Header errno.h

Header	Opis zawartych funkcji	Przykładowe funkcje
errno.h	Operacje obsługi błędów	errno, perror(), strerror() <-- nie z 'errno.h' lecz z 'string.h'

Prawdą jest, że program napotykając **poważny programistyczny błąd** (***error***) zwykle nie przerywa swojego działania natychmiast, w miejscu jego występowania. Program w tym wypadku czeka na reakcję, skądkolwiek by ona nie pochodziła, aby zgodnie z nią zareagować i taka reakcja wymagana jest natychmiast. Tylko brak takiej reakcji kończy się przez nikogo nie lubianym, niekontrolowanym przerwaniem działania programu.

Ostrzeżenia (***warnings***) natomiast, to takiego typu błędy, na które kompilator ma swoje reakcje domyślne.

#include<errno.h>

- **errno** – makro, predefiniowana zmienna globalna trzymająca stałe skojarzone z numerem błędu. Brak błędu oznacza errno = 0. Jej definicja w errno.h to:

```
extern int _Cdecl errno;
```

Aby wyświetlić komunikat o błędzie podanym przez **errno** wykorzystuje się jedną z dwóch funkcji:

- **perror()** – wyświetla przekazany ciąg znaków, po którym następuje dwukropek i spacja, a następnie opis kodu błędu przechowywanego w **errno** zakończony pustą linią
- **strerror()** – nie występuje w **errno.h** lecz w **string.h** ale tu pasuje do tematu – zwraca wskaźnik do komunikatu o błędzie wygenerowanym przez system, który jest skojarzony z errno [a więc wyświetli to samo co *perror()*]

```
char * _Cdecl strerror(int __errnum);
```

Obie funkcje, **perror()** i **strerror()**, wyświetlają ten sam komunikat dla określonego numeru błędu. Różnica pomiędzy nimi polega na tym, że **perror()** to format tekstowy **errno**, podczas gdy **strerror()** przekazuje numer błędu jako argument funkcji.

```c
// Obsługa błędów

#include<stdio.h>
#include<errno.h>                       // ten temat
#include<locale.h>                      // dla 'setlocale()'

int main(void)
{  setlocale(LC_CTYPE, "Polish");       // polskie znaki
   FILE *fp;                            // fp sugeruje nazwę 'file pointer'
   fp = fopen("Nie_ma_mnie.txt", "r");  // nie ma takiego pliku
   if (fp == NULL)
   {  printf("Numer błędu (wartość 'errno') : %d\n", errno);
      perror("Opis błędu (odczyt z 'perror')");
      return 0;
   } else
      fclose(fp);
}
```

Wynik działania programu:

```
Numer błędu (wartość 'errno') : 2
Opis błędu (odczyt z 'perror'): No such file or directory

--------------------------------
Process exited after 2.06 seconds with return value 0
Press any key to continue . . . _
```

Gdy zmienimy linię dla opisu błędu z

```
      perror("Opis błędu (odczyt z 'perror')");
```

na

```
      perror("");
```

to otrzymamy:

```
Numer błędu (wartość 'errno') : 2
No such file or directory

--------------------------------
Process exited after 1.42 seconds with return value 0
Press any key to continue . . . _
```

Kiedy spodziewamy się wystąpienia błędu i nie zależy nam na szczegółach dodyczących nieudanych operacji, można się obejść bez 'errno.h' jak w przykładzie:

```
    FILE *tymczasowy = tmpfile();        // próba utworzenia pliku tymczasowego
    if (tymczasowy == NULL)
    {  printf("Nieudana operacja utworzenia pliku tymczasowego.\n");
       return 0;
    }
```

A gdy chcemy wiedzieć, co złego się stało, pytamy o to **errno**:

```
// Obsługa błędów

#include<stdio.h>
#include<errno.h>                  // ten temat
#include<locale.h>                 // dla 'setlocale()'

int main(void)
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki
   FILE *tymczasowy = tmpfile();   // próba utworzenia pliku tymczasowego
   if (tymczasowy == NULL)
   {  printf("Numer błędu (wartość 'errno') : %d\n", errno);
      perror("Opis błędu (odczyt z 'perror')");
      return 0;
   } else
        printf("Plik tymczasowy udało się utworzyć ale zostanie on usunięty
automatycznie wraz z końcem działania programu.\n");

      return 0;
}
```

Wynik działania programu:

```
Numer błędu (wartość 'errno') : 13
Opis błędu (odczyt z 'perror'): Permission denied

--------------------------------
Process exited after 1.966 seconds with return value 0
Press any key to continue . . . _
```

lub z wykorzystaniem funkcji `strerror()`

```c
// Obsługa błędów

#include<stdio.h>
#include<errno.h>                  // ten temat
#include<string.h>                 // dla strerror()
#include<locale.h>                 // dla 'setlocale()'

int main(void)
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki
   FILE *tymczasowy = tmpfile();   // próba utworzenia pliku tymczasowego
   if(tymczasowy == NULL)
   { printf("Numer błędu (wartość 'errno') : %d\n", errno);
     fprintf(stderr, "Opis błędu (odczyt przez 'strerror()'): %s\n", strerror(errno));
     return 0;
   } else
        printf("Plik tymczasowy udało się utworzyć ale zostanie on usunięty
automatycznie wraz z końcem działania programu.\n");

   return 0;
}
```

z takim samym rezultatem:

```
Numer błędu (wartość 'errno') : 13
Opis błędu (odczyt przez 'strerror()'): Permission denied

--------------------------------
Process exited after 2.43 seconds with return value 0
Press any key to continue . . . _
```

Zawsze linię z `fprintf()`:

```c
     fprintf(stderr, "Opis błędu (odczyt przez 'strerror()'): %s\n", strerror(errno));
```

można zamienić na linię z `printf()` bo ona automatycznie da wydruk na ekranie monitora:

```c
     printf("Opis błędu (odczyt przez 'strerror()'): %s\n", strerror(errno));
```

z takim samym rezultatem.

Zarówno **stdio.h** jak i **stdlib.h** dysponują operacjami na błędach i nie potrzebują możliwości **errno.h**.

Utwórz na pulpicie plik w notatniku o nazwie '`test.txt`' z linijką dowolnego tekstu.

```c
// Obsługa błędów bez wywołania 'errno.h'

#include<stdio.h>                   // już tu jest dostęp do 'perror' i 'ferror'
#include<locale.h>                  // dla 'setlocale()'

int main()
```

```c
{   setlocale(LC_CTYPE, "Polish");  //polskie znaki

    FILE *fp;
    fp = fopen("C:/Users/Artur/Desktop/test.txt", "r" ); // plik jest tylko do odczytu
    if( !fp )
       perror("Błąd otwarcia pliku \'test.txt\'" );
    else
    {  printf("Plik \'test.txt\' jest otwarty. Można na nim pracować.");
       fwrite("TEKST", 5, 1, fp);   // chcemy do pliku wpisać słowo "TEKST"
       if( ferror(fp) )             // plik jest tylko do odczytu więc będzie błąd
       {  printf("\nBłąd zapisu do pliku \'test.txt\'");
          perror("\nBłąd drukowany przez 'perror' ");
       }
    }
    fclose( fp );
    return 0;
}
```

Wynik działania programu:

```
Plik 'test.txt' jest otwarty. Można na nim pracować.
Błąd zapisu do pliku 'test.txt'
Błąd drukowany przez 'perror' : Bad file descriptor

--------------------------------
Process exited after 0.1129 seconds with return value 0
Press any key to continue . . . _
```

Podejrzewam, że powodem błędu `Bad file descriptor jest` oznaczenie trybu `"r"` zamiast np. `"w"` bo plik jest tylko do odczytu a program chce coś do niego wpisać.

Gdy nie sprawdzimy podejrzanej operacji, jak poniżej...:

```c
// Brak sprawdzenia operacji
#include<stdio.h>
#include<errno.h>       // dołączona do programu ale nie proszona o pomoc, milczy

int main(void)
{  char autor[] = "Henryk Sienkiewicz";
   printf("Drukuj setny znak w nazwie autora: %c", autor[99]);
   return 0;
}
```

Wynik

```
Drukuj setny znak w nazwie autora:
--------------------------------
Process exited after 1.793 seconds with return value 0
Press any key to continue . . . _
```

...kompilator nie wykona zadania i milczy albo... wykona operację wadliwie.

Nawet wpisanie:

```c
   if( autor[99] == NULL )
```

```
   {  printf("Numer błędu (wartość 'errno') : %d\n", errno);
      perror("Opis błędu (odczyt z 'perror')");
   }
```

daje:

```
Numer błędu (wartość 'errno') : 0
Opis błędu (odczyt z 'perror'): No error
```

Dlatego logika operacji, szczególnie na łańcuchach znakowych jest ważniejsza niż umiejętność obsługi błędów w każdym typie programowania. Każdy łańcuch znakowy zawsze powinien być zamknięty znakiem `'\0'` należącym do tego łańcucha i dlatego deklaracja ilości bajtów dla tego łańcucha winna być przynajmniej o 1 większa niż długość łańcucha znakowego.

Zawsze:

- powinieneś być pewnym, że nie odwołujesz się do indeksu łańcucha znakowego, który jest większą liczbą niż `strlen(`tego łańcucha`)`, a łańcuch jest zamknięty uniemożliwiając czytanie 'wyciekających' (nie należących do tego łańcucha) znaków,

- omijasz wszelkie niewykonalne operacje, jak np. dzielenie przez zero, logarytmowanie liczby niedodatniej czy o niedodatniej podstawie, pierwiastkowania liczby ujemnej...,

- nie przekraczasz limitu określającego maksymalne wartości typów liczbowych i znakowych.

Nie przestrzegając tych zasad otrzymasz w wyniku wadliwe wartości, czasem bez wnikliwej analizy niezauważalne i dlatego trudne do wykrycia.

Lista błędów: ich numery, nazwy i opisy

Numer	Nazwa błędu	Opis błędu po polsku	Opis błędu jaki może się pojawić zarówno w `perror` jak i w `strerror( errno )`
0	EZERO	Brak błędu	**No error**
1	EPERM	Niedozwolona operacja	**Operation not permitted**
2	ENOENT	Nie ma takiego pliku ani katalogu - sprawdź ścieżkę dostępu do pliku albo katalogu	**No such file or directory** **(**A component of a path doesn't specify an existing directory**)**
3	ESRCH	Nie ma takiego procesu	**No such process**
4	EINTR	Przerwane wywołanie systemowe	**Interrupted system call**
5	EIO	Błąd wejścia/wyjścia	**Input/output error**
6	ENXIO	Nie ma takiego urządzenia ani adresu	**No such device or address**
7	E2BIG	Lista argumentów za długa	**Argument list too long**
8	ENOEXEC	Błędny format pliku wykonywalnego	**Exec format error**
9	EBADF	Błędne oznaczenie pliku (możliwy niepoprawny tryb otwarcia pliku, np "r" zamiast "w")	**Bad file descriptor**
10	ECHILD	Brak procesów podrzędnych	**No child processes** (No spawned processes)
11	EAGAIN	Zasoby chwilowo niedostępne (niewystarczająca ilość pamięci lub osiągnięty maksymalny poziom zagnieżdżenia)	**Resource temporarily unavailable** (There isn't enough memory, or the maximum nesting level has been reached)
12	ENOMEM	Za mało miejsca w pamięci operacyjnej (np. nie ma wystarczającej ilości pamięci do wykonania procesu podrzędnego)	**Cannot allocate memory** (Not enough core. Eg. insufficient memory is available to execute a child process)
13	EACCES	Brak dostępu (np. próba odczytu z pliku, który nie jest otwarty. Lub przy próbie otwarcia istniejącego pliku tylko do odczytu do zapisu lub otwarcia katalogu zamiast pliku.)	**Permission denied** (eg. an attempt is made to read from a file that isn't open. Or, on an attempt to open an existing read-only file for writing, or to open a directory instead of a file.)
14	EFAULT	Błędny adres	**Bad address**
16	EBUSY	Urządzenie lub zasoby zajęte	**Device or resource busy**
17	EEXIST	Plik już istnieje (próba utworzenia pliku, który już istnieje)	**File exists** (File already exists. An attempt to create a file that already exists)
18	EXDEV	Nieprawidłowy link między urządzeniami	**Invalid cross-device link**

		(próba przeniesienia pliku na inne urządzenie (za pomocą funkcji rename)	(An attempt to move a file to a different device (using the rename function).)
19	ENODEV	Nie ma takiego urządzenia	**No such device**
20	ENOTDIR	To nie katalog	**Not a directory**
21	EISDIR	To jest katalog	**Is a directory**
22	EINVAL	Błędny argument (nieprawidłowa wartość dla argumentu przesyłanego do funkcji)	**Invalid argument** (an invalid value of an argument passing to a function.)
23	ENFILE	Zbyt wiele plików jest otwartych w systemie	**Too many files open in system**
24	EMFILE	Za dużo otwartych plików w procesie	**Too many open files**
25	ENOTTY	Niewłaściwa operacja kontrolna wejścia/wyjścia	**Inappropriate ioctl for device** (Inappropriate I/O control operation)
27	EFBIG	Plik jest za duży	**File too large**
28	ENOSPC	Brak miejsca na urządzeniu (np. dysk jest pełny)	**No space left on device** (eg. disk is full)
29	ESPIPE	Nieprawidłowe wyszukiwanie/ Błędne przesunięcie kursora czytania/pisania	**Invalid seek**
30	EROFS	System plików wyłącznie do odczytu	**Read-only file system**
31	EMLINK	Za dużo linków	**Too many links**
32	EPIPE	Przerwany potok	**Broken pipe**
33	EDOM	Argument funkcji matematycznej nie należy do dziedziny funkcji	**Numerical argument out of domain** (Numerical argument to a math function is not in the domain of the function)
34	ERANGE	Wynik jest zbyt duży. (Argument funkcji matematycznej jest zbyt duży, co powoduje częściową lub całkowitą utratę znaczenia wyniku)	**Numerical result out of range** (An argument to a math function is too large, resulting in partial or total loss of significance in the result)
36	EDEADLK	Wystąpło zakleszczenie (deadlock) zasobów (próba ingerencji w dane, które są zamknięte z powodu ingerencji w te same dane innego użytkownika)	**Resource deadlock avoided** (Resource deadlock would occur)
38	ENAMETOOLONG	Zbyt długa nazwa pliku	**File name too long**
39	ENOLCK	Brak dostępnych blokad	**No locks available**
40	ENOSYS	Niezaimplementowana funkcja / Brak takiej funkcji	**Function not implemented**
41	ENOTEMPTY	Katalog nie jest pusty	**Directory not empty**
42	EILSEQ	Niedozwolona sekwencja bajtów	**Invalid or incomplete multibyte or wide character** (Illegal sequence of bytes)
80	STRUNCATE	Kopiowanie lub łączenie łańcuchów znakowych spowodowało obcięcie łańcucha znakowego	**A string copy or concatenation resulted in a truncated string**
100	EADDRINUSE	Adres jest już w użyciu	**Address already in use**
101	EADDRNOTAVAIL	Adres jest niedostępny	**Cannot assign requested address** (Address not available)
102	EAFNOSUPPORT	Rodzina adresów nie jest obsługiwana	**Address family not supported by protocol**
103	EALREADY	Operacja jest już wykonywana	**Operation already in progress**
105	ECANCELED	Operacja jest anulowana	**Operation canceled**
106	ECONNABORTED	Połączenie zostało przerwane	**Software caused connection abort**
107	ECONNREFUSED	Połączenie zostało odrzucone	**Connection refused**
108	ECONNRESET	Połączenie zerwane przez inną osobę	**Connection reset by peer**
109	EDESTADDRREQ	Wymagany jest adres docelowy	**Destination address required**
110	EHOSTUNREACH	Nie ma drogi do serwera Host (serwer) jest nieosiągalny	**No route to host** (Host unreachable)
112	EINPROGRESS	Operacja jest teraz właśnie wykonywana	**Operation now in progress**
113	EISCONN	Gniazdo już jest połączone	**Transport endpoint is already connected**
114	ELOOP	Zbyt wiele poziomów symbolicznych linków	**Too many levels of symbolic links**
115	EMSGSIZE	Komunikat jest za długi	**Message too long**
116	ENETDOWN	Sieć nie działa (jest wyłączona)	**Network is down**
117	ENETRESET	Resetowanie sieci	**Network dropped connection on reset**
118	ENETUNREACH	Sieć jest niedostępna	**Network is unreachable**
119	ENOBUFS	Brak miejsca w buforze	**No buffer space available**
123	ENOPROTOOPT	Protokół jest niedostępny	**Protocol not available**
126	ENOTCONN	Gniazdo nie jest podłączone	**Transport endpoint is not connected**
128	ENOTSOCK	Jest to problem związany z siecią. Błąd "Operacja gniazda na nie-gnieździe" oznacza, że z jakiegoś powodu stos TCP-IP systemu Windows został przeciążony, a kanał gniazda (który jest używany do	**Socket operation on non-socket** (This is a network-related problem. The "Socket operation on a non-socket" error means that, for some reason, the Windows TCP-IP stack has been overloaded, and the

		komunikacji z Internetem) został nagle zamknięty	socket channel (which is used for communicating with the Internet) has been shut down abruptly.)
129	ENOTSUP	Operacja nie jest obsługiwana	**Operation not supported**
130	EOPNOTSUPP	Operacja nie jest obsługiwana	**Operation not supported**
132	EOVERFLOW	Wartość zbyt duża dla zdefiniowanego typu danych	**Value too large for defined data type**
133	EOWNERDEAD	Właściciel nie żyje	**Owner died**
134	EPROTO	Błąd protokołu	**Protocol error**
135	EPROTONOSUPPORT	Protokół nie jest obsługiwany	**Protocol not supported**
136	EPROTOTYPE	Nieprawidłowy typ protokołu. (Typ protokołu nie pasuje do gniazda)	**Protocol wrong type for socket**
138	ETIMEDOUT	Przekroczony czas oczekiwania na połączenie	**Connection timed out**
140	EWOULDBLOCK	Operacja zostanie zablokowana	**Resource temporarily unavailable** (Operation would block)

(Patrz także tu: "Tablice języka C", Lista błędów i ich numery)