Práce se soubory

Tématicky zaměřený vývoj aplikací v jazyce C
skupina Systémové programování – Linux

Jiří Novosad

Fakulta informatiky
Masarykova univerzita
novosad@fi.muni.cz

Brno, 5. 11. 2014

Práce se soubory

Základy, které znáte a pokročilé techniky, které (možná) neznáte

Základní operace

aneb co byste měli znát

- Otevření/zavření souboru

```
int open(const char *pathname, int flags, ... /* mode_t mode */);
int close(int fd);
```

- Přejmenování souboru

```
int rename(const char *oldpath, const char *newpath);
```

- Smazání souboru (odkazu na soubor)

```
int unlink(const char *pathname);
```

- Čtení/zápis do souboru – `read()`, `write()`, ...

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
off_t lseek(int fd, off_t offset, int whence);
...
```

Základní operace

- Práce s adresáři

```
int mkdir(const char *pathname, mode_t mode);
int rmdir(const char *pathname);
DIR *opendir(const char *name);
struct dirent *readdir(DIR *dirp);
void rewinddir(DIR *dirp);
int closedir(DIR *dirp);
...
```

- Práce s odkazy a se symbolickými odkazy

```
int link(const char *oldpath, const char *newpath);
int symlink(const char *target, const char *linkpath);
ssize_t readlink(const char *pathname, char *buf, size_t bufsiz);
```

Přístupová práva souboru

- Kontrola přístupových práv – `F_OK`, `R_OK`, `W_OK`, `X_OK`
- Kontrolují se podle reálných práv procesu

```
#include <unistd.h>
int access(const char *pathname, int mode);
```

Přístupová práva souboru

- Kontrola přístupových práv – `F_OK`, `R_OK`, `W_OK`, `X_OK`
- Kontrolují se podle reálných práv procesu

```
#include <unistd.h>
int access(const char *pathname, int mode);
```

- Maska práv při vytváření souborů

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

- `permissions & ¬mask`
- `S_ISUID`, `S_ISGID`, `S_ISVTX`
- `S_IRWXU`, `S_IRUSR`, `S_IWUSR`, `S_IXUSR`
- `S_IRWXG`, `S_IRGRP`, `S_IWGRP`, `S_IXGRP`
- `S_IRWXO`, `S_IROTH`, `S_IWOTH`, `S_IXOTH`

Vlastnosti souboru

- Čas přístupu a modifikace souboru

```
#include <sys/types.h>
#include <utime.h>
int utime(const char *filename, const struct utimbuf *times);

#include <sys/time.h>
int utimes(const char *filename, const struct timeval times[2]);

#include <sys/stat.h>
int utimensat(int dirfd, const char *pathname,
              const struct timespec times[2], int flags);
```

Vlastnosti souboru

- Čas přístupu a modifikace souboru

```
#include <sys/types.h>
#include <utime.h>
int utime(const char *filename, const struct utimbuf *times);

#include <sys/time.h>
int utimes(const char *filename, const struct timeval times[2]);

#include <sys/stat.h>
int utimensat(int dirfd, const char *pathname,
              const struct timespec times[2], int flags);
```

- Komplexní informace o souboru

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int stat(const char *path, struct stat *buf);
int fstat(int fd, struct stat *buf);
int lstat(const char *path, struct stat *buf);
```

fcntl()

```
#include <unistd.h>
#include <fcntl.h>
int fcntl(int fd, int cmd, ... /* arg */ );
```

- Rozhraní pro řídící operace na souborových deskriptorech.
  - změna režimu práce (blokující / neblokující)
  - zamykání souborů
  - ...
- Operace v zásadě generické, ale jejich význam se může lišit podle typu souboru

fcntl()

```
#include <unistd.h>
#include <fcntl.h>
int fcntl(int fd, int cmd, ... /* arg */ );
```

- Rozhraní pro řídící operace na souborových deskriptorech.
  - změna režimu práce (blokující / neblokující)
  - zamykání souborů
  - ...
- Operace v zásadě generické, ale jejich význam se může lišit podle typu souboru

Změna režimu práce se souborem

- `F_GETFL`, `F_SETFL`, `F_GETFD`, `F_SETFD`
- Režim můžeme nastavit už při `open()`, ale někdy máme k dispozici jen deskriptor (standardní vstup/výstup, `pipe()`).

Zámykání souborů I

`/proc/locks`

Nevynucené (*Advisory*) zamykání

- dobrovolné, spoléhá na spolupráci všech procesů (podobně jako při synchronizaci a řízení přístupu ke sdíleným zdrojům)
- nemá přímý vliv na I/O operace
- zámek je uvolněn i při zavření kteréhokoliv deskriptoru, který ukazuje na stejný soubor!
- `F_GETLK`, `F_SETLK`, `F_SETLKW`, struktura `flock` pak určuje jak zamykáme (`F_RDLCK`, `F_WRLCK`, `F_UNLCK`) a co všechno (interval bytů)

Zámykání souborů II

Vynucené (*Mandatory*) zamykání

- problematická implementace v Linuxu (race condition, nespolehlivé)
- má přímý vliv na I/O operace
- může způsobit výrazné zpomalení podsystému I/O
- striktní, ale vyžaduje speciální nastavení FS a práv

```
# mount -o mand
# chmod g+s,g-x file
```

Zamykání částí souborů nefunguje se streamovými funkcemi (`stdio.h`), proto je třeba v takovém případě používat nízkoúrovňové funkce `read()`, `write()`

Zámykání souborů III

lockf()

- Zapouzdřené volání `fcntl()` (Linux)

```
#include <unistd.h>
int lockf(int fd, int cmd, off_t len);
```

- Od aktuální pozice v souboru
- Zámek je vždy exkluzivní
- Operace `F_LOCK`, `F_ULOCK`, `F_TEST`, `F_TLOCK`

flock()

- Zamknutí **celého** (již otevřeného) souboru
- Jiný typ zámku než předešlé funkce (Linux, vyjímka: NFS)

```
#include <sys/file.h>
int flock(int fd, int operation);
```

- Operace `LOCK_SH`, `LOCK_EX`, `LOCK_UN` (`| LOCK_NB`)

úkol

- demonstrace zamykání (celého) souboru – program zamkne soubor a uvolní ho až na pokyn uživatele (stisk enteru)
- spusťte 2 instance programu a zkontrolujte jeho funkčnost

Kopírování dat mezi soubory

- Přímé přenesení dat z jednoho souboru do druhého – zero copy
- Operace bez mezipřenosu do uživatelského prostoru
- Možné problémy s přenositelností – běžný soubor může být výstupním souborem až od verze jádra 2.6.22

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

Kopírování dat mezi soubory

- Přímé přenesení dat z jednoho souboru do druhého – zero copy
- Operace bez mezipřenosu do uživatelského prostoru
- Možné problémy s přenositelností – běžný soubor může být výstupním souborem až od verze jádra 2.6.22

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

- Od jádra 2.6.17 jsou dostupné funkce pro kontrolu kernel bufferu
- `splice(2)`, `tee(2)`, `vmsplice(2)`

Událostmi řízené programování

Zde v kontextu souborů

Čekání na změnu stavu souboru I

- Sledujeme skupinu file deskriptorů a pokud nastane očekávaná událost (příchozí data), obsloužíme ji
- Vyhýbáme se busy-waitingu
- File deskriptory by měly být v neblokujícím režimu

Čekání na změnu stavu souboru II

select()

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
int pselect(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, const struct timespec *timeout,
           const sigset_t *sigmask);
```

- Pro práci s množinami file deskriptorů slouží makra `FD_CLR`, `FD_ISSET`, `FD_SET`, `FD_ZERO`, `FD_SETSIZE`
- Po návratu obsahují množiny pouze deskriptory souborů, kde nastala očekávaná událost
- Před každým voláním je třeba nastavit množiny deskriptorů znovu a po návratu celou množinu postupně kontrolovat

Čekání na změnu stavu souboru III

poll()

```
#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
#define _GNU_SOURCE
int ppoll(struct pollfd *fds, nfds_t nfds,
            const struct timespec *timeout_ts, const sigset_t *sigmask)

struct pollfd {
    int   fd;         /* file descriptor */
    short events;     /* requested events */
    short revents;    /* returned events */
};
```

- Méně přenositelné než `select()`
- Místo masky signálů je použita speciální struktura → pro každý deskriptor je možné specifikovat události, které nás zajímají

Čekání na změnu stavu souboru IV

epoll_*

```
#include <sys/epoll.h>
int epoll_create(int size);
int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);
int epoll_wait(int epfd, struct epoll_event *events,
               int maxevents, int timeout);
```

- Nepřenositelná skupina funkcí
- Nezávislá manipulace s každým deskriptorem
- Informace o nastalých událostech jsou dostupné lineárně (není třeba prohledávat)

Monitoring událostí na souborech – *inotify*

- monitoring souborů a adresářů (ne rekurzivně!)
- od jádra 2.6.13, specifický mechanismus pro Linux
- nahrazuje *dnotify*

```
#include <sys/inotify.h>
int inotify_init(void);
int inotify_init1(int flags);
int inotify_add_watch(int fd, const char *pathname, uint32_t mask);
int inotify_rm_watch(int fd, int wd);
```

- pracujeme s *inotify* **file deskriptorem**
- *flags* mohou být *IN_NONBLOCK* a *IN_CLOEXEC*
- hodnoty pro masku sledovaných událostí viz `man 7 inotify`
- *wd* je watch deskriptor vrácený funkcí `inotify_add_watch()`

Zpracování *inotify* událostí

- čtení (`read(2)`) z *inotify* fd vrací buffer s jednou nebo více strukturami `inotify_event`
- čtení je blokující, lze použít i `select()/poll()/epoll()`
- velikost jednoho záznamu `inotify_event` je `sizeof(struct inotify_event) + len`

```
struct inotify_event {
    int      wd;       /* Watch descriptor */
    uint32_t mask;     /* Mask of events */
    uint32_t cookie;   /* Unique cookie associating related
                          events (for rename(2)) */
    uint32_t len;      /* Size of name field */
    char     name[];   /* Optional null-terminated name */
};
```

Práce se zařízeními

ioctl

```
#include <sys/ioctl.h>
int ioctl(int d, int request, ...);
```

- víceúčelové rozhraní pro řízení technického zařízení
- první argument je deskriptor souboru zařízení
- druhým argumentem je kód požadavku/operace, který dále určuje zbylé argumenty – viz `man 2 ioctl_list`
- operace jsou většinou vysoce specifické podle typu zařízení a vyžadují detailní znalost ovladačů
- Obdoba `fcntl()`

úkol

- naimplementujte vlastní verzi programu `eject(1)`
- využijte postupy reverzního inženýrství (strace)

Závěr

shrnutí, domácí úkoly a zdroje

Domácí úkol

Dropbox – 1.část

- vytvořte program, který monitoruje změny v zadaném adresáři
- `./dropbox source target`
- po spuštění nakopíruje obsah adresáře `source` do adresáře `target`
- neuvažujte podadresáře
- pomocí `inotify` průběžně synchronizujte změny
- po probrání síťové komunikace doplníte aplikaci o synchronizaci se vzdáleným serverem

Zdroje

- www.makelinux.net/ldd3/chp-6-sect-3.shtml
- www.kernel.org/doc/Documentation/filesystems/mandatory-locking.txt
- www.unixguide.net/network/socketfaq/2.14.shtml
- daniel.haxx.se/docs/poll-vs-select.html
- www.kegel.com/c10k.html
- www.gnu.org/s/libc/manual/html_node/IOCTLs.html
- man7.org/linux/man-pages/man7/inotify.7.html
- www.hackinglinuxexposed.com/articles/20030623.html