Instrukcja laboratoryjna 1	Systemy Odporne na Błędy
	Obsługa plików w C, C++ i Java
	Przygotował: mgr inż. Leszek Ciopiński

I Wstęp

Języki programowania oferują różne narzędzia do obsługi plików. Typowym przypadkiem jest użycie funkcji otwierającej dany plik. Nie zawsze jednak otwarcie kończy się powodzeniem. Plik może nie istnieć, może być zabezpieczony przed odczytem lub zapisem lub użytkownik może nie mieć praw dostępu do pliku. Dlatego istotne jest sprawdzenie, czy po otwarciu pliku możliwe jest wykonywanie na nim operacji.

1. C

W języku C do otwarcia pliku używana jest funkcja `fopen()`. Do zgłaszania rodzaju błędu wykorzystuje ona stałą `errno`.

```
FOPEN(3)                 BSD Library Functions Manual                 FOPEN(3)

NAME
     fdopen, fopen, freopen -- stream open functions

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <stdio.h>

     FILE *
     fdopen(int fildes, const char *mode);

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

     FILE *
     freopen(const char *restrict filename, const char *restrict mode,
         FILE *restrict stream);

DESCRIPTION
     The fopen() function opens the file whose name is the string pointed to
     by filename and associates a stream with it.

     The argument mode points to a string beginning with one of the following
     sequences (Additional characters may follow these sequences.):

     ``r''   Open text file for reading.  The stream is positioned at the
             beginning of the file.
```

``r+'' Open for reading and writing. The stream is positioned at the beginning of the file.

``w'' Truncate to zero length or create text file for writing. The stream is positioned at the beginning of the file.

``w+'' Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.

``a'' Open for writing. The file is created if it does not exist. The stream is positioned at the end of the file. Subsequent writes to the file will always end up at the then current end of file, irrespective of any intervening fseek(3) or similar.

``a+'' Open for reading and writing. The file is created if it does not exist. The stream is positioned at the end of the file. Subsequent writes to the file will always end up at the then current end of file, irrespective of any intervening fseek(3) or similar.

The mode string can also include the letter ``b'' either as last character or as a character between the characters in any of the two-character strings described above. This is strictly for compatibility with ISO/IEC 9899:1990 (``ISO C90'') and has no effect; the ``b'' is ignored.

Finally, as an extension to the standards (and thus may not be portable), mode string may end with the letter ``x'', which insists on creating a new file when used with ``w'' or ``a''. If path exists, then an error is returned (this is the equivalent of specifying O_EXCL with open(2)).

Any created files will have mode "S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH" (0666), as modified by the process' umask value (see umask(2)).

Reads and writes may be intermixed on read/write streams in any order; however, a file positioning function must be called when switching between output and input, unless an input operation encounters end-of-file.

The **fdopen**() function associates a stream with the existing file descriptor, fildes. The mode of the stream must be compatible with the mode of the file descriptor. When the stream is closed via fclose(3), fildes is closed also.

The **freopen**() function opens the file whose name is the string pointed to by filename and associates the stream pointed to by stream with it. The original stream (if it exists) is closed. The mode argument is used just as in the **fopen**() function.

If the filename argument is NULL, **freopen**() attempts to re-open the file associated with stream with a new mode. The new mode must be compatible with the mode that the stream was originally opened with:

- Streams originally opened with mode ``r'' can only be reopened with that same mode.
- Streams originally opened with mode ``a'' can be reopened with the same mode, or mode ``w''.

- Streams originally opened with mode ``w'' can be reopened with the same mode, or mode ``a''.

- Streams originally opened with mode ``r+'', ``w+'', or ``a+'' can be reopened with any mode.

The primary use of the **freopen**() function is to change the file associated with a standard text stream (stderr, stdin, or stdout).

RETURN VALUES

Upon successful completion **fopen**(), **fdopen**(), and **freopen**() return a FILE pointer. Otherwise, NULL is returned and the global variable errno is set to indicate the error.

ERRORS

[EINVAL] The mode argument to **fopen**(), **fdopen**(), or **freopen**() was invalid.

The **fopen**(), **fdopen**() and **freopen**() functions may also fail and set errno for any of the errors specified for the routine malloc(3).

The **fopen**() function may also fail and set errno for any of the errors specified for the routine open(2).

The **fdopen**() function may also fail and set errno for any of the errors specified for the routine fcntl(2).

The **freopen**() function may also fail and set errno for any of the errors specified for the routines open(2), fclose(3) and fflush(3).

SEE ALSO

open(2), fclose(3), fileno(3), fseek(3), funopen(3)

STANDARDS

The **fopen**() and **freopen**() functions conform to ISO/IEC 9899:1990 (``ISO C90''). The **fdopen**() function conforms to IEEE Std 1003.1-1988 (``POSIX.1'').

Źródło: BSD Manual

Dostęp do zmiennej errno można uzyskać poprzez dołączenie do programu pliku nagłówkowego errno.h. Najczęściej spotykanymi błędami są:

- EACCES odmowa dostępu
- EAGAIN zasób tymczasowo niedostępny
- EBADF niepoprawny deskryptor plików
- EINTR podczas wykonywania funkcji nastąpiło przerwanie
- EINVAL błędny argument
- ENOTSUP operacja nie jest zaimplementowana
- EPERM operacja niedozwolona
- EPIPE przerwany potok

Źródło: https://pl.wikipedia.org/wiki/Errno [dostęp: 26 II 2019]

2. C++

Jednym z ważniejszych mechanizmów języka C++ są strumienie. Do obsługi plików przeznaczony jest strumień `fstream`. Umożliwia on zarówno odczyt, jak i zapis danych. Podobnie, jak w przypadku języka C, również w C++ nie wystarczy otworzyć strumień, aby mieć pewność co do możliwości odczytu i zapisu danych. Dlatego, tu również należy sprawdzić, czy plik został otworzony przy pomocy metody `is_open()`:

```
// fstream::is_open
#include <iostream>      // std::cout
#include <fstream>       // std::fstream

int main () {
  std::fstream fs;
  fs.open ("test.txt");
  if (fs.is_open())
  {
    fs << "lorem ipsum";
    std::cout << "Operation successfully performed\n";
    fs.close();
  }
  else
  {
    std::cout << "Error opening file";
  }
  return 0;
}
```

Źródło: http://www.cplusplus.com/reference/fstream/fstream/is_open/ [dostęp: 26 II 2019]

Nie jest to jednak jedyna metoda umożliwiająca sprawdzenie, czy otworzenie pliku powiodło się. Inną, ważną metodą, odziedziczoną po `std::ios` , jest `good()`. Jest ona bezparametrowa i zwraca `true`, gdy operacje na pliku są dozwolone lub `false`, jeżeli wystąpił błąd.

3. Java

Kluczowym mechanizmem języka Java jest jego maszyna wirtualna. Umożliwia ona odseparowanie programu od platformy sprzętowej. W niektórych jednak przypadkach, jak np. podczas odczytu plików, współpraca maszyny wirtualnej z systemem operacyjnym jest nieunikniona. Dlatego również Java oferuje mechanizmy umożliwiające sprawdzenie możliwości operowania na plikach.

Podstawowym strumieniem obsługi plików jest w języku Java klasa `File` z pakietu `java.io`. Metodami ważnymi z punktu zapewnienia odporności oprogramowania na błędy są:

public File(String pathname)

Creates a new File instance by converting the given pathname string into an

abstract pathname. If the given string is the empty string, then the result is the empty abstract pathname.
Parameters:
pathname - A pathname string
Throws:
NullPointerException - If the pathname argument is null

canRead

public boolean canRead()
Tests whether the application can read the file denoted by this abstract pathname. On some platforms it may be possible to start the Java virtual machine with special privileges that allow it to read files that are marked as unreadable. Consequently this method may return true even though the file does not have read permissions.
Returns:
true if and only if the file specified by this abstract pathname exists and can be read by the application; false otherwise
Throws:
SecurityException - If a security manager exists and its SecurityManager.checkRead(java.lang.String) method denies read access to the file

canWrite

public boolean canWrite()
Tests whether the application can modify the file denoted by this abstract pathname. On some platforms it may be possible to start the Java virtual machine with special privileges that allow it to modify files that are marked read-only. Consequently this method may return true even though the file is marked read-only.
Returns:
true if and only if the file system actually contains a file denoted by this abstract pathname and the application is allowed to write to the file; false otherwise.
Throws:
SecurityException - If a security manager exists and its SecurityManager.checkWrite(java.lang.String) method denies write access to the file

exists

public boolean exists()
Tests whether the file or directory denoted by this abstract pathname exists.
Returns:
true if and only if the file or directory denoted by this abstract pathname exists; false otherwise
Throws:
SecurityException - If a security manager exists and its SecurityManager.checkRead(java.lang.String) method denies read access to the file

or directory
Źródło: https://docs.oracle.com/javase/10/docs/api/index.html?overview-summary.html
[dostęp 26 II 2019]

II Zadania

1. Napisz program w języku C, w którym wykonasz próbę otwarcia pliku:
   a. nieistniejącego
   b. istniejącego, ale bez prawa dostępu
   c. istniejącego, z wszelkimi uprawnieniami
   Program powinien informować użytkownika o zaistniałym zdarzeniu na podstawie informacji ze zmiennej `errno`.
2. Napisz program w języku C++, w którym wykonasz próbę otwarcia pliku:
   a. nieistniejącego
   b. istniejącego, ale bez prawa dostępu
   c. istniejącego, z wszelkimi uprawnieniami
   Możliwość wykonania odczytu z pliku należy sprawdzić metodami `is_open()` i `good()`.
3. Napisz program w języku Java, w którym wykonasz próbę otwarcia pliku:
   a. nieistniejącego
   b. istniejącego, ale bez prawa dostępu
   c. istniejącego, z wszelkimi uprawnieniami
   Możliwość odczytania danych z pliku należy sprawdzić metodami `exist()`, `canRead()` i przy użyciu wszystkich wyjątków zwracanych przez wymienione metody i konstruktor `File()`.

W powyższych zadaniach, wyświetlanie zawartości pliku nie jest wymagane, ale w przypadku odmowy dostępu konieczne jest określenie powodu (nie istnieje, brak uprawnień, itp.).