται στα προγράμματα της γλώσσας προγραμματισμού Prolog για την ένδειξη έναρξης σχολίων μέχρι το τέλος της γραμμής. Δηλαδή, το πρόγραμμα "`rmperc`" θα μπορεί να χρησιμοποιηθεί σαν ένα φίλτρο "αποσχολιασμού" προγραμμάτων Prolog. Μία ενδεικτική εκτέλεση είναι η εξής:

```
$ cat prog.pl
parent(jim, john). % This is a fact

  parent(john, helen).  % And another fact % and one percent more
   %  These are two rules
grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
mult(X,Y,Z) :- Z is X * Y.    %   Define multiplication
$ ./rmperc < prog.pl
parent(jim, john).

  parent(john, helen).
grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
mult(X,Y,Z) :- Z is X * Y.
$
```

3. Αν το εκτελέσιμο πρόγραμμα που προκύπτει από το πρόγραμμα C στην επόμενη σελίδα ονομάζεται "`ptt`", δώστε ένα πιθανό αποτέλεσμα της εντολής "`ptt 2 3`". Εξηγήστε επίσης, πολύ συνοπτικά, ποια είναι η λειτουργία του προγράμματος.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int m;
void *f(void *argp)
{  pthread_t *thrs;
   int i, retcode, sum = 0, n = (int) argp;
   thrs = malloc(m * sizeof(pthread_t));
   printf("%d %d\n", n, pthread_self());
   if (n--) {
      for (i=0 ; i<m ; i++)
         pthread_create(thrs+i, NULL, f, (void *) n);
      for (i=0 ; i<m ; i++) {
         pthread_join(*(thrs+i), (void **) &retcode);
         sum += retcode; }
      pthread_exit((void *) sum); }
   pthread_exit((void *) pthread_self()); }

main(int argc, char *argv[])
{  pthread_t thr;
   int n, retcode;
   n = atoi(argv[1]);  m = atoi(argv[2]);
   pthread_create(&thr, NULL, f, (void *) n);
   pthread_join(thr, (void **) &retcode);
   printf("%d\n", retcode);
   pthread_exit(NULL); }
```

Ποιο θα είναι το αποτέλεσμα της εκτέλεσης της εντολής "`ptt 3 5 | wc -l`"; Γενικά, τι θα περιμένατε να εκτυπώνεται από την εντολή "`ptt` $\langle n \rangle$ $\langle m \rangle$ `| wc -l`", για οποιαδήποτε $\langle n \rangle$ και $\langle m \rangle$, και γιατί;

4. Γράψτε ένα πρόγραμμα C (υποθέστε ότι ονομάζεται "`countfiles`") το οποίο όταν καλείται με ένα πλήθος από κόμβους (καταλόγους, αρχεία ή άλλου είδους κόμβους) του ιεραρχικού συστήματος αρχείων στη γραμμή εντολής, να μετρά, για καθένα απ' αυτούς τους κόμβους, πόσοι κόμβοι υπάρχουν κάτω απ' αυτόν, συμπεριλαμβανομένου και του ίδιου, καθώς και συνολικά. Αν το πρόγραμμα κληθεί χωρίς ορίσματα, να μετρά τους κόμβους κάτω από τον τρέχοντα κατάλογο. Φυσικά, "κάτω" από ένα κόμβο που δεν είναι κατάλογος υπάρχει ακριβώς ένας κόμβος, ο εαυτός του, ενώ κάτω από ένα κατάλογο που περιέχει δύο αρχεία και ένα κατάλογο, ο οποίος περιέχει τρία αρχεία, υπάρχουν επτά κόμβοι. Για παράδειγμα:

```
$ ./countfiles ~syspro/2005 ~syspro/sp_programs ..
  11 file(s) under /home/users/syspro/2005
  91 file(s) under /home/users/syspro/sp_programs
3919 file(s) under ..
4021 file(s) found in total
$ ./countfiles
  20 file(s) found in total
$ ./countfiles ~syspro/sp_programs/* countfiles*
  83 file(s) under /home/users/syspro/sp_programs/c_progs
   7 file(s) under /home/users/syspro/sp_programs/sh_scripts
   1 file(s) under countfiles
   1 file(s) under countfiles.c
  92 file(s) found in total
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Α΄ Περιόδου 2006

1. (α΄) Παρατηρήστε την παρακάτω διαλογική επικοινωνία με το κέλυφος και συμπληρώστε εντολές ή αποτελέσματα εντολών, όπου υπάρχουν αποσιωπητικά, έτσι ώστε η εικόνα που θα δημιουργήσετε να είναι συνεπής. Σε περιπτώσεις μηνυμάτων λάθους από εντολές, δεν ζητείται η επακριβής διατύπωσή τους, απλώς η ουσία τους. Μπορείτε να συμπεράνετε, από τα δεδομένα που έχετε, πόσοι υποκατάλογοι περιέχονται στον κατάλογο `../syspro/dir2`; Αιτιολογήστε τις απαντήσεις σας.

```
$ whoami
billy
$ ls -ld ../syspro/dir?
drwxr--r--    2 syspro   users       512 Sep  8 12:38 ../syspro/dir1
drwx--x--x    4 syspro   users       512 Sep  8 12:44 ../syspro/dir2
$ ls ../syspro/dir1
file1.txt
$ ls -l ../syspro/dir1/file1.txt
.....................................................
$ ls ../syspro/dir2
.....................................................
$ ls -l ../syspro/dir2/file2.txt
-rw-r--r--    1 syspro   users .....................................
$ cat ../syspro/dir2/file2.txt
A test file
$ ls ../syspro/dir2/*.txt
.....................................................
$ ln ../syspro/dir1/file1.txt .
.....................................................
$ .....................................................
$ ls -l ../syspro/dir2/file2.txt
.........    2 ...................... Sep  8 12:45 ../syspro/dir2/file2.txt
```

   (β΄) "*Όταν μία γονική διεργασία δημιουργεί διεργασίες-παιδιά, δεν είναι δυνατόν να επιτύχουμε επικοινωνία μεταξύ γονέα και παιδιών μέσω εξωτερικών/καθολικών μεταβλητών, γιατί οι μεταβλητές αυτές αντιστοιχίζονται σε διαφορετικές θέσεις μνήμης κατά τη δημιουργία των παιδιών. Είναι δυνατόν όμως να πετύχουμε κάτι τέτοιο αν κάνουμε δυναμική δέσμευση μνήμης στο γονέα (με* `malloc`*), γιατί ο χώρος στον οποίο δεσμεύεται αυτή (σωρός) είναι κοινός για το γονέα και για τα παιδιά. Αντίστροφα, όταν σε μία διεργασία το αρχικό νήμα δημιουργήσει ένα πλήθος από άλλα νήματα, κάθε νήμα έχει διαφορετικό σωρό, οπότε δεν μπορούμε να πετύχουμε επικοινωνία μεταξύ των νημάτων με δυναμική δέσμευση μνήμης. Ενώ, με εξωτερικές/καθολικές μεταβλητές, μπορούμε να έχουμε επικοινωνία μεταξύ των νημάτων, αφού ο χώρος που φυλάσσονται αυτές είναι κοινός για όλα τα νήματα μίας διεργασίας.*" Σχολιάστε τα προηγούμενα σε **5 γραμμές το πολύ**.

2. Γράψτε ένα πρόγραμμα κελύφους Bourne (έστω ότι ονομάζεται "`teams`") που να συμβουλεύεται ένα αρχείο κειμένου, το όνομα-μονοπάτι του οποίου είναι η τιμή της μεταβλητής περιβάλλοντος `RESULTSFILE`, στο οποίο περιέχονται αποτελέσματα ποδοσφαιρικών αγώνων, ένα σε κάθε γραμμή του αρχείου, στη μορφή $\langle teamA\rangle$-$\langle teamB\rangle$:$\langle goalsA\rangle$-$\langle goalsB\rangle$ (τα ονόματα ομάδων δεν περιέχουν - ή :). Το πρόγραμμα που θα γράψετε να υπολογίζει και να εκτυπώνει, για κάθε ομάδα που του δίνεται στη γραμμή εντολής, το πλήθος των αγώνων που έχει δώσει, το σύνολο των βαθμών που έχει πάρει (για κάθε νίκη, μία ομάδα παίρνει 3 βαθμούς, για κάθε ισοπαλία παίρνει 1 βαθμό, ενώ με ήττα δεν παίρνει βαθμούς), καθώς και πόσα τέρματα έχει επιτύχει και πόσα έχει δεχθεί. Θεωρήστε ότι το `RESULTSFILE` περιέχει εγγυημένα μόνο γραμμές στη σωστή τους μορφή. Μία ενδεικτική εκτέλεση φαίνεται στην επόμενη σελίδα.

```
$ RESULTSFILE=results.txt ; export RESULTSFILE
$ cat results.txt
Cote d'Ivoire-Serbia and Montenegro:3-2
Serbia and Montenegro-Netherlands:0-1
Netherlands-Cote d'Ivoire:2-1
Netherlands-Argentina:0-0
$ ./teams Netherlands "Cote d'Ivoire" Brazil
Matches: 3   Points:  7   Goals for: 3   Goals against: 1   Team: Netherlands
Matches: 2   Points:  3   Goals for: 4   Goals against: 4   Team: Cote d'Ivoire
Matches: 0   Points:  0   Goals for: 0   Goals against: 0   Team: Brazil
```

3. Γράψτε ένα πρόγραμμα C (υποθέστε ότι ονομάζεται "**thrseq**") το οποίο να καλείται με έναν αριθμό $\langle n \rangle$ στη γραμμή εντολής. Το αρχικό νήμα της διεργασίας να δημιουργεί ένα νέο νήμα, το οποίο να δημιουργεί ένα άλλο, αυτό ένα τρίτο, και ούτω καθεξής, έως ότου δημιουργηθούν συνολικά $\langle n \rangle$ νήματα, πλην του αρχικού. Κάθε νήμα, πλην του τελευταίου, να περιμένει να τερματίσει το νήμα που δημιούργησε και μετά να τερματίζει και αυτό επιστρέφοντας σαν κωδικό εξόδου το άθροισμα της ταυτότητάς του και του κωδικού εξόδου του νήματος που είχε δημιουργήσει. Το τελευταίο νήμα να επιστρέφει μόνο την ταυτότητά του. Το αρχικό νήμα δεν επιστρέφει κάτι. Μία ενδεικτική εκτέλεση είναι η εξής:

```
$ ./thrseq 3
Level 3 (thread 16384): I created thread with id 16386
Level 2 (thread 16386): I created thread with id 32771
Level 0 (thread 49156): I didn't create any thread
Level 1 (thread 32771): I created thread with id 49156
Level 1 (thread 32771): Thread with id 49156 returned 49156
Level 2 (thread 16386): Thread with id 32771 returned 81927
Level 3 (thread 16384): Thread with id 16386 returned 98313
```

4. Δίνεται στη συνέχεια τμήμα C προγράμματος, το εκτελέσιμο του οποίου ονομάζεται "**prshmem**". Δώστε ένα πιθανό αποτέλεσμα της εντολής "**prshmem** 5". Αιτιολογήστε συνοπτικά την απάντησή σας.

```
main(int argc, char *argv[]) { ..................................
  struct sembuf op[3] = {{0, 0, 0}, {0, 0, 0}, {0, 1, 0}};  is = 0;
  n = atoi(argv[1]);  op[0].sem_op = -n;   op[1].sem_op = n;
  mid = shmget(SHMKEY, n*sizeof(int), PERMS | IPC_CREAT);
  sid = semget(SEMKEY, 1, PERMS | IPC_CREAT);
  reg = shmat(mid, (char *) 0, 0);
  semctl(sid, 0, SETVAL, &is);
  numb = getpid();
  for (i=1 ; i<n ; i++) {
    pid = fork();
    if (!pid) {
      *((int *) reg + i - 1) = numb;
      semop(sid, &op[2], 1);  semop(sid, &op[0], 1);
      printf("%d %d\n", getpid(), *((int *) reg + i));
      semop(sid, &op[1], 1);  exit(0); }
    numb = pid; }
  *((int *) reg + n - 1) = numb;
  semop(sid, &op[2], 1);  semop(sid, &op[0], 1);
  printf("%d %d\n", getpid(), *((int *) reg));
  semop(sid, &op[1], 1);
  ..................................................................... }
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Β′ Περιόδου 2005

1. (α′) Παρατηρήστε την παρακάτω διαλογική επικοινωνία με το κέλυφος και συμπληρώστε τι θα εκτυπωθεί όπου υπάρχουν αποσιωπητικά. Δικαιολογήστε τις απαντήσεις σας. Αν στην τελευταία εντολή, η επιλογή στην "`ls`" ήταν "`-l`" (αντί για "`-al`"), θα παίρνατε το ίδιο αποτέλεσμα; Αν όχι, έχετε να σχολιάσετε κάτι;

```
$ ls -ld *
drwxr-x---   12 spro     users         512 Sep  4 15:13 mydir
-rw-r-----    1 spro     users        7672 Sep  4 15:08 myfile
$ cp myfile file1
$ ls -l file1
-rw-r-----    1 spro     users        7672 Sep  4 15:14 file1
$ ln myfile file2
$ chmod 700 myfile
$ rm myfile
$ ln file2 file3
$ ln -s mydir newdir
$ cp file1 file4
$ ls -l file4
-rw-r-----    1 spro     users        7672 Sep  4 15:14 file4
$ ls -ld *
.....................................................
$ ls -al mydir | grep '^d' | wc -l
.....................................................
$
```

   (β′) Δύο διεργασίες P_1 και P_2 επικοινωνούν μέσω υποδοχών ροής (stream sockets). Η διεργασία P_1 κάνει τρεις κλήσεις εγγραφής στην υποδοχή της S_1, σε απροσδιόριστες φάσεις της εκτέλεσής της, και γράφει, κατά σειρά, τα μηνύματα M_1, M_2 και M_3. Η διεργασία P_2 κάνει τρεις κλήσεις ανάγνωσης από την υποδοχή της S_2, τις R_1, R_2 και R_3, με τη σειρά αυτή, πάλι σε απρόβλεπτες χρονικές στιγμές. Ποια από τα παρακάτω ενδεχόμενα είναι πιθανό να συμβούν και ποια αποκλείονται παντελώς;

   i. Η R_1 θα διαβάσει το μήνυμα M_1, η R_2 το M_2 και η R_3 το M_3.
   ii. Η R_1 θα διαβάσει το μήνυμα M_3, η R_2 το M_1 και η R_3 το M_2.
   iii. Η R_1 θα διαβάσει το μήνυμα M_1M_2 και η R_2 το M_3.
   iv. Η R_2 θα διαβάσει το μήνυμα M_1M_3 και η R_3 το M_2.
   v. Η R_1 θα διαβάσει το μήνυμα $M_1M_2M_3$.
   vi. Η R_1 θα διαβάσει το μήνυμα M_1 και η R_2 το M_2M_3.

   Δικαιολογήστε την απάντησή σας. Υπάρχουν άλλα πιθανά ενδεχόμενα, εκτός από αυτά που αναφέρονται και επιλέξατε; Ποιες θα ήταν οι απαντήσεις σας αν η επικοινωνία μεταξύ των διεργασιών γινόταν μέσω τηλεγραφικών υποδοχών (datagram sockets); Αν γινόταν μέσω σωλήνων;

2. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`contwrd`") το οποίο να διαβάζει από την προκαθορισμένη είσοδο (stdin) ένα κείμενο το οποίο να μεταφέρει στην προκαθορισμένη έξοδο (stdout), έχοντας όμως διαγράψει συνεχόμενες εμφανίσεις ίδιων λέξεων περισσότερο από μία φορά. Κάθε λέξη στην έξοδο να ακολουθείται από τη σήμανση /⟨N⟩, όπου ⟨N⟩ είναι το πλήθος των συνεχόμενων επαναλήψεων της λέξης στην είσοδο. Επιπλέον κενά μεταξύ των λέξεων, στην αρχή ή στο τέλος γραμμών, ή ακόμα και κενές γραμμές, να μην μεταφέρονται στην έξοδο, στην οποία όμως να

διατηρείται η αρχική δόμηση σε γραμμές της εισόδου, φυσικά με την εξαίρεση των λέξεων που έχουν διαγραφεί. Ένα παράδειγμα εκτέλεσης φαίνεται παρακάτω, στο Σχήμα 1.

```
$ cat testf.txt

This   is   a test test
test    test    test


   test file    file     for
  the  contwrd Bourne   Bourne
script.  Have a look  look
  look at  at  at it it it.
$ ./contwrd < testf.txt
This/1 is/1 a/1 test/6
file/2 for/1
the/1 contwrd/1 Bourne/2
script./1 Have/1 a/1 look/3
at/3 it/2 it./1
$
```

Σχήμα 1

```
while (!pthread_mutex_lock(&mtx)) {
   if (is_empty_queue(myqueue)) {
.....................................
      pthread_cond_wait(&cvar, &mtx);
.....................................
      continue;
   }
   item = remove_from_queue(myqueue);
.....................................
   while(needs_processing(item)) {
      newitem = process_item(&item);
.....................................
      insert_in_queue(myqueue, newitem);
.....................................
   }
}
```

Σχήμα 2

3. Σε μία πολυνηματική εφαρμογή, το αρχικό νήμα τοποθετεί στοιχεία προς επεξεργασία σε μία καθολικά προσπελάσιμη ουρά και, στη συνέχεια, δημιουργεί `NWorkers` νήματα-εργάτες, τα οποία διαχειρίζονται την ουρά για να επεξεργασθούν τα δεδομένα που περιέχει. Κάθε εργάτης, όταν δεν έχει δουλειά, εξάγει κάποιο στοιχείο από την ουρά, το επεξεργάζεται και, αναλόγως με το αποτέλεσμα της επεξεργασίας, ενδέχεται να προσθέσει στην ουρά και κάποια νέα στοιχεία για επεξεργασία από εργάτες, πιθανώς και τον ίδιο. Μετά συνεχίζει παίρνοντας νέο στοιχείο από την ουρά για επεξεργασία. Η κοινή προσπάθεια των εργατών τερματίζει όταν δεν υπάρχει, και είναι βέβαιο ότι δεν θα υπάρξει, άλλο στοιχείο στην ουρά για επεξεργασία. Το απόσπασμα κώδικα C που φαίνεται παραπάνω (Σχήμα 2) προέρχεται από τη συνάρτηση των νημάτων-εργατών. Συμπληρώστε το απόσπασμα αυτό, όπου υπάρχουν αποσιωπητικά, για να επιτευχθεί η επιθυμητή συμπεριφορά. Μπορείτε να χρησιμοποιήσετε και επιπλέον, εξωτερικές ή όχι, μεταβλητές, όχι όμως σηματοφόρους ή μεταβλητές συνθήκης, πλην αυτών που περιλαμβάνονται ήδη στο απόσπασμα. Ό,τι συναρτήσεις χρησιμοποιούνται, θεωρήστε ότι έχουν οριστεί κατάλληλα.

4. Γράψτε ένα πρόγραμμα C (υποθέστε ότι ονομάζεται "`sigcyc`") που να καλείται με δύο ορίσματα $\langle N \rangle$ και $\langle M \rangle$ στη γραμμή εντολής. Η διεργασία-γονέας να δημιουργεί αρχικά $\langle N \rangle$ διεργασίες-παιδιά και στη συνέχεια να στέλνει ένα σήμα `SIGUSR1` στο $\langle N \rangle$-οστό παιδί. Αυτό, όταν παραλάβει το σήμα, να στείλει επίσης ένα σήμα `SIGUSR1` στο προηγούμενό του, το $\langle N \rangle - 1$ τάξης, αυτό στο προηγούμενό του και ούτω καθ' εξής. Το 1ο παιδί να στείλει το σήμα στο γονέα τους. Αυτή η κυκλική διάδοση του σήματος να γίνει συνολικά $\langle M \rangle$ φορές. Ένα παράδειγμα εκτέλεσης είναι το εξής:

```
$ ./sigcyc 3 2
Cycle 1: Process 0 (pid=11651) sending signal to 11654
Cycle 1: Process 3 (pid=11654) sending signal to 11653
Cycle 1: Process 2 (pid=11653) sending signal to 11652
Cycle 1: Process 1 (pid=11652) sending signal to 11651
Cycle 2: Process 0 (pid=11651) sending signal to 11654
Cycle 2: Process 3 (pid=11654) sending signal to 11653
Cycle 2: Process 2 (pid=11653) sending signal to 11652
Cycle 2: Process 1 (pid=11652) sending signal to 11651
$
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Α΄ Περιόδου 2005

1. (α΄) Η εντολή "`find` ⟨*dir*⟩ `-name` ⟨*name*⟩ `-print`" ψάχνει στον κατάλογο ⟨*dir*⟩, και αναδρομικά σε όλους τους υποκαταλόγους του, για κόμβους στο ιεραρχικό σύστημα αρχείων που το όνομα τους ταιριάζει με το ⟨*name*⟩ και εκτυπώνει τα ονόματα μονοπάτια των κόμβων που βρίσκει. Παρατηρήστε τη διαλογική επικοινωνία με το κέλυφος στο Σχήμα 1 και βρείτε τί θα εκτυπωθεί όπου υπάρχουν αποσιωπητικά. Δικαιολογήστε στοιχειωδώς την απάντησή σας.

   (β΄) Ποια ήταν, κατά τη γνώμη σας, η πρόθεση του προγραμματιστή που έγραψε το πρόγραμμα C στο Σχήμα 2; Δώστε μία πιθανή εκτύπωση του αντίστοιχου εκτελέσιμου προγράμματος (αν όλες οι κλήσεις συστήματος λειτουργήσουν χωρίς πρόβλημα). Υπάρχει κάποιο πολύ σοβαρό σχεδιαστικό λάθος στο πρόγραμμα αυτό; Αν ναι, εξηγήστε ποιο είναι και προτείνετε πολύ σύντομα τουλάχιστον δύο τρόπους με τους οποίους θα μπορούσε να διορθωθεί το πρόβλημα αυτό.

```
$ ls . dir
.:
dir  exfind.c

dir:
a_file.c  exfind.c
$ find . -name '*.c' -print
.............................
$ cat exfind.c
#include <stdio.h>
main()
{ execlp("find", "find", ".",
  "-name", "*.c", "-print", NULL);
   perror("execlp"); }
$ cc -o exfind exfind.c
$ find . -name *.c -print
.............................
$ ./exfind
.............................
```

Σχήμα 1

```
#include <stdio.h>
#include <signal.h>
int p[4] = {0, 0, 0, 0};
void handler(int sig)
{  printf("%d: %d %d %d %d\n",
          getpid(), p[0], p[1],
          p[2], p[3]); }
main()
{  int i, pid, st;
   signal(SIGUSR1, handler);
   for (i=0 ; i<4 ; i++) {
      pid = fork();
      if (!pid) { pause(); exit(0); }
      p[i] = pid; }
   for (i=0 ; i<4 ; i++)
      kill(p[i], SIGUSR1);
   kill(getpid(), SIGUSR1);
   for (i=0 ; i<4 ; i++)
      wait(&st); }
```

Σχήμα 2

2. Ένα κείμενο σε φυσική γλώσσα είναι αποδεκτό από πλευράς εμφάνισης όταν υπάρχει ακριβώς ένα κενό μεταξύ των λέξεων και όταν δεν υπάρχουν κενά πριν από τα σημεία στίξης (τελείες, κόμματα, κλπ.). Επίσης, όταν ένα σημείο στίξης ακολουθείται από λέξη, πρέπει να υπάρχει ακριβώς ένα κενό μεταξύ αυτού και της λέξης που ακολουθεί. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`punct`") το οποίο να διαβάζει από την προκαθορισμένη είσοδο (stdin) ένα αρχείο κειμένου και, εφαρμόζοντας το προηγούμενο "αισθητικό" αξίωμα, να το ξαναγράφει στην προκαθορισμένη έξοδο (stdout) με κομψό τρόπο. Υποθέστε ότι όλα τα πιθανά σημεία στίξης είναι καταχωρημένα στη μεταβλητή περιβάλλοντος `PMARKS`. Ένα παράδειγμα εκτέλεσης είναι το ακόλουθο:

```
$ PMARKS='.,?!' ; export PMARKS
$ cat inp_file
Oh , my   God ! !! What
  an ugly file , isn't
it?  Yes  ,it is.
```

```
$ ./punct < inp_file
Oh, my God!!! What
an ugly file, isn't
it? Yes, it is.
$
```

3. Ποια ήταν, κατά τη γνώμη σας, η πρόθεση του προγραμματιστή που έγραψε το παρακάτω πρόγραμμα C; Παρότι το πρόγραμμα αυτό μεταγλωττίζεται με επιτυχία, έχει διάφορα σοβαρά λογικά λάθη. Γράψτε μία σωστή εκδοχή του, με βάση την απάντησή σας στην αρχική ερώτηση.

```
#include <stdio.h>
#include <pthread.h>
int i, j, count = 0;  pthread_t ker_id, pi_id;
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cvar = PTHREAD_COND_INITIALIZER;
void *kernighan(void *argp)
{  printf("I am Kernighan version %d\n", *(int *) argp);
   for (j=0 ; j<30000 ; j++) { pthread_mutex_lock(&mtx);  count++;
      if (count == 25000) {
         pthread_cond_signal(&cvar);  printf("%d: Signaled\n", ker_id); }
      pthread_mutex_unlock(&mtx); } }
void *pike(void *argp)
{  printf("I am Pike the only one\n");
   pthread_cond_wait(&cvar, &mtx);  printf("%d: Woke up\n", pi_id); }
main()
{  for (i=0 ; i<5 ; i++) pthread_create(&ker_id, NULL, kernighan, (void *) &i);
   pthread_create(&pi_id, NULL, pike, NULL);
   for (i=0 ; i<5 ; i++) pthread_join(ker_id, NULL);
   pthread_join(pi_id, NULL); }
```

4. Γράψτε δύο προγράμματα C (έστω ότι ονομάζονται "`filein`" και "`fileout`") τέτοια ώστε το πρώτο να διαβάζει την προκαθορισμένη είσοδό του (stdin) και να την στέλνει στο δεύτερο μέσω μίας ουράς μηνυμάτων. Το πρόγραμμα "`fileout`" να στέλνει ό,τι διαβάζει από την ουρά μηνυμάτων στην προκαθορισμένη έξοδό του (stdout). Το κλειδί που θα αντιστοιχεί στην κοινή ουρά να είναι ο inode ενός αρχείου του οποίου το όνομα δίνεται σαν όρισμα και στα δύο προγράμματα. Πληροφοριακά, η κλήση συστήματος "`msgrcv`" επιστρέφει στο όνομά της, σε περίπτωση επιτυχούς εκτέλεσης, το πραγματικό μέγεθος του μηνύματος που διάβασε. Ενδεικτικές εκτελέσεις των δύο προγραμμάτων είναι οι εξής:

```
$ wc -c test_file
    104 test_file
$ cat test_file
This is a text file to test the
filein/fileout cooperation. Binary
files should be transmitted OK, too.
$ ./filein /etc/services < test_file
/etc/services inode is 11166
Referencing msgq with id 360448
Transferring data... Done!
Sent 104 bytes
$ wc -c filein
   6254 filein
$ ./filein /etc/fstab < filein
/etc/fstab inode is 4210
Referencing msgq with id 393217
Transferring data... Done!
Sent 6254 bytes
$
```

```
$ ./fileout /etc/services > outp_file
/etc/services inode is 11166
Referencing msgq with id 360448
Transferring data... Done!
Received 104 bytes
Removed msgq with id 360448
$ ./fileout /etc/fstab > new_filein
/etc/fstab inode is 4210
Referencing msgq with id 393217
Transferring data... Done!
Received 6254 bytes
Removed msgq with id 393217
$ wc -c outp_file
    104 outp_file
$ wc -c new_filein
   6254 new_filein
$ diff test_file outp_file
$ cmp filein new_filein
$
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Β′ Περιόδου 2004

1. (α′) Παρατηρήστε την παρακάτω διαλογική επικοινωνία με το κέλυφος:

```
$ ls -l
total 10
-rw-r--r--    1 spro     users          62 Jul 22 15:02 >!
-rwxr-xr-x    1 spro     users        7921 Sep  5 14:05 a_file
-rw-r--r--    1 spro     users         100 Sep 28 09:58 redir.c
$ cat redir.c
#include <stdio.h>
main()
{  execlp("ls", "ls", "-l", ">!", "a_file", NULL);
   perror("execlp"); }
$ cc -o redir redir.c
$ ./redir
.....................................
$ ls -l
.....................................
```

Θα εκτυπωθεί κάτι στη θέση των αποσιωπητικών μετά την εκτέλεση της εντολής "`./redir`"; Αν ναι, τί και γιατί; Αν όχι, γιατί; Δώστε και ένα πιθανό αποτέλεσμα της εντολής "`ls -l`" στο τέλος.

(β′) "*Μπορούμε πριν το γράψιμο στο κατάλληλο άκρο ενός σωλήνα (ή σε μία υποδοχή ροής) του επιθυμητού μηνύματος, να γράψουμε το μέγεθος του μηνύματος (π.χ. σαν έναν ακέραιο 2 bytes), οπότε ...*". Αν το προηγούμενο είναι η αρχή μίας απάντησης, τότε ποια είναι η ερώτηση και πώς θα πρέπει να συμπληρωθεί η απάντηση;

2. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`encol`") το οποίο να διαβάζει την είσοδό του από την προκαθορισμένη είσοδο, να γράφει την έξοδό του στην προκαθορισμένη έξοδο και το οποίο, με βάση αριθμούς στηλών που του δόθηκαν από τη γραμμή εντολής, να κατασκευάζει την έξοδό του σαν παράθεση των κολονών από την είσοδο που ορίζονται από τους δεδομένους αριθμούς στηλών. Δηλαδή, αν το "`encol`" κληθεί σαν "`encol` $\langle col.1\rangle$ $\langle col.2\rangle$ ... $\langle col.N\rangle$", στην έξοδο να εκτυπωθούν πρώτα οι στήλες 1 έως $\langle col.1\rangle - 1$ της εισόδου, μετά οι στήλες $\langle col.1\rangle$ έως $\langle col.2\rangle - 1$ και ούτω καθεξής και τέλος οι στήλες $\langle col.N\rangle$ έως την τελευταία. Φυσικά, αν κάποια γραμμή της εισόδου δεν είναι αρκετά πλατιά, οι μη υπάρχουσες κολόνες αυτής της γραμμής θα πρέπει να προκαλούν την εγγραφή στην έξοδο κενών γραμμών. Με άλλα λόγια, ανεξάρτητα από το πλάτος των γραμμών στην είσοδο, αν αυτές είναι $\langle M\rangle$ στο πλήθος, η έξοδος πρέπει να έχει $\langle M\rangle \times (\langle N\rangle + 1)$ γραμμές. Ένα αρχείο εισόδου και ένα παράδειγμα εκτέλεσης του "`encol`" φαίνονται στα σχήματα:

```
$ cat inp_file
1. 234 George B. 4. 334 Helen B.
2. 247 Julia R.
3. 327 Robert R. 6. 389 Mary C. 9. 480 Peter G.
$
```

```
$ ./encol 18 33 < inp_file
1. 234 George B.
2. 247 Julia R.
3. 327 Robert R.
4. 334 Helen B.

6. 389 Mary C.



9. 480 Peter G.
$
```

3. Γράψτε ένα πρόγραμμα C (υποθέστε ότι ονομάζεται "`thread_tree`") που όταν καλείται με έναν αριθμό $\langle n\rangle$ στη γραμμή εντολής, το αρχικό νήμα να δημιουργεί ένα δυαδικό δέντρο από νήματα βάθους $\langle n\rangle$ (ή $\langle n\rangle+1$ επιπέδων). Συγκεκριμένα, το αρχικό νήμα να δημιουργεί το νήμα-ρίζα (επιπέδου $\langle n\rangle$) του δέντρου, το οποίο να δημιουργεί δύο νήματα (επιπέδου $\langle n\rangle - 1$) και ούτω καθεξής έως ότου δημιουργηθούν $2^{\langle n\rangle}$

νήματα (επιπέδου 0), δηλαδή τα φύλλα του δέντρου. Κάθε νήμα τερματίζοντας να επιστρέφει σαν κωδικό εξόδου το άθροισμα των ταυτοτήτων όλων των νημάτων που είναι άμεσοι ή έμμεσοι απόγονοί του. Το αρχικό νήμα να εκτυπώνει το άθροισμα των ταυτοτήτων όλων των νημάτων του δέντρου, χρησιμοποιώντας το μηχανισμό των κωδικών εξόδου που περιγράφτηκε. Ένα παράδειγμα εκτέλεσης:

```
$ ./thread_tree 2
Initial thread (id = 1024) created thread with id 1026
Level 0 thread (id = 3076) did not create any thread
Level 0 thread (id = 5126) did not create any thread
Level 1 thread (id = 4101) created threads with ids 5126 and 6151
Level 0 thread (id = 6151) did not create any thread
Level 2 thread (id = 1026) created threads with ids 2051 and 4101
Level 1 thread (id = 2051) created threads with ids 3076 and 7176
Level 0 thread (id = 7176) did not create any thread
Initial thread computes total sum of thread ids: 28707
```

4. Αν το εκτελέσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα ονομάζεται "iosem", δώστε ένα πιθανό αποτέλεσμα της εντολής "iosem 5 10 1", καθώς επίσης και της "iosem 5 10 0". Εξηγήστε επίσης, πολύ συνοπτικά, ποια είναι η λειτουργία του προγράμματος.

```
/* All includes are here */
union semun {int val; struct semid_ds *buff; unsigned short *array; };
main(int argc, char *argv[])
{  int i, j, n, m, sf, sid, fd, pid, st;    long rnd, sofar, tot = 0;
   struct sembuf op[2] = {0, -1, 0, 0, 1, 0};    union semun arg;
   n = atoi(argv[1]);    m = atoi(argv[2]);    sf = atoi(argv[3]);
   sid = semget((key_t) 11111, 1, 0600 | IPC_CREAT);
   arg.val = 1;    semctl(sid, 0, SETVAL, arg);
   fd = open("my_file", O_CREAT | O_TRUNC | O_WRONLY, 0600);
   srandom((unsigned int) getpid());    rnd = random() % 100;
   printf("pid = %d  rnd = %ld\n", getpid(), rnd);
   write(fd, (char *) &rnd, sizeof(long));    close(fd);
   for (i=0 ; i<n ; i++) {
      pid = fork();
      if(!pid) {
         sid = semget((key_t) 11111, 1, 0600);
         srandom((unsigned int) getpid());
         for (j=0 ; j<m ; j++) {
            if (sf) semop(sid, &op[0], 1);
            fd = open("my_file", O_RDONLY);
            read(fd, (char *) &sofar, sizeof(long));    close(fd);
            rnd = random() % 100;    sofar = sofar + rnd;
            fd = open("my_file", O_WRONLY);
            write(fd, (char *) &sofar, sizeof(long));    close(fd);
            if (sf) semop(sid, &op[1], 1);
            tot = tot + rnd; }
         printf("pid = %d  tot = %ld\n", getpid(), tot);    exit(0); } }
   for (i=0 ; i<n ; i++) wait(&st);
   semctl(sid, 0, IPC_RMID, 0);
   fd = open("my_file", O_RDONLY);
   read(fd, (char *) &rnd, sizeof(long));    close(fd);
   printf("pid = %d  rnd = %ld\n", getpid(), rnd);    unlink("my_file"); }
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Α΄ Περιόδου 2004

1. (α΄) Σχολιάστε σε 10 το πολύ γραμμές συνολικά τους παρακάτω ισχυρισμούς:
   i. Ο συνολικός χώρος που καταλαμβάνουν τα αρχεία και οι κατάλογοι ενός χρήστη σ΄ ένα σύστημα αρχείων είναι 600 MB. Ένα απ΄ αυτά τα αρχεία, το "`foo`", έχει μέγεθος 10 MB. Μετά την εκτέλεση της εντολής "`ln foo bar1`", στον κατάλογο που βρίσκεται το "`foo`", ο συνολικός χώρος που καταλαμβάνουν τα αρχεία και οι κατάλογοι του χρήστη είναι πλέον 610 MB και μετά την εκτέλεση της εντολής "`cp foo bar2`", ο συνολικός χώρος είναι 620 MB.
   ii. Ως γνωστόν, κάποια από τις πληροφορίες που φυλάσσονται στον inode ενός αρχείου είναι και οι διευθύνσεις των blocks στα οποία είναι καταχωρημένα τα περιεχόμενα του αρχείου. Ομοίως, στα blocks τα οποία αναφέρονται από τον inode ενός καταλόγου είναι καταχωρημένα τα περιεχόμενα του καταλόγου, δηλαδή τα περιεχόμενα των αρχείων και των καταλόγων που περιέχονται σ΄ αυτόν τον κατάλογο.

   (β΄) Σε μία πολυνηματική εφαρμογή, ένα νήμα, ο "*προϊστάμενος*", μπορεί να εισέλθει σε κάποιο τμήμα του κώδικά του, μόνο αφού μία συνθήκη που εμπλέκει κάποιες εξωτερικές μεταβλητές ισχύσει. Τις μεταβλητές της συνθήκης τις επηρεάζουν ένα πλήθος από άλλα νήματα, οι "*υπάλληλοι*", τα οποία επίσης φροντίζουν να ενημερώσουν κατάλληλα τον "*προϊστάμενο*" μόλις η συνθήκη ισχύσει. Υποθέστε ότι οι μεταβλητές που εμπλέκονται στη συνθήκη μεταβάλλονται με τέτοιο τρόπο ώστε αν κάποια στιγμή η συνθήκη ισχύσει, θα ισχύει για πάντα στη συνέχεια. Διατυπώστε σε μία "ψευδογλώσσα" της επιλογής σας τον κώδικα του "*προϊσταμένου*" και τον κώδικα των "*υπαλλήλων*", εστιάζοντας φυσικά στα τμήματα που έχουν να κάνουν με τις απαιτούμενες ενημερωτικές ενέργειες των "*υπαλλήλων*" και με το "ξεμπλοκάρισμα" του "*προϊσταμένου*" όταν ισχύσει η συνθήκη. Εννοείται ότι δεν υπάρχει καμία εκ των προτέρων γνώση η εγγύηση για το σχετικό χρονισμό των εμπλεκόμενων νημάτων.

2. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`reformat`") το οποίο, εκτελούμενο με μία παράμετρο $\langle n \rangle$ στη γραμμή εντολής, να διαβάζει από την προκαθορισμένη είσοδο (stdin) ένα κείμενο δομημένο σε γραμμές από λέξεις (οι λέξεις χωρίζονται με έναν η περισσότερους κενούς χαρακτήρες) και να το ξαναγράφει στην προκαθορισμένη έξοδο (stdout) έτσι ώστε καμία γραμμή να μην ξεπερνά τους $\langle n \rangle$ χαρακτήρες, μεταξύ των λέξεων να υπάρχει ακριβώς ένας κενός χαρακτήρας, όλες οι γραμμές να αρχίζουν με μη κενό χαρακτήρα και να μην τελειώνουν με κενούς χαρακτήρες. Υποθέστε ότι στην είσοδό σας δεν υπάρχει λέξη με περισσότερους από $\langle n \rangle$ χαρακτήρες. Ένα παράδειγμα εκτέλεσης είναι το ακόλουθο:

```
$ cat file.txt
   This is a text file to
test   the Bourne script for
reformatting.  Use it    with
  care!
$ ./reformat 17 < file.txt
This is a text
file to test the
Bourne script for
reformatting. Use
it with care!
$
```

3. Αν το εκτελέσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα ονομάζεται "`mythreads`", δώστε ένα πιθανό αποτέλεσμα της εντολής "`mythreads 3`", υποθέτοντας ότι όλες οι συναρτήσεις νημάτων που καλούνται θα λειτουργήσουν κανονικά.

```
#include <stdio.h>
#include <pthread.h>

void *f(void *argp)
{  pthread_t thr;
   int n = (int) argp;
   while (n--) {
      pthread_create(&thr, NULL, f, (void *) n);
      printf("%d %d %d\n", n, pthread_self(), thr);
      pthread_join(thr, NULL); }
   pthread_exit(NULL); }

main(int argc, char *argv[])
{  pthread_t thr;
   int n = atoi(argv[1]);
   pthread_create(&thr, NULL, f, (void *) n);
   pthread_join(thr, NULL);
   pthread_exit(NULL); }
```

Ποιο θα είναι το αποτέλεσμα εκτέλεσης της εντολής "`mythreads 10 | wc -lw`";

4. Γράψτε ένα πρόγραμμα C (υποθέστε ότι ονομάζεται "`mybros`") το οποίο να καλείται με έναν αριθμό $\langle n \rangle$ στη γραμμή εντολής. Η αρχική διεργασία-γονέας να δημιουργεί $\langle n \rangle$ διεργασίες-παιδιά. Η κάθε διεργασία-παιδί να εκτυπώνει την "τάξη" της (1ο παιδί, 2ο παιδί, κλπ.) και την ταυτότητά της. Ειδικά το 1ο παιδί να εκτυπώσει επιπλέον και τις ταυτότητες των επόμενων διεργασιών-παιδιών, δηλαδή των αδελφών του. Ένα παράδειγμα εκτέλεσης είναι το παρακάτω:

```
$ ./mybros 6
I am child no 2 with pid 4215
I am child no 4 with pid 4217
I am child no 3 with pid 4216
I am child no 5 with pid 4218
I am child no 1 with pid 4214 and my brothers are:
   brother no 1 with pid 4215
   brother no 2 with pid 4216
   brother no 3 with pid 4217
   brother no 4 with pid 4218
   brother no 5 with pid 4219
I am child no 6 with pid 4219
$
```

Εννοείται ότι δεν υπάρχει καμία απαίτηση εγγύησης για τη σειρά των εκτυπώσεων.

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Β′ Περιόδου 2003

1. (α′) Παρατηρήστε την παρακάτω διαλογική επικοινωνία με το κέλυφος και δώστε το αποτέλεσμα που παραλείψαμε στην τελευταία εντολή. Εξηγήστε, πολύ σύντομα, την απάντησή σας.

```
$ ls -l
total 4
-rw-r--r--    1 spro     users          91 Sep  9 15:29 exwild.c
$ cat exwild.c
#include <stdio.h>
main()
{  execlp("ls", "ls", "-l", "*.*", NULL);
   perror("execlp"); }
$ cc -o exwild exwild.c
$ ./exwild
.....................................
```

   (β′) Ποια ήταν, κατά τη γνώμη σας, η πρόθεση του προγραμματιστή που έγραψε το πρόγραμμα C από το οποίο προέρχεται το παρακάτω απόσπασμα; Υπάρχει κάποιο πολύ σοβαρό σχεδιαστικό λάθος στο πρόγραμμα αυτό; Αν ναι, εξηγήστε ποιο είναι και προτείνετε πώς θα μπορούσε να διορθωθεί.

```
.........................................
main()
{  .....................................
   if (pipe(fd) == -1) { ................... }
   if ((pid = fork()) == -1) { ............. }
   if (pid == 0) { write(fd[1], top, strlen(top)+1);
                   b = read(fd[0], mess, sizeof(mess));
                   printf("Child: Read %d bytes: %s\n", b, mess); ........ }
           else { write(fd[1], toc, strlen(toc)+1);
                   b = read(fd[0], mess, sizeof(mess));
                   printf("Parent: Read %d bytes: %s\n", b, mess); ....... }
.........................................
```

2. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`remdupl`") το οποίο να διαβάζει γραμμές από την προκαθορισμένη είσοδο (stdin) και να τις γράφει στην προκαθορισμένη έξοδο (stdout), αλλά στην περίπτωση συνεχόμενων ίδιων γραμμών στην είσοδο, να μεταφέρεται στην έξοδο η μία μόνο απ′ αυτές. Να δώσετε επιπλέον στο πρόγραμμά σας τη δυνατότητα να καλείται, προαιρετικά, και με την επιλογή "`-c`", οπότε, στην περίπτωση αυτή, στην αρχή κάθε γραμμής στην έξοδο να εκτυπώνεται και των πλήθος των γραμμών της εισόδου από τις οποίες προήλθε. Η διαλογική επικοινωνία στο Σχήμα 1 της επόμενης σελίδας είναι ενδεικτική για τη λειτουργικότητα του "`remdupl`". (*Σημείωση:* Απαγορεύεται η χρήση της εντολής "`uniq`".)

3. Γράψτε μία συνάρτηση C, την "`void swapsighandl(int sig1, int sig2)`", η οποία, όταν καλείται, να ανταλλάσσει τους διαχειριστές σήματος για τα σήματα `sig1` και `sig2`. Δηλαδή, μετά την κλήση της, ο διαχειριστής για το σήμα `sig1` να είναι αυτός που είχε προηγουμένως το σήμα `sig2` και τούμπαλιν. Γράψτε επίσης και ένα πρόγραμμα C της επιλογής σας, το οποίο να επιδεικνύει τη χρήση της συνάρτησης `swapsighandl`. Για παράδειγμα, στο Σχήμα 2 της επόμενης σελίδας, φαίνεται η εκτέλεση ενός προγράμματος το οποίο, αφού ορίσει κατάλληλους διαχειριστές για τα σήματα που προκύπτουν πατώντας Control-C ή Control-Z στο πληκτρολόγιο, περιμένει την παραλαβή των δύο αυτών σημάτων και, τέλος, επαναλαμβάνει τη διαδικασία αυτή, αφού πρώτα εναλλάξει του διαχειριστές τους μέσω της `swapsighandl`.

```
$ cat test.txt
A line
A line
Another   line
Final line
Final line
Final line
$ cat test.txt | ./remdupl
A line
Another   line
Final line
$ cat test.txt | ./remdupl -c
2 A line
1 Another   line
3 Final line
```

Σχήμα 1

```
$ ./sigswap
Press Control-C or Control-Z
^C
I got a Control-C
Press Control-C or Control-Z
^Z
I got a Control-Z
Swapping handlers...
Press Control-C or Control-Z
^C
I got a Control-Z
Press Control-C or Control-Z
^Z
I got a Control-C
Finished...
```

Σχήμα 2

4. Αν το εκτελέσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα ονομάζεται "mqdemo", δώστε ένα πιθανό αποτέλεσμα της εντολής "mqdemo 5", υποθέτοντας ότι όλες οι κλήσεις συστήματος θα λειτουργήσουν κανονικά. Εξηγήστε επίσης στοιχειωδώς τη λειτουργία του προγράμματος.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>

void comm(int i, int q)
{ struct message { long mtype; char mtext[8]; } messrcv, messsnd;
  messsnd.mtype = (long) getpid();
  sprintf(messsnd.mtext, "%d", getppid());
  msgsnd(q, &messsnd, strlen(messsnd.mtext)+1, 0);
  msgrcv(q, &messrcv, 8, (long) i, 0);
  printf("%d %s\n", getpid(), messrcv.mtext); }

main(int argc, char *argv[])
{ int i, n, pid, orig, qid, status;
  n = atoi(argv[1]);
  orig = getpid();
  qid = msgget((key_t) 23456, 0600 | IPC_CREAT);
  for(i=0 ; i<n ; i++) {
    if ((pid = fork()) == -1) { perror("fork"); exit(1); }
    if (pid) {
      printf("%2d ", i);
      comm(pid, qid);
      wait(&status);
      if(!i) {
        msgctl(qid, IPC_RMID, (struct msqid_ds *) 0); }
      exit(0); } }
  printf("   ");
  comm(orig, qid); }
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Α΄ Περιόδου 2003

1. (α΄) Παρατηρήστε την παρακάτω διαλογική επικοινωνία με το κέλυφος και δώστε το αποτέλεσμα που παραλείψαμε στην τελευταία εντολή. Εξηγήστε, πολύ σύντομα, την απάντησή σας.

```
$ ls -l
total 4
-rw-r--r--    1 spro     users          95 Jun 18 23:46 expipe.c
$ cat expipe.c
#include <stdio.h>
main()
{  execlp("ls", "ls", "-l", "|", "wc", NULL);
   perror("execlp"); }
$ cc -o expipe expipe.c
$ ./expipe
.....................................
```

   (β΄) Το πρόγραμμα C από το οποίο προέρχεται το παρακάτω απόσπασμα μεταγλωττίζεται με επιτυχία, έχει όμως δύο πολύ σοβαρά λογικά λάθη. Εξηγήστε ποια είναι τα λάθη αυτά, σε σχέση με τη γνώμη σας για τις προθέσεις του προγραμματιστή, και προτείνετε πώς θα μπορούσαν να διορθωθούν.

```
.................................................
pthread_t thrid, tids[5];
void *thr_f1(void *argp)
{  printf("I am thread with id %d\n", thrid);
   ..................... }
void *thr_f2(void *argp)
{  printf("I am thread no %d\n", *((int *) argp));
   ..................... }
main()
{  ......................
   pthread_create(&thrid, NULL, thr_f1, NULL);
   for (i=0 ; i<5 ; i++)
      pthread_create(&tids[i], NULL, thr_f2, (void *) &i);
   ..................... }
```

2. Σ΄ ένα Πρόγραμμα Μεταπτυχιακών Σπουδών (ΠΜΣ), τα προσφερόμενα μαθήματα, καθώς και κάποιες πληροφορίες σχετικές με αυτά, είναι καταχωρημένα σ΄ ένα αρχείο, το όνομα-μονοπάτι του οποίου υποθέστε ότι φυλάσσεται σαν τιμή της μεταβλητής περιβάλλοντος "PGCOURSES". Κάθε γραμμή του αρχείου αυτού αντιστοιχεί σ΄ ένα μάθημα και περιλαμβάνει, με αυτή τη σειρά, τον κωδικό του μαθήματος, το όνομά του (συνήθως περισσότερες από μία λέξη), τον τύπο του μαθήματος (B αν είναι βασικό ή E αν είναι επιλογής) και τις διδακτικές μονάδες του. Για παράδειγμα:

```
$ PGCOURSES=pgcourses.txt ; export PGCOURSES ; cat $PGCOURSES
512 Systhmata Polymeswn kai Ypermeswn E 3
527 Diktya Optikwn Epikoinwniwn B 4
519 Katanemhmena Systhmata B 2
.....................................
```

   Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "pgstat") το οποίο να διαβάζει από την προκαθορισμένη είσοδο (stdin) το φάκελλο ενός φοιτητή, δίνοντας στην πρώτη γραμμή της

εισόδου τον αριθμό μητρώου του φοιτητή και το ονοματεπώνυμό του (όχι, κατ΄ ανάγκη, δύο λέξεις) και σε κάθε επόμενη γραμμή τον κωδικό ενός μαθήματος στο οποίο έχει εξετασθεί ο φοιτητής και το βαθμό που έχει πάρει. Ένας φοιτητής θεωρείται ότι έχει τελειώσει το ΠΜΣ όταν από τα μαθήματα που έχει περάσει τη βάση (βαθμός ≥ 5) έχει συμπληρώσει τουλάχιστον 40 διδακτικές μονάδες, από τις οποίες τουλάχιστον 25 να προέρχονται από βασικά μαθήματα. Το πρόγραμμα "`pgstat`" να εκτυπώνει αν ο φοιτητής συνεχίζει ή έχει τελειώσει τις σπουδές του, καθώς και το μέχρι στιγμής, ή τελικό, μέσο όρο βαθμολογίας, με αποκοπή στα δύο δεκαδικά ψηφία. Για τον υπολογισμό του μέσου όρου, λαμβάνονται υπόψη όλα τα μαθήματα στα οποία ο φοιτητής έχει περάσει τη βάση (ακόμα και αν έχει συμπληρώσει περισσότερες διδακτικές μονάδες από τις απαιτούμενες) και ο κάθε βαθμός συνεισφέρει στο μέσο όρο με βαρύτητα τις διδακτικές μονάδες του αντίστοιχου μαθήματος. Για παράδειγμα:

```
$ ./pgstat
5234 John Smith
527 7
540 4
512 10
519 8
^D
John Smith (id = 5234) not finished.
Average mark: 8.22
```

```
$ ./pgstat
8 Bill Gates
532 6
510 8
527 10
.....................
^D
Bill Gates (id = 8) got the diploma.
Average mark: 7.67
```

Αν χρησιμοποιούσε το πρόγραμμά σας η Γραμματεία που διαχειρίζεται το ΠΜΣ, θα έπρεπε κάθε φορά που θα ήθελε να ελέγξει την κατάσταση ενός φοιτητή να πληκτρολογεί το φάκελλό του, ή θα μπορούσε να κάνει κάτι πιο πρακτικό, χωρίς να αλλάξει βέβαια το πρόγραμμα; Υποθέστε ότι όλα τα δεδομένα σας είναι απολύτως συνεπή. Δεν χρειάζεται να κάνετε ελέγχους ορθότητας.

3. Δώστε μία πιθανή εκτύπωση του εκτελέσιμου προγράμματος που προέρχεται από το ακόλουθο πρόγραμμα C. Εννοείται, ότι θα υποθέσετε ότι όλες οι "`fork()`" θα λειτουργήσουν χωρίς πρόβλημα.

```
#include <stdio.h>
main()
{  int s;
   printf("%d %d %d %d\n", getpid(), getppid(), fork(), fork());
   printf("%d %d %d\n", getpid(), wait(&s), wait(&s)); }
```

4. Γράψτε ένα πρόγραμμα C (υποθέστε ότι ονομάζεται "`smplfind`") το οποίο να λειτουργεί σαν μία απλή εκδοχή της εντολής "`find`", ουσιαστικά υλοποιώντας μόνο το κριτήριο εύρεσης με βάση το όνομα. Συγκεκριμένα, όταν το πρόγραμμα αυτό καλείται σαν "`smplfind` $\langle name \rangle_1 \langle name \rangle_2 \ldots \langle name \rangle_k$ `-dirs` $\langle dir \rangle_1 \langle dir \rangle_2 \ldots \langle dir \rangle_m$" να ψάχνει κάτω από τους καταλόγους $\langle dir \rangle_j$ (σε οποιοδήποτε βάθος) κόμβους με όνομα κάποιο από τα $\langle name \rangle_i$. Το τμήμα από το `-dirs` και μετά μπορεί και να μην δίνεται, οπότε η αναζήτηση τότε θα πρέπει να γίνεται στον τρέχοντα κατάλογο. Για παράδειγμα:

```
$ ./smplfind file1.c bla.txt foo.dat -dirs sp ../dir /home/users/me
sp/bla.txt
../dir/newdir/file1.c
../dir/other/another/bla.txt
/home/users/me/dir/foo.dat
```

Με βάση το πρόγραμμα που γράψατε, δώστε ένα πιθανό αποτέλεσμα της εντολής:

```
$ ./smplfind *.c ????.txt
.....................................
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Β′ Περιόδου 2002

1. (α′) Παρατηρήστε την παρακάτω διαλογική επικοινωνία με το κέλυφος Bourne, που δεν είναι όμως σωστή. Σε κάποια σημεία, σε **αποτελέσματα** εντολών μόνο, εκτός απ′ αυτά της πρώτης εντολής, έγινε εκ των υστέρων επέμβαση. Ποια είναι αυτά τα σημεία, γιατί είναι λάθος και πώς ήταν η πραγματική πρωτότυπη εκδοχή; Επίσης, δώστε το αποτέλεσμα που παραλείψαμε στην τρίτη εντολή.

```
$ ls -ald . .. ???
drwxr-xr-x    4 spro     users        4096 Oct  5 16:02 .
drwx------    3 spro     users        4096 Oct  5 15:55 ..
-rw-r--r--    1 spro     users         471 Sep 24 13:22 bla
-rwxr-xr-x    1 spro     users       14415 Oct  5 14:18 foo
$ ln foo bar
$ ls -ald . .. ???
.....................................................
$ chmod 710 foo
$ ls -ald . .. ???
drwxr-xr-x    4 spro     users        4096 Oct  5 16:03 .
drwx------    3 spro     users        4096 Oct  5 15:55 ..
-rwxr-xr-x    2 spro     users       14415 Oct  5 14:18 bar
-rw-r--r--    1 spro     users         471 Sep 24 13:22 bla
-rwx--x---    1 spro     users       14415 Oct  5 14:18 foo
$ rm foo
$ ls -ald . .. ???
drwxr-xr-x    3 spro     users        4096 Oct  5 16:04 .
drwx------    3 spro     users        4096 Oct  5 15:55 ..
-rwxr-xr-x    2 spro     users       14415 Oct  5 14:18 bar
-rw-r--r--    1 spro     users         471 Sep 24 13:22 bla
```

(β′) Όταν αναπτύσσουμε client-server εφαρμογές μέσω υποδοχών, σε κάποιες περιπτώσεις υλοποιούμε τον εξυπηρετητή (server) S έτσι ώστε μία διεργασία-παιδί του να αναλαμβάνει την εξυπηρέτηση του πελάτη (client) C, ενώ ο S περιμένει αιτήσεις εξυπηρέτησης από άλλους πελάτες. Τότε λέμε ότι ο S είναι *ταυτόχρονος εξυπηρετητής* (concurrent server). Σε άλλες περιπτώσεις, υλοποιούμε τον S έτσι ώστε ο ίδιος να αναλάβει να εξυπηρετήσει τον C, και αφού τελειώσει η εξυπηρέτηση, περιμένει αιτήσεις από άλλους πελάτες. Τότε λέμε ότι έχουμε *σειριακό εξυπηρετητή* (sequential server). Πότε ακολουθούμε την πρώτη προσέγγιση και πότε τη δεύτερη; Δώστε δύο παραδείγματα συγκεκριμένων "χρήσιμων" υπηρεσιών, ένα για κάθε περίπτωση. Σχετίζεται, για τεχνικούς ή άλλους λόγους, το είδος των υποδοχών που θα χρησιμοποιήσουμε (TCP ή UDP) με την επιλογή μας για το αν θα ακολουθήσουμε την εκδοχή του ταυτόχρονου ή του σειριακού εξυπηρετητή;

2. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`elections`") το οποίο να λειτουργεί σαν ένα απλό σύστημα διαχείρισης και παρουσίασης εκλογικών αποτελεσμάτων. Το πρόγραμμα αυτό να καλείται με δύο τρόπους. Ο πρώτος να χρησιμοποιείται για την περίπτωση καταχώρισης ενός μερικού αποτελέσματος, για παράδειγμα από κάποιο εκλογικό τμήμα, με την εντολή "`elections` $\langle party\rangle_1$ $\langle votes\rangle_1$ $\langle party\rangle_2$ $\langle votes\rangle_2$ ... $\langle party\rangle_n$ $\langle votes\rangle_n$", όπου για κάθε i, $\langle party\rangle_i$ είναι το όνομα ενός κόμματος και $\langle votes\rangle_i$ είναι ο αριθμός των ψήφων που πήρε το κόμμα αυτό στο συγκεκριμένο εκλογικό τμήμα. Φυσικά, η σειρά με την οποία δίνονται τα ζευγάρια ονομάτων κομμάτων και αριθμών ψήφων στη γραμμή εντολής δεν είναι απαραίτητο να είναι πάντα η ίδια, ούτε είναι απαραίτητο να έχουμε το ίδιο πλήθος ζευγών σε διαφορετικές εκτελέσεις του προγράμματος, γιατί μπορεί κάποιο κόμμα να μην έχει πάρει καμία ψήφο σε κάποιο εκλογικό τμήμα. Ως προς το πού και πώς θα καταχωρούνται τα αποτελέσματα αυτά, έχετε απόλυτη ελευθερία κινήσεων. Βέβαια, η προφανής ιδέα για το "πού"

είναι η δευτερεύουσα μνήμη. Ο δεύτερος τρόπος εκτέλεσης του "`elections`" να είναι με το μοναδικό όρισμα "`-results`", οπότε να μας δίνει μία εικόνα των μέχρι στιγμής συνολικών αποτελεσμάτων, συμπεριλαμβανομένων και ποσοστών. Δείτε τα ακόλουθα παραδείγματα:

```
$ ./elections "To komma mas" 187 "Oi alloi" 225 "Oi kakoi" 247
$ ./elections "Oi kaloi" 150 "Oi kakoi" 10 "Oi asxhmoi" 304 "To komma mas" 57
$ ./elections "Oi alloi" 404 "To komma mas" 192
$ ./elections -results
Oi alloi: votes 629, i.e. 35%
Oi asxhmoi: votes 304, i.e. 17%
Oi kakoi: votes 257, i.e. 14%
Oi kaloi: votes 150, i.e. 8%
To komma mas: votes 436, i.e. 24%
```

3. Δώστε μία πιθανή εκτύπωση του εκτελέσιμου προγράμματος που προέρχεται από το ακόλουθο πρόγραμμα C. Σχεδιάστε το δέντρο των διεργασιών που θα δημιουργηθούν, περιλαμβάνοντας στο σχήμα σας και τη διεργασία του κελύφους κάτω από το οποίο εκτελέστηκε το πρόγραμμα. Εννοείται, ότι θα υποθέσετε ότι όλες οι "`fork()`" θα λειτουργήσουν χωρίς πρόβλημα. Σημειώνεται ότι ο κωδικός εξόδου μίας διεργασίας είναι πάντα ένα byte, το λιγότερο σημαντικό του ακεραίου που επιστρέφει κατά τον τερματισμό της. Επίσης, θεωρήστε ότι όταν η "`int wait(int *p)`" επιστρέφει `-1`, το περιεχόμενο της διεύθυνσης `p` δεν αλλάζει.

```
#include <stdio.h>
main()
{ int q, r, s, t;
  s = 0;
  q = !fork(); t = wait(&s); r = !fork(); t = wait(&s);
  printf("%d  %d  q = %d  r = %d  s = %d  t = %d\n",
         getpid(), getppid(), q, r, s, t);
  exit(getpid()); }
```

4. Γράψτε ένα πρόγραμμα C (υποθέστε ότι ονομάζεται "`mqf`") το οποίο να καλείται με έναν αριθμό $\langle N \rangle$ στη γραμμή εντολής. Η αρχική διεργασία (επιπέδου 0) να ορίζει μία ουρά μηνυμάτων και στη συνέχεια να δημιουργεί μία διεργασία-παιδί (επιπέδου 1), η οποία, με τη σειρά της, να δημιουργεί μία διεργασία-παιδί (επιπέδου 2) και ούτω καθεξής, έως ότου δημιουργηθεί και η διεργασία επιπέδου $\langle N \rangle$ σαν παιδί της διεργασίας επιπέδου $\langle N \rangle - 1$. Κάθε διεργασία να εκτυπώνει το βάθος της (`depth`), την ταυτότητά της (`pid`), την ταυτότητα του γονέα της (`ppid`) και την ταυτότητα της διεργασίας-εγγονού της (`gcpid`). Για την πληροφόρηση σχετικά με το `gcpid`, να χρησιμοποιηθεί η ουρά μηνυμάτων που όρισε η αρχική διεργασία. Το `gcpid` για τις δύο τελευταίες διεργασίες να είναι 0. Κάθε διεργασία, μετά την εκτύπωση των πληροφοριών που ζητήθηκαν, να περιμένει να τερματίσει το παιδί της και μετά να τερματίζει και αυτή. Τέλος, η ουρά μηνυμάτων να απελευθερώνεται από την αρχική διεργασία που την όρισε. Ένα παράδειγμα εκτέλεσης είναι το εξής:

```
$ ./mqf 5
Process with pid=2268 created message queue with qid=98304
I am process at depth 0 with pid=2268 ppid=2113 gcpid=2270
I am process at depth 2 with pid=2270 ppid=2269 gcpid=2272
I am process at depth 5 with pid=2273 ppid=2272 gcpid=0
I am process at depth 1 with pid=2269 ppid=2268 gcpid=2271
I am process at depth 4 with pid=2272 ppid=2271 gcpid=0
I am process at depth 3 with pid=2271 ppid=2270 gcpid=2273
Process with pid=2268 destroyed message queue with qid=98304
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Α΄ Περιόδου 2002

1. (α΄) Ποια είναι διαφορά στην **εσωτερική** λειτουργία ενός κελύφους όταν εκτελούμε κάτω απ΄ αυτό μία εντολή στο παρασκήνιο (π.χ. "`ls -l &`") σε σχέση με την εκτέλεση μίας εντολής στο προσκήνιο (π.χ. "`ls -l`");

   (β΄) Γνωρίζετε ότι μία από τις διαφορές μεταξύ των υποδοχών ροής (stream sockets) και των τηλεγραφικών υποδοχών (datagram sockets) είναι ότι ενώ στις δεύτερες το πρωτόκολλο (UDP) υποστηρίζει όρια μεταξύ των αποστελλόμενων μηνυμάτων, στις πρώτες το αντίστοιχο πρωτόκολλο (TCP) δεν παρέχει κάτι τέτοιο. Ποια είναι ακριβώς η πρακτική επίπτωση αυτής της διαφοράς; Αν χρειαζόμασταν σε κάποια εφαρμογή να χρησιμοποιήσουμε, για κάποιο σημαντικό λόγο, υποδοχές ροής, αλλά ταυτόχρονα μας ενδιέφερε να έχουμε και όρια μεταξύ των μηνυμάτων, πώς θα μπορούσαμε να λύσουμε το πρόβλημα;

2. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`myrm`") το οποίο να υποκαθιστά την εντολή "`rm`" του Unix ως εξής: Το "`myrm`", αντί να διαγράφει αρχεία, να τα μεταφέρει στον κατάλογο "`.waste`" που βρίσκεται στον κατάλογο αφετηρίας του χρήστη. Αν δεν υπάρχει ο κατάλογος "`.waste`", να τον δημιουργεί. Για παράδειγμα, αυτό να γίνεται με την εντολή "`myrm` $\langle file\rangle_1$ $\langle file\rangle_2$ ... $\langle file\rangle_n$", όπου $n \geq 1$. Η "`myrm`" να υποστηρίζει επίσης και τις επιλογές "`-l`", "`-p`" και "`-u`" (από τα list, purge και undelete, αντίστοιχα), που σημαίνουν τα εξής: Το "`myrm -l`" να εκτυπώνει τα ονόματα των αρχείων που έχουν μεταφερθεί στον κατάλογο "`.waste`" μαζί με όλες τις πληροφορίες που δίνει η "`ls -l`". Το "`myrm -p`" να διαγράφει οριστικά όλα τα αρχεία του καταλόγου "`.waste`". Το "`myrm -u` $\langle file\rangle$" να επαναφέρει το αρχείο $\langle file\rangle$ από τον κατάλογο "`.waste`" στον τρέχοντα κατάλογο. Υπάρχουν περιπτώσεις που δεν θα λειτουργήσει σωστά το πρόγραμμα που γράψατε, εφόσον βέβαια κληθεί σύμφωνα με τις προδιαγραφές, και ποιες είναι αυτές; Ή μήπως είναι, κατά τη γνώμη σας, τέλειο;

3. Γράψτε ένα πρόγραμμα C (υποθέστε ότι ονομάζεται "`nch`") το οποίο να καλείται με έναν αριθμό $\langle N\rangle$ στη γραμμή εντολής. Η αρχική διεργασία-γονέας να δημιουργεί $\langle N\rangle$ διεργασίες-παιδιά, εκτυπώνοντας σχετικό μήνυμα μετά από κάθε δημιουργία, ενώ κάθε διεργασία-παιδί να εκτυπώνει μήνυμα που να περιλαμβάνει την "τάξη" της (1ο παιδί, 2ο παιδί, κλπ.), την ταυτότητά της, καθώς και την ταυτότητα της προηγούμενης διεργασίας-παιδιού (το 1ο παιδί δεν έχει προηγούμενη διεργασία-παιδί). Ένα παράδειγμα εκτέλεσης είναι το εξής:

```
$ ./nch 3
I am parent and I created child 1 with pid=3224
I am child 1 with pid=3224, prev=0
I am parent and I created child 2 with pid=3225
I am child 2 with pid=3225, prev=3224
I am child 3 with pid=3226, prev=3225
I am parent and I created child 3 with pid=3226
$
```

   Πόσο εύκολα πιστεύετε ότι θα μπορούσατε να τροποποιήσετε το πρόγραμμα που γράψατε έτσι ώστε κάθε διεργασία-παιδί να εκτυπώνει εκτός από την ταυτότητα του προηγούμενου παιδιού και αυτήν του επόμενου; Αντίστοιχα με ό,τι ισχύει για το πρώτο παιδί, το τελευταίο παιδί δεν έχει επόμενη διεργασία-παιδί. Αν είναι απλή η τροποποίηση επισημάνετέ την επάνω στο πρόγραμμα που ήδη γράψατε με κατάλληλες παραπομπές και, πάντως, με ευανάγνωστο τρόπο. Αν πιστεύετε ότι χρειάζονται ριζικές προσθήκες ή τροποποποιήσεις, απλώς περιγράψτε τις, χωρίς να τις υλοποιήσετε. Ένα υποθετικό παράδειγμα εκτέλεσης από το επεκτεταμένο πρόγραμμα "`nch`" (έστω ότι ονομάζεται "`newnch`") είναι αυτό που φαίνεται στη συνέχεια:

```
$ ./newnch 3
I am parent and I created child 1 with pid=3228
I am parent and I created child 2 with pid=3229
I am child 1 with pid=3228, prev=0, next=3229
I am parent and I created child 3 with pid=3230
I am child 3 with pid=3230, prev=3229, next=0
I am child 2 with pid=3229, prev=3228, next=3230
$
```

4. Αν το εκτελέσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα ονομάζεται "smch", δώστε ένα πιθανό αποτέλεσμα της εντολής "smch 4", υποθέτοντας ότι όλες οι κλήσεις συστήματος θα λειτουργήσουν κανονικά. Εξηγήστε επίσης στοιχειωδώς (το πολύ σε 10 γραμμές) τη λειτουργία του προγράμματος.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <stdio.h>

main(int argc, char *argv[])
{ int i, j, n, mid, sid, pid, st, is = 0;
  char *reg;
  struct sembuf op[2] = {0, 0, 0, 0, 1, 0};
  n = atoi(argv[1]);
  op[0].sem_op = -n;
  mid = shmget((key_t) 12345, 256, 0600 | IPC_CREAT);
  sid = semget((key_t) 23456, 1, 0600 | IPC_CREAT);
  semctl(sid, 0, SETVAL, &is);
  for (i=0 ; i<n ; i++) {
    pid = fork();
    if (pid) {
      printf("%d %d\n", getpid(), getppid());
      if (!i) {
        reg = shmat(mid, (char *) 0, 0);
        semop(sid, &op[0], 1);
        for (j=0 ; j<n ; j++)
          printf("%d ", *((int *) reg + j));
        printf("\n");
        shmctl(mid, IPC_RMID, (struct shmid_ds *) 0);
        semctl(sid, 0, IPC_RMID, 0);
      }
      printf("%d\n", wait(&st));
      exit(0);
    }
    reg = shmat(mid, (char *) 0, 0);
    *((int *) reg + (n-i-1)) = getpid();
    semop(sid, &op[1], 1);
    shmdt(reg);
  }
}
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Β′ Περιόδου 2001

1. (α′) Γνωρίζουμε ότι οι διαχειριστές σήματος είναι συνήθως `void` συναρτήσεις χωρίς ορίσματα. Σε αρκετά Unix συστήματα όμως, μπορούμε να έχουμε ένα ακριβώς όρισμα, τον αριθμό σήματος που ενεργοποίησε το διαχειριστή. Πότε πιστεύετε ότι θα ήταν χρήσιμο κάτι τέτοιο; Πάντως, σε καμία περίπτωση, δεν υπάρχει η δυνατότητα να έχουμε άλλα ορίσματα σ΄ ένα διαχειριστή σήματος, πλην του αριθμού σήματος που τον ενεργοποίησε, με σκοπό την ανταλλαγή δεδομένων με το υπόλοιπο πρόγραμμα. Πιστεύετε ότι μας χρειάζεται κάποια ανταλλαγή αυτού του είδους; Αν ναι, πώς λύνουμε το πρόβλημά μας;

   (β′) Σ΄ ένα κέλυφος δίνουμε μία εντολή με ανακατεύθυνση της προκαθορισμένης εξόδου σε αρχείο. Το κέλυφος ανοίγει το αρχείο με την "`open`", η οποία επιστρέφει ένα περιγραφέα αρχείου, βάζει με την "`dup2`" τον περιγραφέα αυτό να δείξει στο 1, κάνει "`fork`" και η διεργασία-παιδί που δημιουργείται αντικαθιστά με κάποια "`execXX`" τον κώδικά της με τον κώδικα της εντολής που δόθηκε στο κέλυφος. Είναι σωστή η προηγούμενη διαδικασία; Αν όχι, πού είναι το λάθος και γιατί;

2. Θεωρήστε ότι στο Unix ένα αρχείο μηνυμάτων ηλεκτρονικού ταχυδρομείου είναι ένα απλό αρχείο κειμένου, στο οποίο τα διάφορα μηνύματα είναι χωρισμένα μεταξύ τους με μία κενή γραμμή και η πρώτη γραμμή κάθε μηνύματος έχει σαν πρώτη λέξη την "`From`" και σαν δεύτερη λέξη τη διεύθυνση ⟨*address*⟩ του αποστολέα του μηνύματος. Φυσικά, στο σώμα κάθε μηνύματος μπορούν να υπάρχουν κενές γραμμές. Ένα απόσπασμα από κάποιο αρχείο μηνυμάτων ηλεκτρονικού ταχυσδρομείου θα μορούσε να ήταν το ακόλουθο:

```
% cat a_mail_folder
From costas@di.uoa.gr ..........
...................
...................
  Bye

From nick@acm.org ..............
...................
...................
  See you

..............................
..............................
..............................

From costas@di.uoa.gr ..........
...................
  Costas

%
```

   Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`sliptmf`") το οποίο να δέχεται σαν μοναδικό όρισμα ένα όνομα καταλόγου ⟨*dir*⟩ (υπαρκτού ή όχι — αν δεν υπάρχει, να δημιουργείται). Το πρόγραμμα αυτό να διαβάζει από την προκαθορισμένη είσοδο ένα αρχείο μηνυμάτων ηλεκτρονικού ταχυδρομείου και να διαχωρίζει τα μηνύματα του αρχείου αυτού σε διαφορετικά αρχεία μηνυμάτων, που να τα δημιουργεί στον κατάλογο ⟨*dir*⟩, ανάλογα με τον αποστολέα τους. Συγκεκριμένα, όλα τα μηνύματα που προέρχονται από τη διεύθυνση ⟨*address*⟩, να καταχωρούνται στο αρχείο μηνυμάτων ⟨*dir*⟩`/`⟨*address*⟩. Για παράδειγμα:

```
% splitmf my_dir < a_mail_folder
Creating my_dir
Creating my_dir/costas@di.uoa.gr
Creating my_dir/nick@acm.org
.............................
%
```

3. Έστω ότι τα εκτελέσιμα προγράμματα που αντιστοιχούν στα παρακάτω προγράμματα C, αριστερά και δεξιά, είναι τα "`whfork1`" και "`whfork2`", αντίστοιχα.

```
#include <stdio.h>
main(int argc, char *argv[])
{  int n, st;
   n = atoi(argv[1]);
   while(n*fork()) {
     printf("%d  %d\n",
            getpid(), getppid());
     n--;
     printf("%d\n", wait(&st)); }
}
```

```
#include <stdio.h>
main(int argc, char *argv[])
{  int n, st;
   n = atoi(argv[1]);
   while(n*!fork()) {
     printf("%d  %d\n",
            getpid(), getppid());
     n--; }
   printf("%d\n", wait(&st));
}
```

Δώστε μία πιθανή εκτύπωση της εντολής "`whfork1 3 ; whfork2 3`". Σχεδιάστε το δέντρο των διεργασιών που θα δημιουργηθούν από την εκτέλεση της εντολής αυτής, περιλαμβάνοντας στο σχήμα σας και τη διεργασία του κελύφους κάτω από το οποίο εκτελέστηκε η εντολή. Εννοείται, ότι θα υποθέσετε ότι όλες οι κλήσεις συστήματος θα λειτουργήσουν χωρίς πρόβλημα.

4. Ένας τρόπος να γνωρίζουμε στο Unix αν έχουμε καινούργια μηνύματα ηλεκτρονικού ταχυδρομείου, χωρίς να ανοίξουμε το γραμματοκιβώτιό μας, είναι να συγκρίνουμε την ώρα τελευταίας τροποποίησης του γραμματοκιβωτίου με αυτήν της τελευταίας προσπέλασης σ΄ αυτό. Αν η πρώτη είναι μεταγενέστερη της δεύτερης, τότε έχουμε καινούργια μηνύματα. Γράψτε ένα πρόγραμμα C (έστω ότι ονομάζεται "`chkmail`") το οποίο να χρησιμοποιεί το παραπάνω τρυκ και να εκτυπώνει κατάλληλο μήνυμα, ανάλογα αν έχουμε ή όχι καινούργια μηνύματα ηλεκτρονικού ταχυδρομείου. Θεωρήστε ότι το όνομα του αρχείου που παίζει το ρόλο του γραμματοκιβωτίου ενός χρήστη σ΄ ένα Unix σύστημα είναι η τιμή της μεταβλητής περιβάλλοντος `MAIL` και ότι μπορούμε μέσα από ένα πρόγραμμα C να πάρουμε την τιμή μίας μεταβλητής περιβάλλοντος με τη συνάρτηση βιβλιοθήκης `char *getenv(const char *)`, δίνοντάς της σαν όρισμα το όνομα της μεταβλητής και παίρνοντας στο όνομα της συνάρτησης την τιμή της μεταβλητής. Μία πιθανή εκτέλεση του προγράμματος "`chkmail`" θα μπορούσε να ήταν η εξής:

```
% chkmail
You have new mail
% pine
.....................
.....................
% chkmail
You don't have new mail
% ...................
.....................
% ...................
.....................
% chkmail
You have new mail
%
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Α΄ Περιόδου 2001

1. (α΄) Στον κατάλογο αφετηρίας μας, που έχει δικαιώματα προστασίας `rwxr-xr-x`, έχουμε ένα αρχείο με δικαιώματα προστασίας `rw-------`. Κάποιος δημιουργεί στο δικό του κατάλογο αφετηρίας ένα σκληρό σύνδεσμο στο αρχείο μας, αλλάζει τα δικαιώματα του σκληρού συνδέσμου και μας "κλέβει" τα δεδομένα μας. "Πάσχει" κάπου το προηγούμενο σενάριο; Αν ναι, πού και γιατί;

   (β΄) Σε κάποιο τμήμα ενός προγράμματος, επιθυμούμε να κάνουμε αποκλειστική χρήση ενός πόρου (π.χ. κοινής μνήμης) και για το σκοπό αυτό τον "κλειδώνουμε" πριν τη χρήση και τον "ξεκλειδώνουμε" μετά. Γνωρίζετε ότι μία μέθοδος για "κλείδωμα" και "ξεκλείδωμα" είναι μέσω σηματοφόρων. Μπορείτε να προτείνετε έναν άλλο τρόπο για να γίνουν αυτές οι ενέργειες, για παράδειγμα με τη βοήθεια του συστήματος αρχείων; Αν ναι, με ποιες εντολές σ΄ ένα πρόγραμμα C θα το κατορθώνατε;

2. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`runexps`") το οποίο να δέχεται σαν πρώτο όρισμα έναν αριθμό $\langle N \rangle$ και τα υπόλοιπα ορίσματά του να είναι το όνομα ενός προγράμματος $\langle prog \rangle$ και τα ορίσματα του προγράμματος $\langle arg1 \rangle$, $\langle arg2 \rangle$, ... Υποθέστε ότι το πρόγραμμα $\langle prog \rangle$ κάνει κάποια λειτουργία και παράγει διάφορα αποτελέσματα στην προκαθορισμένη έξοδο, μεταξύ των οποίων και μία ακριβώς γραμμή της μορφής `***Total time***:` $\langle S \rangle$ `seconds`, όπου $\langle S \rangle$ είναι ο χρόνος εκτέλεσης του προγράμματος. Το "`runexps`" που θα γράψετε να εκτελεί $\langle N \rangle$ φορές το πρόγραμμα $\langle prog \rangle$ (με τα ορίσματά του) και να μας πληροφορεί σχετικά με όλους τους χρόνους εκτέλεσης, τον ελάχιστο απ΄ αυτούς, το μέγιστο και τη μέση τιμή τους. Για παράδειγμα:

```
% experim ls -Rl /usr
.............................
***Total time***: 48 seconds
.............................
% runexps 3 experim ls -Rl /usr
Experiment 1: 49 seconds
Experiment 2: 42 seconds
Experiment 3: 46 seconds

Minimum time: 42 seconds
Maximum time: 49 seconds
Average time: 45 seconds
```

3. Γράψτε ένα πρόγραμμα C (υποθέστε ότι ονομάζεται "`succs`") το οποίο να καλείται με έναν αριθμό $\langle N \rangle$ στη γραμμή εντολής. Η αρχική διεργασία-γονέας να δημιουργεί μία διεργασία-παιδί, η οποία να δημιουργεί, με τη σειρά της, μία δική της διεργασία-παιδί, κ.ο.κ. μέχρι να δημιουργηθούν $\langle N \rangle$ απόγονοι της αρχικής διεργασίας. Τόσο οι διεργασίες απόγονοι, όσο και η αρχική διεργασία να εκτυπώνουν μηνύματα, όπως φαίνεται στην παρακάτω εκτέλεση:

```
% succs 3
I am process with PID = 20125 and PPID = 20124
I am process with PID = 20126 and PPID = 20125
I am process with PID = 20127 and PPID = 20126
I am original process (PID = 20124) and I have 3 successors
     with PIDS = 20125 20126 20127
```

Προσέξτε ότι οι γραμμές "`I am process ...`" εκτυπώνονται από τις διεργασίες απογόνους, ενώ οι γραμμές "`I am original process ... with PIDS = ...`" από την αρχική διεργασία. Για να επιτευχθεί αυτό πιθανώς να χρειαστεί να προγραμματίσετε και κάτι περισσότερο απ΄ αυτά που λέγονται ρητά στην εκφώνηση.

4. Αν το εκτελέσιμο πρόγραμμα που προκύπτει από το C πρόγραμμα παρακάτω ονομάζεται "`mfo`", δώστε ένα πιθανό αποτέλεσμα της εντολής "`mfo 3 < test.dat`", υποθέτοντας ότι όλες οι κλήσεις συστήματος θα λειτουργήσουν κανονικά. Τα περιεχόμενα του αρχείου "`test.dat`" φαίνονται δίπλα από το πρόγραμμα. Εξηγήστε επίσης στοιχειωδώς (το πολύ σε 10 γραμμές) τη λειτουργία του προγράμματος.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#define MSGSIZE 1024

void trnsf(char, char, char, char *);
struct message {
  long mtype;
  char mtext[MSGSIZE]; };

main(int argc, char *argv[])
{ int i, k, n, qid, st, pid[100]; struct message mess;
  qid = msgget((key_t) 13562, 0600 | IPC_CREAT);
  n = atoi(argv[1]);
  for (i=0 ; i<n ; i++) {
    pid[i] = fork();
    if (!pid[i]) {
      printf("%d: %d\n", i+1, getpid());
      msgrcv(qid, &mess, MSGSIZE, (long) getpid(), 0);
      while (strcmp(mess.mtext, "END")) {
        printf("%d: %ld: %s\n", i+1, mess.mtype, mess.mtext);
        if (getpid()%2)
          trnsf('A', 'Z', 'a', mess.mtext);
        else
          trnsf('a', 'z', 'A', mess.mtext);
        mess.mtype = (long) getppid();
        msgsnd(qid, &mess, strlen(mess.mtext)+1, 0);
        msgrcv(qid, &mess, MSGSIZE, (long) getpid(), 0); }
      exit(0); } }
  i = 0; k = 0;
  while (gets(mess.mtext) != NULL) {
    mess.mtype = (long) pid[i];
    msgsnd(qid, &mess, strlen(mess.mtext)+1, 0);
    i = (i+1)%n;
    k++; }
  strcpy(mess.mtext, "END");
  for (i=0 ; i<n ; i++) {
    mess.mtype = (long) pid[i];
    msgsnd(qid, &mess, 4, 0); }
  for (i=0 ; i<n ; i++)
    wait(&st);
  for (i=0 ; i<k ; i++) {
    msgrcv(qid, &mess, MSGSIZE, (long) getpid(), 0);
    printf("%s\n", mess.mtext); }
  msgctl(qid, IPC_RMID, (struct msqid_ds *) 0); }

void trnsf(char c1, char c2, char c3, char *s)
{ while (*s != 0) {
    if (*s>=c1 && *s<=c2)
      *s = *s+c3-c1;
    s++; } }
```

```
% cat test.dat
This is a FILE
for testing the
program of the
Syspro Exams on
Friday June 15th.
% mfo 3 < test.dat
.................
.................
.................
.................
.................
.................
.................
.................
.................
.................
.................
.................
.................
```