PB173 Linux

09 Súbory

Roman Lacko xlacko1@fi.muni.cz

2022-11-18

Obsah

Súborový systém

Súborový systém

Súborový systém

- Spôsob uloženia dát na disku
- Prístup k objektom (súbory, adresáre, ...)

Aspekty

- Spôsob prideľovania pamäte
- Podpora hierarchií
- Metadáta

Súborový systém: UNIX

Virtual File System

- Rozhranie jadra
- Abstrakcia rozdielov medzi súborovými systémami

Vlastnosti

- Jeden strom
- Koreň v adresári /
- Ostatné systémy pripojené na rôzne cesty

Súborový systém: FHS

Filesystem Hierarchy Standard (FHS)

- 1. úroveň

`/bin`	Aplikácie nutné pre beh
`/etc`	Konfiguračné súbory
`/lib`	Systémové knižnice, tiež niekedy `/lib32` a `/lib64`
`/sbin`	Aplikácie pre superpoužívateľov
`/tmp`	Dočasné súbory, ktoré nebudú chýbať

Súborový systém: FHS

- 2. úroveň

  `/usr/{bin,lib,sbin,…}`
  Doplnkové aplikácie,
  nie nutné pre beh systému

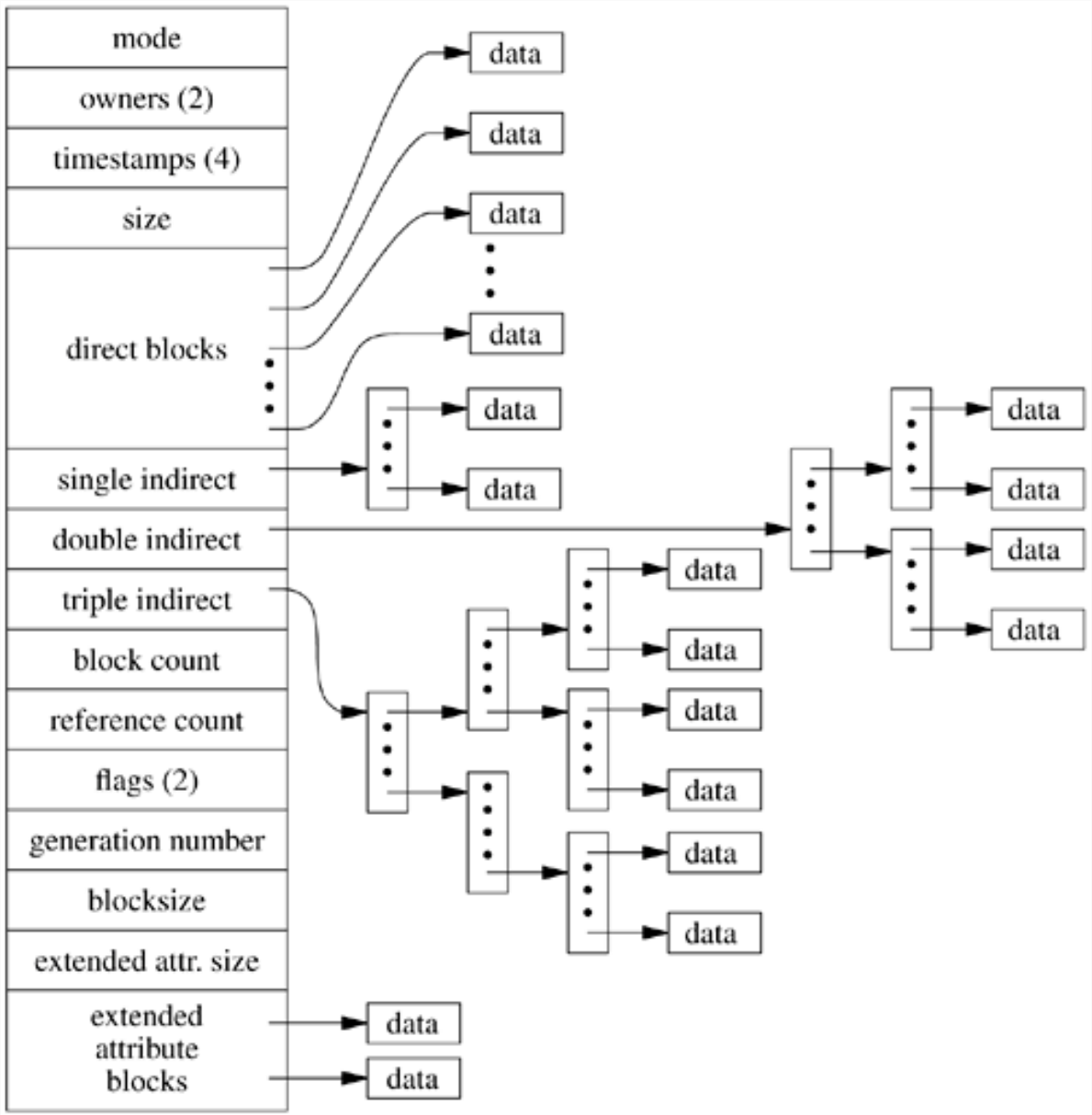
- 3. úroveň

  `/usr/local/{bin,lib,sbin,…}`
  Aplikácie získané mimo distribučné balíky

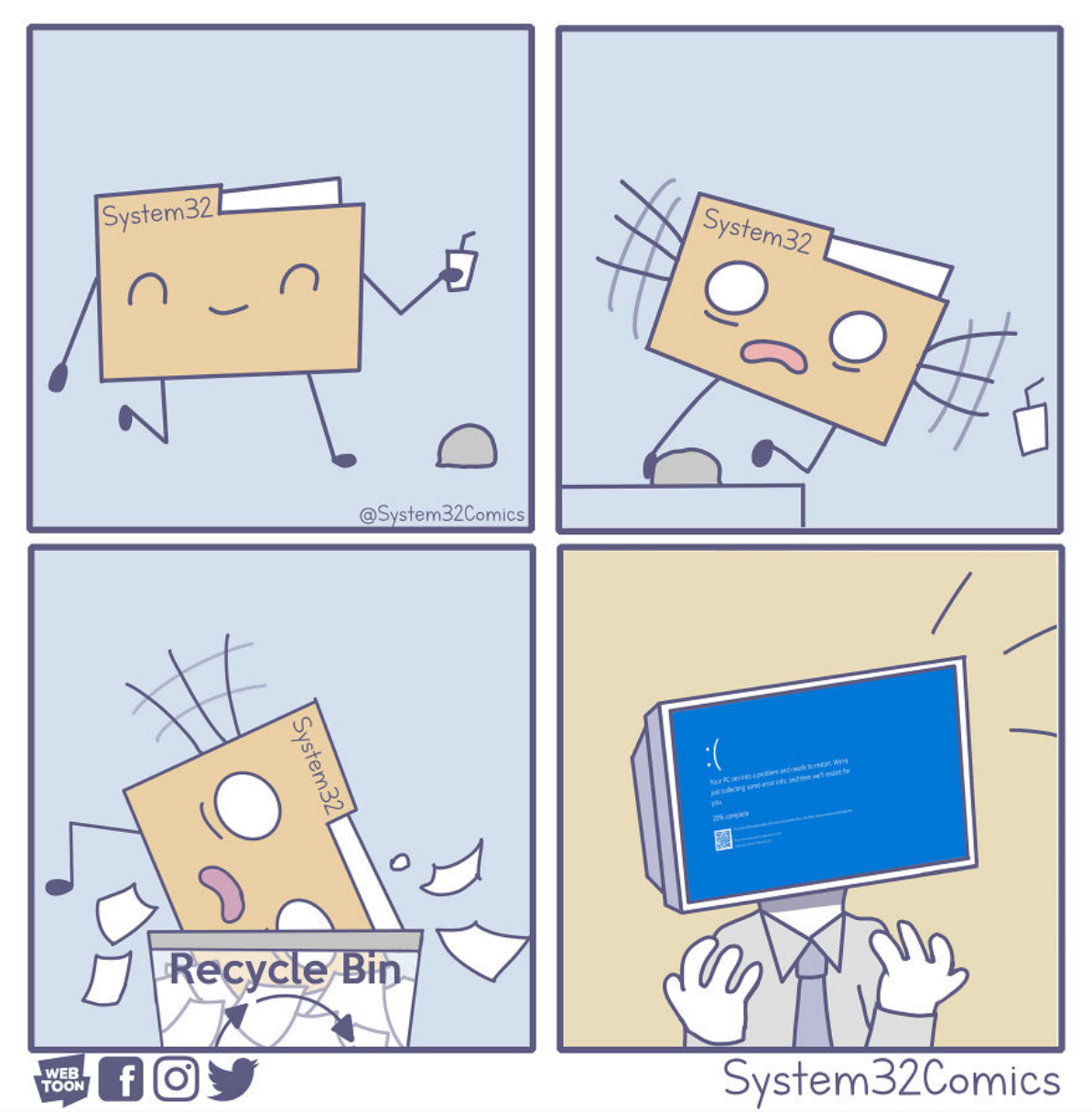
💬 *Unified* /usr

Niektoré distribúcie majú 1. a 2. úroveň zjednotenú
a poskytujú symbolické odkazy /α → /usr/α.

Súborový systém: i-uzol



Súborový systém: Objekty



Súborový systém: Bežné súbory

Regular File

- Obsahuje bajty, význam záleží od aplikácie
- *I-uzol* obsahuje oprávnenia a mapovanie na bloky disku
  - **Neobsahuje** názov súboru!
  - Veľkosť v i-uzle môže byť väčšia, než mapované bloky (*sparse files*)

💬 *File vs Regular File*

- *File* (súbor) je všeobecný pojem pre akýkoľvek objekt FS.
- *Regular file* (bežný súbor) je objekt, ktorý obsahuje používateľom prístupné dáta.

Často *file* ≈ *regular file* podľa kontextu.

Súborový systém: Bežné súbory

Podobnosť C99 vs POSIX.1-2008

Pozor na rozdielnu sémantiku, viď manuál.

C99	POSIX
`#include <stdio.h>`	≈ `#include <unistd.h>`
`FILE *file`	≈ `int fd`
`fopen(path, mode)`	≈ `open(path, flags, 0666)`
`fclose(file)`	≈ `close(fd)`
`fread(buf, size, n, file)`	≈ `read(fd, buf, size * n)`
`fwrite(buf, size, n, file)`	≈ `write(fd, buf, size * n)`

Súborový systém: Deskriptor

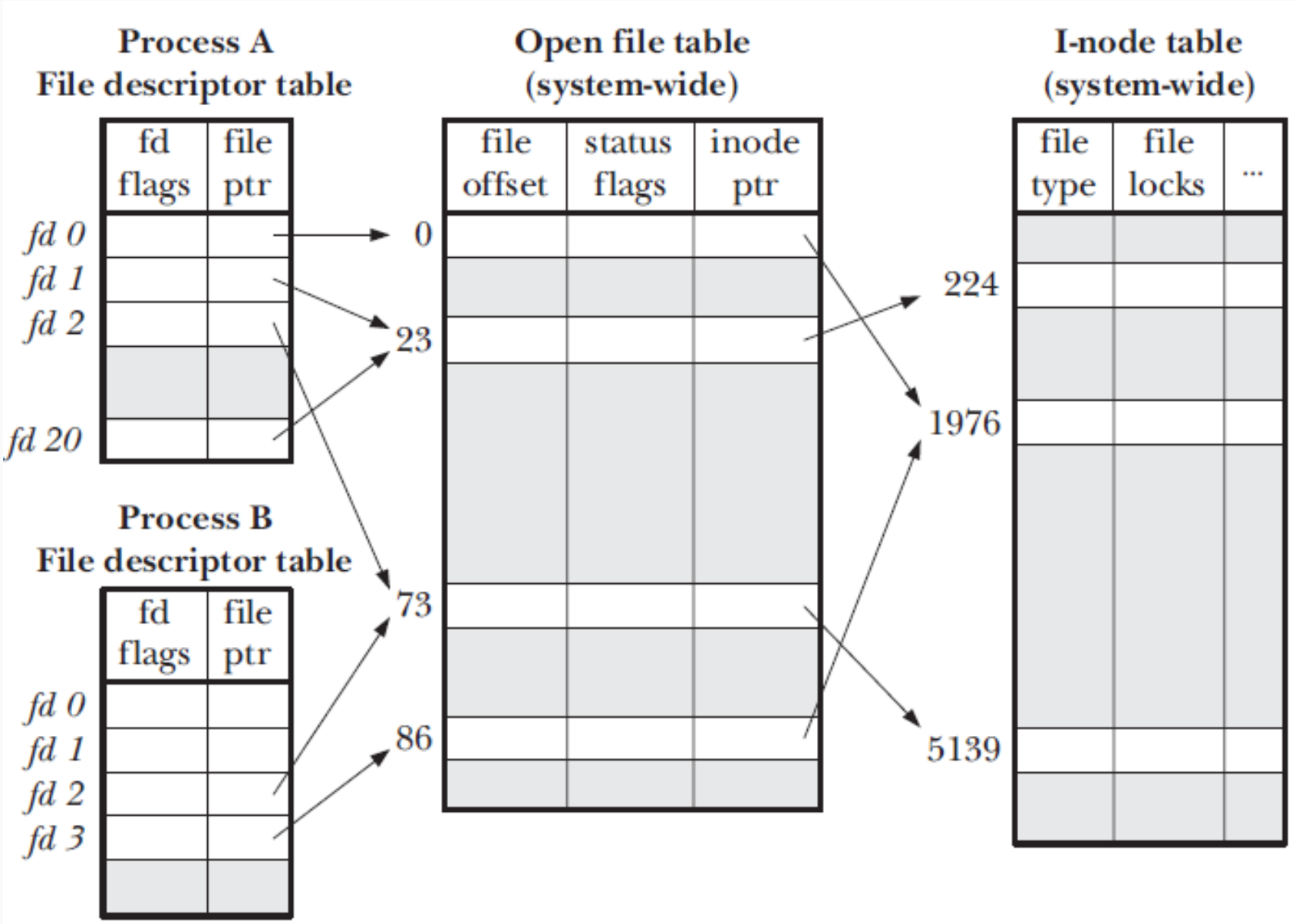
File Descriptor

- Abstraktná reprezentácia zdroja v UNIXe
  - Súbor, rúra, sieťové spojenie, monitory, ...
- Malé nezáporné číslo typu `int`
- Porovnajte s `HANDLE` vo WinAPI

(Open) File Description

- Záznam kernelu o otvorenom súbore
- Môže byť odkazovaný viacerými deskriptormi
- Môže byť zdieľaný medzi procesmi

Súborový systém: Deskriptor



Súborový systém: Konvencie

Konvencie názvov funkcií

`ζ()`	Základná funkcia
`lζ()`	**Ne**dereferencuje odkazy
`fζ()`	Pracuje s deskriptorom namiesto cesty
`fdζ()`	Alternatíva `fζ()`
`ζat()`	Relatívne umiestnenie od zadaného adresára
`ζdir()`	Operácia s adresárom

Občas sa môžu nájsť aj kombinácie, napr. `fζat()`.

Nie vždy presné, čítajte manuál.

Súborový systém: Bežné súbory

```c
#include <fcntl.h>  /* open(), O_a, S_a */
#include <unistd.h> /* close() */

int open(const char *path, int flags /*, mode_t mode */);
int close(int fd);

#define O_RDONLY    /* … */  /* Access mode (one of) */
#define O_WRONLY    /* … */
#define O_RDWR      /* … */

#define O_APPEND    /* … */  /* Additional flags */
#define O_CLOEXEC   /* … */
#define O_CREAT     /* … */ // Requires <mode>!
#define O_EXCL      /* … */
#define O_NONBLOCK  /* … */
#define O_TRUNC     /* … */
```

Súborový systém: Bežné súbory

```c
#include <fcntl.h>
int openat(int dirfd, const char *path, int flags, /* mode_t mode
#define AT_FDCWD /* … */
```

```
open(π, φ, μ) ≈ openat(AT_FDCWD, π, φ, μ)
```

```c
int creat(const char *path, mode_t mode);
```

```
creat(π, μ)
≈ openat(AT_FDCWD, π, O_WRONLY | O_CREAT | O_TRUNC, μ)
```

```c
#include <stdio.h>
FILE *fdopen(int fd, const char *mode);
int fileno(FILE *stream);
```

Súborový systém: Bežné súbory

Prístupové práva k vytvorenému súboru sú dané kombináciou

`mode` Parameter pre `open()` alebo `creat()`

`umask` (*user*) *file mode creation mask*, atribút procesu

```
#include <sys/stat.h>
mode_t umask(mode_t mask);

//              Read      Write    eXecute     RWX
// user:     S_IRUSR   S_IWUSR   S_IXUSR    S_IRWXU
// group:    S_IRGRP   S_IWGRP   S_IXGRP    S_IRWXG
// others:   S_IROTH   S_IWOTH   S_IXOTH    S_IRWXO
//             ↑         ↑         ↑             ↑

// special: S_ISUID   S_ISGID   S_ISVTX
```

Súborový systém: Bežné súbory

```c
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

- Používané aj pre iné typy zdrojov (rúry, monitory, ...)
- `O_NONBLOCK`

Môžu zapísať alebo prečítať menej dát, než `count`.
To nie je chyba.

Súborový systém: Bežné súbory

```c
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
#define SEEK_SET    /* … */
#define SEEK_CUR    /* … */
#define SEEK_END    /* … */

#define SEEK_DATA   /* … */ // Linux specific
#define SEEK_HOLE   /* … */

ssize_t pread(int fd, void *buf, size_t size, off_t off);
ssize_t pwrite(int fd, const void *buf, size_t size, off_t off);
```

Rúry

Rúry

Rúra

- Jednosmerný prúd dát
- Dvojica deskriptorov (zápis → čítanie)
- Obmedzená veľkosť (`PIPE_BUF`)
- Nepomenované
  - `ls "Futurama-*.mkv" | shuf | xargs -d'\n' mpv`
  - IPC medzi príbuznými procesmi
- Pomenované
  - Objekty v súborovom systéme
  - IPC medzi (obvykle) nepríbuznými procesmi

Rúry: Proces

```c
#include <stdio.h>

FILE *popen(const char *command, const char *type);
int pclose(FILE *stream);
```

V princípe rovnaké ako volanie `system(command)`.

Rúry: Nepomenované

```c
#include <unistd.h>
int pipe(int pipefd[2]);
```

`pipefd[0]` Čítajúci koniec
0 ako stdin

`pipefd[1]` Zapisujúci koniec
1 ako stdout

Odbočka: Kópia deskriptora

```c
#include <unistd.h>

int dup(int fd);
int dup2(int fd, int newfd);
int dup3(int fd, int newfd, int flags);
```

Typické použitie

Nahradenie štandardných deskriptorov

Rúry: Pomenované

Pomenované rúry

- Špeciálne súbory v súborovom systéme
- Práca skoro ako s bežným súborom
- Shell:

```
mkfifo FILE
```

- C:

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

/* mknod(…), mknodat(…) + S_IFIFO */

int mkfifo(const char *pathname, mode_t mode);
int mkfifoat(int dirfd, const char *pathname, mode_t mode);
```

Adresáre

Adresáre

- Mapujú názvy na i-uzly,
  $\Sigma^+ \to \mathbb{N}$
- Špeciálne záznamy:

  `.` aktuálny adresár

  `..` nadradený adresár

Adresáre

```c
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *name);
int closedir(DIR *dir);

int dirfd(DIR *dir);
DIR *fdopendir(int fd);
```

Adresáre

```c
#include <dirent.h>
void rewinddir(DIR *dirp);
struct dirent *readdir(DIR *dirp);

struct dirent {
    ino_t d_ino;
    off_t d_off;
    unsigned char d_type;
    char d_name[256];
};

#define DT_DIR     /* … */
#define DT_LNK     /* … */
#define DT_REG     /* … */
#define DT_UNKNOWN /* … */ // Must be checked!
```

Adresáre

```c
#include <sys/stat.h> /* + other headers for modes */
int mkdir(const char *pathname, mode_t mode);
int mkdirat(int dirfd, const char *pathname, mode_t mode);

#include <unistd.h>
int rmdir(const char *pathname);
// rmdirat() → unlinkat(…, …, AT_REMOVEDIR)
```

Adresáre

Pracovný adresár

- Vlastnosť procesu
- Rezolúcia relatívnych ciest

```
char *getcwd(char *buf, size_t size);
/* char *getwd(char *buf); */        // AVOID
char *get_current_dir_name(void);   // GNU

int chdir(const char *path);
int fchdir(int fd);
```

Adresáre

Linux

Nepoužívať priamo, toto je len pre ukážku.

```
int open(const char *path, int flags, /* mode_t mode */);
#define O_DIRECTORY /* … */

ssize_t getdents64(int fd, void *dirp, size_t count);

struct linux_dirent64 {
    ino64_t d_ino;
    /* … */
    char d_name[];
};
```

Odkazy

Odkazy

Názov súboru je záznam v adresári, nie je uložený v i-node súboru.

Pevný odkaz Priradenie názvu k i-uzlu v zázname adresára.

Symbolický odkaz Súbor, ktorý obsahuje cestu (jednu z možných) k súboru.

Odkazy

```c
#include <unistd.h>
int link(const char *target, const char *linkpath);
int linkat(int fd1, const char *p1, … fd2, … *p2, int flags);


int symlink(const char *target, const char *linkpath);
int symlinkat(… *target, int newdirfd, … *linkpath);

ssize_t readlink(const char *pathname, char *buf, size_t bufsz);
ssize_t readlinkat(int dirfd, … *pathname, … *buf, … bufsz);

#include <stdlib.h>
char *realpath(const char *path, char *resolved_path);
#define PATH_MAX /* … */    // In ‹limits.h›
```

Odkazy

```c
#include <unistd.h>
int access(const char *path, int mode);
int faccessat(int dirfd, … *pathname, … mode, int flags);
#define F_OK /* … */
#define R_OK /* … */
#define W_OK /* … */
#define X_OK /* … */
```

! Pozor na súbeh

Po skončení access() nemusí byť informácia aktuálna.

Odkazy

```
int unlink(const char *pathname);
int unlinkat(int dirfd, … *pathname, int flags);

#include <stdio.h>
int rename(const char *oldpath, const char *newpath);
int renameat(int oldfd, const char *oldpath, … newfd, … *newpath);
int renameat2(… oldfd, … oldpath, … newfd, … newpath, int flags);
```

! Pozor

Premenovanie nahradí existujúci odkaz!

Odkazy

Ako bezpečne premenovať súbor?

```c
#include <stdio.h>
#include <unistd.h>

int rename_safe(const char *oldpath, const char *newpath) {
    if (access(newpath, F_OK) == 0) {
        errno = EEXIST; /* File already exists */
        return -1;
    }

    return rename(oldpath, newpath);
}
```

Aký problém je v tejto ukážke?

Odkazy

Ako bezpečne pepísať súbor?

```
int replace_file(const char *file, size_t *bsize,
        const char data[bsize]) {
    int fd = open(file, O_WRONLY | O_TRUNC | O_CREAT, …);
    if (fd == -1)
        return -1;

    while (…) {
        write(fd, …);
    }

    close(fd);
    return 0;
}
```

Aký problém by mohol nastať v tejto ukážke?

Vlastnosti

Vlastnosti: Zistenie

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

int stat(const char *path, struct stat *buf);
int lstat(const char *path, struct stat *buf);
int fstat(int fd, struct stat *buf);
int fstatat(int dirfd, … *path, … *buf, int flags);

struct stat {
    mode_t st_mode;     /* File type and mode */
    uid_t  st_uid;      /* Owner */
    gid_t  st_gid;      /* Group */
    off_t  st_size;
    /* … */
};
```

Vlastnosti: Zmena

Vlastník, skupina a práva

```
#include <unistd.h>
int chown(const char *path, uid_t owner, gid_t group);
int lchown(const char *path, uid_t owner, gid_t group);
int fchown(int fd, uid_t owner, gid_t group);
int fchownat(int dirfd, … *path, … owner, … group, int flag);

#include <sys/stat.h>
int chmod(const char *path, mode_t mode);
int fchmod(int fd, mode_t mode);
int fchmodat(int dirfd, … *path, mode_t mode, int flags);
```

Prečo neexistuje `lchmod()`?

Vlastnosti: Zmena

Veľkosť a obsadenie blokov

```c
#include <unistd.h>
int truncate(const char *path, off_t length);
int ftruncate(int fd, off_t length)

#include <fcntl.h>
int fallocate(int fd, int mode, off_t offset, off_t len);
#define FALLOC_FL_PUNCH_HOLE /* … */
#define FALLOC_FL_COLLAPSE_RANGE /* … */
#define FALLOC_FL_ZERO_RANGE /* … */

int posix_fallocate(int fd, off_t offset, off_t len);
```

`ftruncate()` vie pracovať aj na iných objektoch
napr. na zdieľanom segmente pamäte

Vlastnosti: Zmena

Čas posledného prístupu a modifikácie

```
#include <sys/time.h>
#include <sys/stat.h>
int utime(const char *path, const struct utimbuf *time);

int utimes(const char *path, const struct timeval times[2]);
int lutimes(const char *path, … times[2]);
int futimes(int fd, … times[2]);
int futimesat(int dirfd, const char *path, … times[2]);

int futimens(int fd, … times[2]);
int utimensat(int dirfd, const char *pathname, … times[2], int fla
```

Ďalšie možnosti

Rýchle kopírovanie dát

Kopírovanie medzi soketmi a súbormi

```
#include <sys/sendfile.h>
ssize_t sendfile(int ofd, int ifd, off_t *offset, size_t count);
```

- `ifd` objekt okrem soketu
- `ofd` soket alebo bežný súbor

```
#define _GNU_SOURCE
#include <unistd.h>
ssize_t copy_file_range(int fd_in, loff_t *off_in,
    int fd_out, loff_t *off_out,
    size_t len, unsigned int flags)
```

- Oba deskriptory musia byť bežné súbory

Rýchle kopírovanie dát

Prenos dát medzi rúrami

```
#include <fcntl.h>
ssize_t splice(int fd_in, loff_t *off_in,
    int fd_out, loff_t *off_out,
    size_t len, unsigned int flags);
```

- Aspoň jeden z deskriptorov musí byť rúra.

```
ssize_t tee(int fd_in, int fd_out, size_t len,
        unsigned int flags);
```

- **Kopíruje** dáta z jednej rúry do druhej
- V zdrojovej rúre dáta zostanú

Rýchle kopírovanie dát

Kopírovanie stránok pamäte do rúry

```
#include <fcntl.h>
ssize_t vmsplice(int fd, const struct iovec *iov,
    unsigned long nr_segs, unsigned int flags);
```

Rekurzívne prechádzanie adresárov

File Tree Walk

Push-based prechádzanie adresárovej štruktúry

```
#include <ftw.h>
int nftw(const char *path,
    int (*fn)(const char *fpath, const struct stat *sb,
    int typeflag, struct FTW *ftwbuf),
    int nopenfd, int flags);

int ftw(const char *path,
    int (*fn)(const char *fpath, const struct stat *sb,
    int typeflag, struct FTW *ftwbuf),
    int nopenfd);
```

Rekurzívne prechádzanie adresárov

POSIX scandir

Push-based prechádzanie adresárovej štruktúry

```
#include <dirent.h>
int scandir(const char *dirp, struct dirent ***namelist,
    int (*filter)(const struct dirent *),
    int (*cmp)(const struct dirent **, const struct dirent **));

int alphasort(const struct dirent **a, const struct dirent **b);

/* GNU extensions */
int versionsort(const struct dirent **a, const struct dirent **b);
int scandirat(int dirfd, const char *dirp,
    /* rest same as for scandir() */);
```

Rekurzívne prechádzanie adresárov

POSIX scandir

Pull-based prechádzanie adresárovej štruktúry

```
#include <fts.h>
FTS *fts_open(char *const *path_argv, int options,
    int (*cmp)(const FTSENT **, const FTSENT **));
int fts_close(FTS *ftsp);

FTSENT *fts_read(FTS *ftsp);
FTSENT *fts_children(FTS *ftsp, int options);
int fts_set(FTS *ftsp, FTSENT *f, int instr);
```

Špeciálne objekty

- Zariadenia
- Pomenované rúry (FIFO)
- Sokety

Ďalšie podľa podpory OS a súborového systému

- *Doors*, Solaris
- *Forks*, MacOS (HFS), Windows (NTFS)
  tiež *Alternate Data Streams*
- *Junctions*, Windows (NTFS)

Záver

Zdroje

- Understanding Linux Filesystems
- Everything you ever wanted to know about inodes