Pliki

W celu wykonania jakiejkolwiek operacji na istniejącym pliku, plik ten musi zostać **otwarty**, natomiast jeśli plik jeszcze nie istnieje, to musi zostać **utworzony**.
Plik może zostać otwarty w trybie:

- do zapisu — możliwa jest wówczas tylko operacja zapisu,
- do odczytu — możliwa jest wówczas tylko operacja odczytu
- do zapisu i odczytu — możliwa jest wówczas zarówno operacja odczytu jak i zapisu.

Każdemu otwartemu plikowi odpowiada jedna pozycja w tablicy i-węzłów w jądrze. Z każdym otwarciem pliku skojarzona jest też pozycja w tablicy otwartych plików jądra. Ponieważ plik może zostać otwarty wielokrotnie, kilka pozycji w tablicy otwartych plików może wskazywać na ten sam i-węzeł.

Otwarty plik identyfikowany jest przez ***deskryptor***, czyli indeks odpowiedniej pozycji w tablicy deskryptorów procesu. Każdy otwarty plik ma zatem przydzielone miejsce w tablicy deskryptorów. Tablica deskryptorów zawiera z kolei wskaźnik (indeks) do tablicy otwartych plików jądra. Tablica deskryptorów ma ograniczony rozmiar i w zależności od wersji systemu możliwe jest jednoczesne otwarcie przez proces od kilkunastu do kilkudziesięciu plików, w szczególności może to być jednoczesne wielokrotne otwarcie tego samego pliku lub powielenie samego deskryptora.

System przydziela zawsze **wolny deskryptor o najniższej wartości** (wolną pozycję o najniższym numerze w tablicy deskryptorów). W momencie utworzenia procesu otwarte są już trzy pliki o deskryptorach 0, 1 i 2, odpowiadające standardowemu wejściu, standardowemu wyjściu i standardowemu wyjściu diagnostycznemu. Wszystkie te deskryptory najczęściej związane są z plikiem specjalnym, jakim jest terminal.

Funkcje tworzące pliki i operujące na nich opisane są w części 2 pomocy systemowej.

Tworzenie i otwieranie plików:

`open`	otwarcie pliku (uogólniona funkcja `open` umożliwia również utworzenie pliku),
`creat`	utworzenie pliku i otwarcie do zapisu,
`dup`	utworzenie kopii deskryptora i nadanie jej pierwszego wolnego numeru z tablicy otwartych plików
`dup2`	utworzenie kopii deskryptora, umożliwiające określenie jej identyfikatora przez użytkownika
`close`	zamknięcie deskryptora otwartego pliku,
`unlink`	usunięcie dowiązania do pliku.

Operacje na plikach:

`read`	odczyt fragmentu pliku,
`write`	zapis fragmentu pliku,
`lseek`	przesunięcie wskaźnika bieżącej pozycji

```
#include <fcntl.h>
```

Funkcja CREAT

```
PROTOTYPE: int creat( const char *pathname, mode_t mode );
  RETURNS: success : deskryptor otwartego pliku
           error: -1

   errno =EEXIST(plik o takiej nazwie już istnieje, a użyto
                 flag O_CREAT i O_EXCL)
          EFAULT(nazwa pathname wskazuje poza dostępną
                 przestrzeń adresową)
          EACCES(żądany dostęp do pliku nie jest dozwolony)
          EMFILE(proces już otworzył maksimum plików)
          ENFILE(osiągnięto limit otwartych plików w
                 systemie)
          EROFS (żądane jest otwarcia w trybie zapisu pliku
                 będącego plikiem tylko do odczytu)
```

PARAMETRY:

1. `pathname`- wskaźnik do napisu zawierającego nazwę ścieżki pliku, który ma być otwarty (nazwa bezwzględna lub względna)
2. `mode` - prawa dostępu (np. 0640)

UWAGI:

Funkcja tworzy plik, którego lokalizację wskazuje parametr `pathname` Prawa dostępu do utworzonego pliku ustawiane są zgodnie z parametrem `mode`. Jeśli plik o takiej nazwie już istnieje a proces wywołujący funkcję `creat` ma prawo do zapisu tego pliku, to jego zawartość jest usuwana (następuje obcięcie pliku). Plik wskazywany przez `pathname` otwierany jest w trybie do zapisu.

Funkcja OPEN

```
PROTOTYPE: int open( const char *pathname, int flags );
           int open( const char *pathname, int flags ,
                     mode_t mode);
   RETURNS: success : deskryptor otwartego pliku
            error: -1

   errno = analogicznie do funkcji open
```

PARAMETRY:

1. `pathname`- wskaźnik do napisu zawierającego nazwę ścieżki pliku, który ma być otwarty (nazwa bezwzględna lub względna)

2. `flags` - metoda dostępu

   O_RDONLY otwarcie w trybie tylko do odczytu
   O_WRONLY otwarcie w trybie tylko do zapisu
   O_RDWR otwarcie w trybie do odczytu i do zapisu

   Argument `flags` może być połączony bitowym OR z jedną (lub więcej) z następujących wartości:

   O_CREAT utworzenie pliku, jeśli plik jeszcze 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.
3. `mode` – prawa dostępu

UWAGI:
Parametr wejściowy `pathname` jest nazwą (w szczególności pełną nazwą ścieżkową) pliku, parametr wejściowy `flags` oznacza tryb otwarcia pliku pliku i może mieć następujące wartości: O_RDONLY, O_WRONLY, O_RDWR .
Dodatkowo w trybie zapisu możliwe jest użycie flagi O_APPEND, która jest sumowana bitowo z O_WRONLY lub O_RDWR i powoduje, że zapis wykonywany jest zawsze na końcu pliku. Dane są więc dopisywane do pliku i system gwarantuje, że nie nastąpi nadpisanie danych zapisanych wcześniej.

Poza funkcjami `open` i `creat` istnieje uogólniona, trzyparametrowa wersja funkcji `open`, która łączy cechy obu tych funkcji. Dodatkowy parametr prawa określa prawa dostępu do pliku (podobnie jak dla funkcji `creat`) i wykorzystywany jest wówczas, gdy tryb otwarcia wymusza tworzenie pliku. Przydatne są wówczas dodatkowe flagi umieszczane w trybie otwarcia: O_CREAT, O_TRUNC, O_EXCL.

Funkcja `creat` jest równoważna uogólnionej funkcji `open` z parametrem tryb równym O_WRONLY|O_CREAT|O_TRUNC, czyli poniższe wywołania są równoważne:

```
creat( nazwa, prawa );
open( nazwa, O_WRONLY|O_CREAT|O_TRUNC, prawa );
```

Funkcja CLOSE

```
PROTOTYPE: int close( int fd );
   RETURNS: success : 0
            error: -1
   errno = EBADF(fd nie jest prawidłowym deskryptorem otwartego
                 pliku)
```

PARAMETRY:

1. `fd`- deskryptor zamykanego pliku

UWAGI:

Zamknięcie deskryptora pliku. Funkcja zamyka deskryptor pliku przekazany przez parametr `fd`. Po zamknięciu pliku zwalniana jest pozycja w tablicy deskryptorów i może ona zostać ponownie wykorzystana przy otwarciu kolejnego pliku, czyli nowo otwarty plik może otrzymać ten sam deskryptor, który miał plik wcześniej zamknięty. Ponadto zmniejszany jest o 1 licznik deskryptorów w tablicy otwartych plików. Jeśli `fd` jest ostatnią kopią deskryptora pliku, to zasoby z nim związane zostają zwolnione, natomiast jeśli deskryptor był ostatnia referencją do pliku, który usunięto komendą `unlink`**,** plik jest kasowany.

Funkcja DUP

```
PROTOTYPE: int dup( int oldfd );
 RETURNS: success : nowy deskryptor
          error: -1

 errno = EBADF (oldfd nie jest deskryptorem otwartego
                pliku, lub newfd jest poza dozwolonym
                zasięgiem deskryptorów plików)
         EMFILE(proces już osiągnał maksymalną liczbę
                otwartych deskryptorów plików)
```

PARAMETRY:

1. `oldfd`- deskryptor zamykanego pliku

UWAGI:

Funkcja tworzy kopię pozycji w tablicy deskryptorów na innej, wolnej pozycji o najniższym indeksie. W ten sposób otrzymujemy nowy deskryptor związany z tym samym otwartym plikiem. Nowa pozycja w tablicy deskryptorów wskazuje na tą samą pozycję w tablicy otwartych plików, stąd stary i nowy deskryptor mogą być używane zamiennie. Deskryptory dzielą pozycję pliku i flagi, np. jeśli pozycja pliku zmieniła się po użyciu funkcji `lseek` na jednym z deskryptorów, zmieniła się ona także na drugim.

Funkcja DUP2

```
PROTOTYPE: int dup2( int oldfd, int newfd );
   RETURNS: success : nowy deskryptor
            error: -1
   errno = analogicznie do funkcji dup
```

PARAMETRY:

1. `oldfd`- deskryptor zamykanego pliku
2. `newfd` – nowy deskryptor

UWAGI:

Utworzenie kopii deskryptora. Podobnie jak w przypadku funkcji `dup` tworzony jest nowy deskryptor otwartego pliku identyfikowanego przez `oldfd`. `Newfd` staje się nowym, dodatkowym deskryptorem, a jeśli przed wywołaniem `dup2` identyfikował on inny plik, następuje zamknięcie tego deskryptora przed powieleniem `oldfd`. Funkcja zwraca wartość nowego deskryptora.

Wykonanie operacji `close(1); dup(fd);` jest równoważne operacji `dup(1,fd)`

Funkcja UNLINK

```
PROTOTYPE: int unlink( const char *pathname );
   RETURNS: success : 0
            error: -1
   errno = EFAULT (nazwa pathname wskazuje poza dostępną
                   przestrzeń adresową)
           EACCES (żądany dostęp do pliku nie jest
                   dozwolony)
```

PARAMETRY:

1. `pathname`- wskaźnik do napisu zawierającego nazwę ścieżki pliku, który ma być otwarty (nazwa bezwzględna lub względna)

UWAGI:

Funkcja powoduje usunięcie dowiązania do pliku. Wskazana przez parametr `pathname` nazwa pliku jest usuwana, a dodatkowo - jeśli było to jedyne dowiązanie tego pliku następuje usunięcie pliku z systemu.

Funkcja READ

```
PROTOTYPE: int read( inf fd, void *buf, size_t count );
  RETURNS: success : rzeczywista liczba bajtów, jaką udało
                     się odczytać
           error: -1

   errno = EINTR(wywołanie zostało przerwane sygnałem przed
                 odczytaniem danych)
           EAGAIN(przy użyciu O_NONBLOCK wybrano
                 nieblokujące I/O, a nie ma akurat danych
                 dostępnych do odczytania natychmiast)
           EIO   (błąd I/O. Zdarza się to np. jeśli proces
                 jest w grupie procesów tła próbuje czytać z
```

kontrolującego tty, lub ignoruje sygnał SIGTIN, lub jego grupa procesów jest osierocona.

EISDIR (fd odnosi się do katalogu)

EBADF (fd nie jest prawidłowym deskryptorem pliku, lub nie jest otwarty dla odczytu)

EINVAL(fd wskazuje na obiekt nieodpowiedni do odczytu)

EFAULT(buf wskazuje poza dostępną przestrzeń adresową)

PARAMETRY:

1. `fd`- deskryptor pliku z którego mają zostać odczytane dane
2. `buf` – adres bufora znajdującego się w segmencie danych procesu, do którego zostaną przekazane dane odczytane z pliku w wyniku wywołania funkcji `read`
3. `count` – ilość bajtów do odczytania

UWAGI:

Odczyt z pliku. Funkcja powoduje odczyt `count` bajtów z otwartego pliku, identyfikowanego przez deskryptor `fd`, począwszy od bieżącej pozycji wskaźnika do pliku i umieszczenie ich pod adresem `buf` w przestrzeni adresowej procesu. Funkcja zwraca liczbę bajtów na której udało się wykonać operację (zero oznacza koniec pliku).
Odczyt powoduje zmianę wskaźnika bieżącej pozycji w pliku. Po otwarciu pliku wskaźnik ten ustawiony jest na 0, czyli na początek pliku, a po kolejnych operacjach przesuwa się w kierunku końca pliku o tyle bajtów ile udało się odczytać

Funkcja WRITE

PROTOTYPE: int write(inf fd, void *buf, size_t count);

RETURNS: success : rzeczywista liczba bajtów, jaką udało się odczytać

error: -1

errno =**EBADF**(fd nie jest prawidłowym deskryptorem pliku, lub nie jest otwarty dla odczytu)

EINVAL(fd wskazuje na obiekt nieodpowiedni do zapisu)

EFAULT(buf jest poza dostępną przestrzenią adresową)

EPIPE (fd jest podłączony do potoku, lub gniazda, którego drugi koniec jest zamknięty. Gdy zdarzy się taka sytuacja, proces otrzyma sygnał **SIGPIPE;** jeśli jednak go przechwytuje, blokuje lub ignoruje, zwrócony zostanie błąd **EPIPE**)

EAGAIN(wybrano nieblokujący I/O (przy użyciu **O_NONBLOCK**) a do potoku lub gniazda fd nie można natychmiast zapisać danych)
EINTR (wywołanie zostało przerwane sygnałem przed zapisaniem danych)
ENOSPC (Urządzenie, zawierające plik fd nie ma miejsca na dane)

PARAMETRY:
1. `fd`- deskryptor pliku do którego mają zostać zapisane dane
2. `buf` – adresem bufora znajdującego się w segmencie danych procesu, z którego zostaną pobrane dane zapisane przez funkcję `write`
3. `count` – ilość bajtów do zapisania

UWAGI:
Zapis danych do pliku. Funkcja powoduje zapis `count` bajtów do otwartego pliku, identyfikowanego przez deskryptor `fd`, począwszy od bieżącej pozycji wskaźnika do pliku i umieszczenie ich pod adresem `buf` w przestrzeni adresowej procesu. Funkcja zwraca liczbę bajtów na której udało się wykonać operację.
Podobnie jak dla funkcji `read` zapis powoduje zmianę wskaźnika bieżącej pozycji w pliku.

Funkcja LSEEK

```
#include <uinstd.h>
PROTOTYPE: long lseek( inf fd, off_t offset, int whence );
  RETURNS: success : nową bieżącą pozycje, licząc względem
                     początku pliku
           error: -1
  errno = EBADF(fd nie jest prawidłowym deskryptorem pliku)
          ESPIPE(fd jest związany z potokiem, gniazdem, lub
                          FIFO)
          EINVAL(Whence jest nieprawidłową wartością)
```

PARAMETRY:
1. `fd`- deskryptor pliku
2. `offset` – liczba bajtów, o jaką należy przesunąć wskaźnik
3. `whence` – parametr określający pozycję względem której jest przesuwany wskaźnik

UWAGI:
Przesunięcie wskaźnika bieżącej pozycji. Funkcja powoduje zmianę wskaźnika bieżącej pozycji w otwartym pliku o deskryptorze *fd*. Nowa pozycja jest bajtem o numerze *offset* liczonym odpowiednio względem :
- początku pliku, gdy *whence* = **SEEK_SET**
- końca pliku , gdy *whence* = **SEEK_END**

- bieżącej pozycji, gdy *whence* = **SEEK_CUR**

Wartość parametru *offset* < 0 oznacza przesunięcie w kierunku początku pliku (niższych indeksów), a wartość *offset* >0 oznacza przesunięcie w kierunku końca pliku (wyższych indeksów). Funkcja zwraca aktualną wartość wskaźnika bieżącej pozycji (po przesunięciu) liczoną względem początku pliku. Działanie funkcji sprowadza się do modyfikacji zawartości odpowiedniego pola w tablicy otwartych plików.
Nie można przesunąć wskaźnika na pozycję przed początkiem pliku, można za to wyszczególnić pozycję poza końcem pliku. Jeśli później w tym miejscu są zapisane jakieś dane, to kolejne odczyty danych z luki zwrócą bajty zerowe (aż dane nie zostaną rzeczywiście zapisane w luce).