Systemy Operacyjne #8

Podstawowe usługi plikowe w systemach Windows i Linux

swernikowski@zut.edu.pl

(wersja: 2021-10-1)

{lin,win} Reprezentowanie plików na poziomie podstawowych usług systemu operacyjnego

- otwarty plik reprezentowany jest przez tzw. **deskryptor pliku** (ang. *file descriptor*)
- deskryptor to nieujemna dana całkowita
- startujący proces otrzymuje od systemu trzy wstępnie otwarte deskryptory i są to:
  `0` – przypisany do `stdin`
  `1` – przypisany do `stdout`
  `2` – przypisany do `stderr`

{lin,win} Otwarcie pliku – funkcja open()

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

// Windows
#include <io.h>

int open(const char *path, int oflag [, int mode ]);
```

- funkcja dokonuje skojarzenia pomiędzy plikiem o nazwie określonej w pierwszym argumencie, a nowo utworzonym deskryptorem pliku
- funkcja zwraca:
  - wynik≥0: nowy deskryptor otwartego pliku w przypadku powodzenia (o najmniejszej, dopuszczalnej dla danego procesu, wartości - w obrębie procesu deskryptory są unikalne)
  - wynik<0: błąd (dokładniejszą informację o przyczynie błędu zawiera errno)

{lin,win} Otwarcie pliku – funkcja open()

```
int open(const char *path, int oflag, ...);
```

- sposób otwarcia pliku oraz postępowanie z plikami nieistniejącymi określa drugi argument, składający się pewnej liczby stałych symbolicznych (masek bitowych) łączonych ze sobą operatorem |
- stałe te określają tzw. *status flags*
- pierwsze trzy z nich muszą być użyte wyłącznie tzn. nie wolno ich ze sobą łączyć w jednym wywołaniu
- trzeci argument ma sens tylko wtedy, gdy w wyniku wywołania funkcji zostanie utworzony nowy plik i jest to wartość praw dostępu tego pliku

{lin,win} Tryby otwarcia pliku

O_RDONLY
- plik otwierany jest w trybie tylko do odczytu
- próba pisania do pliku otwartego jako O_RDONLY kończy się błędem

O_WRONLY
- plik otwierany jest w trybie tylko do zapisu
- próba czytania z pliku otwartego jako O_WRONLY kończy się błędem

O_RDWR
- plik otwierany jest w trybie zapis/odczyt

UWAGA!
- jeśli użyto tylko jednej z w/w flag, plik musi istnieć, aby można go było pomyślnie otworzyć!
- po pomyślnym wykonaniu open() z jedną z powyższych flag wskaźnik pliku ustawiany jest na 0

{lin,win} Tryby otwarcia pliku - modyfikatory

O_APPEND

- użyta wraz z O_WRONLY lub O_RDWR powoduje, że przed każdą operacją zapisu wskaźnik pliku będzie ustawiany na końcu (za ostatnim bajtem pliku)

O_CREAT

- jeśli plik istnieje, zastosowanie tej flagi nie powoduje żadnych skutków, no chyba, że użyto jej wraz z O_EXCL
- jeśli użyto O_CREAT, obowiązkowe jest podanie trzeciego argumentu, określającego uprawnienia tworzonego pliku

O_EXCL

- jeśli plik istnieje i użyto O_CREAT, wywołanie open() zakończy się błędem

O_TRUNC

- jeśli użyto O_WRONLY albo O_RDWR i plik istnieje, jego dotychczasowa zawartość zostanie usunięta

{lin,win} Czytanie z pliku – funkcja read()

```
// Linux
#include <unistd.h>

// Windows
#include <io.h>

int read(int fd, void *buf, int nbyte);
```

- funkcja odczytuje z pliku o deskryptorze fd nbyte bajtów i umieszcza je w pamięci pod adresem buf
- wskaźnik pliku jest przesuwany do przodu o liczbę faktycznie odczytanych bajtów
- funkcja zwraca:
  - wynik≥0: liczba faktycznie odczytanych bajtów (jeśli jest mniejszy od nbyte, świadczy to o osiągnięciu końca pliku)
  - wynik<0: błąd (dokładniejszą informację o przyczynie błędu zawiera errno)

{lin,win} Pisanie do pliku – funkcja write()

```
// Linux
#include <unistd.h>

// Windows
#include <io.h>

int write(int fd, void *buf, int nbyte);
```

- funkcja zapisuje do pliku o deskryptorze fd nbyte bajtów spod adresu buf
- wskaźnik pliku jest przesuwany do przodu o liczbę faktycznie zapisanych bajtów
- funkcja zwraca:
  - wynik≥0: liczba faktycznie zapisanych bajtów (jeśli jest mniejszy od nbyte, świadczy to o błędzie, np. o wyczerpaniu miejsca w systemie plików)
  - wynik<0: błąd (dokładniejszą informację o przyczynie błędu zawiera errno)

{lin,win} Ustawienie wskaźnika pliku – funkcja lseek()

```
// Linux
#include <unistd.h>

// Windows
#include <io.h>

int lseek(int fd, int offset, int whence);
```

- funkcja ustawia wskaźnik pliku o deskryptorze fd na pozycję offset względem punktu odniesienia opisanego w whence:
  - SEEK_SET – ustawienie nowej pozycji względem początku pliku
  - SEEK_END – ustawienie nowej pozycji względem końca pliku
  - SEEK_CUR – ustawienie nowej pozycji względem obecnej pozycji

- funkcja zwraca:
  - wynik≥0: aktualna pozycja pliku
  - wynik<0: błąd (dokładniejszą informację o przyczynie błędu zawiera errno)
  - zauważ: wywołanie `lseek(fd, 0, SEEK_END)` dostarcza informację o aktualnym rozmiarze pliku

{lin,win} Zamknięcie pliku – funkcja close()

```
// Linux
#include <unistd.h>

// Windows
#include <io.h>

int close(int fd);
```

- funkcja odłącza deskryptor do pliku i zamyka plik; od tej chwili jakakolwiek operacja na zwolnionym deskryptorze wywoła błąd
- funkcja zwraca:
  - wynik=0: wykonanie poprawne
  - wynik<0: błąd (dokładniejszą informację o przyczynie błędu zawiera errno)
  - wbrew dość powszechnemu przekonaniu close() może zakończyć się błędem :)

{lin,win} Usunięcie pliku – funkcja unlink()

```
// Linux
#include <unistd.h>

// Windows
#include <io.h>

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

- funkcja usuwa plik o podanej nazwie
- funkcja zwraca:
  - wynik=0: wykonanie poprawne
  - wynik<0: błąd (dokładniejszą informację o przyczynie błędu zawiera errno)

{lin,win} Pobranie informacji o pliku – funkcje stat() i fstat()

```
// Linux
#include <unistd.h>

// Windows
#include <io.h>

int stat(const char *pathname, struct stat *statbuf);
int fstat(int fd, struct stat *statbuf);
```

- funkcje pobierają metadane pliku:
  - stat() - używając nazwy pliku
  - fstat() - używając deskryptora otwartego pliku
- funkcje zwracają:
  - wynik=0: wykonanie poprawne
  - wynik<0: błąd (dokładniejszą informację o przyczynie błędu zawiera errno)

{lin,win} Struktura stat:

```
struct stat {
    dev_t     st_dev;         /* ID of device containing file; drive number in
MSW (A:->0) */
    ino_t     st_ino;         /* Inode number; no meaning in MSW */
    mode_t    st_mode;        /* File type and mode; limited meaning in MSW */
    nlink_t   st_nlink;       /* Number of hard links; always 1 on non-NTFS in
MSW */
    uid_t     st_uid;         /* User ID of owner; always 0 in MSW */
    gid_t     st_gid;         /* Group ID of owner; always 0 in MSW */
    dev_t     st_rdev;        /* Device ID (if special file); same as st_dev in
MSW */
    off_t     st_size;        /* Total size, in bytes */
    blksize_t st_blksize;     /* Block size for filesystem I/O; absent in MSW */
    blkcnt_t  st_blocks;      /* Number of 512B blocks allocated; absent in MSW
*/
    struct timespec st_atim;  /* Time of last access; valid only for NTFS in MSW
*/
    struct timespec st_mtim;  /* Time of last modification; valid only for NTFS
in MSW */
    struct timespec st_ctim;  /* Time of last status change; valid only for NTFS
```

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

#define OUT(s) write(1, s, strlen(s))
#define ERR(s) write(2, s, strlen(s))
#define SIZE     8192

int main(int argc, char *argv[]) {
    if(argc != 3) {
        ERR("Usage: ");
        ERR(argv[0]);
        ERR(" <src_file> <dst_file>\n");
        return 1;
    }
    int fdi = open(argv[1], O_RDONLY);
    if(fdi < 0) {
        ERR("Cannot open source file\n");
        return 2;
    }
     :
     :
```

```
    :
     :
     struct stat stat;
    fstat(fdi, &stat);
    int fdo = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, stat.st_mode);
    if(fdo < 0) {
        ERR("Cannot create destination file\n");
        close(fdi);
        return 3;
    }
    char buff[SIZE];
    int  rdin;
    do {
        rdin = read(fdi, buff, SIZE);
        write(fdo, buff, rdin);
    } while(rdin > 0);
    close(fdi);
    close(fdo);
    OUT("Finished ok");
    return 0;
}
```