System plików, operacje na plikach

Unixowy system plików jest hierarchicznym uporządkowaniem katalogów i plików.

Plik – struktura danych zapisana na dysku i identyfikowana za pomocą nazwy.

Ogólny schemat operacji na plikach obejmuje:

- otwarcie pliku (przygotowujące do zapisywania lub odczytywania informacji, skojarzenia zmiennej plikowej z plikiem),
- wykonanie operacji zapisu lub odczytu danych,
- zamknięcie pliku (przerwanie skojarzenia pomiędzy zmienną plikową i plikiem).

Funkcje systemowe

Funkcje systemowe operujące na plikach w systemie UNIX oparte są na pojęciu deskryptora. Deskryptor to nieujemna zmienna typu int przypisana przez system w danym procesie do danego pliku, unikalna w obrębie tego procesu. Każdy proces dostaje 3 domyślne deskryptory: (stałe zdefiniowane w `<unistd.h>`)

- standardowe wejście `STDIN_FILENO` (0)
- standardowe wyjście `STDOUT_FILENO` (1)
- standardowe wyjście diagnostyczne `STDERR_FILENO` (2)

Korzystanie z funkcji systemowych do obsługi plików wymaga dołączenia bibliotek:
`<fcntl.h> <unistd.h> <sys/types.h> <sys/stat.h>`

Otwieranie i tworzenie plików

Funkcje systemowe odpowiadające za otwieranie i tworzenie plików to **open** oraz **creat**:

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

Lista możliwych flag dla funkcji **open**:

O_RDONLY	Otwiera plik do odczytu
O_WRONLY	Otwiera plik do zapisu
O_RDWR	Otwiera plik do zapisu i odczytu.

powyższe flagi można łączyć bitowym OR z poniższymi:

O_CREAT	Utworzenie pliku, jeżeli nie istnieje.
O_TRUNC	Obcięcie pliku, jeśli plik istnieje i otwierany jest w trybie O_WRONLY lub O_RDWR
O_EXCL	Powoduje zgłoszenie błędu jeśli plik już istnieje i otwierany jest z flagą O_CREAT
O_APPEND	Operacje pisania odbywają się na końcu pliku

Odczyt i zapis pliku

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

- próbuje wczytać podana liczbę bajtów (`count`) z pliku o podanym deskryptorze (`fd`) do podanego bufora (`buf`); bieżąca pozycja w pliku przesuwa się o tyle, ile bajtów przeczytano,
- funkcja `read` zwraca ilość bajtów naprawdę przeczytanych (zawracana wartość może być mniejsza od nbytes !)
- gdy "bieżąca pozycja" przekroczy koniec pliku, to read zwraca 0

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

- zapis zawartości bufora do pliku, argumenty analogiczne do `read`.

Ustawianie pozycji w pliku

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

Argumenty:

`fd` – deskryptor do pliku na którym operujemy
`offset` – nowa pozycja w pliku
`whence` – parametr służący interpretacji drugiego parametru. Musi być to liczba równa 0, 1 lub 2
Parametr `whence` funkcji `lseek` przyjmuje jedną z wartości:

- SEEK_SET – początek pliku
- SEEK_END – koniec pliku
- SEEK_CUR – aktualna pozycja wskaźnika

Na podstawie tej wartości wylicza nową pozycję wskaźnika po przesunięciu o *offset*

Wyniki: W przypadku powodzenia funkcja zwraca nowa pozycje w pliku, w przeciwnym wypadku wartość mniejsza od zera.

Zamykanie pliku

```
int close(int fd);
```

Przykład 1

Program kopiujący znak po znaku z wykorzystaniem funkcji niskopoziomowych

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

int main()
{
  char c;
  int we,wy;
  we=open("we", O_RDONLY);
  wy=open("wy",O_WRONLY|O_CREAT,S_IRUSR|S_IWUSR);
  while(read(we,&c,1)==1)
        write(wy,&c,1);

}
```

Przykład 2

Program kopiujący blokami o rozmiarze 1024B z wykorzystaniem funkcji niskopoziomowych

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

int main()
{
  char blok[1024];
  int we, wy;
  int liczyt;
  we=open("we", O_RDONLY);
  wy=open("wy",O_WRONLY|O_CREAT,S_IRUSR|S_IWUSR);
  while((liczyt=read(we,blok,sizeof(blok)))>0)
      write(wy,blok,liczyt);

}
```

Funkcje biblioteki standardowej wejścia-wyjścia

Funkcje z biblioteki standardowej C mapują funkcje systemowe. Zaletą tego rozwiązania jest przenośność kodu, na innym systemie funkcje z C zaimplementowane będą jako inne funkcje systemowe jednak na poziomie języka działanie będzie jednakowe w każdym wypadku.

Otwarcie pliku

Aby otworzyć plik używamy funkcji **fopen**:

```
FILE * fopen ( const char * filename, const char * mode );
```

Atrybuty z jakimi można otworzyć plik:

r	Otwiera plik do odczytu
w	Otwiera plik do zapisu (kasuje ewentualny poprzedni)
a	Otwiera plik do zapisu. Nie kasuje poprzedniego pliku i ustawia wskaźnik na końcu.
r+	Otwiera plik do zapisu i odczytu. Plik musi istnieć.
w+	Otwiera plik do zapisu i odczytu. Jeśli plik istniał to nadpisuje.
a+	Otwiera plik do odczytu i dopisywania. Nie można pisać wcześniej niż na końcu.
[rwa+]b	Otwiera plik jako binarny nie tekstowy.

Zapis i odczyt pliku

`size_t fread (void * ptr, size_t size, size_t count, FILE * file)`

Argumenty:

- `ptr` – wskaźnik na tablicę
- `size` – rozmiar elementu tablicy
- `count` – liczba elementów do odczytu
- `file` – plik, na którym wykonywana jest operacja

Funkcja `fread` kopiuje `count` elementów z podanego pliku do tablicy. Kopiowanie kończy się w przypadku wystąpienia błędu, końca pliku lub po skopiowaniu podanej liczby elementów. Wskaźnik pliku jest przesuwany, tak by wskazywał pierwszy nieodczytany element.

Wartość zwracana: liczba faktycznie wczytanych elementów.

`size_t fwrite (const void * ptr, size_t size, size_t count, FILE * file);`

Funkcja `fwrite` kopiuje `count` elementów z poddanej tablicy do pliku. Kopiowanie kończy się w przypadku wystąpienia błędu lub po skopiowaniu podanej liczby elementów. Wskaźnik pliku jest przesuwany, tak by wskazywał pierwszy element po ostatnim zapisanym.

Wartość zwracana: Liczba faktycznie zapisanych elementów.

Ustawianie pozycji w pliku

```
int fseek ( FILE * file, long int offset, int mode);
```

Funkcja `fseek` ustawia pozycję w pliku file na offset w zależności od wartości argumentu mode.

- `mode` = 0 – offset liczony jest od początku.
- `mode` = 1 – offset przesuwany od aktualnej pozycji,
- `mode`=2 – przesuwany o offset od końca pliku (wskaźnik pliku jest przesuwany do pozycji będącej sumą rozmiaru pliku i parametru `offset`).

Zwraca: Zero gdy funkcja wykonała się pomyślnie, w przypadku błędu wartość niezerowa.

```
int fsetpos (FILE* file, fpos_t* pos);
```

Funkcja zmienia aktualną pozycję wskaźnika do pliku `file` na `pos`.
Zwraca: Zero gdy funkcja wykonała się pomyślnie, EOF w przypadku wystąpienia błędu

```
int fgetpos (FILE* file, fpos_t* pos);
```

Funkcja umieszcza w `pos` aktualną pozycję wskaźnika do pliku `file`.
Zwraca: Zero gdy funkcja wykonała się pomyślnie, EOF w przypadku wystąpienia błędu

Zamykanie pliku

```
int fclose ( FILE * stream );
```

Przykład 3

```
#include <stdio.h>

int main ()
{
  char napis[20];
  FILE *plik=fopen("nazwa1.txt", "a+");
  if(plik)
  {
    fread(napis,1, 15,plik);
    printf("%s",napis);
    printf("\n");
    fwrite("Zdanie drugie.", 1, 14, plik);
    rename("nazwa1.txt","nazwa2.txt");
    fclose(plik);
  }
  return 0;
}
```

Katalogi

Katalogi są w systemach Unix traktowane prawie tak samo jak pliki. Jedyna, ale bardzo ważna różnica to hierarchia, jaką tworzą katalogi. Katalogi porządkują system plików tworząc drzewo katalogów. Katalog jest kontenerem dla plików. Katalogami są również `.` i `..` – są to łącza do bieżącego katalogu i jego przodka. Katalog / jest korzeniem.

Operacje na katalogach

W bibliotece `dirent.h` istnieją następujące definicje:

- `DIR` – struktura reprezentująca strumień katalogowy
- `struct dirent` – struktura, która zawiera:
  - `ino_t d_ino` – numer i–węzła pliku
  - `char d_name[]` – nazwa pliku

`DIR* opendir(const char* dirname)`

Otwiera strumień do katalogu znajdującego się pod ścieżką `dirname`. Po prawidłowym wykonaniu, `opendir` zwraca wskaźnik do obiektu typu DIR, inaczej zwraca NULL.

`int closedir(DIR* dirp)`

Zamyka strumień katalogowy `dirp`. Po prawidłowym wykonaniu, funkcja zwraca wartość 0, inaczej zwraca –1 i zapisuje kod błędu w zmiennej `errno`.

`struct dirent* readdir(DIR* dirp)`

Zwraca wskaźnik do struktury reprezentującej plik w obecnej pozycji w strumieniu `dirp` i awansuje pozycję na następny plik w kolejce. Zwrócony wskaźnik do obiektu `struct dirrent` nie powinien być zwolniony. Jeśli nie ma już więcej plików w katalogu, wartość `NULL` jest zwrócona. Gdy wystąpi błąd, wartość `NULL` także jest zwrócona i powód jest zapisany w zmiennej `errno`

`void rewinddir(DIR* dirp)`

Ustawia strumień katalogowy na początek.

`void seekdir(DIR* dirp, long int loc)`

Zmienia pozycję strumienia katalogowego.

```
int stat (const char *path, struct stat *buf);
```

Pobranie statusu pliku

Argumenty: `path` – nazwa sprawdzanego pliku
`buf` – bufor na strukturę `stat`

Funkcja zwraca: `err` – Po sukcesie zwracane jest zero. Po błędzie –1 i ustawiane jest 'errno':

- `EBADF` – `'filedes'` jest nieprawidłowy.
- `ENOENT` – Plik nie istnieje.

```
int lstat(const char *ścieżka, struct stat *statystyka);
```

Identyczna jak funkcja `stat`(), lecz nie zwraca on statusu plików, wskazywanych przez linki, a status samego linku.

Struktura stat

```
struct stat
    {
        dev_t           st_dev;      /* urządzenie */
        ino_t           st_ino;      /* inode */
        umode_t         st_mode;     /* ochrona */
        nlink_t         st_nlink;    /* liczba hardlinków */
        uid_t           st_uid;      /* ID użytkownika właściciela */
        gid_t           st_gid;      /* ID grupy właściciela */
        dev_t           st_rdev;     /* typ urządzenia (jeśli urządzenie inode)
*/
        off_t           st_size;     /* całkowity rozmiar w bajtach */
        unsigned long   st_blksize;  /* wielkość bloku dla I/O systemu plików */
        unsigned long   st_blocks;   /* ilość zaalokowanych bloków */
        time_t          st_atime;    /* czas ostatniego dostępu */
        time_t          st_mtime;    /* czas ostatniej modyfikacji */
        time_t          st_ctime;    /* czas ostatniej zmiany */
    };
```

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

Tworzenie katalogu z uprawnieniami podanymi w `mode`

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

Usuwanie katalogu

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

Zmiana katalogu bieżącego na `path`.

```
char *getcwd (char *folder_name, ssize_t size);
```

Funkcja wpisuje do `folder_name` bieżący katalog roboczy o rozmiarze `size`.

```
int chmod (const char *path, mode_t new_mode);
```

Zmiana uprawnień do pliku.

```
int chown (const char *path, uid_t id_owner, gid_t id_group);
```

Zmiana właściciela.

```
int link (const char *path, const char *new_path);
```

Stworzenie nowego dowiązania do pliku. Usunięcie łącza – funkcja `ulink`.

```
int nftw(const char *dir, int(*fn) (const char *,
        const struct stat *, int, struct FTW *), int nopend, int flags)
```

Przeglądanie drzewa katalogów

wejście: `dir` – katalog główny drzewa do przeglądnięcia
`fn` – funkcja wywoływana dla każdego przeglądanego elementu w drzewie
`nopenfd` – maksymalna ilość otwieranych przez funkcję deskryptorów
`flags` – znaczniki definiujące zachowanie funkcji

Funkcja zwraca: 0 w przypadku powodzenia,
–1 w przypadku błędu.

Funkcja `ftw`() przechodzi przez drzewo katalogów startując z określonego katalogu `dir`. Dla każdej znalezionej pozycji w drzewie, wywołuje funkcję `fn` z pełną nazwą ścieżki do pozycji, wskaźnik na strukturę otrzymaną z funkcji `stat`(2) dla tej pozycji oraz flagę `flag` której wartość jest jedną z poniższych wartości:

- `FTW_F` – pozycja jest normalnym plikiem
- `FTW_D` – pozycja jest katalogiem
- `FTW_DNR` – pozycja jest katalogiem który nie może być czytany
- `FTW_SL` – pozycja jest linkiem symbolicznym
- `FTW_NS` – operacja `stat` nie powiodła się na pozycji która nie jest linkiem symbolicznym

Ryglowanie rekordów

Stevens rozdział 12.3 str 435-453