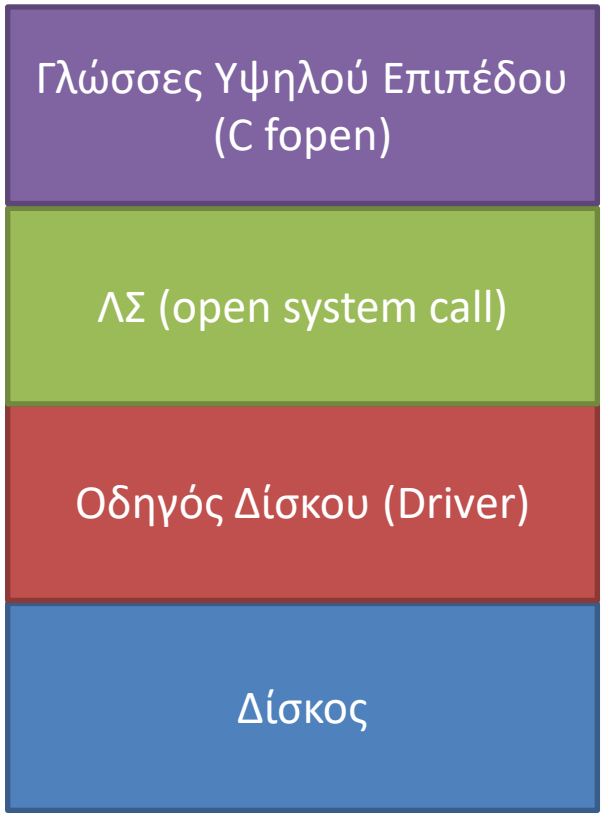
Εργαστήριο 5

Γενική Πυραμίδα



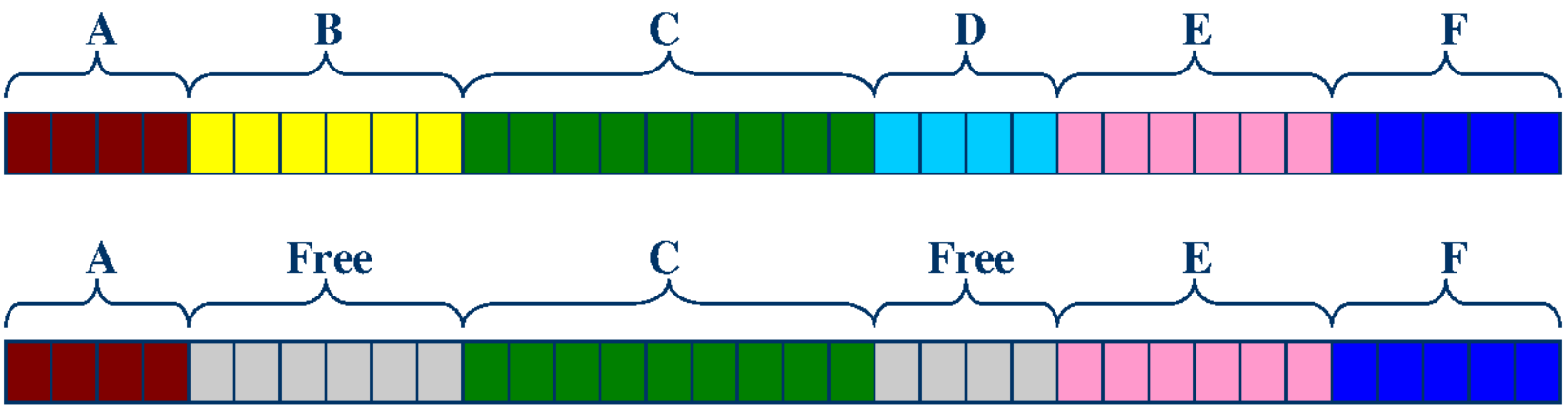
Τρόπος γενικής χρήσης αρχείων ανεξάρτητα από
χρησιμοποιούμενο ΛΣ ή τύπο δίσκου

Τρόπος καταγραφής τοποθεσίας μπλοκ και χρήσης driver
Γνώση ορίων αρχείων, δικαιωμάτων κλπ. Πρέπει να ξέρετε το API του driver
Συνήθως αυτά τα API είναι τυποποιημένα οπότε ουσιαστικά
δεν αλλάζουν από δίσκο σε δίσκο (απλά πρέπει να έχετε το όνομα του driver
που θα χρησιμοποιηθεί)

Τρόπος υλοποίησης API προς το ΛΣ για μετάβαση/λειτουργία
σε ένα σημείο του δίσκου. Πρέπει να ξέρετε το εσωτερικό του δίσκου
Δεν γνωρίζετε όρια αρχείων κλπ

Φυσική Αποθήκευση σε μπλοκ/τοποθεσίες
Καμία γνώση για το ποια αρχεία ή προγράμματα αφορούν

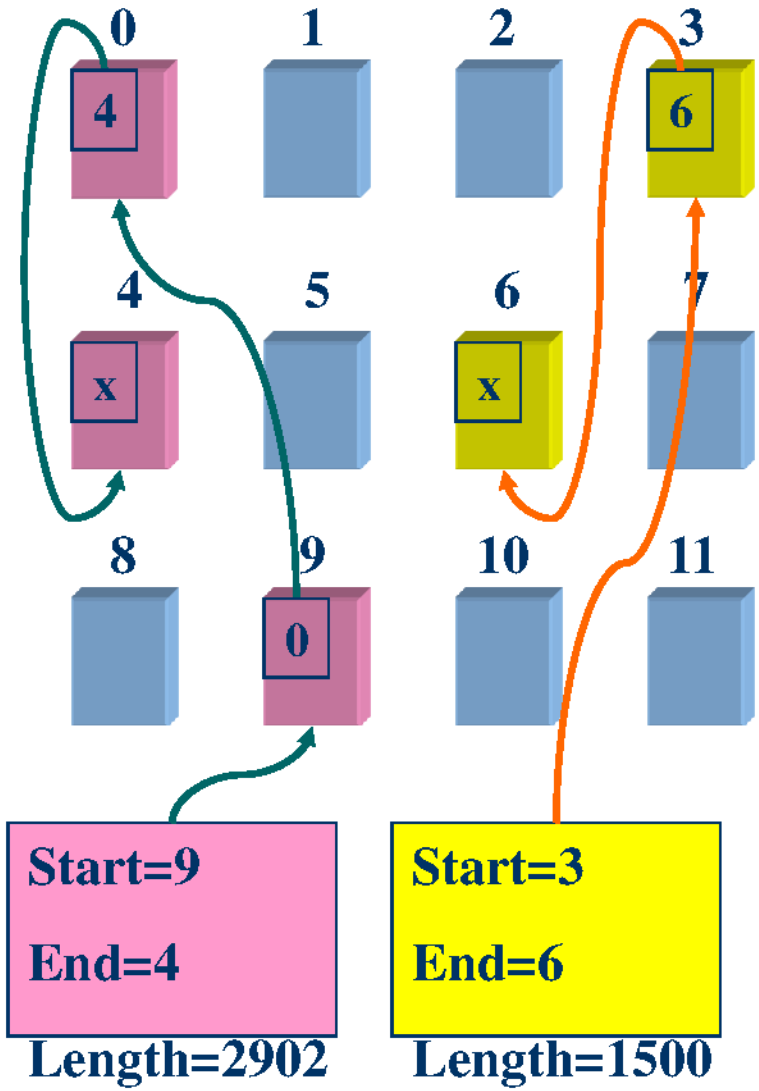
Διαμόρφωση Δίσκου



- Είναι τα μπλοκ ενός αρχείου σειριακά αποθηκευμένα στο δίσκο??
- Με τη διαγραφή αρχείων δημιουργούνται κενά
- Επίσης τα αρχεία επεκτείνονται!

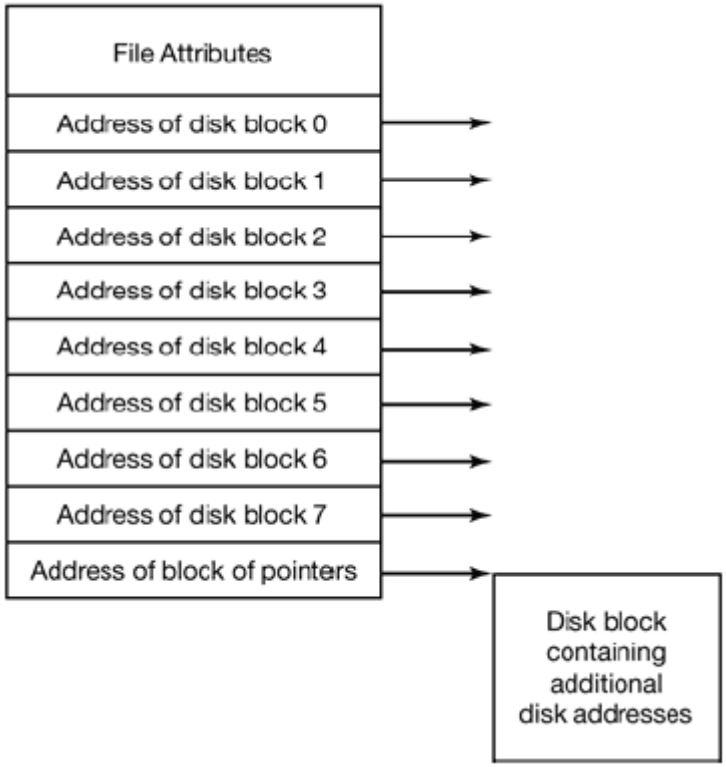
Λίστες από μπλοκ

- Τα αρχεία είναι στην πράξη λίστες από ανεξάρτητα μπλοκ στο δίσκο
  - Τα μπλοκ περιέχουν δεδομένα και σύνδεσμο στο επόμενο μπλοκ.
  - Δεν υπάρχει περιορισμός στο μέγεθος των αρχείων
- Νέα μπλοκ αποδίδονται σε ένα αρχείο όποτε χρειαστεί
  - Προστίθενται στη «λιστα» μπλοκ του αρχείου



i-node

- Δομή δεδομένων σε περιβάλλον Unix που περιγράφει ένα αντικείμενο του συστήματος αρχείων (αρχείο ή κατάλογο)
  - Metadata attributes
    - File size, file owner, group στο οποίο ανήκει
    - File access rights, link count, time stamps(last modified time, last accessed time, last changed time).
  - Data disk block locations
- Στη περίπτωση των καταλόγων αποθηκεύεται και το όνομα του γονέα καταλόγου καθώς και καθενός από τα παιδιά καταλόγους



System calls

- Βιβλιοθήκες λειτουργιών/API που μας δίνει το κάθε λειτουργικό σύστημα ώστε να μπορούμε να χρησιμοποιήσουμε τις «υπηρεσίες» του (όπως την εύρεση φακέλων, αρχείων κλπ) χωρίς να χρειάζεται να γνωρίζουμε όλες τις παραπάνω πληροφορίες ή τρόπους χειρισμού

Παράδειγμα open, read, write system calls (Linux)

- Open

SYNOPSIS top

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

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

- Read

SYNOPSIS top

```
#include <unistd.h>

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

DESCRIPTION top

read() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

- \

SYNOPSIS top

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

DESCRIPTION top

write() writes up to *count* bytes from the buffer starting at *buf* to the file referred to by the file descriptor *fd*.

Σημεία προσοχής

- Κάθε άνοιγμα αρχικοποιεί έναν file descriptor με offset default 0

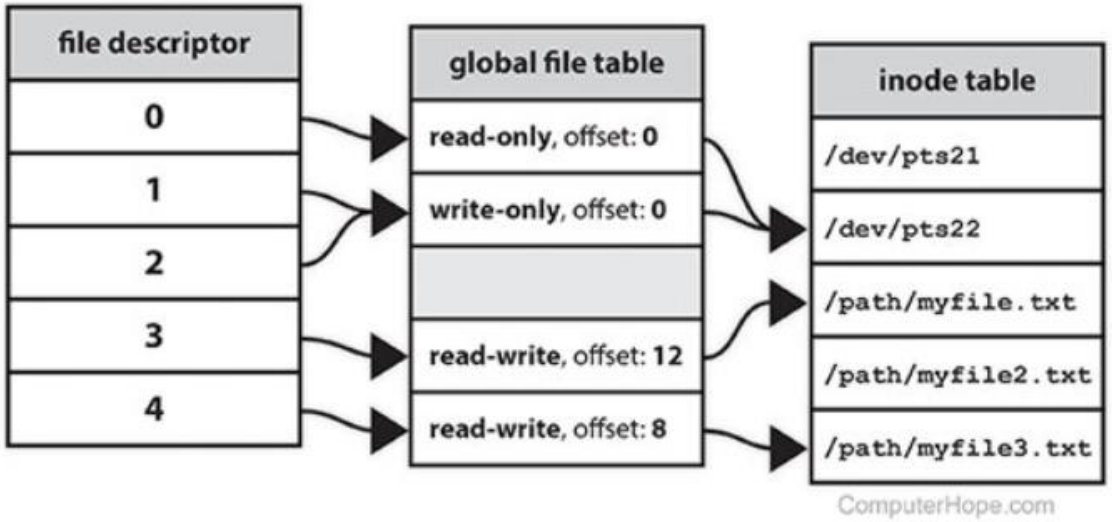
- Η πρόσβαση ξεκινάει από την αρχή του file εκτός αν μεταθέσουμε τον pointer του file descriptor κατά το άνοιγμα με την open system call
  - Στην fopen δεν υπάρχει αυτή η παράμετρος
  - Πρέπει να χρησιμοποιήσετε μετά την fseek ή να πάτε μόνοι σας τόσα byte μετά την αρχή

- Και στα system calls χειρισμού αρχείων και στην fopen έχουμε arguments σε σχέση με το τι mode ανοίγουμε το αρχείο
  - Read/write/append

```
00700 user (file owner) has read, write, and execute permission

00400 user has read permission

00200 user has write permission

00100 user has execute permission

00070 group has read, write, and execute permission

00040 group has read permission

00020 group has write permission

00010 group has execute permission

00007 others have read, write, and execute permission

00004 others have read permission

00002 others have write permission

00001 others have execute permission
```

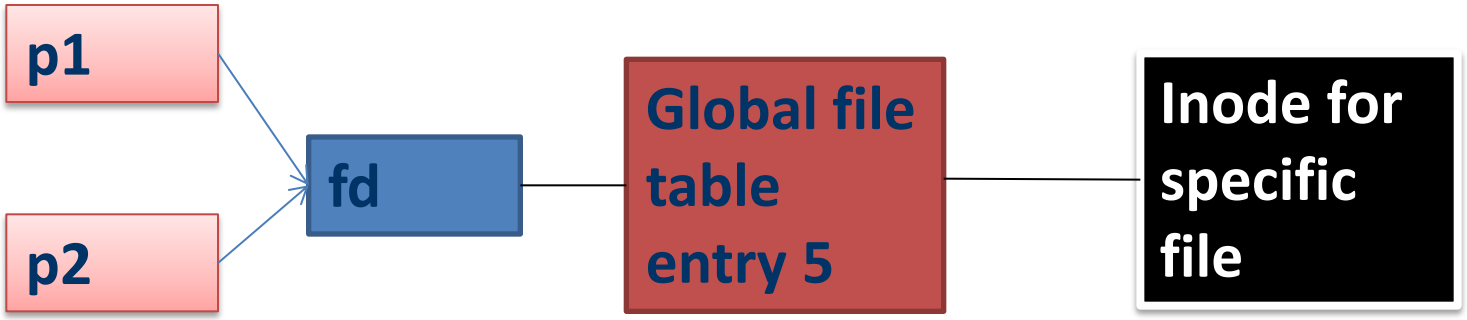
Μορφή δομής δεδομένων πληροφορίας ανοικτών αρχείων ΛΣ

- Κάθε φορά που ανοίγουμε ένα αρχείο δημιουργείται ένας file descriptor
  - Που δείχνει σε μια δομή που κρατάει τον τρόπο που έχει ανοίξει το αρχείο (mode, offset κλπ)
  - Επειδή έχετε πολλά προγράμματα μπορεί να έχετε πολλαπλά ανοίγματα οπότε και πολλαπλά entries στον global file table που να δείχνουν στο ιδιο αρχείο



Κοινός file descriptor?

- Μπορούμε επίσης να μοιραζόμαστε τον ίδιο fd που πάει προς το ίδιο file table entry
  - Ποια χρήση θα είχε αυτό?
  - Θα είχαμε το ίδιο offset, το οποίο σημαίνει ότι θα μπορούσαμε να έχουμε πολλαπλά κομμάτια κώδικα που να διαβάζουν σειριακά
    - Δηλαδή με μια λογική εκεί που σταματάει ο ένας να συνεχίζει ο άλλος



  - Αυτό είτε γίνεται συστηματικά (πχ fork) είτε με ειδική system call (dup)
  - Στη C μπορείτε απλά να χρησιμοποιείτε τον ίδιο FILE *fd στο ίδιο πρόγραμμα

Buffering (Επίπεδο C του προγράμματος σας)

- Χρήση ενδιάμεσης προσωρινής μνήμης για κάποια εργασία

- Χρήσεις (μεταξύ άλλων):
  - Συγχρονισμός μεταξύ εργασιών: αν δύο εργασίες δίνουν η μία στην άλλη, χρησιμοποιείται ο ενδιάμεσος buffer σαν σημείο αποθήκευσης και συγχρονισμού αν υπάρχει διαφορά στην ταχύτητα
  - Χειρισμός μεγαλύτερων blocks πληροφορίας στη C από ότι χρειάζεστε μια δεδομένη χρονική στιγμή για βελτίωση της απόδοσης
    - Πχ η πρόσβαση σε δεδομένα στο δίσκο θέλει χρήση της αντίστοιχης system call (χρονοβόρα διαδικασία), μετακίνηση στο δίσκο για ανάγνωση (πολύ χρονοβόρα διαδικασία) και την πραγματική ανάγνωση και μεταφορά των δεδομένων (μικρή καθυστέρηση)
    - Άρα αν θελουμε ένα συγκεκριμένο σημείο σε ένα αρχείο, καλό είναι να φέρνουμε και τις κοντινές περιοχές δεδομένων γιατί πολύ πιθανό να χρειαστούν
      - Χωρική και χρονική τοπικότητα
      - Στην επόμενη ανάγνωση του γειτονικού Byte δεν πάμε στο δίσκο αλλά στη μνήμη (και όχι μέσω system call)
    - Έτσι δεχόμαστε μια αύξηση στη μικρή καθυστέρηση της ανάγνωσης (διαβάζουμε πχ 100 bytes αντί για 1) αλλά γλυτώνουμε τα μεγάλα κομμάτια καθυστέρησης στα επόμενα Bytes που θα χρειαστούμε
  - Αυτός ο τρόπος χειρισμού (σε blocks)είναι κοινός στα υπολογιστικά συστήματα΄, πχ RAM pages

Buffering (Επίπεδο ΛΣ)

- Συνήθως οι stdin, stderr είναι buffered
- Οι άλλες ροές ανάλογα με την υλοποίηση

Προσοχή στο Offset

- Ένα αρχείο μπορεί να έχει ανοιχθεί πολλές φορές σε διαφορετικά σημεία του κώδικα (ή και σε άλλα προγράμματα)
  - Κάθε διαφορετικό άνοιγμα θα δείχνει σε διαφορετικό file descriptor
- Το Offset είναι το σημείο στο οποίο βρισκόμαστε κάθε στιγμή μέσα στο αρχείο σε ΑΥΤΟΝ τον file descriptor που χρησιμοποιούμε
- Μπορεί να μεταφερθεί με κατάλληλες εντολές (fseek)
- Κάθε εντολή ανάγνωσης ή εγγραφής ορίζει ποιον file descriptor θα χρησιμοποιήσει και χρησιμοποιεί την τιμή του offset ΑΥΤΟΥ του file descriptor για να πραγματοποιηθεί
- Άλλο το offset του desciptor του ΛΣ και άλλο του FILE στη C. Γιατί?

Συνολικές διαφορές fopen(C) από open (ΛΣ)

- Διαφορά fopen από open system call
  - fopen: C library, open: Linux system call
  - Η fopen επιστρέφει την πληροφορία σε ένα struct τύπου FILE με διάφορες πληροφορίες
    - Στην ουσία κάνει abstract τις πληροφορίες στο επίπεδο της C
    - Τα πεδία εξαρτώνται από την βιβλιοθήκη της C που χρησιμοποιεί ο κάθε compiler
  - Η open επιστρέφει σε έναν file descriptor τύπου int (index στον table του ΛΣ)
  - Η open μπορεί να μην υπάρχει με αυτόν τον ορισμό σε άλλου τύπου ΛΣ οπότε ο C κώδικας μπορεί να μην είναι portable
  - Η fread/write χρησιμοποιεί Buffering, που έχει πλεονέκτημα ταχύτητας σε σειριακή ανάγνωση ή ανάγνωση με τοπικότητα αλλά αν γενικά θέλουμε τυχαία ακανόνιστη προσπέλαση (οπότε είναι εκτός των ορίων του buffer η επόμενη θέση που θέλουμε) τότε κάθε φορά θα διαβάζουμε χωρίς αποτέλεσμα περισσότερα bytes από ότι χρειάζεται (μικρό πλεονέκτημα system calls από θέμα απόδοσης)
    - Λόγω του buffering εσείς μπορεί να χρειάζεστε 1 byte, αλλά να έχετε φέρει 100
    - Αυτά τα 100 μπορεί εσείς να μην τα έχετε διαβάσει όλα
    - **Για το λόγο αυτό το FILE έχει το δικό του offset που δείχνει μέχρι που έχετε διαβάσει από τον buffer**
    - **Όλη η λογική του buffering και της επικοινωνίας με το ΛΣ είναι κρυμμένη μέσα στην stdio της C**

- Σημασία fclose
  - Οι λειτουργίες γίνονται μέσω buffering (ενδιάμεση μνήμη)
  - Ένα write γράφει στον buffer, αλλά σε μια δεδομένη χρονική στιγμή μπορεί τα περιεχόμενα αυτού να μην έχουν μεταφερθεί στο αρχείο
  - Με το fclose γίνεται η αποθήκευση για ένα κομμάτι που είναι ακόμα στο Buffer

Συνολική ροή

- Καλείτε την C fopen σε ένα αρχείο
- Αυτή καλεί την open system call
  - Η open δημιουργεί το entry στο file table του ΛΣ
- Καλείτε την C fread για 1 Byte
- Αυτή καλεί την read system call για <buffer_size> bytes
- Η read βρίσκει από τον file table και το inode το block του δίσκου
- Φέρνει <buffer_size> bytes και το αποθηκεύει στην θέση του buffer που δείχνει το struct FILE
  - Προφανώς έχει γίνει το αντίστοιχο malloc κατά το άνοιγμα του αρχείου
  - Αλλάζει και το offset στο δείκτη ΛΣ από 0-> + <buffer_size> bytes
- Η fread διαβάζει από τον buffer 1 byte
  - Αλλάζει το offset κατά 1 στον FILE descriptor (από 0->1)
- Αν διαβάσετε αρκετές φορές και περάσει το όριο του buffer επαναλαμβάνεται η παραπάνω διαδικασία (μετά το κομμάτι του open)

Εργαστήριο 5

- Μορφή FILE struct (glibc 2.2.5)

- Προσοχή ότι αυτό μπορεί να αλλάζει σε διαφορετικές εκδόσεις της βιβλιοθήκης
- Συνήθως το _ πριν το όνομα μιας μεταβλητής υποδηλώνει ότι μπορείτε να έχετε πχ ένα Pointer προς αυτήν αλλά δεν πρέπει να αλλάξετε τα περιεχόμενά της
  - Απλά πρέπει να περάσετε τον Pointer προς μια άλλη function που καταλαβαίνει πώς να μεταβληθεί το περιεχόμενο με ασφάλεια

- Παράδειγμα χρήσης και πρόσβασης άλλων πεδίων της FILE από ένα ανοικτό αρχείο
  - ΔΕΝ ΤΟ ΧΡΗΣΙΜΟΠΟΙΟΥΜΕ ΠΟΤΕ ΑΥΤΟ

```
1  struct _IO_FILE {
2    int _flags;        /* High-order word is _IO_MAGIC; rest is flags. */
3  #define _IO_file_flags _flags
4
5    /* The following pointers correspond to the C++ streambuf protocol. */
6    /* Note:  Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
7    char* _IO_read_ptr;    /* Current read pointer */
8    char* _IO_read_end;    /* End of get area. */
9    char* _IO_read_base;    /* Start of putback+get area. */
10   char* _IO_write_base;    /* Start of put area. */
11   char* _IO_write_ptr;    /* Current put pointer. */
12   char* _IO_write_end;    /* End of put area. */
13   char* _IO_buf_base;    /* Start of reserve area. */
14   char* _IO_buf_end;    /* End of reserve area. */
15   /* The following fields are used to support backing up and undo. */
16   char *_IO_save_base; /* Pointer to start of non-current get area. */
17   char *_IO_backup_base;  /* Pointer to first valid character of backup area */
18   char *_IO_save_end; /* Pointer to end of non-current get area. */
19
20   struct _IO_marker *_markers;
21
22   struct _IO_FILE *_chain;
23
24   int _fileno;
25 #if 0
26   int _blksize;
27 #else
28   int _flags2;
29 #endif
30   _IO_off_t _old_offset; /* This used to be _offset but it's too small.  */
31
32 #define __HAVE_COLUMN /* temporary */
33   /* 1+column number of pbase(); 0 is unknown. */
34   unsigned short _cur_column;
35   signed char _vtable_offset;
36   char _shortbuf[1];
37
38   /*  char* _save_gptr;  char* _save_egptr; */
39
40   _IO_lock_t *_lock;
41 };
```