Προγραμματισμός ΙΙ (ECE116)

#15 λειτουργίες αρχείων της βιβλιοθήκης stdio

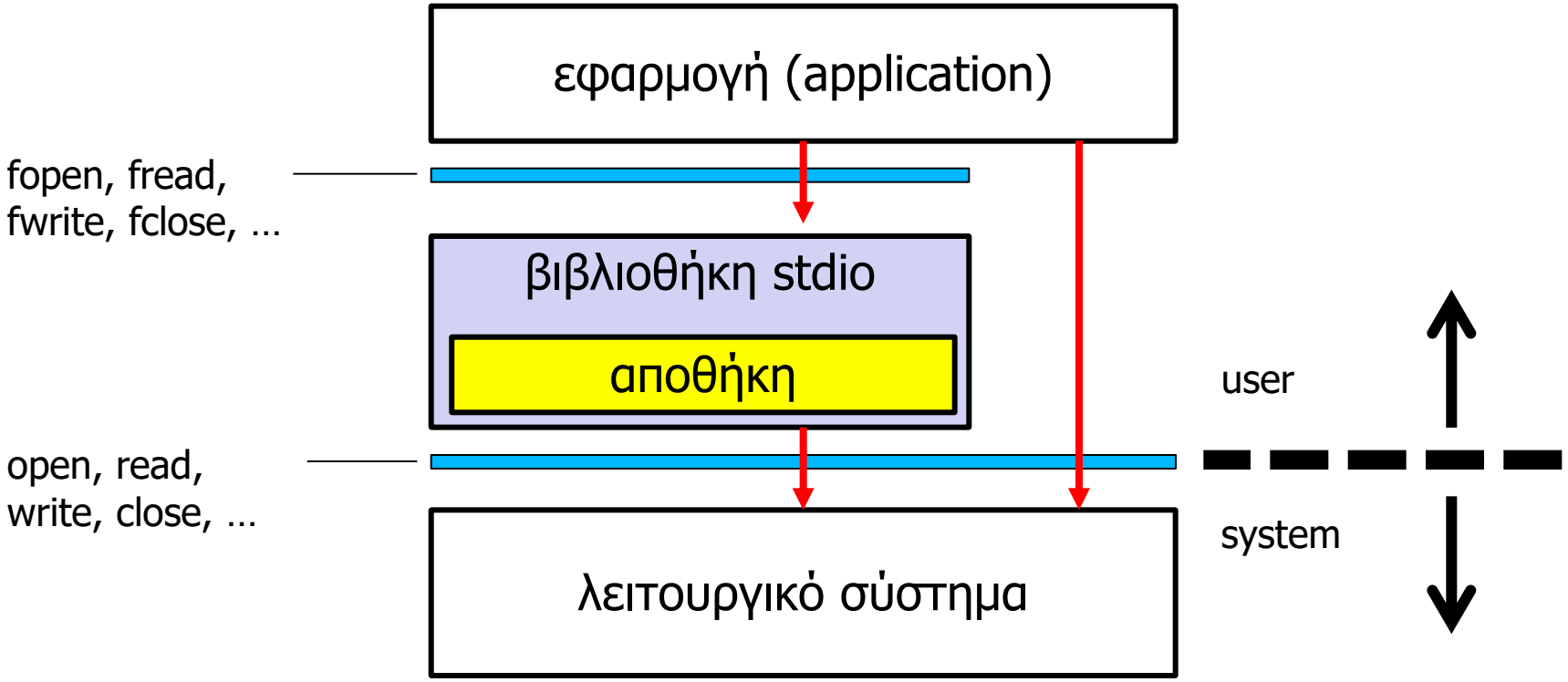
Σπύρος Λάλης
lalis@uth.gr

Η βιβλιοθήκη stdio

- Μέρος του προτύπου ANSI C (D. Ritchie, 1975)
- Λειτουργίες υψηλού επιπέδου για είσοδο/έξοδο πάνω σε λεγόμενες **ροές δεδομένων** (streams)
  - τα αρχεία δίσκου είναι μια ειδική περίπτωση ροής

Βασικές λειτουργίες

- **Μορφοποιημένο** σκανάρισμα/εκτύπωση
  - μετατροπή βασικών τύπων δεδομένων από την δυαδική αναπαράσταση του Η/Υ σε συμβατική μορφή, και το αντίστροφο
- Διάβασμα/γράψιμο bytes
  - κοντά στο πνεύμα των κλήσεων συστήματος
- Εσωτερική υλοποίηση
  - χρησιμοποιεί **κλήσεις συστήματος** του λειτουργικού
  - σε συνδυασμό με μια **εσωτερική** αποθήκη δεδομένων


εφαρμογή (application)
fopen, fread,
fwrite, fclose, …
βιβλιοθήκη stdio
αποθήκη
user
open, read,
write, close, …
system
λειτουργικό σύστημα

Σκανάρισμα/εκτύπωση κειμένου

- **Scan:** διάβασμα δεδομένων από κείμενο (text) και αποθήκευση στην μνήμη του προγράμματος
- **Print:** εκτύπωση δεδομένων του προγράμματος με την μορφή κειμένου
- Για την ροή της **καθιερωμένης εισόδου/εξόδου**
  - `getc()`, `putc()`, `scanf()`, `printf()`, …
- Αλλά και για **οποιαδήποτε άλλη** ροή δεδομένων
  - `fgetc()`, `fputc()`, `fscanf()`, `fprintf()`, …
- Δημιουργούνται **αυτόματα** ανοιχτές ροές για την συμβατική είσοδο, έξοδο και έξοδο λαθών
  - `FILE *stdin, *stdout, *stderr`
  - αντιστοιχίζονται στους συμβατικούς περιγραφείς αρχείων `STDIN_FILENO` (`0`), `STDOUT_FILENO` (`1`), `STDERR_FILENO` (`2`)

Γράψιμο/διάβασμα **bytes** (σε αρχεία)

- Μια ροή δεδομένων **μπορεί** να είναι συσχετισμένη με ένα (πραγματικό) αρχείο στον δίσκο
- Τότε οι λειτουργίες της `stdio` διαβάζουν/γράφουν δεδομένα από το / στο αντίστοιχο αρχείο
  - μέσω ενός περιγραφέα αρχείου και κλήσεων συστήματος

- Είστε ήδη εξοικειωμένοι με τις λειτουργίες για μορφοποιημένη είσοδο/έξοδο (scan/print)
  - δεν θα αναφερθούμε σε αυτές
- Θα εστιάσουμε στις βασικές λειτουργίες της `stdio` για **μη-μορφοποιημένο** διάβασμα/γράψιμο bytes

Άνοιγμα ροής σε αρχείο

- `FILE *fopen(const char *filename, const char *mode)`
- Η παράμετρος `filename` προσδιορίζει το όνομα
- Η παράμετρος `mode` προσδιορίζει τον τρόπο με τον οποίο το πρόγραμμα προτίθεται να χρησιμοποιήσει το αρχείο
  - `r`: διάβασμα (το αρχείο πρέπει να υπάρχει)
  - `w`: δημιουργία αρχείου για γράψιμο (**προσοχή:** αν το αρχείο υπάρχει, αυτομάτως **σβήνεται το περιεχόμενο** του)
  - `a`: προσθήκη στο τέλος (αν το αρχείο δεν υπάρχει, δημιουργείται)
  - `r+`: όπως `r` αλλά και γράψιμο
  - `w+`: όπως `w` αλλά και διάβασμα
  - `a+`: όπως `a` αλλά και διάβασμα
- Άνοιγμα σε **binary mode:** προσθήκη στην `mode` το `b`
- Αν η `fopen` αποτύχει, επιστρέφεται `NULL`

Διάβασμα / γράψιμο

- `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)`
  - ανάγνωση `nmemb` «αντικειμένων» μεγέθους `size` bytes, και αποθήκευση των δεδομένων στην διεύθυνση `ptr`
  - επιστρέφεται ο αριθμός «αντικειμένων» που διαβάστηκαν

- `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)`
  - γράψιμο `nmemb` «αντικειμένων» μεγέθους `size` bytes, με διάβασμα των δεδομένων από την διεύθυνση `ptr`
  - επιστρέφεται ο αριθμός «αντικειμένων» που γράφτηκαν

- `int fflush(FILE *stream)`
  - άδειασμα εσωτερικής αποθήκης

Μετακίνηση θέσης ανάγνωσης/εγγραφής

- `long ftell(FILE *stream)`
  - επιστροφή θέσης γράψιματος/ανάγνωσης
- `int fseek(FILE *stream, long int offset, int whence)`
  - μετακίνηση θέσης ανάγνωσης/γραψίματος
- `void rewind(FILE *stream)`
  - ισοδύναμο με `(void) fseek(stream, 0L, SEEK_SET)`

Έλεγχος για λάθη / κλείσιμο αρχείου

- `int feof(FILE *stream)`
  - έλεγχος τέλους αρχείου (end-of-file)
- `int ferror(FILE *stream)`
  - έλεγχος για λάθος σε προηγούμενη λειτουργία
- `void perror(const char *str)`
  - εκτύπωση μηνύματος λάθους στο `stderr`
- `void clearerr(FILE *stream)`
  - καθάρισμα ένδειξης λάθους και end-of-file

- `int fclose(FILE *stream)`
  - κλείσιμο ροής/αρχείου
  - με απελευθέρωση πόρων (εσωτερικών buffers)

Διαβάστε **προσεκτικά** το manual

- If a file is opened for update (the + mode), an output operation may not be followed by an input operation without flushing the buffer or repositioning, and an input operation may not be followed by an output operation without flushing the buffer or repositioning, unless the input operation has reached end-of-file.

- For input streams, fflush() discards any buffered data that has been fetched from the underlying file, but has not been consumed by the application.

- For output streams, fflush() forces a write of all user-space buffered data for the given output or update stream via the stream's underlying write function – for files, this does not ensure that the data will actually be written on disk.

Κόψιμο/επέκταση αρχείου

- Η stdio **δεν** προσφέρει λειτουργία για το κόψιμο ενός αρχείου
  - ασύμβατο με την έννοια της «ροής» δεδομένων
- Το κόψιμο μπορεί να γίνει μέσω των κλήσεων συστήματος `truncate()` και `ftruncate()`
- Αυτές **δεν** είναι λειτουργίες της βιβλιοθήκης stdio
  - καλό είναι να **αποφεύγεται** η χρήση τους ταυτόχρονα με τις λειτουργίες της stdio
  - αυτό ισχύει και για άλλες λειτουργίες αρχείων του συστήματος

Εσωτερική αποθήκη

- Η stdio διατηρεί ενδιάμεση **εσωτερική αποθήκη**
  - έτσι ώστε να μην χρειάζεται να καλεί κάθε φορά τις λειτουργίες του συστήματος (`write`/`read`)
- Ξεχωριστή εσωτερική αποθήκη για κάθε ροή
  - δημιουργείται με την **πρώτη** λειτουργία ανάγνωσης/γραψίματος

- **Γράψιμο:** τα δεδομένα γράφονται στην αποθήκη και μεταφέρονται στην ροή σε δεύτερο χρόνο
  - όταν η αποθήκη γεμίσει ή το πρόγραμμα καλέσει `fflush()`
- **Διάβασμα:** τα δεδομένα διαβάζονται από την ροή στην αποθήκη, και από εκεί προωθούνται στο πρόγραμμα

Πολιτική εσωτερικής αποθήκευσης

- **Line buffered (LB):** flush σε `\n`
- **Fully buffered (FB):** flush όταν η αποθήκη γεμίσει
- **Not buffered (NB):** χωρίς ενδιάμεση αποθήκευση
- Υπάρχουν **αυτόματες** πολιτικές
  - stdout: LB, stderr: NB, files: FB
- Αν το stdout **ανακατευθυνθεί** σε αρχείο **πριν** την πρώτη χρήση, η πολιτική **γίνεται** FB …
- … και **παραμένει** έτσι ακόμα και αν μετά γίνει ανακατεύθυνση πίσω στο τερματικό
  - αντί το καθιερωμένο LB
- Αν το stdout **ανακατευθυνθεί** σε αρχείο **μετά** την πρώτη χρήση, η πολιτική **παραμένει** LB
  - παρότι η έξοδος γράφεται σε αρχείο

```
int fd;

fd = open("test",O_WRONLY|O_CREAT|O_TRUNC,S_IRWXU);
dup2(fd,STDOUT_FILENO); /* redirect stdout to fd */
printf("hello world\n");
…
```

fully buffered

```
int fd;

printf("something\n");
fd = open("test",O_WRONLY|O_CREAT|O_TRUNC,S_IRWXU);
dup2(fd,STDOUT_FILENO); /* redirect stdout to fd */
printf("hello world\n");
…
```

line buffered

Αλλαγή πολιτικής αποθήκευσης

- Ο προγραμματιστής μπορεί να **προσδιορίσει** την επιθυμητή πολιτική αποθήκευσης, για κάθε ροή (`FILE`)

- ```
  int setvbuf(FILE *stream, char *buffer,
              int mode, size_t size)
  ```
- Καθορίζει την πολιτική αποθήκευσης για την ροή `stream`, το μέγεθος `size` της αποθήκης ή/και την μνήμη `buffer` που θα χρησιμοποιηθεί για την αποθήκη

- Αυτό πρέπει να γίνει (α) **μετά** την `fopen` και (β) **προτού** κληθεί η πρώτη λειτουργία ανάγνωσης ή γραψίματος
  ```

```
int main(int argc , char *argv[]) {
  char str[]="hello\nthere\nhow are you?";
  int i;

  if (!strcmp(argv[1],"LB")) {
    // default
  }
  else if (!strcmp(argv[1],"NB")) {
    setvbuf(stdout,NULL,_IONBF,0);
  }
  else if (!strcmp(argv[1],"FB")) {
    setvbuf(stdout,NULL,_IOFBF,0);
  }

  for (i=0; i<strlen(str); i++) {
    sleep(1);
    printf("%c",str[i]);
  }
  return(0);
}
```

not buffered

fully buffered