4. Pliki

4.1 Pliki i katalogi

Plik jest podstawową abstrakcją używaną w systemach operacyjnych. Pozwala na traktowanie dużego zbioru zasobów w jednolity sposób.

W systemie Linux prawie wszystkie zasoby są plikami. Dane i urządzenia są reprezentowane przez abstrakcję plików. Mechanizm plików pozwala na jednolity dostęp do zasobów tak lokalnych jak i zdalnych za pomocą poleceń i programów usługowych wydawanych z okienka terminala. Plik jest obiektem abstrakcyjnym z którego można czytać i do którego można pisać. Oprócz zwykłych plików i katalogów w systemie plików widoczne są pliki specjalne. Zaliczamy do nich łącza symboliczne, kolejki FIFO, bloki pamięci, urządzenia blokowe i znakowe.

Atrybuty pliku

Oprócz zawartości plik posiada szereg atrybutów pokazanych w poniższej tabeli.

Typ pliku	Plik regularny, katalog, gniazdko, kolejka FIFO, urządzenie blokowe, urządzenie znakowe, link
Prawa dostępu	Prawo odczytu, zapisu, wykonania określone dla właściciela pliku, grupy do której on należy i innych użytkowników systemu
Wielkość	Wielkość pliku w bajtach
Właściciel pliku	UID właściciela pliku
Grupa do której należy właściciel	GID grupy do której należy właściciel pliku
Czas ostatniej modyfikacji	Czas kiedy nastąpił zapis do pliku
Czas ostatniego dostępu	Czas kiedy nastąpił odczyt lub zapis do pliku
Czas ostatniej modyfikacji statusu	Kiedy zmieniano atrybuty takie jak prawa dostępu, właściciel, itp
Liczba dowiązań	Pod iloma nazwami (ang. *links*) występuję dany plik
Identyfikator urządzenia	Identyfikator urządzenia na którym plik jest pamiętany

Tab. 4-1 Atrybuty pliku

Informacje o tym jaki jest numer użytkownika i jaki jest numer grupy uzyskiwana jest z plików **/etc/passwd** i **/etc/group.**
Plik **passwd** składa się z linii. Każda z linii odpowiada jednemu użytkownikowi i składa się z 7 pól oddzielonych dwukropkiem ":".

Nazwa	Nazwa użytkownika
Hasło	Znak x gdy wymagane podanie hasła
UID	Liczbowy identyfikator użytkownika. 0 dla administratora, 1-999 zarezerwowane,
GID	Numer grupy do której należy użytkownik
Informacja	Dodatkowe informacje o użytkowniku
Katalog	Ścieżka określająca katalog domowy – katalog w który będzie bieżący po zalogowaniu
Shell	Ścieżka do interpretera poleceń, zwykle /bin/bash

Tab. 4-2 Zawartość linii pliku **passwd**

Przykład linii pliku passwd:
juka:x:1000:1000:Jedrzej Ulasiewicz :/home/juka:/bin/bash

System umożliwia dostęp do plików w trybie odczytu, zapisu lub wykonania. Symboliczne oznaczenia praw dostępu do pliku dane są poniżej:

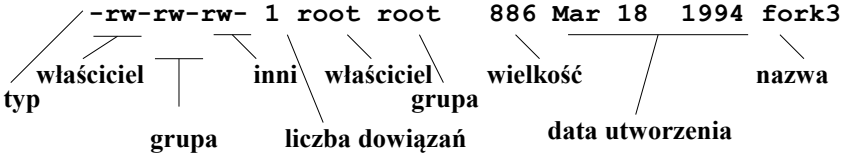
r - Prawo odczytu (*ang. read*)
w - Prawo zapisu (*ang. write*)
x - Prawo wykonania (*ang. execute*)

Prawa te mogą być zdefiniowane dla właściciela pliku, grupy do której on należy i wszystkich innych użytkowników.

u - Właściciela pliku (*ang. user*)
g - Grupy (*ang. group*)
o - Innych użytkowników (*ang. other*)

Atrybuty pliku można uzyskać za pomocą polecenia systemowego ls.

```
$ls -l
-rwxrwxr-x 1 root root  7322 Nov 14  2003 fork3
-rw-rw-rw- 1 root root   886 Mar 18  1994 fork3.c
```



Przykład 4-1 Listowanie zawartości katalogu bieżącego.

4.2 Niskopoziomowe funkcje dostępu do plików

Niskopoziomowe funkcje dostępu do plików zapewniają dostęp do:

- plików regularnych
- katalogów
- łącz nazwanych
- łącz nie nazwanych
- gniazdek
- urządzeń znakowych (porty szeregowe, równoległe).
- Urządzeń blokowych (dyski)

Funkcje opisane w:

- https://www.gnu.org/software/libc/manual/html_node/index.html
- http:/www.kernel.org/docs/man-pages

Nr	Funkcja	Opis
1	`open`	**Otwarcie lub utworzenie pliku**
2	`creat`	**Tworzy pusty plik**
3	`read`	**Odczyt z pliku**
4	`write`	**Zapis do pliku**
5	`lseek`	**Pozycjonowanie bieżącej pozycji pliku**
6	`fcntl`	**Ustawianie i testowanie różnorodnych atrybutów pliku**
7	`fstat`	**Testowanie statusu pliku**
8	`close`	**Zamknięcie pliku**
9	`unlink, remove`	**Usuwa plik**

Tab. 4-3 Ważniejsze niskopoziomowe funkcje dostępu do plików

4.2.1 Otwarcie lub tworzenie pliku – funkcja open

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

path Nazwa pliku lub urządzenia
oflag Tryb dostępu do pliku – składa się z bitów – opis w pliku nagłówkowym <fcntl.h>
mode Atrybuty tworzonego pliku (prawa dostepu)

Funkcja powoduje otwarcie pliku lub urządzenia o nazwie wyspecyfikowanej w parametrze **path**. Otwarcie następuje zgodnie z trybem oflag. Funkcja zwraca deskryptor pliku (uchwyt) będący niewielką liczbą int. Uchwyt pliku służy do identyfikacji pliku w innych funkcjach systemowych.

Funkcja zwraca:
> 0 – uchwyt do pliku (*ang. File handle*) – mała liczba typu int.
- 1 – gdy wystąpił błąd.

O_RDONLY	**Tylko odczyt**
O_WRONLY	**Tylko zapis**
O_RDWR	**Odczyt i zapis**
O_CREAT	**Utwórz plik gdy nie istnieje**
O_NONBLOCK	**Gdy flaga ustawiona odczyt i zapis mogą być operacjami blokującymi**
O_EXCL	**Gdy flaga użyta w połączeniu z O_CREAT to jeśli plik istnieje to funkcja open się nie powiedzie. Mechanizm nie działa w NFS. Mechanizm może być użyty do implementacji blokad**
O_TRUNC	**Jeżeli plik istnieje, jest zwykłym plikiem i tryb otwarcia pozwala na zapis to zostanie on obcięty.**
O_APPEND	**Plik otwarty w trybie dopisywania**
O_SYNC	**Zapisy będą blokowały proces aż do zakończenia fizycznego wpisu.**
O_ASYNC	**Generowanie sygnału (domyslnie SIGIO) gdy odczyt/zapis przez ten deskryptor staje się możliwy. Dostępna dla terminali, pseudoterminali i gniazd.**

Tab. 4-4 Ważniejsze flagi funkcji open

Funkcja **open** może być także użyta do tworzenia nowego pliku. W takim przypadku należy określić prawa dostępu do nowo tworzonego pliku. Prawa dostępu mogą być określone jako:

- Trzeci argument **mode** funkcji **open** w postaci ósemkowej
- Flagi w argumencie **oflag** zgodnie z Tab. 4-5

Wartośc ósemkowa	Nazwa symboliczna	Pozwolenie na
0400	**S_IRUSR**	Odczyt przez właściciela
0200	**S_IWUSR**	Zapis przez właściciela
0100	**S_IXUSR**	Wykonanie przez właściciela
0040	**S_IRGRP**	Odczyt przez grupę
0020	**S_IWGRP**	Zapis przez grupę
0010	**S_IXGRP**	Wykonanie przez grupę
0004	**S_IROTH**	Odczyt przez innych użytkowników
0002	**S_IWOTH**	Zapis przez innych użytkowników
0001	**S_IXOTH**	Wykonanie przez innych użytkowników

Tab. 4-5 Specyfikacja bitów określających prawa dostępu do pliku

```
fd = open(”plik.txt”,O_RDWR|O_CREAT | S_IRUSR |
                        S_IWUSR);
fd = open(”plik.txt”,O_RDWR|O_CREAT,0600);
```

Wartośc ósemkowa	Nazwa symboliczna	Znaczenie
04000	**S_ISUID**	Ustaw EUID na UID właściciela pliku przy wykonaniu
02000	**S_ISGID**	Ustaw EGID na GID właściciela pliku przy wykonaniu
01000	**S_ISVTX**	Bit klejący (sticky bit) Dawniej - po zakończeniu procesu utrzymać segment kodu w przestrzeni wymiany. Obecnie – stosowany do katalogu /tmp, Gdy zastosowany do katalogu tylko właściciel może zmieniać i usuwać zawarte tam pliki

Tab. 4-6 Specyfikacja bitów określających dodatkowe atrybuty pliku

Przykład ustawienia bitu S_ISUID:

```
chmod u+s prog
-rws--------- 1 juka users  256 2014-11-17  prog
```

4.2.2 Utworzenie nowego pliku – funkcja creat

```
int creat(char *path,mode_t mode)
```

path Nazwa pliku lub urządzenia
mode Atrybuty tworzonego pliku (prawa dostępu)

Funkcja powoduje utworzenie pustego pliku o nazwie wyspecyfikowanej w parametrze **path**. Nowy plik będzie miał prawa dostępu zgodnie z parametrem **mode**.

```
fd = creat("plik.txt",0666);
```

4.2.3 Maska tworzenia pliku – funkcja umask

Uprawnienia tworzonego pliku określane są przez parametry funkcji **open** lub **creat**. Jednak przy ostatecznym określaniu praw dostępu uwzględniana jest wartość maski tworzenia pliku (ang. *file creation mask*).

Gdy w masce tworzenia pliku ustawiony jest jakiś bit to przy tworzeniu pliku odpowiadający mu bit będzie wyzerowany.

Jest to zabezpieczenie przed przypadkowym włączeniem uprawnień.

Gdy plik tworzony jest z prawami dostępu **mode** a maska wynosi **umask** to działanie funkcji

```
fd = open("plik.txt",O_RDWR|O_CREAT,mode);
```

równoważne będzie funkcji

```
fd = open("plik.txt",O_RDWR|O_CREAT,(~umask)&mode);
```

Testowanie maski: polecenie **umask**

Typowa wartość umask wynosi 0022 czyli **S_IWGRP | S_IWOTH** a więc zabezpiecza przed nadaniem prawa zapisu dla grupy i innych.

Przykład: 0666 & ~022 = 0644 czyli rw-r--r--

mode	0666	**S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH**
umask	0022	**S_IWGRP | S_IWOTH**
wynik	0644	**S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH**

Do zmiany maski tworzenia pliku służy funkcja **umask**

```
int umask(mode_t maska)
```

maska Nowa maska tworzenia pliku

Funkcja zwraca poprzednią maskę tworzenia pliku.

4.2.4 Odczyt z pliku – funkcja read

```
int read(int fdes, void *bufor, int nbytes)
```

fdes Uchwyt do pliku zwracany przez funkcję open
bufor Bufor w którym umieszczane są przeczytane bajty
nbytes Liczba bajtów którą chcemy przeczytać.

Funkcja powoduje odczyt z pliku identyfikowanego przez fdes, nbytes bajtów i umieszczenie ich w buforze.
Funkcja zwraca:
> 0 – liczbę rzeczywiście przeczytanych bajtów,
- 1 – gdy błąd.

Odczyt i zapis do pliku odbywa się od bieżącej pozycji pliku. Parametr ten pamiętany jest w stowarzyszonej z procesem tablicy otwartych plików. Po odczycie / zapisie n bajtów jest on automatycznie przesuwany o tę wartość. Może być wprost zmieniony funkcją lseek.

```
int pread(int fdes, void *bufor,int size, off_t
offset)
```

Funkcja działa tak jak **read** z tą różnicą że czytanie nie jest od bieżącej pozycji pliku ale od pozycji wskazywanej przez parametr offset.

4.2.5 Zapis do pliku – funkcja write

```
int write(int fdes, void *bufor, int nbytes)
```

fdes Uchwyt do pliku zwracany przez funkcję open
bufor Bufor w którym umieszczane są bajty przeznaczone do zapisu
nbytes Liczba bajtów którą chcemy zapisać

Funkcja powoduje zapis do pliku identyfikowanego przez fdes nbytes bajtów znajdujących buforze.

Funkcja zwraca:
> 0 – liczbę rzeczywiście zapisanych bajtów,
- 1 – gdy błąd.

int pwrite(int fdes, void *bufor,int size, off_t offset)

Funkcja działa tak jak **write** z tą różnicą że zapis nie jest od bieżącej pozycji pliku ale od pozycji wskazywanej przez parametr **offset**.

4.2.6 Zamknięcie pliku – funkcja close

int close(int fdes)

fdes Uchwyt do pliku zwracany przez funkcję open

Funkcja powoduje zamknięcie pliku identyfikowanego przez fdes. Należy ją wykonać gdy nie będą już wykonywane operacje na danym pliku .

```
int main(void) {
  int fd;
  char buf[80];
  fd = open(”moj_plik.txt”,O_RDONLY);
  if(fd < 0) {
       perror(“open”);
       return 0;
  }
  do {
     rd = read(fd,buf,80);
     if(rd < 0) {
        perror(“read”);
        return 0;
     }
     printf(“Odczytano %d bajtów\n”,rd);
  } while(rd > 0);
  close(fd);
  return 0;
}
```

Przykład 4-2 Przykład odczytu pliku

4.2.7 Ustawianie bieżącej pozycji pliku – funkcja lseek

```
off_t lseek(int fdes, off_t offset, int start_flag)
```

fdes	Uchwyt do pliku zwracany przez funkcję open
offset	Nowa pozycja wskaźnika
start_flag	Skąd ma być nowa pozycja liczona

Funkcja powoduje przesunięcie wskaźnika bieżącej pozycji pliku identyfikowanego przez **fdes**, na nową pozycję określaną przez offset. Sposób określania tej pozycji zależy od flagi **start_flag**.

SEEK_SET	Od początku pliku
SEEK_CUR	Od biezącej pozycji wskaźnika pliku
SEEK_END	Od końca pliku

Tab. 4-7

Funkcja zwraca:
- 0 – sukces,
- 1 – gdy błąd.

4.2.8 Testowanie statusu pliku – funkcja stat

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

fd	Uchwyt do pliku zwracany przez funkcję open
buf	Struktura w której umieszczane są informacje o pliku
path	Scieżka do pliku

Funkcja powoduje umieszczenie w buforze **buf** informacji o pliku.

stat	plik identyfikowany przez ścieżkę
fstat	plik identyfikowany przez uchwyt
lstat	plik identyfikowany przez ścieżkę (gdy ścieżka jest linkiem to zostaną podane dane linku)

Funkcja zwraca:
- 0 – ok
- 1 – gdy błąd.

```
struct stat {
 dev_t     st_dev;     /* ID of device containing file */
 ino_t     st_ino;     /* inode number */
 mode_t    st_mode;    /* protection */
 nlink_t   st_nlink;   /* number of hard links */
 uid_t     st_uid;     /* user ID of owner */
 gid_t     st_gid;     /* group ID of owner */
 dev_t     st_rdev;    /* device ID (if special file) */
 off_t     st_size;    /* total size, in bytes */
 blksize_t st_blksize; /* blocksize for file system I/O */
 blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
 time_t    st_atime;   /* time of last access */
 time_t    st_mtime;   /* time of last modification */
 time_t    st_ctime;   /* time of last status change */
};
```

Tab. 4-8 Pola struktury **stat**

Atrybuty pliku mogą być też odczytane poleceniem stat

```
$stat plik.txt
  Plik: „plik.txt”
  rozmiar: 132098       bloków: 264        bloki I/O: 4096   zwykły plik
Urządzenie: fd00h/64768d     inody: 202872461   dowiązań: 1
Dostęp: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Kontekst: unconfined_u:object_r:admin_home_t:s0
Dostęp:      2016-05-12 09:29:38.655878512 +0200
Modyfikacja: 2016-05-12 09:29:31.295951064 +0200
Zmiana:      2016-05-12 09:29:31.295951064 +0200
Utworzenie:  -
```

Przykład 4-3 Testowanie statusu pliku – polecenie stat

```
int main(int argc, char *argv[]) {
    int fd,res;
    struct stat buf;
    fd = open(argv[1],O_RDONLY);
    res = fstat(fd,&buf);
    printf("size %d\n",buf.st_size);
    return 0;
}
```

Przykład 4-4 Testowanie statusu pliku – funkcja stat

4.2.9 Ustawianie parametrów pliku – funkcja fcntl

```
int fcntl(int fdes, int cmd, ...)
```

fdes Uchwyt do pliku zwracany przez funkcję open
cmd Polecenie

Funkcja powoduje wykonanie na pliku fdes operacji określonej parametrem cmd. Dalsze parametry funkcji zależą od tej operacji które określone są w pliku nagłówkowym <fcntl.h>.
Operacje te to zmiana atrybutów i sposobów blokowania pliku.

Funkcja zwraca:
0 – sukces,
- 1 – gdy błąd.

F_DUPFD	Duplikuje deskryptor pliku, nowy numer deskryptora nie może być mniejszy niż wartość argumentu. Zwraca numer deskryptora lub błąd.
F_GETFD	Testuje czy dla pliku flaga close_on_exec jest ustawiona
F_SETFD	Ustawia lub kasuje flage close_on_exec w zależności od argumentu. Gdy flaga ustawiona, proces potomny nie dziedziczy otwartych plików.
F_GETFL	Zwraca flagi pliku
F_SETFL	Ustawia flagi pliku na wartość argumentu **arg**
F_SETLK, F_GETLK, F_SETLKW	Operuje na blokadach typu *POSIX*.
F_GETOWN	Zwraca identyfikator właściciela gniazda.
F_SETOWN	Ustawia identyfikator właściciela gniazda

4.3 Standardowe wejście wyjście

Dla każdego wykonywanego programu system automatycznie otwiera trzy pliki:

- Standardowe wejście
- Standardowe wyjście
- Wyjście komunikatu o błędach

Opis	Wartość uchwytu	Przypisanie początkowe
Standardowe wejście	0	Klawiatura
Standardowe wyjście	1	Ekran
Wyjście komunikatów o błędach	2	Ekran

Tab. 4-9 Standardowe wejście / wyjście programu

Standardowe wejście wyjście posiada przyporządkowanie początkowe do urządzeń pokazane w Tab. 4-9. Może ono jednak zostać zmienione.

$ ./prog < plik_we

$ ./prog > plik_wy

$ ./prog > plik_wy > plik_err

$ ./prog < plik_we > plik_wy

$ ./prog1 | prog2

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 16
int main(int argc, char *argv[])
{
    int fd,rd;
    char buf[SIZE];
    do {
        rd = read(0,buf,SIZE);
        write(1,buf,SIZE);
    } while(rd > 0);
    return 0;
}
```

Program 4-1 Program czytający standardowe wejście i przesyłający wyniki na standardowe wyjście.

4.4 Duplikacja uchwytu pliku – funkcje dup i dup2

```
int dup(int oldfd)
int dup2(int oldfd, int newfd)
```

oldfd Stary uchwyt do pliku
newfd Nowy uchwyt do pliku

Funkcja **dup** tworzy kopię uchwytu ***oldfd***. Nowy uchwyt, będący wolnym uchwytem o najmniejszym numerze (pierwszy wolny) zwracany jest przez funkcję **dup**.

Funkcja **dup2** tworzy nowy uchwyt ***newfd*** będący kopią ***oldfd***. Obydwa uchwyty mogą być używane wymiennie. Gdy ***newfd*** jest już wcześniej używany zostanie on zamknięty.

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

int main() {
    int file,res;
    printf("To jest na stdout \n");
    file = open("myfile.txt", O_APPEND | O_WRONLY | O_CREAT);
    if(file < 0) { perror("open"); return 1; }
    res = dup2(file,1);
    if(res < 0) { perror("dup2"); return 1; }
    printf("To bylo skierowane na stdout (konsole) !\n");
    return 0;
}
```

Przykład 4-5 Przekierowanie stdout do pliku myfile.txt

Po wykonaniu programu w pliku myfile.txt znajdzie się napis:
To bylo skierowane na stdout (konsole) !

```
$dup2examp
To jest na stdout
$cat myfile.txt
To bylo skierowane na stdout (konsole) !
$
```

4.5 Blokady pliku

W sytuacji gdy więcej procesów korzysta współbieżnie z pliku a przynajmniej jeden z nich pisze może dojść do błędnego odczytu jego zawartości. Podobnie dwa (lub więcej) procesy nie powinny jednocześnie pisać do pliku. Aby zapewnić wzajemne wykluczanie w dostępie do pliku stosuje się blokady plików.

Przykład:
Rezerwacja miejsc w samolocie

Są dwa typy blokad:

- Blokady doradcze (ang. *advisory*)
- Blokady obowiązkowe (ang. *mandatory locks*)

Blokady doradcze działają wtedy gdy procesy używające wspólnego pliku stosują prawidłowo mechanizmy ochrony. Gdy jeden z procesów takie mechanizmy stosuje a inny nie to w tym drugim operacje odczytu i zapisu nie spowodują blokady procesu.

Blokady obowiązkowe działają nawet wtedy gdy nie wszystkie procesy używające wspólnego pliku stosują mechanizmy ochrony. W tym przypadku blokada założona na plik (F_RDLCK, F_WRLCK) przez jeden proces będzie działała nawet gdy inny proces nie będzie jej świadomy. W takim przypadku operacja **`read`** / **`write`** dokonana na zablokowanym pliku spowoduje zawieszenie procesu do czasu zdjęcia blokady z pliku. Aby uzyskać blokady obowiązkowe należy ustawić bit group_id (**`ATTR_GID`**) flagi pliku oraz wyzerować bit group_execute z praw dostępu pliku. Od tej chwili blokada *POSIX* założona na ten plik będzie obowiązująca.

Blokady POSIX

```
int fcntl(int fd, int polecenie, struct flock* fl)
```

fd Uchwyt pliku

polecenie Polecenie:
- F_GETLK – pobiera informację o blokadzie
- F_SETLKW – ustawia / zdejmuje blokadę
- F_SETLK – próbuje ustawić / zdjąć blokadę, nie blokuje procesu

fl Struktura specyfikująca szczegóły blokady

```
struct flock {
        ...
  short l_type;  /* Typ blokady: F_RDLCK,    F_WRLCK, F_UNLCK */
  short l_whence;  /* Skąd ustawić:  SEEK_SET, SEEK_CUR,  SEEK_END */
  off_t l_start;   /* Przesunięcie dla startu blokady */
  off_t l_len;     /* Ile bajtów zablokować */
  pid_t l_pid;     /* PID procesu blokującego    (tylko  F_GETLK ) */
        ...
};
```

l_type	Rodzaj blokady • F_RDLCK – blokada do odczytu, wiele procesów może mieć tę blokadę ale próba założenia blokady do zapisu zablokuje proces zakładający te blokadę. Gdy założona blokada do zapisu blokada do odczytu nie uda się. • F_WRLCK – blokada do zapisu, tylko jeden proces może ją posiadać • F_UNLCK – Zdjęcie blokady
l_whence	Skąd ma być liczony początek blokady SEK_SET – początek pliku SET_CUR – bieżąca pozycja pliku SET_END – koniec pliku
l_start	Przesunięcie początku blokady względem **l_whence w bajtach**
l_len	Zakres blokady
l_pid	PID procesu

Polecenia:

- F_GETLK – pobiera informację o blokadzie pliku fd który jest parametrem funkcji. Należy ustawić parametry struktury flock na parametry blokady którą się chce uzyskać. Gdy blokada mogłaby być przyznana struktura flock nie zostanie zmieniona. Funkcja zwraca –1 gdy brak możliwości uzyskania informacji o blokadzie.
- F_SETLKW – ustawia / zdejmuje blokadę. Gdy blokady nie udaje się uzyskać proces jest blokowany do czasu uzyskania blokady.
- F_SETLK – próbuje ustawić / zdjąć blokadę, nie blokuje procesu ale zwraca wartość różną od -1 gdy blokadę założono bądź –1 gdy nie udało się.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[]){
/* l_type   l_whence  l_start  l_len  l_pid   */
   struct flock fl = {F_WRLCK, SEEK_SET, 0, 0, 0 };
    int fd;
    fl.l_pid = getpid();
    if ((fd = open("blokada.c", O_RDWR)) == -1) {
        perror("open");
        exit(0);
    }
    printf("Proba zajecia blokady\n");
    if (fcntl(fd, F_SETLKW, &fl) == -1) {
        perror("fcntl");
        exit(1);
    }
    printf("Blokada zajeta\n");
    printf("Nacisnij <RETURN> aby zwolnic blokade\n");
    getchar();
    fl.l_type = F_UNLCK;  /* Zwolnienie blokady */
    if (fcntl(fd, F_SETLK, &fl) == -1) {
        perror("fcntl");
        exit(1);
    }
    printf("Zwolniona\n");
    close(fd);
    return 0;
}
```

Przykład 4-6 Demonstracja blokady pliku

Użycie funkcji lockf

Funkcja lockf jest obudową funkcji **fcntl** i ułatwia z niej korzystanie.

Funkcja 4-1 **lockf –** zablokowanie dostepu do pliku	
int lockf(int fd, int funkcja, int zakres)	
fd	Uchwyt pliku
funkcja	Specyfikacja operacji: F_LOCK, F_ULOCK, F_TEST, F_TLOCK
ile	Zakres blokowanie

Funkcja zwraca:

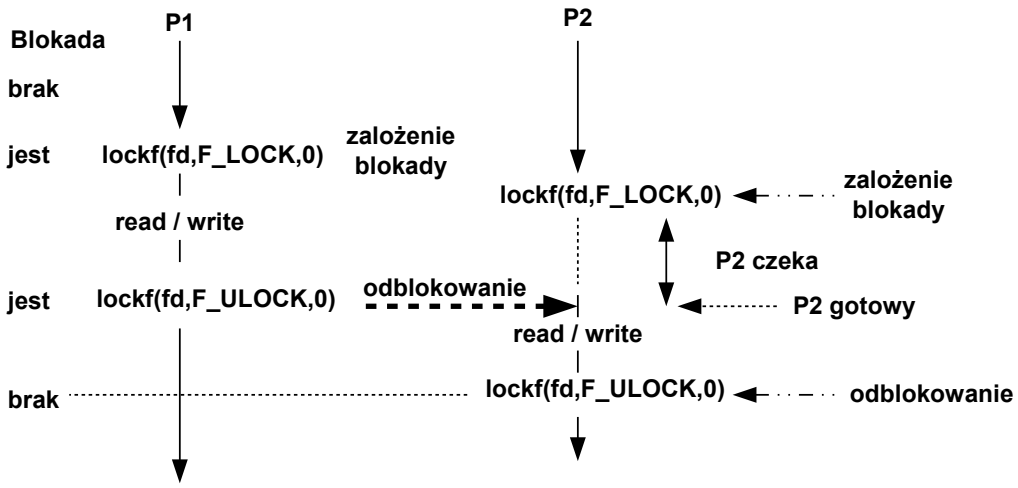
-1	Gdy błąd
>0	Sukces

F_LOCK	Zablokuj dostęp do pliku na długości **zakres** od pozycji bieżącej
F_ULOCK	Zwolnij dostęp do pliku.
F_TEST	Testuj czy fragment pliku jest zablokowany przez inny proces
F_TLOCK	Testuj czy fragment pliku jest zablokowany przez inny proces. Gdy nie to zajmij plik

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

int main(int argc, char * argv[]) {
 int fd,ile,res, num = 444;
 char buf[8];
 fd = open("plik.txt",O_WRONLY | O_CREAT,  S_IRUSR |
           S_IWUSR );
 if(fd < 0) {
      perror("open");
      exit(0);
 }
  printf("Zalozenie blokady \n");
  res = lockf(fd,F_LOCK,0);
  ile = write(fd,&num,0);
  res = lockf(fd,F_ULOCK,0);
  close(fd);
}
```

Przykład 4-7 Blokowanie pliku do zapisu



Rys. 4-1 Przebieg operacji blokowania

4.6 Standardowa biblioteka wejścia / wyjścia

Biblioteka rozszerza możliwości funkcji niskopoziomowych

- Zapewnia wiele rozbudowanych funkcji ułatwiających formatowanie wyjścia i skanowania wejścia
- Obsługuje buforowanie
- Funkcje zadeklarowane są w pliku nagłówkowym stdio.h
- Odpowiednikiem uchwytu jest strumień (ang. stream) widziany w programie jako FILE*

Gdy uruchamiamy program automatycznie otwierane są trzy strumienie:

Opis	Wartość uchwytu		Przypisanie początkowe
Standardowe wejście	0	stdin	Klawiatura
Standardowe wyjście	1	stdout	Ekran
Wyjście komunikatów o błędach	2	stder	Ekran

Tab. 4-10 Standardowe wejście / wyjście programu

Funkcja	**Opis**
`fopen, fclose`	**Otwarcie lub utworzenie pliku, zamknięcie pliku**
`fread`	**Odczyt z pliku**
`fwrite`	**Zapis do pliku**
`fseek`	**Pozycjonowanie bieżącej pozycji pliku**
`fgetc,getc, getchar`	**Odczyt znaku**
`fputc, putc, putchar`	**Zapis znaku**
`frintf, fprintf, sprintf`	**Formatowane wyjście**
`scanf, fscanf, sscanf`	**Skanowanie wejścia**
`fflush`	**Zapis danych na nośnik**

Tab. 4-11 Ważniejsze funkcje wysokiego poziomu dostępu do plików

4.6.1 Otwarcie pliku – funkcja fopen

```
FILE* fopen(char *path,char *tryb)
```

path Nazwa pliku lub urządzenia
tryb Tryb dostępu do pliku

Funkcja powoduje otwarcie pliku lub urządzenia o nazwie wyspecyfikowanej w parametrze **path**. Otwarcie następuje zgodnie z trybem tryb. Funkcja zwraca identyfikator strumienia który służy do identyfikacji pliku w innych funkcjach biblioteki.

	Dostęp	Bieżąca pozycja pliku
r	odczyt	początek
r+	Aktualizacja (odczyt i zapis)	początek
w	Zapis	początek
w+	Odczyt i zapis. Gdy pliku nie ma zostanie utworzony, gdy jest stara zawartość jest ona kasowana.	początek
a	Dopisywanie. Gdy pliku nie ma zostanie utworzony	koniec
a+	Dopisywanie i odczyt. Gdy pliku nie ma zostanie utworzony	koniec

Funkcja zwraca:
Wskaźnik na strumnień gdy sukces
NULL gdy błąd

Podana niżej funkcja fdopen transformuje uchwyt pliku na strumień

```
FILE * fdopen(int filedes, const char *opentype)
```

Gdzie:
filedes – uchwyt pliku
opentype - tryb otwarcia tak jak w **fopen**

W odwrotną stronę działa funkcja **fileno**

```
int fileno(FILE *stream)
```

4.6.2 Odczyt z pliku – funkcja fread

```
size_t fread(void bufor, size_t size, size_t nmemb,
             FILE* stream )
```

bufor Bufor w którym umieszczane są przeczytane bajty
size Wielkość elementu w bajtach
nmemb Liczba elementów którą chcemy przeczytać
stream Identyfikator strumienia zwracany przez funkcję fopen

Funkcja powoduje odczyt z pliku identyfikowanego przez stream , size elementów z których każdy ma nmemb bajtów i umieszczenie ich w buforze. Funkcja zwraca liczbę rzeczywiście przeczytanych elementów (nie bajtów)

4.6.3 Zapis do pliku – funkcja fwrite

```
size_t fwrite(void bufor, size_t size, size_t nmemb,
              FILE* stream )
```

bufor Bufor w którym umieszczane są bajty do zapisu
size Wielkość elementu w bajtach
nmemb Liczba elementów którą chcemyzapisać
stream Identyfikator strumienia zwracany przez funkcję fopen

Funkcja powoduje zapis do pliku identyfikowanego przez stream , size elementów z których każdy ma nmemb bajtów z bufora.
Funkcja zwraca liczbę rzeczywiście zapisanych elementów (nie bajtów)

4.6.4 Ustawianie bieżącej pozycji pliku – funkcja fseek

```
int fseek(FILE* stream, long int offset, int skad)
```

stream Identyfikator strumienia zwracany przez funkcję fopen
offset Nowa pozycja wskaźnika
skad Skąd ma być nowa pozycja liczona

Funkcja powoduje przesunięcie wskaźnika bieżącej pozycji pliku identyfikowanego przez **stream**, na nową pozycję określaną przez offset. Sposób określania tej pozycji zależy od flagi **skad**.

SEEK_SET	Od początku pliku
SEEK_CUR	Od bieżącej pozycji wskaźnika pliku
SEEK_END	Od końca pliku

Tab. 4-12

Funkcja zwraca:
- 0 – sukces,
- 1 – gdy błąd.

4.6.5 Zamknięcie pliku – funkcja fclose

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

stream Identyfikator strumienia zwracany przez funkcję fopen

Funkcja powoduje zamknięcie pliku identyfikowanego przez stream. Należy ją wykonać gdy nie będą już wykonywane operacje na danym pliku .

```
#include <stdio.h>
#define SIZE 80

int main() {
   int ile;
   FILE *f;
   char buf[SIZE];
   f = fopen("fread.c","r");
   if(f == NULL) { perror("fopen"); exit(0);}
   do {
    ile = fread(&buf,sizeof(buf),1,f);
    printf("%s\n",buf);
   } while(ile == 1);
   fclose(f);
   return 0;
}
```

Przykład 4-8 Przykład odczytu pliku funkcją fread

4.6.6 Odczyt znaku ze strumienia – fgetc, getc, getchar

```
int fgetc (FILE* stream )
int getc (FILE* stream )
```

```
int getchar()
```

stream Identyfikator strumienia zwracany przez funkcję fopen

Funkcje czytają jeden znak ze strumienia i zwracają go jako wynik. Gdy nie ma znaków zwracają stałą EOF. Funkcja getchar jest tym samym co getc(stdout).

4.6.7 Zapis znaku do strumienia – funkcje fputc, putc, putchar

```
int fput (int  c, FILE* stream )
int putc (int  c, FILE* stream )
int putchar(int  c)
```

stream Identyfikator strumienia zwracany przez funkcję fopen
c Znak do zapisu

Funkcje piszą jeden znak do strumienia. Funkcja putchar jest tym samym co putc(c,stdout).

4.6.8 Odczyt łańcucha ze strumienia – funkcje fgets, gets

```
char* fgets(char* s,int max, FILE* stream )
char* gets(char* s)
```

stream Identyfikator strumienia zwracany przez funkcję fopen
max Maksymalna liczba znaków
s Łańcuch

Funkcja fgets odczytuje ze strumienia kolejne znaki i kopiuje je do bufora s do chwili aż:

- Napotkany zostanie znak nowej linii
- Wczytane zostanie max-1 znaków
- Plik się skończy

Bufor zostanie uzupełniony znakiem \0 końca łańcucha. Funkcja gets czyta ze standardowego wejścia.

4.6.9 Formatowane wyjście – funkcje fprintf, sprintf, printf

```
int fprintf(FILE* stream, const char* format,…)
int sprintf(char* s, const char* format,…)
int printf(const char* format,…)
```

stream Identyfikator strumienia zwracany przez funkcję fopen
format Łańcuch formatujący
s Bufor na łańcuch wyjściowy

Funkcja **fprintf** pisze do strumienia stream zawartość kolejnych zmiennych zgodnie z łańcuchem formatującym. Funkcja **sprintf** pisze do bufora s a funkcja **printf** na standardowe wyjście.

%d	Liczba całkowita w formacie dziesiętnym
%o	Liczba całkowita w formacie ósemkowym
%x	Liczba całkowita w formacie szesnastkowym
%c	Znak
%s	Łańcuch
%f	Liczba zmiennoprzecinkowa
%g	Liczba zmiennoprzecinkowa w postaci wykładniczej

Tab. 4-13 Ważniejsze specyfikatory konwersji

```
fprintf(plik,"PID: %d Nazwa: %s\n",getpid(),argv[0]);
```

Funkcja zwraca liczbę wypisanych znaków. Gdy wystąpi błąd zwraca –1 i ustawia zmienną **errno**.

4.6.10 Formatowane wejście – funkcje fscanf, sscanf, scanf

```
int fscanf(FILE* stream, const char* format,…)
int sscanf(char* s, const char* format,…)
int scanf(const char* format,…)
```

stream Identyfikator strumienia zwracany przez funkcję fopen
format Łańcuch formatujący
s Bufor na łańcuch wejściowy

Funkcja **fscanf** czyta dane ze strumienia stream i umieszcza je w zmiennych których adresy wyszczególnione są jako parametry. Ważne aby sposób zapisu pola danych był zgodny z typem zmiennej w której to

pole ma być zapisane. Funkcja scanf zwraca liczbę wczytanych elementów.

4.6.11 Testowanie końca pliku – funkcja feof

```
int feof(FILE* stream)
```

stream Identyfikator strumienia zwracany przez funkcję fopen

Funkcja zwraca zero gdy nie został ustawiony wskaźnik końca pliku i nie zero gdy wystapił koniec pliku.

4.6.12 Konwersja strumienia na uchwyt pliku – funkcja fileno

```
int fileno(FILE* stream)
```

stream Identyfikator strumienia zwracany przez funkcję fopen

Funkcja zwraca uchwyt pliku identyfikowanego przez strumień.

```
#include <stdlib.h>
#include <stdio.h>
#define SIZE 512

int main(int argc, char* argv[]){
     FILE * skad, * dokad;
     int odczyt,zapis,ile=0;
     char buffer[SIZE];
     if(argc!=3){
           fprintf(stderr,"Uzycie: filecopy skad dokad\n");
        exit(1);
     }
     // Otwarcie pliku zrodlowego
     skad=fopen(argv[1],"rb");
     if(skad==NULL) { perror("fopen 1"); exit(1);}

     // Otwarcie pliku docelowego
     dokad=fopen(argv[2],"wb");
     if(dokad==NULL){ perror("fopen 2"); exit(1);}
     do {
           // Czytanie do bufora
           odczyt=fread(buffer,1,SIZE,skad);
           ile = ile + odczyt * SIZE;
           // Zapis z bufora
           zapis=fwrite(buffer,1,odczyt,dokad);
           printf("Odczyt %d zapis %d\n",odczyt,zapis);
           if(zapis != odczyt) { perror("fwrite"); exit(1); }
     } while(feof(skad)==0);
     printf("Skopiowano %d bajtow\n",ile);
     fclose(skad);
     fclose(dokad);
     return 0;
}
```

Przykład 4-9 Kopiowanie plików przy użyciu funkcji fread i fwrite

4.7 Zmienna errno i funkcja perror

Funkcje systemowe mogą skończyć się błędem, na co programista powinien być przygotowany. Błąd funkcji sygnalizowany jest jej kodem powrotu, zwykle –1. Numer błędu ostatniej funkcji systemowej przechowywany jest w zmiennej globalnej errno. Ich znaczenie zdefiniowane jest w pliku nagłówkowym <errno.h>

```
#include <ernno.h>
main(int argc, char * argv[]) {
  int fd;
  fd = open(argv[1],O_RDONLY);
  if(fd < 0) {
       printf("Blad %d\n",errno);
       return;
  }
  ...
}
```

Przykład 4-10 Użycie zmiennej errno

Gdy spróbujemy otworzyć powyższym programem nieistniejący plik otrzymamy komunikat:

```
Blad: 2
```

Aby określić znaczenie błędu trzeba przeszukać plik <errno.h> co jest niewygodne. Aby uniknąć tego można skorzystać z funkcji perror.

`void perror(char *napis)`
Gdzie:
`napis` Komunikat wypisywany na **`stderr`**

Funkcja wypisuje na urządzeniu **`stderr`** łańcuch napis a potem słowny opis błędu.

```
#include <ernno.h>
main(int argc, char * argv[]) {
  int fd;
  fd = open(argv[1],O_RDONLY);
  if(fd < 0) {
       perror("open");
       return;
  }
  ...
}
```

Przykład 4-11 Użycie funkcji perror do sygnalizacji błędu

Gdy spróbujemy otworzyć powyższym programem nieistniejący plik otrzymamy komunikat:

```
open: No such file or directory
```

4.8 Katalogi i operowanie na katalogach

4.8.1 Organizacja systemu plików i własności katalogów

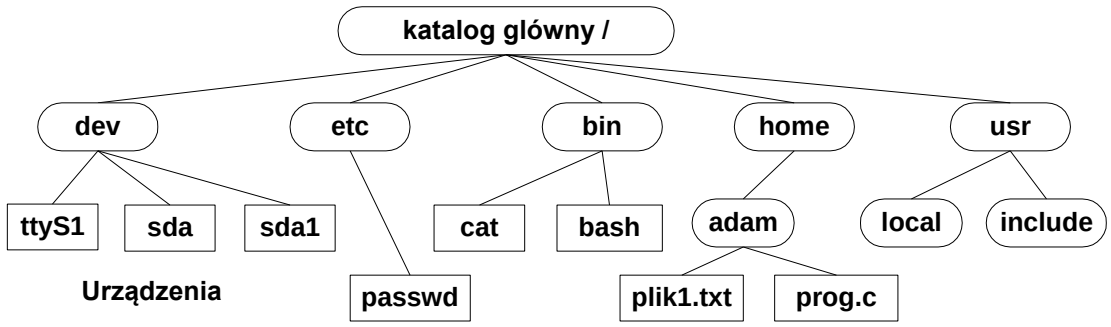
Katalog – zbiór nazw plików.
Własności katalogów

- Katalog jest plikiem specjalnym.
- Katalogi mogą być zagnieżdżane. Każdy podkatalog ma dokładnie jeden katalog macierzysty
- Katalogi maja strukturę drzewa
- Jeden katalog – katalog główny jest wyróżniony. Tworzy on wierzchołek drzewa katalogów a jego katalogiem macierzystym jest on sam.

Celem katalogów jest ułatwienie dostępu do plików i nadanie im uporządkowanej struktury.

System plików – zbiór katalogów i plików. Mają budowę hierarchiczną. Zazwyczaj system plików związany jest z partycją dyskową lub innym urządzeniem pamięciowym.

Pliki specjalne – reprezentują zasoby komputera w tym urządzenia zewnętrzne dołączone do komputera: dyski, porty szeregowe, sieć, klawiaturę, monitor, pamięć.



Rys. 4-2 Fragment systemu plików

System plików ma strukturę drzewa.

- Jego korzeniem jest katalog główny.
- Węzły końcowe są: plikami regularnymi, plikami specjalnymi, katalogami

Położenie pliku w systemie plików definiowane jest przy pomocy ścieżki którą należy przebyć od korzenia systemu plików / do danego pliku. Ścieżka zaczyna się od / i zawiera nazwy podkatalogów oddzielone znakiem ”/ ” a na końcu nazwę pliku. Np.:

`/home /adam/prog.c`

Jest to tak zwana ścieżka bezwzględna – zaczyna się ona od znaku ”/ ”. Atrybutem procesu jest katalog bieżący (w nim domyślnie będą tworzone pliki). Do plików można się również odwoływać przez ścieżki względne. Zaczyna się ona z katalogu bieżącego i prowadzi do danego pliku. Ścieżka od korzenia do katalogu bieżącego dodana będzie automatycznie. Przykładowo gdy katalogiem bieżącym jest /home to do pliku **`/home/adam/prog.c`** można się odwołać jako **`adam/prog.c`**. Wiele poleceń systemowych odnosi się do katalogu bieżącego (np. ls).

Każdy z użytkowników posiada swój katalog macierzysty zdefiniowany w pliku **`/etc/passwd`**.

Korzeń / systemu plików można zmienić za pomocą polecenia chroot. Wykorzystywane jest to między innymi do testowania innych dystrybucji systemu.

Implementacja katalogów:

- Katalogi są plikami.
- Pliki zawierają pozycje odpowiadające plikom lub katalogom.
- Każda z pozycji zawiera co najmniej: nazwę pliku i numer i-węzła. Numer i-węzła jednoznacznie identyfikuje plik w ramach systemu plików.

Katalogi mogą być otwierane i odczytywane za pomocą funkcji **`open`**, **`read, fstat`** lecz nie mogą być tworzone za pomocą funkcji **`open, creat.`**

Kropka i podwójna kropka
W każdym katalogu zawarte są dwie pozycje:
. . podwójna kropka – jest łączem do katalogu macierzystego
. pojedyncza kropka – jest łączem do bieżącego katalogu

Prawa dostępu dotyczące katalogów
Podobnie jak do plików regularnych do katalogów stosują się prawa dostępu dla właściciela, grupy i innych. Mają jednak inne znaczenie.

Odczyt	r	Można uzyskać nazwy plików i katalogów zawartych w tym katalogu
Zapis	w	Można tworzyć i kasować pliki w podkatalogu
Wykonanie	x	Można przejść do katalogu za pomocą polecenia **cd** lub funkcji **chdir**. Aby wykonać program lub utworzyć plik trzeba mieć prawa wykonania na całej bezwzględnej ścieżce biegnącej do pliku.

W operowaniu na katalogach używa się struktury typu **dirent** zdefiniowanej w pliku nagłówkowym **<dirent.h>**. Struktura ta posiada element **d_name[ ]** który zawiera nazwę pliku zawartego w katalogu.

Funkcja	Opis
opendir	Otwarcie katalogu
readdir	Odczyt pozycji katalogu
rewinddir	Przejście do początku katalogu
closedir	Zamknięcie katalogu
mkdir	Tworzenie katalogu
rmdir	Kasowanie katalogu
chdir	Zmiana katalogu bieżącego
getcwd	Odczyt katalogu bieżącego

Tab. 4-14 Ważniejsze funkcje operujące na katalogach

4.8.2 Otwieranie i zamykanie katalogów

Aby uzyskać dostęp do katalogu należy go otworzyć co robi się za pomoca funkcji opendir.

DIR* opendir(char* dirname)

dirname Nazwa katalogu który ma być otwarty

Funkcja zwraca wskaźnik na strukturę DIR która wskazuje na pierwszą pozycję katalogu. Gdy wywołanie się nie uda zwracany jest NULL. Działanie funkcji podobne jest do funkcji **fopen** która otwiera strumień zwracając wskaźnik na strukturę FILE.

Gdy nie będa już wykonywane operacje na katalogach należy go zamknąć wykonując funkcję **closedir** .

int closedir(DIR* dirptr)

dirptr Wskaźnik na strukturę DIR

4.8.3 Czytanie katalogów

Gdy katalog zostanie otwarty można odczytywać kolejne jego pozycje. Robi się to za pomocą funkcji readdir.

`struct dirent* readdir(DIR* dirptr)`

`dirptr` Wskaźnik na strukturę DIR

Funkcja powoduje skopiowanie do struktury **`dirent`** danych o bieżącej pozycji katalogu i przesuwa ten wskaźnik na następną pozycję.

```
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>

int main(int argc, char * argv[]) {
   int ile;
   struct dirent *dp;
   DIR* dirptr;
   dirptr = opendir(argv[1]);
   if(dirptr == NULL) return 0;
   do {
     dp = readdir(dirptr);
     if(dp == NULL) break;
     printf("%s\n",dp->d_name);
   } while(1);
   closedir(dirptr);
   return 1;
}
```

Przykład 4-12 Program pokazuje zawartość katalogu podanego jako argument

4.8.4 Tworzenie i kasowanie katalogu

`int mkdir(char* patchname)`

`patchname` Nazwa tworzonego katalogu (wraz ze ścieżką)

Funkcja zwraca 0 gdy sukces lub –1 gdy błąd. Funkcja tworzy katalog o nazwie **`patchname`**. Tworzy ona także dwa łącza:

.. łącze do katalogu wyższego poziomu
. łącze do bieżącego katalogu

Gdy katalog nie jest potrzebny może być usunięty za pomocą funkcji rmdir.

int rmdir(char* patchname)

patchname Nazwa kasowanego katalogu (wraz ze ścieżką)

Funkcja zwraca 0 gdy sukces lub –1 gdy błąd. Usunięty może być tylko pusty katalog.

4.8.5 Odczyt i zmiana katalogu bieżącego

char* getcwd(char* name, size_t size)

size Maksymalna długość nazwy
name Uzyskana nazwa katalogu bieżącego (wraz ze ścieżką)

Funkcja zwraca nazwę katalogu bieżącego.

int chdir(char* patchname)

patchname Specyfikacja(wraz ze ścieżką) nowego katalogu bieżącego

Funkcja zwraca 0 gdy sukces lub –1 gdy błąd. Funkcja zmienia katalog bieżący na **patchname**.

4.9 Narzędzia do testowania systemu plików

ls	Listowanie zawartości katalogu
lsof	Wyświetla listę otwartych plików
stat	Wyświetla atrybuty pliku
file	Wyświetla typ pliku

Tabela 1 Ważniejsze polecenia dotyczące plików

4.9.1 stat – wyświetlanie atrybutów pliku

Polecenie stat wyświetla atrybuty pliku

```
$stat pot2
Size: 4854          Blocks: 16         IO Block: 4096   zwykły
plik
Device: 801h/2049d  Inode: 1712519     Links: 1
Access: (0755/-rwxr-xr-x)  Uid: ( 1000/    juka)   Gid: (
1000/    juka)
Access: 2014-11-25 23:49:43.000000000 +0100
Modify: 2013-04-26 01:15:06.000000000 +0200
Change: 2013-04-26 01:15:06.000000000 +0200
```

4.9.2 lsof – wyświetlenie listy otwartych plików

Polecenie lsof wyświetla listę otwartych plików

```
$lsof /home/juka
x-session 2029 juka  cwd    DIR    8,1     4096 1541388 /home/juka
gnome-pow 2092 juka  cwd    DIR    8,1     4096 1541388 /home/juka
metacity  2095 juka  cwd    DIR    8,1     4096 1541388 /home/juka
. . .
update-no 2128 juka  cwd    DIR    8,1     4096 1541388 /home/juka
gnome-ter 2225 juka  cwd    DIR    8,1     4096 1541388 /home/juka
```

Otwarte pliki można też uzyskać z katalogu:
/proc/pid_procesu/fdinfo

4.9.3 file – wyświetla typ pliku

```
$file scan
scan: ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/Linux
2.6.18, not stripped
```

4.10 Literatura

https://www.gnu.org/software/libc/manual/html_node/index.html