О целесообразности привлечения функций стандартной библиотеки C для обработки файлов

Несмотря на всю уникальность возможностей Windows, старый добрый язык C и его стандартная библиотека ANSI C по-прежнему могут с успехом использоваться при решении большинства задач, связанных с обработкой файлов. Кроме того, библиотека C (указание на ее соответствие стандарту ANSI C мы будем часто опускать) содержит большое число очень нужных функций, аналогов которых среди системных вызовов нет. К их числу относятся, например, функции, описанные в заголовочных файлах <string.h>, <stdlib.h> и <signal.h>, а также функции форматированного и символьного ввода/вывода. В то же время, имеются и такие функции, как fopen и fread, описанные в заголовочном файле <stdio.h>, для которых находятся близко соответствующие им системные вызовы.

В каких же случаях при обработке файлов можно обойтись библиотекой C, а в каких необходимо использовать системные вызовы Windows? Тот же вопрос можно задать и в отношении использования потоков (streams) ввода/вывода C++ или системы ввода/вывода, которая предоставляется платформой .NET. Простых ответов на эти вопросы не существует, но если во главу угла поставить переносимость программ на платформы, отличные от Windows, то в тех случаях, когда приложению требуется только обработка файлов, а не, например, управление процессами или другие специфические возможности Windows, предпочтение следует отдавать библиотеке C и потокам ввода/вывода C++. Вместе с тем, многими программистами ранее уже делались попытки выработать рекомендации относительно адекватности использования библиотеки C в тех или иных случаях, и эти же рекомендации должны быть применимы и в отношении Windows. Кроме того, с учетом возможностей расширения функциональности, а также повышения производительности и гибкости программ, обеспечиваемые Windows, нередко оказывается более удобным или даже необходимым не ограничиваться библиотекой C, в чем вы постепенно станете убеждаться. К числу возможностей Windows, не поддерживаемых библиотекой C, относятся блокирование и отображение файлов (необходимое для разделения общих областей памяти), асинхронный ввод/вывод, произвольный доступ к файлам чрезвычайно крупных размеров (4 Гбайт и выше) и взаимодействие между процессами.

В случае простых программ вам будет вполне достаточно использовать функции библиотеки C, предназначенные для работы с файлами. Воспользовавшись библиотекой C, можно написать переносимое приложение даже без изучения Windows, однако возможности выбора при этом будут ограниченными. Так, для повышения производительности программы и упрощения программирования применяется отображение файлов, однако библиотека C такие возможности не предоставляет.

Пример: простое последовательное копирование файла

В следующих разделах приведены примеры коротких программ, реализующих простое последовательное копирование содержимого файла тремя различными способами:

1. С использованием библиотеки С.
2. С использованием Windows.
3. С использованием вспомогательной функции Windows — CopyFile.

Кроме того, что эти примеры дают возможность сопоставить между собой различные модели программирования, они также демонстрируют возможности и ограничения, присущие библиотеке С и Windows. Альтернативные варианты реализации усилят программу, увеличивая ее производительность и повышая гибкость.

Последовательная обработка файлов является простейшей, наиболее распространенной и самой важной из возможностей, обеспечиваемых любой операционной системой, и почти в каждой большой программе хотя бы несколько файлов обязательно подвергаются этому виду обработки. Поэтому простая программа обработки файлов предоставляет прекрасную возможность ознакомиться с Windows и принятыми в ней соглашениями.

Копирование файлов, нередко осуществляемое совместно с обновлением их содержимого, и слияние отсортированных файлов являются распространенными формами последовательной обработки файлов. Примерами других приложений, осуществляющих последовательный доступ к файлам, могут служить компиляторы и инструментальные средства, предназначенные для обработки текста.

Несмотря на концептуальную простоту последовательной обработки файлов, эффективная реализация этого процесса, обеспечивающая оптимальную скорость его выполнения, может оказаться нелегкой задачей. Для этого может потребоваться использование перекрывающегося ввода/вывода, отображения файлов, потоков и других дополнительных методов.

Само по себе копирование файлов не представляет особого интереса, однако сравнение программ не только позволит вам быстро оценить, чем отличаются друг от друга различные системы, но и послужит хорошим предлогом для знакомства с Windows. В последующих примерах реализуется ограниченный вариант одной из команд UNIX — cp, осуществляющей копирование одного файла в другой и требующей задания имен файлов в командной строке. В приведенных программах организована лишь простейшая проверка ошибок, которые могут возникать на стадии выполнения, а существующие файлы просто перезаписываются. Эти и другие недостатки будут учтены в последующих Windows-реализациях этой и других программ. *Примечание.* Реализация программы для UNIX находится на Web-сайте книги.

Копирование файлов с использованием стандартной библиотеки C

Как видно из текста программы 1.1, стандартная библиотека C поддерживает объекты потоков ввода/вывода FILE, которые напоминают, несмотря на меньшую общность, объекты Windows HANDLE, представленные в программе 1.2.

Программа 1.1. cpC: копирование файлов с использованием библиотеки C

```
/* Глава 1. Базовая программа копирования файлов cp. Реализация, использующая библиотеку C. */
/* cp файл1 файл2: Копировать файл1 в файл2. */
#include <stdio.h>
#include <errno.h>
#define BUF_SIZE 256

int main(int  argc, char *argv[]) {
 FILE *in_file,  *out_file;
 char rec [BUF_SIZE];
 size_t bytes_in, bytes_out;
 if (argc != 3) {
  printf("Использование: cpC файл1 файл2\n");
  return 1;
 }
 in_file  = fopen(argv [1], "rb");
 if (in_file  == NULL) {
  perror(argv[1]);
  return 2;
 }
 out_file = fopen(argv [2], "wb");
 if (out_file  == NULL) {
  perror(argv [2]);
  return 3;
 }
 /* Обработать входной файл по одной записи за один раз. */
 while  ((bytes_in = fread(rec, 1, BUF_SIZE, in_file))  > 0) {
  bytes_out = fwrite(rec,  1, bytes_in, out_file);
  if (bytes_out != bytes_in) {
   perror("Неустранимая ошибка записи.");
   return 4;
  }
 }
 fclose (in_file);
 fclose (out_file);
 return 0;
}
```

Этот простой пример может служить наглядной иллюстрацией ряда общепринятых допущений и соглашений программирования, которые не всегда применяются в Windows.

1. Объекты открытых файлов идентифицируются указателями на структуры FILE (в UNIX используются целочисленные дескрипторы файлов). Указателю NULL соответствует несуществующий объект. По сути, указатели являются разновидностью дескрипторов объектов открытых файлов.

2. В вызове функции fopen указывается, каким образом должен обрабатываться файл — как текстовый или как двоичный. В текстовых файлах содержатся специфические для каждой системы последовательности символов, используемых, например, для обозначения конца строки. Во многих системах, включая Windows, в процессе выполнения операций ввода/вывода каждая из таких последовательностей автоматически преобразуется в нулевой символ, который интерпретируется в языке C как метка конца строки, и наоборот. В нашем примере оба файла открываются как двоичные.

3. Диагностика ошибок реализуется с помощью функции perror, которая, в свою очередь, получает информацию относительно природы сбоя, возникающего при вызове функции fopen, из глобальной переменной errno. Вместо этого можно было бы воспользоваться функцией ferror, возвращающей код ошибки, ассоциированный не с системой, а с объектом FILE.

4. Функции fread и fwrite возвращают количество обработанных байтов непосредственно, а не через аргумент, что оказывает существенное влияние на логику организации программы. Неотрицательное возвращаемое значение говорит об успешном выполнении операции чтения, тогда как нулевое — о попытке чтения метки конца файла.

5. Функция fclose может применяться лишь к объектам типа FILE (аналогичное утверждение справедливо и в отношении дескрипторов файлов UNIX).

6. Операции ввода/вывода осуществляются в синхронном режиме, то есть прежде чем программа сможет выполняться дальше, она должна дождаться завершения операции ввода/вывода.

7. Для вывода сообщений об ошибках удобно использовать входящую в библиотеку C функцию ввода/вывода printf, которая даже будет использована в первом примере Windows-программы.

Преимуществом реализации, использующей библиотеку C, является ее переносимость на платформы UNIX, Windows, а также другие системы, которые поддерживают стандарт ANSI C. Кроме того, как показано в приложении B, в том, что касается производительности, вариант, использующий функции ввода/вывода библиотеки C, ничуть не уступает другим вариантам реализации. Тем не менее, в этом случае программы вынуждены ограничиваться синхронными операциями ввода/вывода, хотя

влияние этого ограничения будет несколько ослаблено использованием потоков Windows.

Как и их эквиваленты в UNIX, программы, основанные на функциях для работы с файлами, входящих в библиотеку C, способны выполнять операции произвольного доступа к файлам (с использованием функции fseek или, в случае текстовых файлов, функций fsetpos и fgetpos), но это является уже потолком сложности для функций ввода/вывода стандартной библиотеки C, выше которого они подняться не могут. Вместе с тем, Visual C++ предоставляет нестандартные расширения, способные, например, поддерживать блокирование файлов. Наконец, библиотека C не позволяет управлять средствами защиты файлов.

Резюмируя, можно сделать вывод, что если простой синхронный файловый или консольный ввод/вывод — это все, что вам надо, то для написания переносимых программ, которые будут выполняться под управлением Windows, следует использовать библиотеку C.

Копирование файлов с использованием Windows

В программе 1.2 решается та же задача копирования файлов, но делается это с помощью Windows API, а базовые приемы, стиль и соглашения, иллюстрируемые этой программой, будут использоваться на протяжении всей этой книги.

Программа 1.2. cpW: копирование файлов с использованием Windows, первая реализация

```
/* Глава 1. Базовая программа копирования файлов cp. Реализация, использующая Windows. */
/* cpW файл1 файл2: Копировать файл1 в файл2. */
#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 256

int main (int argc, LPTSTR argv []) {
 HANDLE hIn, hOut;
 DWORD nIn, nOut;
 CHAR Buffer [BUF_SIZE];
 if (argc != 3) {
  printf ("Использование: cpW файл1 файл2\n");
  return 1;
 }
 hIn = CreateFile(argv [1], GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
 if (hIn == INVALID_HANDLE_VALUE) {
    printf("Невозможно открыть входной файл. Ошибка: %x\n", GetLastError());
  return 2;
 }
```

```
    hOut = CreateFile(argv[2], GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hOut == INVALID_HANDLE_VALUE) {
      printf("Невозможно открыть выходной файл. Ошибка: %x\n",
GetLastError());
     return 3;
    }
    while (ReadFile(hIn, Buffer, BUF_SIZE, &nIn, NULL) && nIn > 0) {
     WriteFile(hOut, Buffer, nIn, &nOut, NULL);
     if (nIn != nOut) {
      printf ("Неустранимая ошибка записи: %x\n", GetLastError());
      return 4;
     }
    }
    CloseHandle(hIn);
    CloseHandle(hOut);
    return 0;
   }
```

Этот простой пример иллюстрирует некоторые особенности программирования в среде Windows, к подробному рассмотрению которых мы приступим позже.

1. В программу всегда включается файл <windows.h>, в котором содержатся все необходимые определения функций и типов данных Windows.[10]

2. Все объекты Windows идентифицируются переменными типа Handle, причем для большинства объектов можно использовать одну и ту же общую функцию CloseHandle.

3. Рекомендуется закрывать все ранее открытые дескрипторы, если в необходимость в них отпала, чтобы освободить ресурсы. В то же время, при завершении процессов относящиеся к ним дескрипторы автоматически закрываются ОС, и если не остается ни одного дескриптора, ссылающегося на какой-либо объект, то ОС уничтожает этот объект и освобождает соответствующие ресурсы. *(Примечание.* Как правило, файлы подобным способом не уничтожаются.)

4. Windows определяет многочисленные символические константы и флаги. Обычно они имеют длинные имена, нередко поясняющие назначение данного объекта. В качестве типичного примера можно привести имена INVALID_HANDLE_VALUE и GENERIC_READ.

5. Функции ReadFile и WriteFile возвращают булевские значения, а не количества обработанных байтов, для передачи которых используются аргументы функций. Это определенным образом изменяет логику организации работы циклов.[11] Нулевое значение счетчика байтов указывает на попытку чтения метки конца файла и не считается ошибкой.

6. Функция GetLastError позволяет получать в любой точке программы коды системных ошибок, представляемые значениями типа

DWORD. В программе 1.2 показано, как организовать вывод генерируемых Windows текстовых сообщений об ошибках.

7. Windows NT и более старшие версии обладают более мощной системой защиты файлов. В данном примере защита выходного файла не обеспечивается.

8. Такие функции, как CreateFile, обладают богатым набором дополнительных параметров, но в данном примере использованы значения по умолчанию.

Копирование файлов с использованием вспомогательной функции Windows

Для повышения удобства работы в Windows предусмотрено множество вспомогательных функций (convenience functions), которые, объединяя в себе несколько других функций, обеспечивают выполнение часто встречающихся задач программирования. В некоторых случаях использование этих функций может приводить к повышению производительности (см. приложение В). Например, благодаря применению функции CopyFile значительно упрощается программа копирования файлов (программа 1.3). Помимо всего прочего, это избавляет нас от необходимости заботиться о буфере, размер которого в двух предыдущих программах произвольно устанавливался равным 256.

Программа1.3.cpCF: копирование файлов с использованием вспомогательной функции Windows

```
/* Глава 1. Базовая программа копирования файлов ср. Реализация, в
которой для повышения удобства использования и производительности
программы используется функция Windows CopyFile. */
/* cpCF файл1 файл2: Копировать файл1 в файл2. */
#include <windows.h>
#include <stdio.h>

int main (int argc, LPTSTR argv []) {
 if (argc != 3) {
  printf ("Использование: cpCF файл1 файл2\n");
  return 1;
 }
 if (!CopyFile(argv[1], argv[2], FALSE)) {
    printf("Ошибка при выполнении функции CopyFile: %x\n",
GetLastError());
  return 2;
 }
 return 0;
}
```

Резюме

Ознакомительные примеры, в качестве которых были использованы три простые программы копирования файлов, демонстрируют многие из отличий, существующих между программами, в которых применяется с одной стороны библиотека C, а с другой — Windows. Отличия в производительности различных вариантов реализации анализируются в приложении B. Примеры, в которых используется Windows, наглядно демонстрируют стиль программирования и некоторые соглашения, принятые в Windows, но дают лишь отдаленное представление о тех функциональных возможностях, которые Windows предлагает программистам.