Προγραμματισμός συστημάτων UNIX/POSIX

Ανακατευθύνσεις (redirections)

Shell & ανακατεύθυνση εισόδου

- Κατά την εκτέλεση ενός προγράμματος, η είσοδος και η έξοδος ενός προγράμματος μπορούν να ανακατευθυνθούν από/σε κάποιο αρχείο της επιλογής μας. Π.χ. για χρήση του αρχείου “values” **αντί του πληκτρολογίου**, μπορούμε να εκτελέσουμε:

```
$ a.out < values
```

Όλες οι scanf()/gets() του προγράμματος αυτόματα διαβάζουν από το αρχείο αυτό και όχι από το πληκτρολόγιο.

Ανακατεύθυνση εξόδου

- Για να τυπωθούν τα μηνύματα σε ένα αρχείο “res” αντί της οθόνης:

```
$ a.out > res
```

- Μπορούμε να δούμε τα αποτελέσματα απλά με ανάγνωση του αρχείου.
  - Όλα τα `printf()/putchar()/puts()` πάνε αυτόματα στο αρχείο αυτό αντί για την οθόνη

- Αν θέλουμε να δούμε τα λάθη του gcc όταν μεταφράζουμε ένα πρόγραμμα και είναι πολλά:

```
$ gcc test1.c > res
```

Δουλεύει;

Ανακατεύθυνση εξόδου

- Κάθε πρόγραμμα έχει 3 «αρχεία» ανοιχτά όταν εκτελείται:
  - Standard input (stdin)
  - Standard output (stdout)
  - Standard error (stderr)

- `fprintf(stderr, …)`
- Χρησιμοποιείται από προγράμματα ώστε να μην μπλέκεται με τα τακτικά μηνύματα της εφαρμογής προς τον χρήστη.
- Τέλος, τα μηνύματα λάθους τα οποία τυπώνονται μεν στην οθόνη αλλά μέσω του stderr μπορούν επίσης να γραφτούν σε αρχείο με χρήση της εντολής:

```
$a.out 2> err
```

- Και το stdout και το stderr σε ένα αρχείο μαζί:

```
$a.out &> res_and_err
```

Πολλαπλές ανακατευθύνσεις

- Μπορεί κανείς να κάνει ταυτόχρονα πολλαπλές ανακατευθύνσεις:

```
a.out  <  values  >  results
```

Προγραμματισμός συστημάτων UNIX/POSIX

Διαχείριση λαθών
(error handling)

MYY502

errno

- Στα συστήματα POSIX, υπάρχει μία καθολική μεταβλητή η οποία προσδιορίζει το επακριβές σφάλμα που συνέβηκε στην τελευταία κλήση συστήματος στο πρόγραμμά μας.
- Πρέπει να κάνουμε #include <errno.h>
- Για παράδειγμα,

```
#include <errno.h>  /* για το errno κλπ. */

int main() {
  FILE *fp;

  fp = fopen("/tmp/tempfile", "r");
  if (fp == NULL) {
    fprintf(stderr, "Egine lathos: %d\n", errno);
    exit (errno);
  }
  ...
}
```

Συμβολικά

- Επειδή ο αριθμός δεν μας λέει και πολλά, μπορούμε να πάρουμε μία περιγραφή του λάθους με την perror():

```
#include <errno.h>  /* για το errno κλπ. */

int main() {
  FILE *fp;

  fp = fopen("/tmp/tempfile", "r");
  if (fp == NULL) {
    perror("Egine lathos");  /* Εδώ θα προστεθεί το μήνυμα */
    exit (errno);
  }
  ...
}
```

- Υπάρχει πίνακας με τα μηνύματα αυτά, και χρησιμοποιείται το errno αυτόματα για να τυπωθεί το κατάλληλο.

Το μήνυμα του λάθους χωρίς να τυπωθεί.

- Μπορούμε απλά να πάρουμε το μήνυμα λάθους και να μην το τυπώσουμε (η perror() το τυπώνει) ώστε να το εμφανίσουμε με το δικό μας τρόπο
- Αυτό γίνεται με την strerror()

```
#include <errno.h>  /* για το errno κλπ. */

int main() {
  char *msg;
  FILE *fp;

  fp = fopen("/tmp/tempfile", "r");
  if (fp == NULL) {
    msg = strerror(errno);
    fprintf(stderr, "Egine lathos (%d): [%s]", errno, msg);
    exit (errno);
  }
  ...
}
```

Προγραμματισμός συστημάτων UNIX/POSIX

Δυαδικά αρχεία
(binary files)

MYY502

Γενικά (Δυαδικά) αρχεία.

- Τα αρχεία που είδαμε μέχρι στιγμής (fopen()/fclose()) είναι απλά αρχεία κειμένου όπου μπορούμε να κάνουμε «φορμαρισμένη» επικοινωνία (δηλ. διάβασμα/τύπωμα μέσω των format της scanf/printf).
- Βασικά δεν μπορούμε να κάνουμε και πολλά άλλα πράγματα…
- Δεν είναι όμως όλα τα αρχεία απλά αρχεία κειμένου. Π.χ. το a.out είναι αρχείο *δυαδικό (binary).*
- Τα αρχεία αυτά (αλλά γενικότερα οποιοδήποτε «αρχείο» – και το unix θεωρεί «αρχείο» τα πάντα, π.χ. τα αρχεία κειμένου, τα δυαδικά, τις συσκευές εισόδου/εξόδου, τα sockets, κλπ) τα χειριζόμαστε μέσω ειδικών συναρτήσεων «χαμηλότερου» επιπέδου.

Άδειες αρχείων

- ls -l
  - `-rwxr-xr-x  ... a.out ...`
- Τα «`rwxr-xr-x`» περιγράφουν τι άδειες υπάρχουν για το αρχείο αυτό.
- Πρόκειται για 9 bits:

Bit	Σημασία
8	Ανάγνωση από τον κάτοχο
7	Εγγραφή από τον κάτοχο
6	Εκτέλεση από τον κάτοχο
5	Ανάγνωση από την ομάδα
4	Εγγραφή από την ομάδα
3	Εκτέλεση από την ομάδα
2	Ανάγνωση από τους υπόλοιπους
1	Εγγραφή από τους υπόλοιπους
0	Εκτέλεση από τους υπόλοιπους

Άδειες αρχείων

- Το «rwxr-xr-x» επομένως είναι ο δυαδικός αριθμός
  - 111101101
  - Στο οκταδικό, (0)755

- Δίνοντας κατάλληλες τιμές μπορούμε να καθορίσουμε ποιοι έχουν άδεια να κάνουν τι με κάποιο αρχείο μας.

- Με την εντολή chmod μπορούμε να αλλάξουμε τις άδειες αυτές (ο αριθμός πρέπει να είναι οκταδικός), π.χ.
  - `chmod 700 a.out`
  - `ls -l`
  - `-rwx------  ... a.out ...`

Περιγραφείς αρχείων

- Τα αρχεία που χειρίζεται ένα πρόγραμμα αριθμούνται με έναν απλό ακέραιο αριθμό (περιγραφέας – file descriptor).
- Κάθε εκτελούμενο πρόγραμμα έχει 3 αρχεία ανοιχτά αυτόματα, με περιγραφείς 0, 1, 2
  - 0: το standard input
  - 1: το standard output
  - 2: το standard error

- Μπορεί να ανοίξει κι άλλα (υπάρχοντα) με την *open()*
- Μπορεί να δημιουργήσει νέα αρχεία με την *creat()*
- Μπορεί να διαγράψει αρχεία με την *unlink()*
- Κλπ

open() – Άνοιγμα υπάρχοντος αρχείου

- **int open(char *path, int flags);**
- Η πρώτη παράμετρος είναι (ολόκληρο) το μονοπάτι που βρίσκεται το αρχείο.
- Η δεύτερη παράμετρος καθορίζει με τι τρόπο / λειτουργία θα προσπελαύνουμε το αρχείο (δηλ. ανάγνωση, εγγραφή ή και τα δύο).
  - O_RDONLY για μόνο ανάγνωση
  - O_WRONLY για μόνο εγγραφή
  - O_RDWR για ανάγνωση και εγγραφή
- Π.χ.
  - int fd = open("/tmp/tempfile", O_RDWR);
- Αν αποτύχει επιστρέφει αρνητικό αριθμό.

open() – Άνοιγμα υπάρχοντος αρχείου

❖ Παράδειγμα χρήσης:

```
#include <unistd.h>  /* για την open() κλπ. */
#include <fcntl.h>   /* για τα flags O_xxx */

int main() {
  int fd;
  fd = open("/tmp/temfile", O_RDWR);
  if (fd < 0)
    exit(1);
  ...
}
```

❖ Γιατί μπορεί να αποτύχει η open()?

open() – Δημιουργία νέου αρχείου

- Στην περίπτωση αυτή η open() πρέπει να καλείται με 3 ορίσματα:
  - **`int open(char *path, int flags, mode_t permissions);`**
- Ανάμεσα στα flags πρέπει να υπάρχει το O_CREAT.
- Όλα τα διάφορα flags που χρησιμοποιούνται πρέπει να γίνουν binary OR (|) μεταξύ τους. Π.χ.
  - `O_WRONLY | O_CREAT`
- Το O_RDONLY | O_CREAT δεν έχει πολύ νόημα αλλά όντως θα δημιουργηθεί κενό αρχείο στο οποίο δεν μπορούμε να γράψουμε τίποτε...
- Αν το νέο αρχείο που πάμε να δημιουργήσουμε υπάρχει ήδη, δεν θα γίνει τίποτε (απλά ανοίγει το αρχείο – αντίθετα αποτυγχάνει αν έχει δοθεί και O_EXCL). Αν όμως θέλουμε να σβηστεί το παλιό και να δημιουργήσουμε ένα νέο κενό αρχείο από την αρχή, πρέπει να δώσουμε και το flag O_TRUNC, π.χ
  - `O_WRONLY | O_CREAT | O_TRUNC`
- Η τρίτη παράμετρος καθορίζει τις **άδειες** που θα έχει το νέο αρχείο.
  - Αν και παλαιότερα ήταν ένας ακέραιος αριθμός (δινόταν ως οκταδικός συνήθως), πλέον είναι σύνολο από flags.

Ιστορική παρένθεση: creat() για δημιουργία νέου αρχείου

- Η περίπτωση όπου θέλουμε να δημιουργήσουμε νέο αρχείο και να σβηστεί το τυχόν παλιό αν υπάρχει είναι τόσο συνηθισμένη στην πράξη που υπάρχει ειδική συνάρτηση για αυτό, η creat().
- Είναι ΑΚΡΙΒΩΣ σαν την open() μόνο που παίρνει το 1ο και το 3ο όρισμα.
  - `int creat(char *path, mode_t permissions);`
- Είναι σαν να καλούμε την open() με δεύτερο όρισμα τον αριθμό:

  `O_WRONLY | O_CREAT | O_TRUNC`
- Επομένως, την ξεχνάμε και θα χρησιμοποιούμε μόνο την open()
- ***Ιστορικής σημασίας και μόνο.***

open() – Δημιουργία νέου αρχείου

- Η κλήση:
  - `int open(char *path, int flags, mode_t permissions);`
- Η τρίτη παράμετρος καθορίζει τις *άδειες* που θα έχει το νέο αρχείο.
  - Αν και παλαιότερα ήταν ένας ακέραιος αριθμός (δινόταν ως οκταδικός συνήθως), πλέον είναι σύνολο από flags.
  - Για τη χρήση του απαιτείται `#include <sys/stat.h>`

Bit	Σημασία	Τιμή για το mode_t
8	Ανάγνωση από τον κάτοχο	S_IRUSR
7	Εγγραφή από τον κάτοχο	S_IWUSR
6	Εκτέλεση από τον κάτοχο	S_IXUSR
5	Ανάγνωση από την ομάδα	S_IRGRP
4	Εγγραφή από την ομάδα	S_IWGRP
3	Εκτέλεση από την ομάδα	S_IXGRP
2	Ανάγνωση από τους υπόλοιπους	S_IROTH
1	Εγγραφή από τους υπόλοιπους	S_IWOTH
0	Εκτέλεση από τους υπόλοιπους	S_IXOTH

Άδειες

- Επομένως αντί για 0755, θα πρέπει να δώσουμε:

```
S_IRUSR | S_IWUSR | S_IXUSR | S_IRGRP | S_IXGRP |
  S_IROTH | S_IXOTH
```

- Για την περίπτωση που ο χρήστης / η ομάδα / οι υπόλοιποι έχει όλες τις άδειες, μπορούμε αντίστοιχα να δώσουμε

```
S_IRWXU, S_IRWXG ή/και S_IRWXO
```

- Επομένως, το παραπάνω συντομότερα είναι:

```
S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH
```

Διαγραφή αρχείου

- Για διαγραφή ενός αρχείου μπορεί να κληθεί η συνάρτηση:

```
int unlink( char *path );
```

- Αν το αρχείο είναι ανοικτό δε διαγράφεται άμεσα. *Θα διαγραφεί μόλις κλείσει.*

Κλείσιμο αρχείου

- Για να κλείσουμε ένα ανοιχτό αρχείο, πρέπει να κληθεί η συνάρτηση:

```
int close( int fd );
```

- Αν έχει προηγηθεί `unlink()`, το αρχείο θα διαγραφεί μετά την κλήση της `close()`.

Γράψιμο στο αρχείο

- Για γράψιμο (αποθήκευση) στο αρχείο:

```
size_t write(int fd, void *buf, size_t nbytes);
```

- Παράδειγμα:

```
int arr[20];
double x;

write(fd, &x, sizeof(double));
write(fd, arr, 15*sizeof(int));
```

- **<u>Επιστρέφει το πλήθος των bytes που γράφτηκαν ή αρνητικό σε περίπτωση λάθους</u>**.

- ***Πρέπει πάντα να γίνεται έλεγχος της τιμής επιστροφής!!!!***
  - Δίσκος είναι, προβλήματα παρουσιάζονται…
  - Μερικές φορές δεν γράφονται όλα τα bytes, πρέπει να ξαναγράψουμε τα υπόλοιπα.

Διάβασμα από αρχείο

- Για διάβασμα από το αρχείο:

```
size_t read(int fd, void *buf, size_t nbytes);
```

- **Επιστρέφει το πλήθος (>0) των bytes που διαβάστηκαν, 0 για EOF, αρνητικό (<0) σε περίπτωση λάθους**. Παράδειγμα:

```
int arr[20];
double x, sum = 0.0;

read(fd, arr, 15*sizeof(int));
while ( (n = read(fd, &x, sizeof(double))) > 0 ) {
  sum += x;
}
if (n < 0) { perror("Problem: "); exit(1); }
if (n == 0) /* EOF – End of File */
  close(fd);
```

Σχετική θέση αρχείου – lseek()

- Η θέση του αρχείου μετά το open() είναι στο αρχικό byte του(#0).
  - Αν έχουμε δώσει O_APPEND ανάμεσα στα flags της open(), θα είναι αμέσως μετά το τελευταίο byte του.
- Μπορούμε να αλλάξουμε τη θέση με τη συνάρτηση lseek():

```
off_t lseek(int fd, off_t pos, int whence);
```

  (το off_t βασικά είναι ένας long int)
- Επιστρέφει τη νέα θέση.
- Στο pos δίνουμε πόσα bytes να προχωρήσει
- Το whence είναι ένα από:
  - SEEK_SET ώστε η απόλυτη νέα θέση να υπολογίζεται από την αρχή του αρχείου
  - SEEK_CUR ώστε η νέα θέση να υπολογίζεται από την τρέχουσα θέση του αρχείου
  - SEEK_END ώστε η απόλυτη νέα θέση να υπολογίζεται από το τέλος του αρχείου
- Αν η νέα θέση είναι ξεπερνά το μέγεθος του αρχείου, ***το αρχείο μεγαλώνει***.

Παραδείγματα

```
off_t currentpos, filesize;

/* Πήγαινε στην αρχή */
lseek(fd, 0, SEEK_SET);
/* Πήγαινε στο τέλος (ΜΕΤΑ το τελευταίο byte) */
lseek(fd, 0, SEEK_END);
/* Πήγαινε 4 bytes πριν */
lseek(fd, -sizeof(int), SEEK_CUR);

/* Εύρεση τρέχουσας θέσης & μεγέθους αρχείου σε bytes */
currentpos = lseek(fd, 0, SEEK_CUR);    /* Τρέχουσα θέση */
filesize = lseek(fd, 0, SEEK_END);          /* Στο τέλος */
lseek(fd, currentpos, SEEK_SET); /* Γύρνα εκεί που ήσουν */
```

Άδειασμα προσωρινής μνήμης

- Κατά το γράψιμο, *δεν εγγράφονται όλα άμεσα στο δίσκο* (στο close() εξασφαλίζεται ότι όλα θα γραφτούν).
  - Τοποθετούνται σε «προσωρινή μνήμη» και γράφονται από το λειτουργικό σύστημα κάποτε αργότερα
  - Για λόγους ταχύτητας
- Επειδή μπορεί να γίνει κάτι και να μην προλάβουν και γραφτούν τα δεδομένα μπορεί κάποιος να χρησιμοποιήσει τη συνάρτηση:

```
void sync();
```

  - Γράφει ότι δεν έχει γραφτεί ακόμα στο δίσκο, *για όλα τα ανοιχτά αρχεία, όμως*.
- Η συνάρτηση:

```
int fsync(int fd);
```

  - Γράφει ότι δεν έχει γραφτεί ακόμα στο δίσκο, για το συγκεκριμένο αρχείο.

Συγχρονισμένες εγγραφές

- Για να γίνεται πάντα **άμεση** εγγραφή στον δίσκο θα πρέπει κατά το open() να δώσουμε και το flag **O_SYNC**
  - Παράδειγμα:

```
fd = open("/tmp/temfile", O_RDWR | O_SYNC);
```

- Ουσιαστικά θα καλείται αυτόματα η fsync μετά από κάθε εγγραφή με την write.

- Είναι πιο ασφαλές αλλά πολύ πιο αργό!

Δυαδικά αρχεία / αρχεία κειμένου

- Ποια θα είναι τα περιεχόμενα των αρχείων txtfile και binfile?

```c
#include <stdio.h>

int main() {
  FILE *fp;
  int x = 1454654714;
  fp = fopen("txtfile", "w");
  fprintf(fp, "%d", x);
  fclose(fp);
  return 0;
}
```

```c
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

#define Create (O_WRONLY|O_CREAT|O_TRUNC)
#define UserRW (S_IRUSR|S_IWUSR)

int main() {
  int fd;
  int x = 1454654714;
  fd = open("binfile", Create, UserRW);
  write(fd, &x, sizeof(int));
  close(fd);
  return 0;
}
```

Δυαδικά αρχεία / αρχεία κειμένου

- Το txtfile θα περιέχει:

```
$ cat txtfile
1454654714
```

- Γιατί;
  - Διότι γράψαμε *χαρακτήρες*, ακριβώς όπως θα εμφανίζονταν και στην **οθόνη**.
  - Το μέγεθος του αρχείου εξαρτάται από το πλήθος των ψηφίων του x!

```
#include <stdio.h>

int main() {
  FILE *fp;
  int x = 1454654714;
  fp = fopen("txtfile", "w");
  fprintf(fp, "%d", x);
  fclose(fp);
  return 0;
}
```

Δυαδικά αρχεία / αρχεία κειμένου

- Το binfile θα περιέχει:

```
$ cat binfile
V₺D•
```

- Γιατί;
  - Διότι γράψαμε *bytes* (πιθανώς μη εκτυπώσιμα) ακριβώς όπως είναι αποθηκευμένα στη **μνήμη**.
  - Το μέγεθος του αρχείου θα είναι πάντα 4 bytes, όσο πιάνει ένας ακέραιος στη μνήμη.

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

#define Create (O_WRONLY|O_CREAT|O_TRUNC)
#define UserRW (S_IRUSR|S_IWUSR)

int main() {
  int fd;
  int x = 1454654714;
  fd = open("binfile", Create, UserRW);
  write(fd, &x, sizeof(int));
  close(fd);
  return 0;
}
```

Δυαδικά αρχεία / αρχεία κειμένου

- Στα αρχεία κειμένου ουσιαστικά σώζουμε *ακριβώς ότι θα τυπώναμε και στην **οθόνη***, ενώ
  - στα δυαδικά αρχεία σώζουμε *ακριβώς ότι υπάρχει στη **μνήμη** του υπολογιστή*.
- Τα δυαδικά αρχεία τείνουν να είναι πιο μικρά και άρα πιο γρήγορα στην προσπέλασή τους
  - Χρήσιμο όταν έχουμε πραγματικά μεγάλα αρχεία
- Τα δυαδικά αρχεία δεν είναι εύκολα αναγνώσιμα και μεταφέρσιμα (portable)
  - Για μεταφέρσιμα αρχεία συνήθως χρησιμοποιούμε αρχεία κειμένου (π.χ. html).
  - Όμως, για μείωση χρόνου επικοινωνίας χρησιμοποιούμε *συμπιεσμένα* αρχεία, τα οποία είναι σχεδόν πάντα δυαδικά.
- Κάποια είδη αρχείων είναι μόνο δυαδικά (jpeg, mp3 κλπ).
- Κάποιες λειτουργίες του συστήματος (π.χ. sockets) είναι διαθέσιμες μόνο μέσω δυαδικών αρχείων.