ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ
Εξετάσεις Β' Περιόδου 2000

1. (α) Παρατηρήστε την παρακάτω διαλογική επικοινωνία με το κέλυφος C. Ποιο θα είναι το αποτέλεσμα της τελευταίας εντολής "`ls`"; Σχετικά με την αμέσως προηγούμενη εντολή "`mv`", ποια ήταν, κατά τη γνώμη σας, η πρόθεση του χρήστη; Πέτυχε το στόχο του; Αν όχι, γιατί;

```
% ls -al *.txt
-rw-r--r--  1 user1         332 Mar 23 23:28 file1.txt
-rw-r-----  1 user1         165 Sep  4 14:27 file2.txt
% ls -al *.doc
No match.
% mv *.txt *.doc
% ls -al *.txt *.doc
.....
```

   (β) Μπορεί μία διεργασία να είναι ταυτόχρονα ζωντανή-νεκρή (zombie) και ορφανή (orphan); Σε ποια περίπτωση; Τι συμβαίνει τότε;

   (γ) Έστω ότι το εκτελέσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα ονομάζεται "`ro`". Ποια εντολή κάτω από το κέλυφος C, στην οποία δεν εμπλέκεται το πρόγραμμα "`ro`", έχει το ίδιο αποτέλεσμα με την εντολή "`ro` $\langle arg1 \rangle$ $\langle arg2 \rangle$ $\langle arg3 \rangle$ $\langle arg4 \rangle$";

```
#include <stdio.h>
#include <sys/file.h>
main(int argc, char *argv[])
{  int fd;
   fd = open(argv[1], O_WRONLY|O_CREAT|O_TRUNC, 0600);
   dup2(fd, 1);
   execvp(argv[2], &argv[2]); }
```

   (δ) Γιατί, κατά τη γνώμη σας, ενώ με την κλήση συστήματος "`write`" μπορούμε να μεταβάλουμε προγραμματιστικά τα περιεχόμενα κοινών αρχείων, δεν μπορούμε να το κάνουμε αυτό για τα περιεχόμενα καταλόγων, και μάλιστα δεν μας παρέχεται κανένας άμεσος τρόπος (π.χ. μέσω κάποιας κλήσης συστήματος) για να γίνει αυτό στο Unix; Ποιους έμμεσους τρόπους γνωρίζετε με τους οποίους μεταβάλλονται προγραμματιστικά τα περιεχόμενα καταλόγων;

2. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`myspell`"), το οποίο να λειτουργεί σαν ένας πολύ απλός ελεγκτής ορθογραφίας. Θεωρήστε ότι όλες οι ορθογραφικά σωστές λέξεις της γλώσσας που μας ενδιαφέρει βρίσκονται καταχωρημένες σ' ένα αρχείο-λεξικό (κάθε λέξη σε μία γραμμή), το πλήρες όνομα-μονοπάτι του οποίου φυλάσσεται σαν τιμή της μεταβλητής περιβάλλοντος "`DICTIONARY`". Το πρόγραμμα "`myspell`" να διαβάζει το αρχείο, την ορθογραφία του οποίου επιθυμείτε να ελέγξετε, από την προκαθορισμένη είσοδο (*stdin*) γραμμή-γραμμή και να ελέγχει σε κάθε βήμα την τρέχουσα γραμμή, αντί να το διαβάζει ολόκληρο στην αρχή και να το επεξεργάζεται στη συνέχεια σαν σύνολο. Ο λόγος είναι ότι το αρχείο αυτό ενδέχεται να είναι πολύ μεγάλο και να υπάρχει πρόβλημα στην καταχώρησή του στην (πιθανώς) περιορισμένη διαθέσιμη κεντρική μνήμη στο πρόγραμμά σας. Σαν αποτέλεσμα, το πρόγραμμα "`myspell`" να δίνει τις λέξεις εκείνες του κειμένου εισόδου που δεν βρίσκονται μέσα στο λεξικό, καθώς τόσο τη γραμμή όσο και τη θέση μέσα στη γραμμή που βρίσκονται αυτές οι λέξεις στην είσοδο. Η διαλογική επικοινωνία που ακολουθεί διευκρινίζει περισσότερο την επιθυμητή συμπεριφορά του "`myspell`":

```
% setenv DICTIONARY /usr/dict/words
% cat inp
this is a test fille to check
whether script myspell is corect
adn get high grade out of it
```

```
% myspell < inp
Line 1, Word 5: fille
Line 2, Word 3: myspell
Line 2, Word 5: corect
Line 3, Word 1: adn
%
```

(*Υπόδειξη:* Μέσα από ένα πρόγραμμα κελύφους Bourne, ο τρόπος με τον οποίο, όταν διαβάζουμε με την εντολή "`read`" γραμμές από την προκαθορισμένη είσοδο, μπορούμε να αναγνωρίσουμε το end-of-file είναι να ελέγξουμε τον κωδικό εξόδου της "`read`", που, στο κέλυφος Bourne, είναι προσπελάσιμος με το "`$?`", δηλαδή το αντίστοιχο του "`$status`" του κελύφους C. Όσο η "`read`" δεν βρίσκει end-of-file, επιστρέφει κωδικό εξόδου `0`.)

3. Τι θα εκτυπωθεί από το εκτελέσιμο πρόγραμμα (έστω ότι ονομάζεται "`pipefork`") που προκύπτει από το παρακάτω C πρόγραμμα, όταν κληθεί σαν "`pipefork 5`", εφ' όσον όλες οι κλήσεις συστήματος λειτουργήσουν χωρίς πρόβλημα; Δικαιολογήστε στοιχειωδώς το αποτέλεσμα. Σε τι θα διαφέρει το αποτέλεσμα αν ενεργοποιήσουμε τη γραμμή του προγράμματος που αυτήν τη στιγμή είναι σχολιασμένη;

```
#include <stdio.h>
main(int argc, char *argv[])
{  int i, n, pid, st, fd[2];
   n = atoi(argv[1]);
   for (i=1 ; i<=n ; i++) {
      pipe(fd);
      pid = fork();
      if (pid) {
         write(fd[1], (char *) &pid, sizeof pid);
         wait(&st);
         break; }
      else {
         printf("%2d %2d %2d %2d ", i, fd[0], fd[1], pid);
         read(fd[0], (char *) &pid, sizeof pid);
         /*  close(fd[0]); close(fd[1]);  */
         printf("%5d %5d %5d\n", getpid(), getppid(), pid); } } }
```

4. Μία συνηθισμένη υπηρεσία που προσφέρει ένας server στο Internet είναι η "`FINGER`", η οποία παρέχεται στη θύρα του server με αριθμό 79. Αν ένα πρόγραμμα client συνδεθεί μέσω υποδοχής ροής με τη θύρα αυτή ενός server, γράψει στη υποδοχή αυτή το όνομα ενός χρήστη (σαν συμβολοσειρά τερματισμένη με το χαρακτήρα αλλαγής γραμμής), και διαβάσει από την υποδοχή αυτή ό,τι του σταλεί, αυτό θα είναι διάφορες πληροφορίες σχετικές με το χρήστη αυτό στο μηχάνημα-server, όπως το πλήρες του όνομα, πότε συνδέθηκε τελευταία φορά, κλπ. Γράψτε ένα πρόγραμμα C (έστω ότι ονομάζεται "`myfinger`") το οποίο να συμπεριφέρεται σαν ένας τέτοιος client. Το πρόγραμμα αυτό να καλείται είτε σαν "`myfinger` ⟨*user*⟩`@`⟨*host*⟩" είτε σαν "`myfinger` ⟨*user*⟩". Στην πρώτη περίπτωση να συνδέεται με τον server ⟨*host*⟩ και να ζητά τις σχετικές πληροφορίες για το χρήστη ⟨*user*⟩, ενώ στη δεύτερη περίπτωση να ζητά τις πληροφορίες για το ⟨*user*⟩ από το ίδιο το μηχάνημα στο οποίο εκτελείται ο client. Ενδεικτικές εκτελέσεις του προγράμματος είναι οι εξής:

```
% myfinger syspro@gaia.di.uoa.gr
Login       Name               TTY         Idle    When    Where
syspro   Syspro account        pts/5        <Sep  8 21:19> armagedon.di.uoa
% myfinger georges
Login name: georges                     In real life: George Spiliotis
Directory: /usr/users/georges           Shell: /bin/csh
Last login Sun Mar 17, 1996 on ttyp0 from zeus.di.uoa.gr
No Plan.
%
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Α' Περιόδου 2000

1. (α) Παρατηρήστε την παρακάτω διαλογική επικοινωνία με το κέλυφος C. Τι σημαίνει το 2 που βρίσκεται στην τρίτη γραμμή (πριν το `nick`) και γιατί είναι 2 και όχι κάτι άλλο; Τι θα εκτυπωθεί στη γραμμή με τις τελείες και γιατί;

```
% ls -al bla
total 2
drwxr-xr-x  2 nick          512 Jun 10 08:28 .
drwxr-xr-x 15 nick         1024 Jun 10 08:28 ..
% ls -al | grep '^d' | wc -l
.......................
%
```

   (β) Έστω ότι έχουμε γράψει ένα πρόγραμμα "`pipeline`" που υλοποιεί, με τον τρόπο που γίνεται στα διάφορα κελύφη, μία σωλήνωση μεταξύ N εντολών, τις οποίες δέχεται στα ορίσματά του, χωρισμένες μεταξύ τους με το γνωστό μας σύμβολο της σωλήνωσης. Εκ των υστέρων, παρατηρούμε ότι είτε εκτελέσουμε το πρόγραμμά μας σαν "`pipeline` $\langle C_1 \rangle$ `"|"` $\langle C_2 \rangle$ `"|"` ... `"|"` $\langle C_N \rangle$" είτε σαν "`pipeline` $\langle C_1 \rangle$ `|` $\langle C_2 \rangle$ `|` ... `|` $\langle C_N \rangle$", παίρνουμε τα ίδια αποτελέσματα. Μπορείτε να δώσετε κάποια εξήγηση γι' αυτό; Πώς θα πρέπει να συμπεριφέρεται το πρόγραμμα "`pipeline`" για την περίπτωση $N = 1$, έτσι ώστε να έχουμε το φαινόμενο που προαναφέρθηκε;

   (γ) Τι σχέση έχει η εντολή "`kill`" με την κλήση συστήματος "`kill`"; Τι σχέση έχουν και οι δύο με το όνομά τους (kill = δολοφονώ);

   (δ) Έστω ότι ενδιαφέρεστε να υλοποιήσετε ένα server, ο οποίος όταν δέχεται από κάποιο client μία συμβολοσειρά, να επιστρέφει στον client το μήκος της συμβολοσειράς. Για το σκοπό αυτό, μπορείτε στο πρόγραμμα του server να δημιουργήσετε μία υποδοχή ροής, να συνδέσετε την υποδοχή με την κατάλληλη διεύθυνση και να να εγκαταστήσετε μία ουρά αναμονής αιτήσεων απο clients. Όταν γίνει αποδεκτή από το server μία αίτηση εξυπηρέτησης ενός client, δημιουργεί ο server μία διεργασία-παιδί, η οποία διαβάζει τη συμβολοσειρά που στέλνει ο client, υπολογίζει το μήκος της και το επιστρέφει στον client, ενώ, ταυτόχρονα, η διεργασία-γονέας server περιμένει νέες αιτήσεις σύνδεσης από clients. Έχετε να σχολιάσετε κάτι σ' αυτό το σενάριο; Αν το σχόλιό σας είναι αρνητικό, μπορείτε να κάνετε κάποια αντιπρόταση για την αντιμετώπιση του προβλήματος;

2. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`comp_res`"), το οποίο να δέχεται, κατά σειρά, στα ορίσματά του τα ονόματα αρχείων κάποιων εκτελέσιμων προγραμμάτων, τη λέξη-κλειδί "`-data`" και τα ονόματα μερικών αρχείων δεδομένων. Τα εκτελέσιμα προγράμματα είναι έτσι κατασκευασμένα ώστε να διαβάζουν τα δεδομένα τους από την προκαθορισμένη είσοδο και να εκτυπώνουν τα αποτελέσματά τους στην προκαθορισμένη έξοδο. Όσον αφορά τη λειτουργικότητα του "`comp_res`", θέλουμε να μας πληροφορεί, για κάθε ζευγάρι εκτελέσιμων προγραμμάτων, από αυτά που του δόθηκαν στη γραμμή εντολής, και για κάθε περίπτωση δεδομένων εισόδου, όπως αυτά περιέχονται στα αρχεία δεδομένων που επίσης δόθηκαν στη γραμμή εντολής, αν τα προγράμματα αυτά δίνουν ακριβώς τα ίδια αποτελέσματα. Ένα παράδειγμα εκτέλεσης του προγράμματος είναι το εξής:

```
% comp_res exfile1 exfile2 exfile3 exfile4 -data data1 data2
exfile1 and exfile2 give different results with data1
exfile1 and exfile2 give the same  results with data2
exfile1 and exfile3 give the same  results with data1
exfile1 and exfile3 give the same  results with data2
exfile1 and exfile4 give different results with data1
exfile1 and exfile4 give the same  results with data2
exfile2 and exfile3 give different results with data1
exfile2 and exfile3 give the same  results with data2
```

```
exfile2 and exfile4 give different results with data1
exfile2 and exfile4 give the same  results with data2
exfile3 and exfile4 give different results with data1
exfile3 and exfile4 give the same  results with data2
%
```

(*Υπόδειξη:* Το αντίστοιχο στο κέλυφος Bourne του "`$status`" του κελύφους C είναι το "`$?`". Ισχύει, γενικά, ότι εντολές που τερματίζουν χωρίς πρόβλημα επιστρέφουν στο γονέα τους σαν κωδικό εξόδου το `0`. Για παράδειγμα, αν με την εντολή "`cmp`" δοκιμάσουμε να συγκρίνουμε τα περιεχόμενα δύο αρχείων, αν αυτά είναι εντελώς ίδια, η "`cmp`" θα τερματίσει με κωδικό εξόδου το `0`.)

3. Τι θα εκτυπωθεί από το εκτελέσιμο πρόγραμμα (έστω ότι ονομάζεται "`sigforks`") που προκύπτει από το παρακάτω C πρόγραμμα, όταν κληθεί σαν "`sigforks 3`", εφ' όσον όλες οι κλήσεις συστήματος λειτουργήσουν χωρίς πρόβλημα; Δικαιολογήστε στοιχειωδώς το αποτέλεσμα. Πόσες διεργασίες δημιουργούνται από το πρόγραμμα, όταν αυτό καλείται με τον ακέραιο N στη γραμμή εντολής;

```
#include <stdio.h>
#include <signal.h>
int n; void sh();
main(int argc, char *argv[])
{ int st;
  n = atoi(argv[1]);
  signal(SIGUSR1, sh);
  while (n) kill(getpid(), SIGUSR1);
  while (wait(&st) > 0); }
void sh()
{ printf("%d: %d %d\n", n, getpid(), getppid());
  n--; fork(); }
```

4. Στη συνέχεια φαίνεται (αριστερά) τμήμα ενός προγράμματος C, από το οποίο προκύπτει ένα εκτελέσιμο πρόγραμμα με όνομα "`race`" και (δεξιά) ένα πιθανό αποτέλεσμα εκτέλεσης του προγράμματος αυτού.

```
.....................................
main(int argc, char *argv[])
{ int pid1, pid2, status;
  n = atoi(argv[1]);
  if ((shmid = shmget(SHMKEY, ........
  if ((semid = semget(SEMKEY, ........
  if ((shmem = shmat(shmid, ..........
  if (semctl(..., SETVAL, 1) < 0) { ..
  strcpy(shmem, "1");
  if ((pid1 = fork()) == -1) { .......
  if (pid1 == 0) dochild();
  if ((pid2 = fork()) == -1) { .......
  if (pid2 == 0) dochild();
  ..................................... }
void dochild()
{ ..................................
    printf("%d %s\n", getpid(), shmem);
  ..................................... }
```

```
% race 8
7972 1
7973 2
7972 3
7973 4
7972 5
7973 6
7972 7
7973 8
7972 9
7973 10
7972 11
7972 12
7973 13
7972 14
7973 15
7973 16
%
```

Συμπληρώστε κατάλληλα το πρόγραμμα αυτό έτσι ώστε όταν καλείται με έναν ακέραιο N στη γραμμή εντολής, κάθε διεργασία-παιδί της αρχικής διεργασίας να κάνει N επαναλήψεις, σε κάθε μία από τις οποίες να αυξάνει κατά 1 την αριθμητική τιμή της καταχωρημένης συμβολοσειράς στην κοινή μνήμη (που προσδιορίζεται από το `shmid`). Φροντίστε, σε κάθε επανάληψη, κάθε διεργασία-παιδί να ελέγχει την πρόσβασή της στην κοινή μνήμη μέσω ενός σηματοφόρου (που προσδιορίζεται από το `semid`).

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Β' Περιόδου 1999

1. (α) Έχει νόημα να έχουμε για κάποιο αρχείο δικαίωμα εκτέλεσης (`x`), όχι όμως και δικαίωμα ανάγνωσης (`r`); Για κάποιο κατάλογο;

   (β) Τι θα εκτυπωθεί στις διακεκομμένες γραμμές που φαίνονται στη συνέχεια; Δικαιολογήστε την απάντησή σας.

```
% pwd
/usr/users/spro
% cat mycd
#! /bin/sh5
cd $1
pwd
% mkdir newdir
% mycd newdir
.................
% pwd
.................
%
```

   (γ) Υπάρχουν περιπτώσεις που δεν μας εξυπηρετούν οι κλήσεις συστήματος "`execl`" (ή "`execlp`") και χρειαζόμαστε τις "`execv`" (ή "`execvp`"); Το αντίστροφο συμβαίνει ποτέ; Αν ναι, πότε; Αν όχι, γιατί;

   (δ) "*Ενόσω είμαστε συνδεδεμένοι σ' ένα σύστημα Unix, αν πατήσουμε Ctrl-Z στο πληκτρολόγιο, η διεργασία που εκτελείται στο προσκήνιο σταματά*". Είναι σωστό αυτό; Αν όχι, διορθώστε το.

2. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`maxfile`"), το οποίο, αφού διαβάσει το όνομα ενός καταλόγου και ένα όνομα αρχείου κατά την εκτέλεση του, εκτός αν κάποιο απ' αυτά (ή και τα δύο) του έχουν δοθεί από τη γραμμή εντολής με τις επιλογές "`-dir`" και "`-file`" αντίστοιχα, να βρίσκει στην ιεραρχική δομή κάτω από τον κατάλογο ποιο αρχείο με το δεδομένο όνομα έχει το μεγαλύτερο μέγεθος σε bytes. Ενδεικτικά παραδείγματα εκτέλεσης του προγράμματος είναι τα εξής:

```
% maxfile
Give directory to search in: ./hw
Give filename to search for: sendfile.c
Maximum file: ./hw/950/sendfile.c
Size:     8511 bytes
% maxfile -file cycle
Give directory to search in: hw
Maximum file: hw/924/cycle
Size:    60220 bytes
% maxfile -file ourprinters -dir ~spro/hw
Maximum file: /usr/users/spro/hw/936/ouprinters
Size:     5953 bytes
%
```

3. Τι θα εκτυπωθεί από το εκτελέσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα, εφ' όσον οι τρεις `fork()` λειτουργήσουν χωρίς πρόβλημα; Δικαιολογήστε στοιχειωδώς το αποτέλεσμα.

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

4. Γράψτε δύο προγράμματα C (έστω ότι ονομάζονται "`sender`" και "`receiver`") τέτοια ώστε το πρώτο να στέλνει ένα αρχείο στο δεύτερο μέσω μίας ουράς μηνυμάτων. Το όνομα του αρχείου για αποστολή να δίνεται στο πρόγραμμα "`sender`" σαν ένα όρισμα, ενώ το πρόγραμμα "`receiver`" να εκτυπώνει τα περιεχόμενα του αρχείου που παραλαμβάνει μέσω της ουράς μηνυμάτων στην προκαθορισμένη έξοδο. Για να εξασφαλιστεί ότι τα δύο προγράμματα θα προσπελάσουν την ίδια ουρά μηνυμάτων, το κλειδί (τύπου `key_t`) που θα αντιστοιχεί στην ουρά αυτή να είναι ο inode του αρχείου `/etc/passwd` (τον οποίο φυσικά πρέπει να βρίσκουν τα δύο προγράμματα). Η ουρά μηνυμάτων να δημιουργείται από εκείνο από τα δύο προγράμματα ("`sender`" ή "`receiver`") που αρχίζει να εκτελείται πρώτο, ενώ αυτό που εκτελείται δεύτερο απλώς θα προσπελαύνει την ήδη δημιουργημένη ουρά. Φροντίστε επίσης, μετά τη χρησιμοποίησή της, η ουρά να καταστρέφεται (από το πρόγραμμα "`receiver`" – *γιατί;*). Πληροφοριακά, η κλήση συστήματος "`msgrcv`" επιστρέφει στο όνομά της, σε περίπτωση επιτυχούς εκτέλεσης, το πραγματικό μέγεθος του μηνύματος που διάβασε. Προσέξτε επίσης ότι στην επικοινωνία μέσω ουρών μηνυμάτων δεν υφίσταται η έννοια του EOF, οπότε πρέπει να βρείτε έναν τρόπο να καταλάβει ο "`receiver`" πότε ο "`sender`" δεν έχει να του στείλει άλλα δεδομένα. Ενδεικτικές εκτελέσεις των δύο προγραμμάτων φαίνονται στη συνέχεια:

```
% cat test_file
This is a file to be used
for testing the server and
receiver programs.
% sender test_file
/etc/passwd inode is 4095
Referencing msgq with id 502
Sending 72 bytes
Done!
%
```

```
% receiver
/etc/passwd inode is 4095
Referencing msgq with id 502
Receiving 72 bytes
This is a file to be used
for testing the server and
receiver programs.
Done!
Removed msgq with id 502
%
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Α' Περιόδου 1999

1. (α) Ως γνωστόν, στη μεταβλητή περιβάλλοντος `PATH` περιέχονται οι κατάλογοι εκείνοι οι οποίοι θέλουμε να εξερευνώνται, κατά σειρά από τον πρώτο προς τον τελευταίο, για τον εντοπισμό προγραμμάτων των οποίων ζητάμε την εκτέλεση. Πιστεύετε ότι είναι καλή ιδέα να υπάρχει μεταξύ των καταλόγων αυτών και ο τρέχων κατάλογος ".", ή η πρόταση αυτή εμπεριέχει κάποιους κινδύνους; Εξηγήστε την άποψή σας. Αν υπάρχει ο τρέχων κατάλογος στη μεταβλητή `PATH`, είναι καλό να είναι πρώτος ή τελευταίος και γιατί;

   (β) Η εντολή `mv` μπορεί να χρησιμοποιηθεί είτε για τη μετονομασία ενός αρχείου ή καταλόγου από `f1` σε `f2` (δίνοντας "`mv f1 f2`") είτε για τη μετακίνηση ενός πλήθους αρχείων ή καταλόγων `f1`, `f2`, ..., `fn` σε ένα κατάλογο `d1` (δίνοντας "`mv f1 f2 ... fn d1`"). Αν υποθέσουμε ότι δουλεύουμε με το κέλυφος C, εξηγήστε γιατί δεν είναι δυνατόν να μετονομάσουμε όλα τα αρχεία του τρέχοντος καταλόγου με επέκταση `bak` έτσι ώστε να έχουν επέκταση `old` με την εντολή "`mv *.bak *.old`". Περιγράψτε μία κατάσταση περιεχομένων του τρέχοντος καταλόγου στην οποία η εντολή αυτή θα δώσει μήνυμα λάθους, μία δεύτερη κατάσταση στην οποία η εντολή θα κάνει (περίπου) αυτό που θα θέλαμε και μία τρίτη στην οποία η εντολή θα κάνει κάτι πραγματικά ανεπιθύμητο (ίσως και τραγικό) για εμάς.

   (γ) Ως γνωστόν, για τον προγραμματισμό στο κέλυφος Bourne έχουμε τη δυνατότητα να χρησιμοποιήσουμε τα `$1`, `$2`, `$3`, ..., `$9` για να προσπελάσουμε το 1ο, 2ο, 3ο, ..., 9ο όρισμα, αντίστοιχα, ενός προγράμματος κελύφους. Δεν είναι περιοριστικό αυτό όσον αφορά τον αριθμό των ορισμάτων που μπορούμε να δώσουμε σε ένα πρόγραμμα κελύφους Bourne; Εσείς στα προγράμματα κελύφους Bourne που έχετε γράψει, ποια από τα `$1`, `$2`, ... χρησιμοποιήσατε; Μέχρι πόσα ορίσματα δίνατε στα προγράμματα αυτά όταν τα εκτελούσατε;

   (δ) Ποια πιστεύετε ότι ήταν η πρόθεση του προγραμματιστή που έγραψε το παρακάτω τμήμα προγράμματος για το κέλυφος Bourne; Τι λάθη έκανε;

```
#! /bin/sh5
.................
if [ ! -f ~/.login ]
then
   wc /etc/hosts /etc/machines >& file
fi
.................
```

2. Η εντολή "`finger`" μπορεί να δώσει ένα πλήθος από πληροφορίες για ένα χρήστη με δεδομένη ταυτότητα. Ας υποθέσουμε ότι σε ένα συγκεκριμένο σύστημα η μορφή της εξόδου της εντολής αυτής είναι η εξής:

```
% finger georges
Login name: georges                     In real life: George Spiliotis
Directory: /usr/users/georges           Shell: /bin/csh
Last login Sun Mar 17, 1996 on ttyp0 from zeus.di.uoa.gr
No Plan.
%
```

   Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`namesof`"), το οποίο, για ένα σύνολο από χρήστες των οποίων τις ταυτότητες δίνουμε στη γραμμή εντολής κατά την εκτέλεσή του, να μας πληροφορεί για τα πραγματικά τους ονόματα. Για το σκοπό αυτό, το πρόγραμμα να εκμεταλλεύεται τις πληροφορίες των κατάλληλων εντολών "`finger`". Θεωρήστε ότι στην έξοδο τέτοιων εντολών, το πραγματικό όνομα ενός χρήστη βρίσκεται μετα το "`In real life:`" και πριν το "`Directory:`". Ένα παράδειγμα εκτέλεσης θα μπορούσε να ήταν το εξής:

```
% namesof georges phys30 ea11 marianna
Name of georges is George Spiliotis
Name of phys30 is physics
Name of ea11 is User from EA
Name of marianna is Marianna Vlastou
%
```

3. Γράψτε ένα πρόγραμμα C (υποθέστε ότι ονομάζεται "`pingpong`") το οποίο να καλείται με έναν αριθμό $\langle N \rangle$ στη γραμμή εντολής. Η αρχική διεργασία-γονέας του προγράμματος να δημιουργεί μία διεργασία-παιδί και, στη συνέχεια, να αρχίσουν οι δύο διεργασίες να ανταλλάσσουν μεταξύ τους σήματα `SIGUSR1` και `SIGUSR2`. Συγκεκριμένα, αρχικά ο γονέας να στείλει ένα σήμα `SIGUSR1` στο παιδί, το οποίο, αφού το λάβει, να απαντήσει στο γονέα με ένα σήμα `SIGUSR2`. Όταν ο γονέας λάβει το `SIGUSR2`, να ξαναστείλει στο παιδί ένα `SIGUSR1` και ούτω καθεξής. Αυτές οι ανταλλαγές σημάτων να συνεχιστούν μέχρι ο γονέας να έχει στείλει $\langle N \rangle$ σήματα `SIGUSR1` και να έχει λάβει και $\langle N \rangle$ απαντήσεις `SIGUSR2` από το παιδί. Όταν το παιδί στείλει και το τελευταίο σήμα να τερματίσει και, αφού ο γονέας αντιληφθεί τον τερματισμό του παιδιού, να τερματίσει και αυτός. Ένα παράδειγμα εκτέλεσης του προγράμματος είναι το εξής:

```
% pingpong 3
2207 sending SIGUSR1 to 2208 - ping 1
2208 sending SIGUSR2 to 2207 - pong 1
2207 sending SIGUSR1 to 2208 - ping 2
2208 sending SIGUSR2 to 2207 - pong 2
2207 sending SIGUSR1 to 2208 - ping 3
2208 sending SIGUSR2 to 2207 - pong 3
%
```

4. Μία συνηθισμένη υπηρεσία που προσφέρει ένας server στο Internet ειναι η "`DAYTIME`", η οποία παρέχεται στη θύρα του server με αριθμό 13. Αν ένα πρόγραμμα client συνδεθεί μέσω υποδοχής ροής με τη θύρα αυτή ενός server και διαβάσει από την υποδοχή αυτή ό,τι του σταλεί, αυτό θα είναι η τοπική ημερομηνία και ώρα του server. Γράψτε ένα πρόγραμμα C (έστω ότι ονομάζεται "`daytimeat`") το οποίο να συμπεριφέρεται σαν ένας τέτοιος client. Δηλαδή, αν δίνουμε στη γραμμή εντολής του προγράμματος αυτού ένα όνομα server (κατά τα καθιερωμένα συμβολικά ονόματα του Internet), το πρόγραμμα αυτό να ζητά από το server την τοπική του ημερομηνία και ώρα και να μας πληροφορεί γι' αυτήν. Ενδεικτικές εκτελέσεις του προγράμματος είναι οι εξής:

```
% daytimeat gaia.di.uoa.gr
Daytime at gaia.di.uoa.gr is Sun Jun 20 19:17:05 1999
% daytimeat archie.au
Daytime at archie.au is Mon Jun 21 02:18:11 1999
% daytimeat doc.ic.ac.uk
Daytime at doc.ic.ac.uk is Sun Jun 20 17:18:20 1999
% daytimeat www.acm.org
Daytime at www.acm.org is Sun Jun 20 12:16:34 1999
%
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Β' Περιόδου 1998

1. (α) Ποιος είναι ο ρόλος των αρχείων `.login` και `.cshrc` στο Unix; Σε ποιο κατάλογο έχει νόημα να βρίσκονται, πότε μας ενδιαφέρει να προσθέσουμε κάτι στο ένα και πότε στο άλλο;

   (β) Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`results`"), το οποίο να διαβάζει από την προκαθορισμένη είσοδο, άρα και από αρχείο με ανακατεύθυνση, γραμμές που δίνει ο διδάσκων ενός μαθήματος, η κάθε μία από τις οποίες περιλαμβάνει τον αριθμό μητρώου ενός φοιτητή και το βαθμό που πήρε στο μάθημα αυτό (από `0` έως `10`). Θεωρήστε σαν τέλος της εισόδου μία γραμμή που περιέχει μόνο το `0`. Για τη διευκόλυνσή σας μπορείτε να υποθέσετε ότι η είσοδος δίνεται με γνωστή εκ των προτέρων στοίχιση, για παράδειγμα ότι ο αριθμός μητρώου είναι τετραψήφιος στις στήλες 1-4, η 5η στήλη είναι κενή και ο βαθμός βρίσκεται στη στήλη 6 (ή στις στήλες 6-7 αν είναι `10`). Το πρόγραμμα "`results`" να στέλνει σε κάθε φοιτητή ένα μήνυμα ηλεκτρονικού ταχυδρομείου για να τον πληροφορήσει για το βαθμό που πήρε στο μάθημα μαζί με κάποιο σύντομο σχόλιο για την επίδοσή του (αντίστοιχο με το βαθμό του). Η διεύθυνση του φοιτητή με αριθμό μητρώου $\langle xxxx \rangle$ μπορείτε να υποθέσετε ότι είναι η `stud`$\langle xxxx \rangle$. (*Υπόδειξη:* Με την εντολή "`mail -s` $\langle subject \rangle$ $\langle address \rangle$" αποστέλλεται με ένα μήνυμα ηλεκτρονικού ταχυδρομείου η προκαθορισμένη είσοδός της στη διεύθυνση $\langle address \rangle$, όπου $\langle subject \rangle$ είναι η επικεφαλίδα του μηνύματος.) Στη συνέχεια φαίνονται ένα αρχείο εισόδου και μία ενδεικτική εκτέλεση του προγράμματος.

```
% cat results.dat
2312 7
2134 10
2257 3
0
% results < results.dat
Informing 2312: Very good 2312, you got 7.
Informing 2134: Excellent 2134, you got 10.
Informing 2257: Next time you have to study more 2257, you got 3.
%
```

2. (α) Όταν μία διεργασία καλέσει την κλήση συστήματος `exit(47)`, τι συμβαίνει; Τι είναι αυτό το `47` και σε ποιον μπορεί να είναι χρήσιμο;

   (β) Τι θα εκτυπωθεί από το εκτελέσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα, εφ'όσον και οι τρεις `fork()` λειτουργήσουν χωρίς πρόβλημα;

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

3. (α) Δώστε το απαιτούμενο τμήμα προγράμματος C, μαζί με τις δηλώσεις τυχόν μεταβλητών που χρειάζονται, με το οποίο να ανταλάσσονται οι διαχειριστές σήματος των σημάτων `SIGUSR1` και `SIGUSR2`.

   (β) Έστω ότι το εκτελέσιμο πρόγραμμα που αντιστοιχεί στο παρακάτω C πρόγραμμα ονομάζεται "`mytee`".

```
#include <stdio.h>
main()
{  char s[80];
   gets(s);
   printf("%d %d %s\n", getpid(), getppid(), s);
}
```

   Τι θα εκτυπωθεί με κάθε μία από τις παρακάτω εντολές;

```
% mytee | mytee | mytee
.............................
.............................
% mytee ; mytee ; mytee
.............................
.............................
% (mytee ; mytee ; mytee)
.............................
.............................
%
```

   Όπου το "`mytee`" χρειάζεται, λόγω της `gets(s)`, να διαβάσει μία γραμμή εισόδου, θεωρήστε ότι δίνεται το: `test`.

4. (α) Ποιες μεθόδους επικοινωνίας τύπου System V μεταξύ διεργασιών γνωρίζετε και σε ποιες περιπτώσεις είναι χρήσιμες;

   (β) Γράψτε ένα πρόγραμμα C (έστω ότι το αντίστοιχο εκτελέσιμο ονομάζεται "`mode`"), το οποίο όταν καλείται με όρισμα ένα όνομα αρχείου, καταλόγου κ.λ.π., να βρίσκει τα δικαιώματα προστασίας του, με πρόσβαση στην εσωτερική οργάνωση του συστήματος αρχείων μέσω κατάλληλης κλήσης συστήματος, και να τα εκτυπώνει. Για παράδειγμα:

```
% mode a_file
rw-r-----
% mode /usr/lib
rwxr-xr-x
%
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Α' Περιόδου (Ιουνίου) 1998

1. (α) Ενόσω ένας χρήστης κάποιου συστήματος Unix αλληλεπιδρά διαλογικά με το κέλυφος C, είναι πιθανόν να θελήσει να διευκρινήσει κάποιο σημείο της λειτουργικότητας του κελύφους Bourne. Για παράδειγμα, θα μπορούσε να διερωτηθεί αν στη συμβολοσειρά που διαβάζεται από την είσοδο με την εντολή `read` διατηρούνται οι κενοί χαρακτήρες που ίσως έχουν δοθεί. Εκτός από το να καταφύγει σε βιβλία και εγχειρίδια και εκτός από το να γράψει κάποιο πρόγραμμα για το κέλυφος Bourne που, αφού το εκτελέσει, να δώσει απάντηση στο ερώτημά του, μπορεί να κάνει κάτι συντομότερο και απλούστερο για να μάθει αυτό που θέλει;

   (β) Ο διδάσκων ενός μαθήματος έχει ζητήσει από τους φοιτητές να παραδώσουν 9 εργασίες, κάθε μία από τις οποίες συνίσταται στη συγγραφή ενός προγράμματος C σ'ένα σύστημα Unix. Ο κωδικός του διδάσκοντα στο σύστημα αυτό είναι `teacher` και των φοιτητών `stud`$\langle xxxx \rangle$, όπου $\langle xxxx \rangle$ είναι ο αριθμός μητρώου του καθενός. Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται `subhw`) το οποίο να διευκολύνει τη διαδικασία παράδοσης των εργασιών, υπό την έννοια ότι θα πρέπει να εκτελείται από κάποιο φοιτητή με την εντολή "`subhw` $\langle N \rangle$ $\langle file \rangle$" όταν αυτός θέλει να υποβάλει το αρχείο $\langle file \rangle$ σαν απάντηση στην $\langle N \rangle$-οστή εργασία. Σαν αποτέλεσμα της εκτέλεσης του προγράμματος, θα πρέπει να αντιγραφεί το αρχείο $\langle file \rangle$ στον κατάλογο "`/users/teacher/hw`". Το όνομα με το οποίο θα αντιγραφεί το αρχείο πρέπει να είναι "`erg-`$\langle N \rangle$`-`$\langle xxxx \rangle$`.c`". Τέλος, το πρόγραμμα `subhw` να ενημερώνει μέσω του ηλεκτρονικού ταχυδρομείου τον `teacher` ότι έγινε η συγκεκριμένη υποβολή και στον `stud`$\langle xxxx \rangle$ να επιστρέφει, επίσης μέσω ηλεκτρονικού ταχυδρομείου, το $\langle file \rangle$ που υπέβαλε. (*Υποδείξεις:* Με την εντολή "`mail -s` $\langle subject \rangle$ $\langle address \rangle$" αποστέλλεται με ένα μήνυμα ηλεκτρονικού ταχυδρομείου η προκαθορισμένη είσοδός της στη διεύθυνση $\langle address \rangle$, όπου $\langle subject \rangle$ είναι η επικεφαλίδα του μηνύματος. Με την εντολή "`whoami`" εκτυπώνεται ο κωδικός του χρήστη που την εκτελεί.) Υπάρχει, κατά τη γνώμη σας, κάποιο σοβαρό πρόβλημα ουσίας σχετικό με τα δικαιώματα προστασίας που πρέπει να έχει το αρχείο αφού αντιγραφεί στον επιθυμητό κατάλογο; Αν ναι, απλώς επισημάνετέ το, χωρίς να προσπαθήσετε να δώσετε λύση.

2. (α) Είναι γνωστό ότι με την εντολή `ln` δημιουργείται ένας νέος σκληρός σύνδεσμος προς ένα ήδη υπάρχον αρχείο (π.χ. "`ln oldfile newlink`"). Τι θα εκτυπωθεί στις διάστικτες γραμμές παρακάτω; Αν νομίζετε ότι κάτι παρουσιάζει ενδιαφέρον, σχολιάστε το.

```
% ls -l file*
-rw-------  1 user          261 Jun 24 16:18 file1
% ln file1 file2
% chmod 644 file2
% ls -l file*
.................................................
.................................................
%
```

   (β) Τι θα εκτυπωθεί από το εκτελέσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα, εφ'όσον και οι τρείς `fork()` λειτουργήσουν χωρίς πρόβλημα;

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

3. (α) Ποια από τα παρακάτω είναι σωστά και ποια λάθος; Σ'αυτά που είναι λάθος, διορθώστε την υπογραμμισμένη έκφραση. Με την κλήση συστήματος

   - `signal`, η καλούσα διεργασία αποστέλλει ένα σήμα σε μία διεργασία.
   - `pause`, η καλούσα διεργασία τίθεται σε αναμονή μέχρι την παραλαβή ενός σήματος.
   - `kill`, η καλούσα διεργασία προκαλεί το βίαιο τερματισμό μίας διεργασίας.
   - `alarm`, η καλούσα διεργασία αποστέλλει ένα σήμα `SIGALRM` σε μία διεργασία.

   (β) Συμπληρώστε το παρακάτω τμήμα προγράμματος C του οποίου όταν το προκύπτον εκτελέσιμο καλείται σαν "`pipeline` ⟨$c1$⟩ "`|`" ⟨$c2$⟩ "`|`" ... "`|`" ⟨cN⟩", το αποτέλεσμα να είναι το ίδιο με αυτό της εκτέλεσης της σωλήνωσης "⟨$c1$⟩ | ⟨$c2$⟩ | ... | ⟨cN⟩" στο κέλυφος. Γιατί δίνουμε τα | στο πρόγραμμα `pipeline` μέσα σε "; Τι θα γινόταν αν τα αφήναμε χωρίς ";

```
int argind, commstart, pipeno, i, pid, status, fd[20][2];        /* Δηλώσεις */
argind = 1;                                                    /* Αρχικό όρισμα */
pipeno = 0;                                                 /* A/A τρέχοντος σωλήνα */
while (argind != argc+1) {                                     /* Τέλος ορισμάτων; */
   commstart = argind;                                          /* Αρχή εντολής */
   while (argind < argc && strcmp(argv[argind], "|"))        /* Τέλος εντολής; */
      argind++;                                               /* Επόμενο όρισμα */
   argv[argind] = NULL;                              /* Για τις ανάγκες της execvp */
   argind++;                                          /* Αρχή επόμενης εντολής */
   pipeno++;                                                /* Νέος A/A σωλήνα */
   if (argind != argc+1)                                    /* Χρειάζεται σωλήνας; */
      pipe(fd[pipeno-1]);                                   /* Ναι, δημιούργησέ τον */
   pid = fork();                                            /* Δημιούργησε παιδί */
   if (pid == 0) {                                                  /* Το παιδί */
      if (pipeno != 1)      { ............ }
      if (argind != argc+1) { ............ }
      execvp(.............................); }   /* Εκτέλεσε τρέχουσα εντολή */
   else {                                                          /* Ο πατέρας */
      if (pipeno != 1)      { ............ }
      if (argind != argc+1) { ............ } } }
.......................................    /* Αναμονή τερματισμού παιδιών */
```

4. (α) Γιατί μας χρειάζονται οι κλήσεις συστήματος `recvfrom` και `sendto` όταν προγραμματίζουμε επικοινωνία μεταξύ διεργασιών μέσω τηλεγραφικών υποδοχών και δεν μας αρκούν οι `read` και `write`;

   (β) Έστω ότι μέσω των κατάλληλων κλήσεων συστήματος είναι υλοποιημένο το παρακάτω σύνολο συναρτήσεων C για τη διαχείριση σηματοφόρων:

   - int **newsem**(key_t *key*): Επιστρέφει έναν προσδιοριστή για το σηματοφόρο με κλειδί *key*.
   - int **seminc**(int *semid*): Αυξάνει, με ατομική πράξη, το σηματοφόρο με προσδιοριστή *semid* κατά 1.
   - int **semdec**(int *semid*): Μειώνει, με ατομική πράξη, το σηματοφόρο με προσδιοριστή *semid* κατά 1, αν έχει τιμή θετική, αλλοιώς μπλοκάρει μέχρι να γίνει.

   Με τη βοήθεια αυτών των συναρτήσεων, γράψτε μία συνάρτηση C, η οποία θα παίρνει σαν όρισμα ένα όνομα αρχείου και ένα κλειδί που αντιστοιχεί σε σηματοφόρο. Η συνάρτηση αυτή αφού "δεσμεύσει" το αρχείο μέσω του σηματοφόρου που αντιστοιχεί στο δοθέν κλειδί, θα διαβάζει την πρώτη γραμμή του αρχείου, η οποία θα περιέχει έναν ακέραιο, και στην οποία θα ξαναγράφει τον ακέραιο που διάβασε αυξημένο κατά 1. Μετά, βέβαια, πρέπει να "αποδεσμεύσει" το αρχείο. Θεωρήστε ότι ο χρησιμοποιούμενος σηματοφόρος έχει αρχικοποιηθεί κατάλληλα από κάποια διεργασία.

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Α' Περιόδου (Φεβρουαρίου) 1998

1. (α) Κάποιος χρήστης ενός συστήματος UNIX είναι ιδιοκτήτης ενός καταλόγου στον οποίο θέλει να δώσει δικαιώματα προστασίας `rwx--x--x`, ενώ στα αρχεία και καταλόγους που περιέχονται σ'αυτόν τον κατάλογο θέλει να δώσει δικαιώματα προστασίας `rw-r--r--`. Τι σημαίνουν αυτά τα δικαιώματα και με ποιες εντολές μπορούν να δοθούν; Για ποιο λόγο πιστεύετε ότι ο συγκεκριμένος χρήστης θα ήθελε να δώσει στον παραπάνω κατάλογο και στα περιεχόμενά του αυτά τα δικαιώματα;

   (β) Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`splitlin`"), το οποίο να καλείται με δύο ονόματα αρχείων για ορίσματα, δηλαδή "`splitlin` ⟨*file*1⟩ ⟨*file*2⟩", και το οποίο, διαβάζοντας γραμμές από την προκαθορισμένη είσοδο, μέχρι να του δοθεί μία κενή γραμμή, να αποθηκεύει τις γραμμές εκείνες που περιέχουν τουλάχιστον ένα μικρό γράμμα (`a`, `b`, ..., `z`) στο αρχείο ⟨*file*1⟩, από τις υπόλοιπες, εκείνες που περιέχουν τουλάχιστον ένα ψηφίο (`0`, `1`, ..., `9`) να τις αποθηκεύει στο αρχείο ⟨*file*2⟩, ενώ όλες τις υπόλοιπες να τις αγνοεί. Επίσης, στο τέλος να εκτυπώνει πόσες γραμμές διάβασε και πόσες αγνόησε. Για παράδειγμα:

```
% splitlin lets digs
9-3-1998
My phone number is 7217941
THIS IS A TEST

3 line(s) read
1 line(s) ignored
% cat lets
My phone number is 7217941
% cat digs
9-3-1998
%
```

2. (α) Σε client-server εφαρμογές που έχουν υλοποιηθεί με τη βοήθεια υποδοχών ροής (stream sockets) στο πεδίο του Internet, δώστε την αλληλουχία ενεργειών του client και αυτήν του server για την αποκατάσταση της μεταξύ τους επικοινωνίας, χωρίς να αναφερθείτε σε κλήσεις συστήματος.

   (β) Γράψτε ένα πρόγραμμα C (υποθέστε ότι ονομάζεται "`outerr`"), το οποίο όταν καλείται από κάποιον με "`outerr` [`-1` ⟨*file*1⟩] [`-2` ⟨*file*2⟩] `-com` ⟨*command*⟩", να εκτελεί, μέσω κάποιας κλήσης συστήματος της ομάδας `execXX`, την εντολή ⟨*command*⟩, με ανακατεύθυνση της προκαθορισμένης εξόδου της (standard output) στο αρχείο ⟨*file*1⟩, αν έχει δοθεί το όρισμα "`-1` ⟨*file*1⟩", και της προκαθορισμένης εξόδου της για διαγνωστικά μηνύματα (standard error) στο αρχείο ⟨*file*2⟩, αν έχει δοθεί το όρισμα "`-2` ⟨*file*2⟩". Δηλαδή, τα ορίσματα αυτά είναι δυνατόν να μην δοθούν ή να δοθεί μόνο το ένα ή και τα δύο, με οποιαδήποτε σειρά. Για τη διαχείριση των ανακατευθύνσεων θα χρησιμοποιήσετε την κλήση συστήματος `dup2`. Παραδείγματα εκτέλεσης είναι τα εξής:

```
% outerr -com ls -l outerr nofile
nofile not found                                        /* Εξοδος σε stderr */
-rwxr-x---  1 user        24576 Mar  5 14:15 outerr     /* Εξοδος σε stdout */
% outerr -2 err -1 out -com ls -l outerr nofile
% cat out
-rwxr-x---  1 user        24576 Mar  5 14:15 outerr
% cat err
nofile not found
% outerr -1 out -com ls -l outerr nofile
nofile not found                                        /* Εξοδος σε stderr */
%
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Β' Περιόδου 1997

1. (α) Είναι δυνατόν να γράψει κάποιος ένα πρόγραμμα `whatisit` (σε κέλυφος Bourne, γλώσσα C κ.λ.π.), το οποίο όταν εκτελείται από το κέλυφος C να έχει την παρακάτω συμπεριφορά; Αν ναι, γράψτε το. Αν όχι, εξηγείστε γιατί.

```
% whatisit +
It's a cross
% whatisit *
It's a star
% whatisit ?
It's a questionmark
%
```

(β) Έστω ένα πρόγραμμα "`progr`" το οποίο χρειάζεται δύο ορίσματα στη γραμμή εντολής όταν εκτελείται, δηλαδή "`progr` ⟨*arg*1⟩ ⟨*arg*2⟩". Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`ontopofprogr`"), στο οποίο να έχουμε τη δυνατότητα να δώσουμε μέσω των επιλογών "`-opt1`" και "`-opt2`" κατάλληλα ορίσματα για το πρόγραμμα "`progr`", δηλαδή αυτό να καλείται με "`ontopofprogr` [`-opt1` ⟨*arg*1⟩] [`-opt2` ⟨*arg*2⟩]". Τα ορίσματα αυτά μπορούν να δοθούν με οποιαδήποτε σειρά ή και να μην δοθούν καθόλου (το ένα από τα δύο ή και τα δύο). Σ'αυτήν την περίπτωση, να θεωρούνται default τιμές για τα ⟨*arg*1⟩ και ⟨*arg*2⟩ οι 10 και 20 αντίστοιχα. Τέλος, το "`ontopofprogr`" να εκτελεί το "`progr`" δίνοντάς του τα κατάλληλα ορίσματα. Ενδεικτικές εκτελέσεις του προγράμματος είναι οι εξής:

```
% ontopofprogr -opt2 40 -opt1 25
Executing: progr 25 40
25 + 40 = 65
% ontopofprogr -opt1 34
Executing: progr 34 20
34 + 20 = 54
% ontopofprogr
Executing: progr 10 20
10 + 20 = 30
%
```

2. (α) Ποιες διαφορές υπάρχουν στην επικοινωνία μεταξύ διεργασιών μέσω σωλήνων (pipes) και υποδοχών (sockets);

(β) Αν το εκτελέσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα ονομάζεται "`desc`", τι εκτυπώνεται με την εντολή "`desc 5`"; Εξηγείστε στοιχειωδώς τη λειτουργία του προγράμματος. Όταν και αν χρειάζονται ταυτότητες διεργασιών, να δίνετε δικούς σας αυθαίρετους αριθμούς.

```
#include <stdio.h>
main(int argc, char *argv[])
{  int i, pid, status;
   for (i = 0 ; i < atoi(argv[1]) ; i++) {
      pid = fork();
      if (pid != 0) {
         wait(&status);
         printf("%d: %d\n", i, pid);
         exit(0); }
      printf("%d: %d %d\n", i, getpid(), getppid()); }
}
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Α' Περιόδου 1997

1. (α) Πώς από έναν κατάλογο στον οποίο έχετε πολλά αρχεία (ίσως και εκατοντάδες) θα τα διαγράφατε όλα εκτός από ένα συγκεκριμένο; Είτε περιγράψτε τη διαδικασία που θα ακολουθούσατε είτε δώστε την ακολουθία από τις εντολές του Unix που την υλοποιούν.

   (β) Γράψτε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`execall`") το οποίο να παίρνει από τη γραμμή εντολής ένα σύνολο από εντολές Unix, με αυθαίρετο αριθμό ορισμάτων η κάθε μία, και οι οποίες να χωρίζονται μεταξύ τους με το "`-new`". Το πρόγραμμα αυτό να εκτελεί αυτές τις εντολές, τη μία μετά την άλλη, και, στο τέλος, να μας πληροφορεί για το πλήθος των εντολών που εκτελέστηκαν. Εννοείται ότι καμία εντολή δεν μπορεί να έχει κάποιο όρισμα που να ονομάζεται "`-new`". Μία ενδεικτική εκτέλεση του προγράμματος μπορεί να είναι η εξής:

```
% execall ls -alg -new ps -new wc -w myfile -new echo a b c d
Going to execute: ls -alg
.....................................
Going to execute: ps
.....................................
Going to execute: wc -w myfile
.....................................
Going to execute: echo a b c d
.....................................
4 command(s) were executed
%
```

2. (α) Γιατί χρειαζόμαστε τις κλήσεις συστήματος "`open`", "`read`", "`write`" και "`close`" αφού έχουμε στη διάθεσή μας τις συναρτήσεις της πρότυπης (προκαθορισμένης) βιβλιοθήκης "`stdio`" της C για είσοδο-έξοδο, όπως, για παράδειγμα, τις "`scanf`", "`fprintf`", "`fgets`", "`putc`", "`fopen`", "`fclose`" κ.λ.π.;

   (β) Γράψτε ένα πρόγραμμα C (έστω ότι ονομάζεται "`3processes`") στο οποίο η αρχική διεργασία-πατέρας (A) να δημιουργήσει μία διεργασία-παιδί (B) η οποία, με τη σειρά της, να δημιουργήσει μία δική της διεργασία-παιδί (C). Στη συνέχεια, η διεργασία C να στείλει ένα σήμα SIGUSR1 στη διεργασία A, η οποία όταν το λάβει να εκτυπώσει ένα κατάλληλο μήνυμα. Μετά την αποστολή του σήματος, η C να τερματίσει και αφού η B αποδεχθεί τον τερματισμό της να τερματίσει και αυτή. Τέλος, αφού η A αποδεχθεί τον τερματισμό της B, να αυτο-αντικατασταθεί από την εντολή "`ls`" με ορίσματα που είχαν δοθεί από τη γραμμή εντολής κατά την εκτέλεση του προγράμματος "`3processes`". Ακολουθήστε στην απάντησή σας, όσο το δυνατόν πιο πιστά, την παρακάτω ενδεικτική εκτέλεση, δηλαδή, για κάθε εκτύπωση, δώστε ποια διεργασία την κάνει και για κάθε διεργασία, δώστε και την ταυτότητά της:

```
% 3processes -l -g /var/spool/oki
C(11291): Sending SIGUSR1 to A(11289)
A(11289): Received SIGUSR1
C(11291): Terminating
B(11290): My child C(11291) terminated
B(11290): Terminating
A(11289): My child B(11290) terminated
A(11289): Executing ls -l -g /var/spool/oki
total 2
-rw-r--r--  1 root     system         28 Jan 31 15:14 lock
-rw-rw-r--  1 root     system         37 Jan 31 15:14 status
%
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Β' Περιόδου 1996

1. Σε ένα σύστημα UNIX, για δύο χρήστες με ονόματα λογαριασμών `kostas` και `nikos` έχει ορισθεί να επικοινωνούν διαλογικά με το σύστημα μέσω του κελύφους C. Ο `kostas` έχει στον κατάλογο αφετηρίας του ένα σύνολο από πηγαία προγράμματα C (αρχεία με επέκταση "`.c`"), κάποια αρχεία κειμένου (επέκταση "`.txt`") και έναν υποκατάλογο `bin`. Όλα τα αρχεία στον υποκατάλογο `bin` είναι εκτελέσιμα προγράμματα. Ποιες εντολές πρέπει να εκτελέσει ο `kostas` ώστε να εξασφαλίσει ότι ο `nikos` θα μπορεί να αντιγράψει τα προγράμματα C, δεν θα έχει καθόλου πρόσβαση στα αρχεία κειμένου και θα μπορεί να εκτελέσει τα προγράμματα στον υποκατάλογο `bin`; Στη συνέχεια, πώς θα αντιγράψει ο `nikos` τα παραπάνω προγράμματα C στο δικό του κατάλογο αφετηρίας και πώς θα εκτελέσει ένα πρόγραμμα `testprog` που βρίσκεται στον υποκατάλογο `bin` που έχει ο `kostas`;

2. Τι συμβαίνει όταν ενώ εκτελείται κάποιο πρόγραμμα σε κάποιο τερματικό πατήσουμε στο πληκτρολόγιο Control-C;

3. Έστω ότι έχετε στη διάθεσή σας δύο προγράμματα `printto` και `linesof`. Όταν το πρώτο καλείται σαν "`printto` $\langle file \rangle$ $\langle printer \rangle$", εκτυπώνει το αρχείο $\langle file \rangle$ στον εκτυπωτή $\langle printer \rangle$. Η εντολή "`linesof` $\langle file \rangle$" εκτυπώνει στην προκαθορισμένη έξοδο τον αριθμό των γραμμών του αρχείου $\langle file \rangle$. Να γράψετε ένα πρόγραμμα "`filtlpr`" για το κέλυφος Bourne το οποίο όταν καλείται σαν "`filtlpr -lim` $\langle lines \rangle$ $\langle file1 \rangle$ $\langle file2 \rangle$ ... $\langle fileN \rangle$", να εκτυπώνει στον εκτυπωτή "`sp`" εκείνα από τα αρχεία $\langle file1 \rangle$, $\langle file2 \rangle$, ..., $\langle fileN \rangle$ που περιέχουν λιγότερες από $\langle lines \rangle$ γραμμές, ενώ τα υπόλοιπα να τα εκτυπώνει στον εκτυπωτή "`lw`". Στο τέλος, να εκτυπώνει στην προκαθορισμένη έξοδο ένα μήνυμα που να πληροφορεί πόσα αρχεία εκτυπώθηκαν σε κάθε εκτυπωτή. Να θεωρήσετε ότι το όρισμα "`-lim` $\langle lines \rangle$" είναι προαιρετικό, δηλαδή το πρόγραμμά σας να μπορεί να κληθεί και σαν "`filtlpr` $\langle file1 \rangle$ $\langle file2 \rangle$ ... $\langle fileN \rangle$". Σ'αυτήν την περίπτωση, υποθέστε ότι όριο διάκρισης είναι οι 1000 γραμμές.

4. Να γράψετε ένα πρόγραμμα C το οποίο να επιδεικνύει πώς μία διεργασία-πατέρας μπορεί να δημιουργήσει δύο διεργασίες-παιδιά οι οποίες να είναι σε θέση να επικοινωνήσουν μέσω ενός σωλήνα. Περιλάβετε στο πρόγραμμά σας και κάποια τέτοια επικοινωνία, π.χ. αποστολή ενός μηνύματος από τη μία διεργασία-παιδί στην άλλη. Έχοντας προβλέψει την εκτύπωση κατάλληλων διαγνωστικών μηνυμάτων, δώστε το αποτέλεσμα της εκτέλεσης του προγράμματός σας.

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Α' Περιόδου 1996

1. Από καθαρή αφηρημάδα εκτέλεσα την εντολή "`rm f`", όπου το αρχείο "`f`" ήταν πολύ σημαντικό για τη δουλειά μου. Τι μπορώ να κάνω τώρα (εκτός από το να το δημιουργήσω πάλι από την αρχή, κάτι που μπορεί να είναι από πολύ επίπονο έως αδύνατο); Πώς μπορώ στο μέλλον να αποφύγω τις ανεπιθύμητες συνέπειες από τέτοιου είδους επιπολαιότητες (εκτός από το να προσπαθήσω να είμαι πιο προσεκτικός);

2. Ποια είναι η σκοπιμότητα των κλήσεων συστήματος της ομάδας "`exec`"; Σε τι διαφέρουν οι "`execl`" / "`execlp`" από τις "`execv`" / "`execvp`"; Πότε είναι πιο βολικό να χρησιμοποιούμε τις πρώτες και πότε τις δεύτερες;

3. Να γράψετε ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`notif`") το οποίο να στέλνει σε ένα σύνολο από χρήστες του συστήματος κάποιο μήνυμα μέσω ηλεκτρονικού ταχυδρομείου. Το πρόγραμμα αυτό να καλείται ως "`notif` $\langle user1\rangle$ $\langle user2\rangle$ ... $\langle userN\rangle$ `-message` $\langle message\rangle$" με σκοπό την αποστολή του μηνύματος $\langle message\rangle$ στους χρήστες $\langle user1\rangle$, $\langle user2\rangle$, ..., $\langle userN\rangle$. Σημειώστε ότι το μήνυμα μπορεί προφανώς να αποτελείται από περισσότερες της μίας λέξεις. Δεν είναι απαραίτητο να προβλέψετε τις πιθανές περιπτώσεις λανθασμένης χρήσης και να προγραμματίσετε την κατάλληλη αντίδραση. Μία ενδεικτική εκτέλεση του προγράμματος μπορεί να είναι η εξής:

```
% notif george anna jim -message Hello there...
Message sent to george
Message sent to anna
Message sent to jim
```

4. Αν το εκτελέσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα ονομάζεται "`feb96`", τι εκτυπώνεται με την εντολή "`feb96 5 abcde fgh ijkl mn`"; Εξηγείστε στοιχειωδώς τη λειτουργία του προγράμματος. Όταν και αν χρειάζονται ταυτότητες διεργασιών να δίνετε δικούς σας αυθαίρετους αριθμούς.

```
#include <stdio.h>
#include <signal.h>
int alf = 0;
void al();

main(int argc, char *argv[])
{  int n, i, p, s;
   n = atoi(argv[1]);
   signal(SIGALRM, al);
   argv++;
   argc--;
   argv[argc] = NULL;
   alarm(n);
   printf("%d:%d:\n", getpid(), getppid());
   for (i = 1 ; i < argc ; i++) {
      p = fork();
      if (p == 0) {
         printf("%d:%d: ", getpid(), getppid());
         fflush(stdout);
         argv[i-1] = "echo";
         execv("/bin/echo", &argv[i-1]); }
      wait(&s); }
   printf("%d:%d:\n", getpid(), getppid());
   while (!alf)
      pause();
   write(1, "mess1\n", 6); }

void al()
{  alf = 1;
   write(1, "mess2\n", 6); }
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Β' Περιόδου 1995

1. Έστω ότι στον τρέχοντα κατάλογο έχουμε ένα πλήθος αρχείων με κατάληξη ".bar" τα οποία θα θέλαμε να τα μετονομάσουμε ώστε να έχουν πλέον κατάληξη ".foo". Είναι δυνατόν αυτό να γίνει με την εντολή "`mv *.bar *.foo`"; Αν όχι, εξηγήστε το λόγο και περιγράψτε τη λειτουργία αυτής της εντολής. Ενισχύστε την άποψή σας συμπληρώνοντας παρακάτω το αποτέλεσμα του τελευταίου "`ls -l`".

```
% mv
usage: mv [-if] f1 f2 or mv [-if] f1 ... fn d1 (‘fn’ is a file or directory)
% ls -l
total 3
-rw-r--r--  1 user1         254 Aug 10 10:02 aa.bar
-rw-r--r--  1 user1        1525 Sep 11 14:30 bb.bar
% mv *.bar *.foo
% ls -l
.................................................
.................................................
%
```

2. Τι είναι οι σωληνώσεις (pipelines) στο Unix; Δώστε ένα παράδειγμα σωλήνωσης που να συνδυάζει 3 τουλάχιστον στοιχειώδεις εντολές και περιγράψτε το αποτέλεσμά της.

3. Έστω ότι υπάρχει ένα πρόγραμμα "`prog`" το οποίο αφού διαβάσει ένα θετικό ακέραιο από την προκαθορισμένη είσοδο εκτελεί κάποιον υπολογισμό και εκτυπώνει τα αποτελέσματά του στην προκαθορισμένη έξοδο (στις διάστικτες γραμμές, όπως φαίνεται παρακάτω).

```
% prog
5
.................................................
.................................................
%
```

   Να γράψετε ένα πρόγραμμα για το κέλυφος Bourne το οποίο να παίρνει σαν ορίσματα ένα θετικό ακέραιο N και ένα όνομα αρχείου F και να εκτελεί το πρόγραμμα "`prog`" για κάθε δυνατή είσοδο 1, 2, ..., N γράφοντας όλα τα αποτελέσματα στο αρχείο F. Για παράδειγμα, αν το πρόγραμμα αυτό ονομάζεται "`mprog`", τότε το "`mprog 6 /tmp/results`" εκτελεί το "`prog`" για τις εισόδους 1, 2, 3, 4, 5, 6 και γράφει τα αποτελέσματα όλων των εκτελέσεων στο αρχείο "`/tmp/results`".

4. (3 μονάδες) Να γράψετε ένα πρόγραμμα C το οποίο να διαβάζει επαναληπτικά από την προκαθορισμένη είσοδο αριθμούς (μέχρι να του δοθεί το 0, οπότε θα τερματίζει) και για κάθε έναν από αυτούς, μόλις διαβασθεί, να δημιουργεί μία διεργασία-παιδί που να ελέγχει αν είναι ή όχι πρώτος αριθμός και να εκτυπώνει το κατάλληλο μήνυμα. Πριν τερματίσει η διεργασία-παιδί να στέλνει ένα σήμα SIGUSR1 στην αρχική διεργασία-πατέρα η οποία όταν το λαμβάνει να αποδέχεται τον τερματισμό του παιδιού. Θεωρείστε ότι υπάρχει δεδομένη μία συνάρτηση `int prime(long numb)` η οποία επιστρέφει 1 ή 0 αν ο `numb` είναι ή όχι πρώτος αριθμός αντίστοιχα και την οποία δεν χρειάζεται να υλοποιήσετε.

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Α' Περιόδου 1995

1. Στο Σχήμα 1 συμπληρώστε κατάλληλα τις διάστικτες γραμμές έτσι ώστε να προκύπτει μία σωστή ακολουθία εκτέλεσης εντολών του Unix και εκτύπωσης των αντίστοιχων αποτελεσμάτων.

2. Εστω ότι το εκτελέσιμο αρχείο "`shpr`" περιέχει το πρόγραμμα για το κέλυφος Bourne που φαίνεται στο Σχήμα 2. Δώστε το αποτέλεσμα που έχει η εκτέλεση του "`shpr 3 51 22 9 1`". Στη συνέχεια είτε να εξηγήσετε την απάντησή σας είτε να δώσετε ένα δικό σας παράδειγμα εκτέλεσης (γραμμή εντολής και αποτελέσματα που εκτυπώνονται).

3. Να γράψετε ένα πρόγραμμα C τέτοιο ώστε η αρχική διεργασία του αντίστοιχου εκτελέσιμου προγράμματος να δημιουργεί N διεργασίες-παιδιά με το N να δίνεται στη γραμμή εντολής. Κάθε διεργασία-παιδί, αφού εκτυπώσει την ταυτότητά της, να καθυστερήσει K δευτερόλεπτα, όπου K είναι η "τάξη" της (1 για το 1ο παιδί, 2 για το 2ο κ.ο.κ.), να στείλει αμέσως μετά το σήμα SIGUSR1 στην αρχική διεργασία-πατέρα και στη συνέχεια να τερματίσει. Κάθε φορά που η διεργασία-πατέρας λαμβάνει το σήμα SIGUSR1 να εκτυπώνει κάποιο μήνυμα. Μόλις λάβει και τα N σήματα να αποδεχθεί τον τερματισμό των παιδιών της και να τερματίσει και αυτή. Μία ενδεικτική εκτέλεση του προγράμματος (έστω ότι ονομάζεται "`cpr`") φαίνεται στο Σχήμα 3.

4. Πότε χρησιμεύουν οι σωλήνες (pipes) στο Unix; Με ποια κλήση συστήματος μπορούν να δημιουργηθούν μέσα από ένα πρόγραμμα C;

```
% ls -l my_file
-rwxr-x---  1 user1         261 Feb  1 16:51 my_file
% chmod go+r my_file
% ls -l my_file
.................................................
% ......................
% ls -l my_file
-r--rw-r-x  1 user1         261 Feb  1 16:51 my_file
% chmod ........ my_file
% ls -l my_file
.................................................
%
```

Σχήμα 1

```
#! /bin/sh5
echo $0:$#:$*
x=$1
y=0
while test $x -ge 1
do
   y=`expr $y + $x + $2`
   x=`expr $x - 1`
   shift
done
echo $y
z="$# $*"
set - $z
for u
do
   if [ -d . -a ! $u -lt $1 ]
   then
      echo +
   else
      echo -
   fi
done
echo $0:$#:$*
```

Σχήμα 2

```
% cpr 3
I am child no-1 with pid 19532
I am child no-2 with pid 19533
I am child no-3 with pid 19534
I am father and I got a SIGUSR1
I am father and I got a SIGUSR1
I am father and I got a SIGUSR1
%
```

Σχήμα 3

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Β' Περιόδου 1994

1. (α) Τι πληροφορίες συνάγετε από το ακόλουθο αποτέλεσμα της εκτέλεσης της εντολής "`ls -lg`";

```
% ls -lg
total 2
-rwxrw-r--  1 root     staff         236 Apr 22 14:16 entry1
drwxr-x---  2 bin      system        512 Sep 19 14:25 entry2
```

(β) Ως γνωστόν, ο κειμενογράφος οθόνης (screen editor) "`vi`" δεν κρατάει εφεδρικές (backup) εκδόσεις των αρχείων που χειρίζεται. Να γραφεί ένα πρόγραμμα για το κέλυφος Bourne (έστω ότι ονομάζεται "`myvi`") το οποίο να υποκαθιστά τον "`vi`" έχοντας και την επιπλέον δυνατότητα διαχείρισης των τεσσάρων τελευταίων εκδόσεων ενός αρχείου (την τρέχουσα συν τρείς εφεδρικές — "`<filename>`", "`<filename>.bak.1`", "`<filename>.bak.2`", "`<filename>.bak.3`"). Για παράδειγμα, η εντολή "`myvi <filename>`" να λειτουργεί όπως και η "`vi <filename>`" με την επιπλέον δυνατότητα να κρατάει την τρέχουσα έκδοση του αρχείου σαν πρώτη εφεδρική, την πρώτη σαν δεύτερη κ.ο.κ. Να ληφθεί υπόψη ότι για κάποιο αρχείο μπορεί να μην έχουν δημιουργηθεί και οι τρείς εφεδρικές εκδόσεις του σε κάποια φάση χρησιμοποίησης του προγράμματος "`myvi`".

2. (α) Ποια είναι η λειτουργία της κλήσης συστήματος "`fork`"; Τι επιστρέφει στο όνομά της;

(β) Ποιες διεργασίες λέγονται ορφανές (orphan) και ποιες ζωντανές-νεκρές (zombie);

(γ) Αν το εκτελεσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα ονομάζεται "`guess`", τι εκτυπώνεται κατά την εκτέλεση του προγράμματος με όρισμα το 3, δηλαδή "`guess 3`"; Να εξηγηθεί στοιχειωδώς η λειτουργία του προγράμματος. Οταν δημιουργούνται νέες διεργασίες και χρειάζονται οι ταυτότητες τους – PIDs – να δίνονται αυθαίρετοι αριθμοί.

```
#include <stdio.h>
main(int argc, char *argv[])
{  int i, d, n, p1, p2, s;
   d = atoi(argv[1]);
   n = 1;
   for(i=1 ; i<=d ; i++) {
      printf("%2d ---- %5d ---- %5d\n", n, getpid(), getppid());
      switch (p1 = fork()) {
         case 0:
            n = 2*n;
            break;
         default:
            switch (p2 = fork()) {
               case 0:
                  n = 2*n+1;
                  break;
               default:
                  wait(&s);
                  wait(&s);
                  exit(0); } } } }
```

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

Εξετάσεις Α' Περιόδου 1994

1. (α) Ποια είναι η λειτουργία της εντολής "`umask`" στο κέλυφος C; Ποιες είναι οι επιπτώσεις ενός "`umask 027`" μέσα στο αρχείο "`.cshrc`" ενός χρήστη;

   (β) Να γραφεί ένα πρόγραμμα για το κέλυφος Bourne (sh) το οποίο να δημιουργεί αντίγραφα δεδομένων αρχείων ή καταλόγων στον ίδιο κατάλογο που ανήκουν. Για κάθε αρχείο ή κατάλογο με όνομα "`<name>`" το αντίγραφό του να ονομάζεται "`<name>.orig`". Η δημιουργία του "`<name>.orig`" να είναι ελεγχόμενη, δηλαδή, αν υπάρχει ήδη, να ζητείται η επιβεβαίωση του χρήστη πριν γίνει η απαιτούμενη ενέργεια. Τα ονόματα των αρχικών αρχείων ή καταλόγων να δίνονται είτε από τη γραμμή εντολής είτε να διαβάζονται από το πρόγραμμα κατά την εκτέλεσή του. Για παράδειγμα, αν το ζητούμενο πρόγραμμα ονομάζεται "`original`", τότε να είναι αποδεκτές και οι δύο παρακάτω χρήσεις:

   ```
   % original file1 file2 dir1 file3 dir2
   ......


   % original
   Dwse ta onomata: file1 file2 dir1 file3 dir2
   ......
   ```

   (*Σημείωση:* Στην εντολή αντιγραφής "`cp`" η επιλογή "`-R`" χρησιμοποιείται για την αναδρομική αντιγραφή καταλόγων, αλλά δεν βλάπτει και την απλή αντιγραφή αρχείων, και η επιλογή "`-i`" χρησιμοποιείται για να ζητηθεί η επιβεβαίωση του χρήστη όταν υπάρχει αρχείο ή κατάλογος με το δεδομένο όνομα προορισμού.)

2. Αν το εκτελεσιμο πρόγραμμα που προκύπτει από το παρακάτω C πρόγραμμα ονομάζεται "`exam`", τι εκτυπώνεται κατά την εκτέλεση του προγράμματος με ορίσματα "3" και "2", δηλαδή "`exam 3 2`"; Εξηγείστε στοιχειωδώς τη λειτουργία του προγράμματος.
   (*Σημείωση:* Οταν δημιουργούνται νέες διεργασίες και χρειάζεστε τις ταυτότητες τους – PIDs – να δίνετε δικούς σας αυθαίρετους αριθμούς.)

```
#include <stdio.h>
#include <signal.h>
int bla, foo;
void fun();
main(int argc, char *argv[])
{  int i, ppp, tak, st;
   ppp = atoi(argv[1]);
   foo = atoi(argv[2]);
   signal(SIGUSR1, fun);
   bla = getpid();
   for (i=1 ; i<=ppp ; i++) {
      tak = fork();
      if (tak == 0) {
         while (foo != 0) pause();
         exit(0); }
      else
         bla = tak; }
   kill(bla, SIGUSR1);
   for (i=1 ; i<=ppp ; i++) wait(&st); }

void fun()
{  printf("%d\n", getpid());
   if (foo != 0) {
      kill(bla, SIGUSR1);
      foo--; } }
```