Міністерство освіти і науки України

Житомирський державний технологічний університет

Лабораторна робота №1

з курсу «Системне програмування»

на тему: «Програми копіювання файлів»

м. Житомир 2011 р.

**Зміст**

1. Навчитися користуватись програмами copy1.exe та copy2.exe.

2. Розібрати роботу програм

2.1 Робота програми «COPY1.EXE»

2.2 Робота програми «COPY1.EXE»

3. Експерименти з «copy1.c»

4. Експерименти з «copy2.c»

Контрольні запитання

Висновок

Список використаної літератури

# 1. Навчитися користуватись програмами copy1.exe та copy2.exe

Програми «COPY1.С» та «COPY2.С» призначені для копіювання файлів. Програма «COPY1.С» використовує функції роботи з file handles, а «COPY2.С» - функції потокового вводу-виводу.

Для користування програмами:

* їх необхідно скомпілювати та отримати виконавчі файли «COPY1.EXE» та «COPY2.EXE»;
* запустити необхідну програму за допомогою командного рядка, ввівши відповідно "ім’я\_прогрми" "файл\_з\_якого\_копіювати" "файл\_в\_який\_копіювати";

програма копіювання файл

**2. Розібрати роботу програм**

**2.1** **Робота програми «COPY1.EXE»**

Підключаємо заголовочні файли для використання функцій та змінних:

**#include <io.h> //open(), eof(), read(), write(), close()**

**#include <conio.h> //getch()**

**#include <stdio.h> //printf()**

**#include <stdlib.h> //exit()**

**#include <fcntl.h> //O\_BINARY, O\_RDONLY, O\_WRONLY, O\_CREAT, O\_EXCL, O\_TRUNC**

**#include <sys\types.h> //**

**#include <sys\stat.h> //S\_IREAD, S\_IWRITE**

**#include <alloc.h> //**

**#include <errno.h> //EEXIST**

Макрос, що визначає розміру масиву

**#define BUFSIZE 10000**

Функція main() має два аргументи: цілочисельний аргумент argc який містить кількість аргументів у командному рядку та покажчик на масив покажчиків на рядки, де кожен вказує на певний аргумент командного рядка.

**void main( int argc, char \*\*argv ) {**

Оголошуємо змінні:

**int source, target;**

**int i;**

**char \*buffer;**

**int count;**

Перевіряємо чи правильна кількість аргументів у командному рядку. Якщо аргументів не три (програма, файл для копіювання, файл-копія), виводиться повідомлення про правильність заповнення командного рядка і виконання програми завершується.

**if( argc != 3 ){**

**printf( "\n"**

**"Usage: COPY1 [d:][\\path]source\_file [d:][\\path]target\_file\n" );**

**getch();**

**exit( 1 );**

**}**

Відкриваємо файл, який будемо копіювати за допомогою функції open(). Для цього передаємо в якості параметрів покажчик на імя файлу (argv[ 1 ]) та вказуємо типи доступу до файлу. O\_BINARY - прапор доступу у бінарному режимі. O\_RDONLY - прапор доступу до файлу "лише для читання". Функція open() у випадку успішного виконання повертає file handle і встановлює курсор на початок файлу, а у випадку помилки повертає -1 та встановлює errno в один із наступних станів: ENOENT, EMFILE, EACCES, EINVACC.

Даним if-ом перевіряємо успішність відкриття файлу, який варто копіювати.

**if( ( source = open( argv[ 1 ], O\_BINARY | O\_RDONLY ) ) == -1 ) {**

**printf( "\nOpen source file error: %d", errno );**

**getch();**

**exit( 2 );**

**}**

Відкриваємо файл, в який будемо копіювати за допомогою тієї ж функції open().

Прапор доступу до файлу встановлюємо в один із положень:

* O\_BINARY - бінарний;
* O\_WRONLY - для запису;
* O\_CREAT - створюється;
* O\_EXCL - перезаписується.

Прапор способу відкриття встановлюємо в один із положень:

* S\_IREAD - може бути прочитаний;
* S\_IWRITE - може бути записаний.

**target = open( argv[ 2 ], O\_BINARY | O\_WRONLY | O\_CREAT | O\_EXCL,**

**S\_IREAD | S\_IWRITE );**

Якщо при відкритті(створенні) файлу до якого буде відбуватися копіювання, errno дорівнює EEXIST (тобто файл з таким ім’ям вже існує), повідомити про існування файлу та запитати про необхідність його перезапису.

**if( errno == EEXIST ) {**

**printf( "\nFile already exists. Overwrite? (Y/N)\n" );**

**i = getch();**

**if( ( i == 'y' ) || ( i == 'Y' ) )**

**target = open( argv[ 2 ], O\_BINARY | O\_WRONLY | O\_CREAT | O\_TRUNC,**

**S\_IREAD | S\_IWRITE );**

**}**

Перевірити чи вдалося створити файл .

**if( target == -1 ) {**

**printf( "\nOpen target file error: %d", errno );**

**getch();**

**exit( 2 );**

**}**

Цілочисельній змінній count присвоїти значення макросу BUFSIZE.

**count = BUFSIZE;**

Виділити пам’ять на масив buffer розміром count, та перевірити чи пам’ять дійсно виділена.

**if( ( buffer = ( char\* )malloc( count ) ) == NULL ) {**

**printf( "\nNot enough memory" );**

**getch();**

**exit( 3 );**

**}**

За допомогою оператора while, який буде виконуватися доти, доки функція eof() не знайде закінчення файлу, який копіюється, будемо виконувати читання та запис з вихідного файлу у вхідний.

**while( !eof( source ) ) {**

За допомогою функції read() зчитаємо із файлу source до масиву buffer count байтів.

**if( ( count = read( source, buffer, count ) ) == -1 ) {**

**printf( "\nRead file error: %d", errno );**

**getch();**

**exit( 4 );**

**}**

За допомогою функції write() запишемо до файлу target із масиву buffer count байтів.

**if( ( count = write( target, buffer, count ) ) == -1 ) {**

**printf( "\nWrite file error: %d", errno );**

**getch();**

**exit( 5 );**

**}**

**}**

Закриємо відкриті файли та звільнимо пам’ять із буферу.

**close( source );**

**close( target );**

**free( buffer );**

Проінформуємо про успішне копіювання.

**printf("File copy...");**

**getch();**

**}**

**2.2** **Робота програми «COPY1.EXE»**

Підключаємо заголовочні файли для використання функцій та змінних:

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <time.h>**

Оголошуємо прототип функції

**void filecpy( FILE \*stream\_from, FILE \*stream\_to );**

Оголошуємо два символьних масиви розміром BUFSIZ\*10;

**char buf1[ BUFSIZ \* 10 ];**

**char buf2[ BUFSIZ \* 10 ];**

Функція main() має два аргументи: цілочисельний аргумент argc який містить кількість аргументів у командному рядку та покажчик на масив покажчиків на рядки, де кожен вказує на певний аргумент командного рядка.

**void main( int argc, char \*argv[] ) {**

Оголосимо змінні:

**time\_t start, end;**

**FILE \*stream\_from, \*stream\_to;**

Перевіряємо чи правильна кількість аргументів у командному рядку. Якщо аргументів не три (програма, файл для копіювання, файл-копія), виводиться повідомлення про правильність заповнення командного рядка і виконання програми завершується.

**if( argc < 3 ) {**

**printf( "\nUsage:"**

**" COPY2 [d:][\\path]source\_file [d:][\\path]target\_file\n" );**

**exit( 1 );**

**}**

Відкриваємо файл, який варто копіювати, для читання.

**if( ( stream\_from = fopen( argv[ 1 ], "rt" ) ) == NULL ) {**

**printf( "\nOpen source file error: %d", errno );**

**exit( 1 );**

**}**

Відкриваємо файл, в який варто копіювати, для запису(якщо файлу не існує, він створюється, якщо існує - перезаписується).

**stream\_to = fopen( argv[ 2 ], "wt+" );**

Вмикаємо тактовий лічильник, і його стартове значення присвоюємо змінній start.

**start = clock();**

Викликаємо функцію для копіювання і передаємо їй покажчик на файл який варто копіювати і на файл до якого необхідно копіювати.

**filecpy( stream\_from, stream\_to );**

Вимикаємо тактовий лічильник, і його фінальне значення присвоюємо змінній end.

**end = clock();**

Друкуємо інформацію про затрачений час на виконання функції і про розмір буферу який довелося використати.

**printf( "Copying time is %5.1f. Buffer size is %d bytes\n",**

**( ( float )end - start ) / CLK\_TCK, BUFSIZ );**

Відкриваємо файл, який варто копіювати, та файл до якого варто копіювати.

**if( ( stream\_from = fopen( argv[ 1 ], "rt") ) == NULL )**

**exit( 1 );**

**stream\_to = fopen( argv[ 2 ], "wt+" );**

Задаємо буфери за допомогою функції setvbuf(), які будуть використовуватися у якості буферів потоків для операцій вводу/виводу. Причому для буферизації будемо використовувати повний об’єм буфера, оскільки використовується режим \_IOFBF.

**setvbuf( stream\_from, buf1, \_IOFBF, sizeof( buf1 ) );**

**setvbuf( stream\_to, buf2, \_IOFBF, sizeof( buf2 ) );**

Викликаємо функцію для копіювання, при цьому підраховуємо затрачений на її виклик час та використану пам'ять.

**start = clock();**

**filecpy( stream\_from, stream\_to );**

**end = clock();**

**printf( "Copying time is %5.1f. Buffer size is %d bytes\n",**

**( ( float )end - start ) / CLK\_TCK, BUFSIZ \* 10 );**

Відкриваємо файл, який варто копіювати, та файл до якого варто копіювати.

**if( ( stream\_from = fopen( argv[ 1 ], "rt") ) == NULL )**

**exit( 1 );**

**stream\_to = fopen( argv[ 2 ], "wt+" );**

Задаємо буфери. Але потіки не буферизується, оскільки використовується режим \_**IONBF**.

**setvbuf( stream\_from, NULL, \_IONBF, 0 );**

**setvbuf( stream\_to, NULL, \_IONBF, 0 );**

Викликаємо функцію для копіювання, при цьому підраховуємо затрачений на її виклик час та використану пам'ять.

**start = clock();**

**filecpy( stream\_from, stream\_to );**

**end = clock();**

**printf( "Copying time is %5.1f. Buffers is not used\n",**

**( ( float )end - start ) / CLK\_TCK );**

**exit( 0 );**

**}**

Функція копіювання файлів з використанням потоку вводу/виводу.

Аргументи: FILE \*stream\_from - покажчик на потік звідки копіювати;

FILE \*stream\_to - покажчик на потік в який копіювати;

Функція нічого не повертає і не використовує глобальних змінних.

**void filecpy( FILE \*stream\_from, FILE \*stream\_to ) {**

Оголошуємо символьний масив на 256 знаків

**char linebuf[ 256 ];**

За допомогою оператора while та функції feof() шукаємо кінець потоку з якого копіюємо.

**while( !feof( stream\_from ) ) {**

Використовуючи функцію fgets(), до масиву linebuf заносимо значенням із вихідного потоку stream\_from.

**if( fgets( linebuf, 255, stream\_from ) == NULL ) break;**

Використовуючи функцію fputs(), із масив linebuf виштовхуємо значенням до вхідного потоку stream\_to.

**if( fputs( linebuf, stream\_to ) == EOF ) break;**

**}**

Закриваємо файли.

**fclose( stream\_from );**

**fclose( stream\_to );**

**}**

**3.** **Експерименти з «copy1.c»**

**В програмі copy1.c замінити бінарний режим доступу до обох файлів на текстовий і запустити програму для копіювання досить великого бінарного файлу. Результати експерименту зафіксувати в зошиті. Пояснити причину ефекту**

При зміні з бінарного режиму доступу до обох файлів на текстовий відбувається не повне копіювання (створений файл менший), а лише до символу Ctrl-Z (ASCII 1Ah), адже вважається, що досягнуто кінець файлу (умова EOF).

**В програмі copy1.c замінити бінарний режим доступу до файлу, який записується, на текстовий і запустити програму. Результати експерименту зафіксувати в зошиті. Пояснити причину ефекту.**

При зміні з бінарного режиму доступу до файлу, який записується, на текстовий створюється не достовірна копія (створений файл більший), оскільки відбувається відображення кожного символу який був зчитаний у бінарному режимі. Таким чином пара, наприклад, символів CR LF не перетворюється в один символ нового рядка “\n”, а виводиться окремо як два символи.

**4.** **Експерименти з «copy2.c»**

**В програмі copy2.c знайти оптимальний розмір буферу. Обґрунтувати вибір.**

На мою думку, оптимальним розміром буферу буде розмір кратний числу 512. Це можна поясноит тим, що функція дає виграш в продуктивності при перенесенні відразу цілої групи байтів (блоку) за одне звернення до функції. Максимальний виграш досягається тоді, коли розмір блоку який переноситься з програми в файл кратний розміру сектора диску, а саме 512 байт.

**В програмі copy2.c модифікувати програму так, щоб вона виводила на екран вміст файлу за допомогою функцій puts(), fputs(), printf(), fwrite(). Результати експерименту зафіксувати в зошиті. Пояснити причину ефекту.**

При використанні функцій виведення на екран, відбувається виведення не екран інформації з вихідного файлу у тому вигляді, який передбачений відповідною функцією. При цьому затрачається більше часу на виконання програми.

**Контрольні запитання**

* 1. **Чим відрізняється текстовий режим доступу до файлу від бінарного?**

Як для потокових так і для префіксних функцій файлового введення-виведення можливі два різних режими доступу до файлу: текстовий та бінарний. В текстовому режимі виконується трансляція символів CR LF (0Dh 0Ah). При читанні інформації з файлу в цьому режимі пара символів CR LF перетворюється в один символ нового рядка “\n” а при записі ‑ символ нового рядка перетворюється в пару символів CR LF. Крім того, як тільки з файлу зчитується символ Ctrl-Z (ASCII 1Ah), вважається, що досягнуто кінець файлу (умова EOF). Таким чином, в текстовому режимі не вдається прочитати інформацію, розташовану після символу Ctrl-Z. При виконанні файлового введення-виведення в бінарному режимі жодного перетворення символів не відбувається, а всі вони розглядаються як такі що не мають якогось особливого сенсу. Режим доступу до файлу задається під час відкриття файлу через бібліотечну функцію відкриття або спеціальною зовнішньою змінною \_fmode.

* 1. **Що таке дескриптор(handle) файлу?**

Відкриття файлів виконує функція АН = 3Dh MS-DOS. Пара регістрів DS:DX вказує на ASCIIZ-рядок, що містить специфікацію файлу, що відкривається як регулярний, або ім’я драйверу символьного пристрою. В регістрі AL задається режим відкриття файлу. Значення яке повертається функцією в регістр АХ є ціле число, це префікс (handle) або дескриптор файлу. При всіх наступних операціях доступу до відкритого файлу в MS-DOS для ідентифікації файлу повідомляється тільки префікс.

1. **Чим відрізняється префіксний доступ до файлу від потокового?**

Бібліотечні функції Turbo C для роботи з файлами можна поділити на дві групи: потокові та префіксні. Як потокові, так і префіксні функції звертаються, в принципі, до тих самих викликам функцій MS-DOS. Однак, потокові функції виконують додаткову буферизацію інформації. Це призводить до подвійної буферизації інформації: на рівні бібліотечної функції і на рівні MS-DOS. Префіксні функції не виконують додаткову буферизацію, а відразу звертаються до префіксних функцій MS-DOS.

1. **В яких випадках більше доцільний потоковий доступ? Чому?**

Префіксні функції дають виграш в продуктивності при перенесенні відразу цілої групи байтів (блоку) за одне звернення до функції. Максимальний виграш досягається тоді, коли розмір блоку який переноситься з програми в файл кратний розміру сектора диску (512 байт). Перенесення інформації між файлом і Сі-програмою по символам або по рядкам виявляється більш ефективним при використанні функцій потокового файлового введення-виведення.

# 

# Висновок

Отже, при виконанні даної лабораторної роботи було використано дві програми копіювання файлів «COPY1.С» та «COPY2.С». Програма «COPY1.С» використовує функції роботи з file handles, а «COPY2.С» - функції потокового вводу-виводу. Було проведено повний розбір роботи обох кодів програм, їх тестування, модифікацію та оптимізацію.

Загалом у ході виконання лабораторної роботи були отримані всі необхідні знання та практичні навички для роботи з файлами.

**Список використаної літератури**

1. **Касаткін А.І. Управление ресурсами. - Минск: Вышейшая школа, 1992.**
2. **Касаткін А.І. Системное программирование. - Минск: Вышейшая школа, 1991.**
3. **Власенко О.В., Данильченко О.М., Северин О.О. Системне прогрмамування. Курс лекцій. Частина 1. (бібліотека ЖІТІ)**