Книга: Язык программирования Си. Издание 3-е, исправленное (https://wm-help.net/lib/b/book/1357034433/)

8.6 Пример. Печать каталогов

8.6 Пример. Печать каталогов

При разного рода взаимодействиях с файловой системой иногда требуется получить *только* информацию о файле, а не его содержимое. Такая потребность возникает, например, в программе печати каталога файлов, работающей аналогично команде *ls* системы UNIX. Она печатает имена файлов каталога и по желанию пользователя другую дополнительную информацию (размеры, права доступа и т. д.). Аналогичной командой в MS-DOS является *dir*.

Так как в системе UNIX каталог - это тоже файл, функции *ls*, чтобы добраться до имен файлов, нужно только его прочитать. Но чтобы получить другую информацию о файле (например узнать его размер), необходимо выполнить системный вызов. В других системах (в MS-DOS, например) системным вызовом приходится пользоваться даже для получения доступа к именам файлов. Наша цель - обеспечить доступ к информации по возможности системно-независимым способом несмотря на то, что реализация может быть существенно системно-зависима. Проиллюстрируем сказанное написанием программы *fsize*. Функция *fsize* - частный случай программы *ls*: она печатает размеры всех файлов, перечисленных в командной строке. Если какой-либо из файлов сам является каталогом, то, чтобы получить информацию о нем, *fsize* обращается сама к себе. Если аргументов в командной строке нет, то обрабатывается текущий каталог.

Для начала вспомним структуру файловой системы в UNIXe. *Каталог* - это файл, содержащий список имен файлов и некоторую информацию о том, где они расположены. "Место расположения" - это индекс, обеспечивающий доступ в другую таблицу, называемую "списком узлов **inode**". Для каждого файла имеется свой *inode*, где собрана вся информация о файле, за исключением его имени. Каждый элемент каталога состоит из двух частей: из имени файла и номера узла *inode*.

К сожалению, формат и точное содержимое каталога не одинаковы в разных версиях системы. Поэтому, чтобы переносимую компоненту отделить от непереносимой, разобьем нашу задачу на две. Внешний уровень определяет структуру, названную *Dirent*, и три подпрограммы *opendir*, *readdir* и *closedir*: в результате обеспечивается системно-независимый доступ к имени и номеру узла *inode* каждого элемента каталога. Мы будем писать программу *fsize*, рассчитывая на такой интерфейс, а затем покажем, как реализовать указанные функции для систем, использующих ту же структуру каталога, что и Version 7 и System V UNIX. Другие варианты оставим для упражнений.

Структура *Dirent* содержит номер узла *inode* и имя. Максимальная длина имени файла равна NAME_MAX - это значение системно-зависимо. Функция *opendir* возвращает указатель на структуру, названную *DIR* (по аналогии с *FILE*), которая используется функциями *readdir* и *closedir*. Эта информация сосредоточена в заголовочном файле *dirent.h*.

```
#define NAME_MAX 14 /* максимальная длина имени файла */
 /* системно-зависимая величина */
typedef struct { /* универс. структура элемента каталога: */
 long ino; /* номер inode */
 char name[NAME_MAX+1]; /* имя + завершающий '' */
} Dirent;
typedef struct { /* минимальный DIR: без буферизации и т.д. */
 int fd; /* файловый дескриптор каталога */
 Dirent d; /* элемент каталога */
} DIR;
DIR *opendir(char *dirname);
Dirent *readdir(DIR *dfd);
void closedir(DIR *dfd);
```

Системный вызов *stat* получает имя файла и возвращает полную о нем информацию, содержащуюся в узле *inode*, или -1 в случае ошибки. Так,

```
 char *name;
struct stat stbuf;
int stat(char *, struct stat *);
stat(name, &stbuf);
```

заполняет структуру *stbuf* информацией из узла *inode* о файле с именем *name*. Структура, описывающая возвращаемое функцией *stat* значение находится в ‹sys/stat.h› и выглядит примерно так:

```
 struct stat /* информация из inode, возвращаемая stat */
{
 dev_t st_dev; /* устройство */
 ino_t st_ino; /* номер inode */
 short st_mode; /* режимные биты */
 short st_nlink; /* число связей с файлом */
 short st_uid; /* имя пользователя-собственника */
 short st_gid; /* имя группы собственника */
 dev_t st_rdev; /* для специальных файлов */
 off_t st_size; /* размер файла в символах */
 time_t st_atime; /* время последнего использования */
 time_t st_mtime; /* время последней модификации */
 time_t st_ctime; /* время последнего изменения inode */
};
```

Большинство этих значений объясняется в комментариях. Типы, подобные *dev_t* и *ino_t*, определены в файле ‹sys/types.h›, который тоже нужно включить посредством #*include*.

Элемент *st_mode* содержит набор флажков, составляющих дополнительную информацию о файле. Определения флажков также содержатся в ‹sys/stat.h› нам

потребуется только та его часть, которая имеет дело с типом файла

```
 #define S_IFMT  0160000 /* тип файла */
#define S_IFDIR 0040000 /* каталог */
#define S_IFCHR 0020000 /* символьно-ориентированный */
#define S_IFBLK 0060000 /* блочно-ориентированный */
#define S_IFREG 0100000 /* обычный */
```

Теперь мы готовы приступить к написанию программы *fsize*. Если режимные биты (*st_mode*), полученные от *stat*, указывают, что файл не является каталогом, то можно взять его размер (*st_size*) и напечатать. Однако если файл - каталог, то мы должны обработать все его файлы, каждый из которых в свою очередь может быть каталогом. Обработка каталога - процесс рекурсивный.

Программа *main* просматривает параметры командной строки, передавая каждый аргумент функции *fsize*.

```
 #include ‹stdio.h›
#include ‹string.h›
#include "syscalls.h"
#include ‹fcntl.h› /* флажки чтения и записи */
#include ‹sys/types.h› /* определения типов */
#include ‹sys/stat.h› /* структура, возвращаемая stat */
#include "dirent.h"
void fsize(char *);
/* печатает размер файлов */
main(int argc, char **argv) {
 if (argc == 1) /* по умолчанию берется текущий каталог */
  fsize(".");
 else
  while (--argc › 0)
   fsize(*++argv);
 return 0;
}
```

Функция *fsize* печатает размер файла. Однако, если файл - каталог, она сначала вызывает *dirwalk*, чтобы обработать все его файлы. Обратите внимание на то, как используются имена флажков *S_IFMT* и *S_IFDIR* из ‹sys/stat.h› при проверке, является ли файл каталогом. Здесь нужны скобки, поскольку приоритет оператора & ниже приоритета оператора ==.

```
 int stat(char *, struct stat *);
void dirwalk(char *, void (*fcn)(char *));
/* fsize: печатает размер файла "name" */
void fsize(char *name)
{
 struct stat stbuf;
 if (stat(name, &stbuf) == -1) {
  fprintf(stderr, "fsize: нет доступа к %sn", name);
  return;
 }
 if ((stbuf.st_mode & S_IFMT) == S_IFDIR)
  dirwalk(name, fsize);
 printf("%8ld%sn", stbuf.st_size, name);
}
```

Функция *dirwalk* - это универсальная программа, применяющая некоторую функцию к каждому файлу каталога. Она открывает каталог, с помощью цикла перебирает содержащиеся в нем файлы, применяя к каждому из них указанную функцию, затем закрывает каталог и осуществляет возврат. Так как *fsize* вызывает *dirwalk* на каждом каталоге, в этих двух функциях заложена косвенная рекурсия.

```
 #define MAX_PATH 1024
/* dirwalk: применяет fcn ко всем файлам из dir */
void dirwalk(char *dir, void (*fcn)(char *))
{
 char name[MAX_PATH];
 Dirent *dp;
 DIR *dfd;
 if ((dfd = opendir(dir)) == NULL) {
  fprintf(stderr, "dirwalk: не могу открыть %sn", dir);
  return;
 }
 while ((dp = readdir(dfd)) != NULL) {
  if (strcmp(dp-›name, ".") == 0 || strcmp(dp-›name, "…") == 0)
   continue; /* пропустить себя и родителя */
  if (strlen(dir)+strlen(dp-›name) + 2 › sizeof(name))
   fprintf(stderr, "dirwalk: слишком длинное имя %s/%sn", dir, dp-›name);
  else {
   sprintf(name, "%s/%s", dir, dp-›name);
   (*fcn) (name);
  }
 }
 closedir(dfd);
}
```

Каждый вызов *readdir* возвращает указатель на информацию о следующем файле или NULL, если все файлы обработаны. Любой каталог всегда хранит в себе информацию о себе самом в файле под именем "." и о своем родителе в файле под именем "…": их нужно пропустить, иначе программа зациклится. Обратите внимание: код программы этого уровня не зависит от того, как форматированы каталоги. Следующий шаг - представить минимальные версии *opendir*, *readdir* и

closedir для некоторой конкретной системы. Здесь приведены программы для систем Version 7 и System V UNIX. Они используют информацию о каталоге, хранящуюся в заголовочном файле ‹sys/dir.h›, который выглядит следующим образом:

```
 #ifndef DIRSIZ
#define DIRSIZ 14
#endif
struct direct /* элемент каталога */
{
 ino_t d_ino; /* номер inode */
 char d_name[DIRSIZ]; /* длинное имя не имеет '' */
};
```

Некоторые версии системы допускают более длинные имена и имеют более сложную структуру каталога.

Тип **ino_t** задан с помощью *typedef* и описывает индекс списка узлов *node*. В системе, которой пользуемся мы, этот тип есть *unsigned short*, но в других системах он может быть иным, поэтому его лучше определять через *typedef*. Полный набор "системных" типов находится в ‹sys/types.h›.

Функция *opendir* открывает каталог, проверяет, является ли он действительно каталогом (в данном случае это делается с помощью системного вызова *fstat*, который аналогичен *stat*, но применяется к дескриптору файла), запрашивает пространство для структуры каталога и записывает информацию.

```
 int fstat(int fd, struct stat *);
/* opendir: открывает каталог для вызовов readdir */
DIR *opendir(char *dirname)
{
 int fd;
 struct stat stbuf;
 DIR *dp;
 if ((fd = open(dirname, O_RDONLY, 0)) == -1 || fstat(fd, &stbuf) == -1 || (stbuf.st_mode & S_IFMT) != S_IFDIR || (dp = (DIR *)
malloc(sizeof(DIR))) == NULL)
  return NULL;
 dp->fd = fd;
 return dp;
}
```

Функция closedir закрывает каталог и освобождает пространство.

```
 /* closedir: закрывает каталог, открытый opendir */
void closedir(DIR *dp) {
 if (dp) {
  close(dp->fd);
  free(dp);
 }
}
```

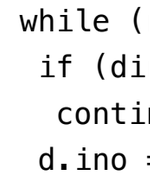
Наконец, *readdir* с помощью *read* читает каждый элемент каталога. Если некий элемент каталога в данный момент не используется (соответствующий ему файл был удален), то номер узла *inode* у него равен нулю, и данная позиция пропускается. В противном случае номер *inode* и имя размещаются в статической (*static*) структуре, и указатель на нее выдается в качестве результата. При каждом следующем обращении новая информация занимает место предыдущей.

```
 #include ‹sys/dir.h› /* место расположения структуры каталога */
/* readdir: последовательно читает элементы каталога */
Dirent *readdir(DIR *dp) {
 struct direct dirbuf; /* структура каталога на данной системе */
 static Dirent d; /* возвращает унифицированную структуру */
 while (read(dp->fd, (char *)&dirbuf, sizeof (dirbuf)) == sizeof(dirbuf)) {
  if (dirbuf.d_ino == 0) /* пустой элемент, не используется */
   continue;
  d.ino = dirbuf.d_ino;
  strncpy(d.name, dirbuf.d_name, DIRSIZ);
  d.name[DIRSIZ] = ''; /* завершающий символ '' */
  return &d;
 }
 return NULL;
}
```

Хотя программа *fsize* - довольно специализированная, она иллюстрирует два важных факта. Первый: многие программы не являются "системными"; они просто используют информацию, которую хранит операционная система. Для таких программ существенно то, что представление информации сосредоточено исключительно в стандартных заголовочных файлах. Программы включают эти файлы, а не держат объявления в себе. Второе наблюдение заключается в том, что при старании системно-зависимым объектам можно создать интерфейсы, которые сами не будут системно-зависимыми. Хорошие тому примеры ~ функции стандартной библиотеки.

Упражнение 8.5. Модифицируйте *fsize* таким образом, чтобы можно было печатать остальную информацию, содержащуюся в узле *inode*.

(https://wm-help.net/lib/b/book/1357034433/100) (https://wm-help.net/lib/b/book/1357034433/102)
Оглавление книги



- 8.6 Пример. Печать каталогов (https://wm-help.net/lib/b/book/1357034433/101)
- 8.7 Пример. Распределитель памяти (https://wm-help.net/lib/b/book/1357034433/102)

Похожие страницы
- Пример установочного скрипта (http://wm-help.net/lib/b/book/1940220047/235)
- Пример из практики (http://wm-help.net/lib/b/book/3346190038/272)
- Заполнение справочников и каталогов (http://wm-help.net/lib/b/book/981810116/57)
- ПРИМЕР ПРОСТОЙ ПРОГРАММЫ НА ЯЗЫКЕ СИ (http://wm-help.net/lib/b/book/1369887060/22)
- Примеры получения статистики (http://wm-help.net/lib/b/book/1940220047/418)
- Пример применения метода «пять почему» (http://wm-help.net/lib/b/book/3711074585/128)
- Пример 12-8. Частота встречаемости отдельных слов (http://wm-help.net/lib/b/book/3933354755/165)
- 1.2.5. Пример программы (http://wm-help.net/lib/b/book/827961078/20)
- 6.1.6. Печать документов (http://wm-help.net/lib/b/book/1759326522/186)
- Пример 17-10. Блочный комментарий (http://wm-help.net/lib/b/book/3933354755/247)
- Примеры (http://wm-help.net/lib/b/book/1369887060/71)
- 2. Пример создания базового отношения в записи на псевдокоде (http://wm-help.net/lib/b/book/4291814739/53)