Лабораторная работа «Дерево каталогов»

UNIX. Профессиональное программирование. У.Ричард Стивенс, Стивен А.Раго

Динамическое выделение памяти для строки пути (Листинг 2.3.)

```
 #include "apue.h"
 #include
 #include
 #ifdef PATH_MAX
static long pathmax = PATH_MAX;
#else static long pathmax = 0;
#endif
static long posix_version = 0;
 static long xsi_version = 0; /* Если константа PATH_MAX не определена */
/* адекватность следующего числа не гарантируется */
 #define PATH_MAX_GUESS 1024
 char * path_alloc(size_t *sizep) /* если удалось выделить память, */
 {
/* возвращает также выделенной объем */
 char *ptr; size_t size;
 if (posix_version == 0) posix_version = sysconf(_SC_VERSION);
 if (xsi_version == 0) xsi_version = sysconf(_SC_XOPEN_VERSION);
if (pathmax == 0) {
    /* первый вызов функции */
    errno = 0;
   if ((pathmax = pathconf("/", _PC_PATH_MAX)) < 0)
     {
       if (errno == 0) pathmax = PATH_MAX_GUESS; /* если константа не определена */
       else err_sys("ошибка вызова pathconf с параметром _PC_PATH_MAX");
     } else
           {
            pathmax++; /* добавить 1, так как путь относительно корня */
            }
```

```
  }
/*
   * До версии POSIX.1-2001 не гарантируется, что PATH_MAX включает
    * завершающий нулевой байт. То же для XPG3.
*/
     if ((posix_version < 200112L) && (xsi_version < 4))
            size = pathmax + 1;
      else size = pathmax;
     if ((ptr = malloc(size)) == NULL)
          err_sys("malloc error for pathname");
    if (sizep != NULL)
         *sizep = size;
     return(ptr);
}
```

Функции семейства stat

Четыре функции семейства stat и информации, которую они возвращают.

```
#include <sys/stat.h>
 int stat(const char *restrict pathname, struct stat *restrict buf);
 int fstat(int fd, struct stat *buf);
 int lstat(const char *restrict pathname, struct stat restrict buf);
 int fstatat(int fd, const char *restrict pathname, struct stat *restrict buf, int flag);
```

SYNOPSIS

```
#include <sys/stat.h>

       int stat(const char *restrict pathname, struct stat *restrict statbuf);
       int fstat(int fd, struct stat *statbuf);
       int lstat(const char *restrict pathname, struct stat *restrict statbuf);

       #include <fcntl.h>  /* Definition of AT_* constants */
       #include <sys/stat.h>

       int fstatat(int dirfd, const char *restrict pathname, struct stat *restrict statbuf, int
 flags);
```

Эти функции возвращают информацию о файле в буфере, на который указывает statbuf. Для самого файла не требуется никаких разрешений, но — в случае stat(), fstatat() и lstat() — разрешение на выполнение (поиск) требуется для всех каталогов в pathname, которые ведут к файлу.

state() и fstatat() извлекают информацию о файле, на который указывает путь; различия для fstatat() описаны ниже.

Функция lstat() идентичена stat(), за исключением того, что если pathname является символической ссылкой, то она возвращает информацию о самой ссылке, а не о файле, на который ссылается ссылка.

Функция fstat() идентична функции stat(), за исключением того, что файл, информация о котором должна быть извлечена, определяется файловым дескриптором fd.

Все четыре функции возвращают 0 в случае успеха, –1 — в случае ошибки.

- Функция **stat()** возвращает структуру с информацией о файле, указанном в аргументе pathname.
- Функция **fstat**() возвращает информацию об открытом файле по его дескриптору fd.
- Функция **lstat**() похожа на функцию stat, но когда ей передается имя символической ссылки, она возвращает сведения о самой символической ссылке, а не о файле, на который она ссылается.
- Функция **fstatat()** возвращает информацию о файле, относительный путь pathname к которому начинается в открытом каталоге, представленном дескриптором fd.

Аргумент flag определяет правила следования по символическим ссылкам: если установлен флаг AT_SYMLINK_NOFOLLOW, функция fstatat не будет следовать по символическим ссылкам, а вернет информацию о самой ссылке. Иначе она будет выполнять переходы и возвращать информацию о файлах, на которые эти ссылки указывают.

Если в аргументе fd передать значение AT_FDCWD, а в аргументе pathname — строку относительного пути, путь к файлу pathname будет откладываться относительно текущего каталога.

Если в аргументе pathname передать строку абсолютного пути, аргумент fd будет игнорироваться. В этих двух случаях fstatat действует подобно stat или lstat, в зависимости от значения аргумента flag.

Второй аргумент, buf, является указателем на структуру, которую функция заполнит информацией.

Определение структуры может отличаться в разных реализациях, но основная ее часть выглядит так:

struct stat

{ mode_t st_mode; /* тип файла и режим (права доступа) */

ino_t st_ino; /* номер индексного узла */

```
dev_t st_dev; /* номер устройства (файловой системы) */
dev_t st_rdev; /* номер устройства для специальных файлов */
nlink_t st_nlink; /* количество ссылок */
uid_t st_uid; /* идентификатор пользователя владельца */
gid_t st_gid; /* идентификатор группы владельца */
off_t st_size; /* размер в байтах, для обычных файлов */
struct timespec st_atim; /* время последнего обращения к файлу */
struct timespec st_mtim; /* время последнего изменения файла */
struct timespec st_ctim; /* время последнего изменения состояния файла */
blksize_t st_blksize; /* оптимальный размер блока ввода/вывода */
blkcnt_t st_blocks; /* количество занятых дисковых блоков */
};
```

The stat structure

All of these system calls return a *stat* structure, which contains the following fields:

```
struct stat {
    dev_t     st_dev;         /* ID of device containing file */
    ino_t     st_ino;         /* Inode number */
    mode_t    st_mode;        /* File type and mode */
    nlink_t   st_nlink;       /* Number of hard links */
    uid_t     st_uid;         /* User ID of owner */
    gid_t     st_gid;         /* Group ID of owner */
    dev_t     st_rdev;        /* Device ID (if special file) */
    off_t     st_size;        /* Total size, in bytes */
    blksize_t st_blksize;     /* Block size for filesystem I/O */
    blkcnt_t  st_blocks;      /* Number of 512B blocks allocated */

    /* Since Linux 2.6, the kernel supports nanosecond
       precision for the following timestamp fields.
       For the details before Linux 2.6, see NOTES. */

    struct timespec st_atim;  /* Time of last access */
    struct timespec st_mtim;  /* Time of last modification */
    struct timespec st_ctim;  /* Time of last status change */

#define st_atime st_atim.tv_sec      /* Backward compatibility */
#define st_mtime st_mtim.tv_sec
#define st_ctime st_ctim.tv_sec
};
```

Стандарт POSIX.1 не требует наличия полей st_rdev, st_blksize и st_blocks. Они определены как расширения XSI в стандарте **Single UNIX Specification**. Структура timespec определяет время в секундах и наносекундах и содержит по меньшей мере два поля: time_t tv_sec; long tv_nsec; До появления редакции стандарта 2008 года поля со значениями времени назывались st_atime, st_mtime и st_ctime и имели тип time_t (время в секундах). Структура timespec позволяет хранить отметки времени (timestamps) с более высокой точностью. Старые имена можно определить в терминах членов tv_sec структуры для совместимости. Например, st_atime можно определить как st_atim.tv_sec.

Обратите внимание, что большинство полей структуры stat имеют элементарный системный тип данных.

Вероятно, наиболее часто функцию stat использует команда ls –l, которая выводит полную информацию о файле.

Чтение каталогов (параграф 4.21 стр. 167)

Прочитать информацию из файла каталога может любой, кто имеет право на чтение этого каталога. Но только ядро может выполнять запись в каталоги, благодаря чему обеспечивается сохранность файловой системы. Известно, что возможность создания и удаления файлов в каталоге определяется битами прав на запись и на выполнение, но это не относится к непосредственной записи в файл каталога. Фактический формат файлов каталогов зависит от реализации UNIX и архитектуры файловой системы.

В ранних версиях UNIX, таких как Version 7, структура каталогов была очень простой — каждая запись имела фиксированную длину 16 байт: 14 байт отводилось для имени файла и 2 байта — для номера индексного узла.

Когда в 4.2BSD была добавлена поддержка более длинных имен файлов, записи стали иметь переменную длину. Это означало, что любая программа, выполняющая прямое чтение данных из файла каталога, попадала в зависимость от конкретной реализации. Чтобы упростить положение дел, был разработан набор функций для работы с каталогами, который стал частью стандарта POSIX.1. Многие реализации не допускают чтения содержимого файлов каталогов с помощью функции read, тем самым препятствуя зависимости приложений от особенностей, присущих конкретной реализации.

#include

DIR *opendir(const char *pathname);

DIR *fdopendir(int fd);

Возвращает указатель в случае успеха или NULL — в случае ошибки

struct dirent *readdir(DIR *dp);

Возвращает указатель в случае успеха, NULL — по достижении конца каталога или в случае ошибки

void rewinddir(DIR *dp); int closedir(DIR *dp);

Возвращает 0 в случае успеха или –1 — в случае ошибки long telldir(DIR *dp); Возвращает значение текущей позиции в каталоге, ассоциированном с dp

```
      void seekdir(DIR *dp, long loc);
```

Листинг. Рекурсивный обход дерева каталогов с подсчетом количества файлов по типам [Стивен,Раго]

```
#include "apue.h"
#include <dirent.h>
#include <limits.h>
/* тип функции, которая будет вызываться для каждого встреченного файла */
 typedef int Myfunc(const char *, const struct stat *, int);
 static Myfunc myfunc;
 static int myftw(char *, Myfunc *);
static int dopath(Myfunc *);
static long nreg, ndir, nblk, nchr, nfifo, nslink, nsock, ntot;
 int main(int argc, char *argv[])
{
     int ret;
    if (argc != 2) err_quit("Использование: ftw ");
     ret = myftw(argv[1], myfunc); /* выполняет всю работу */
    ntot = nreg + ndir + nblk + nchr + nfifo + nslink + nsock;
     if (ntot == 0) ntot = 1;/* во избежание деления на 0 вывести 0 для всех счетчиков */
    printf("обычные файлы = %7ld, %5.2f %%\n", nreg, nreg*100.0/ntot);
    printf("каталоги = %7ld, %5.2f %%\n", ndir, ndir*100.0/ntot);
    printf("специальные файлы блочных устройств = %7ld, %5.2f %%\n", nblk,
    nblk*100.0/ntot); printf("специальные файлы символьных устройств = %7ld, %5.2f
    %%\n", nchr, nchr*100.0/ntot); printf("FIFO = %7ld, %5.2f %%\n", nfifo, nfifo*100.0/ntot);
    printf("символические ссылки = %7ld, %5.2f %%\n", nslink, nslink*100.0/ntot);
    printf("сокеты = %7ld, %5.2f %%\n", nsock, nsock*100.0/ntot); exit(ret);
}
 /*
    * Выполняет обход дерева каталогов, начиная с каталога "pathname".
    * Для каждого встреченного файла вызывает пользовательскую функцию func().
 */
 #define FTW_F 1 /* файл, не являющийся каталогом */
 #define FTW_D 2 /* каталог */
```

```
#define FTW_DNR 3 /* каталог, который недоступен для чтения */
#define FTW_NS 4 /* файл, информацию о котором */
/* невозможно получить с помощью stat */
static char *fullpath; /* полный путь к каждому из файлов */
static size_t pathlen; 182
static int  myftw(char *pathname, Myfunc *func) /* возвращает то, что вернула функция
func() */
{
   fullpath = path_alloc(&len); /* выделить память для PATH_MAX+1 байт */
                                        /* (Стивенс, Стивен: листинг 2.3 стр. 85) */
  if (pathlen <= strlen(pathname))
    {
      pathlen = strlen(pathname) * 2;
     if ((fullpath = realloc(fullpath, pathlen)) == NULL)
        err_sys("ошибка вызова realloc");
    }
  strcpy(fullpath, pathname);
  return(dopath(func));
}
/*
   * Выполняет обход дерева каталогов, начиная с "fullpath".
   * Если "fullpath" не является каталогом, для него вызывается lstat(),
   * func() и затем выполняется возврат.
   * Для каталогов производится рекурсивный вызов функции.
*/
static int dopath(Myfunc* func) /* возвращает то, что вернула функция func() */
{
  struct stat statbuf;
   struct dirent *dirp;
  DIR *dp; int ret, n;
  if (lstat(fullpath, &statbuf) < 0) /* ошибка вызова функции stat */
  return(func(fullpath, &statbuf, FTW_NS));
```

```
  if (S_ISDIR(statbuf.st_mode) == 0) /* не каталог */
  return(func(fullpath, &statbuf, FTW_F));
/*
 * Это каталог. Сначала вызвать функцию func(),
 * а затем обработать все файлы в этом каталоге.
*/
 if ((ret = func(fullpath, &statbuf, FTW_D)) != 0)
        return(ret);
n = strlen(fullpath);
 if (n + NAME_MAX + 2 > pathlen)
 {
    /* увеличить размер буфера */
     pathlen *= 2;
    if ((fullpath = realloc(fullpath, pathlen)) == NULL)
    err_sys("ошибка вызова realloc");
}
   fullpath[n++] = '/';
   fullpath[n] = 0;
    if ((dp = opendir(fullpath)) = = NULL) /* каталог недоступен */
    return(func(fullpath, &statbuf, FTW_DNR));
 while ((dirp = readdir(dp)) != NULL)
  {
     if (strcmp(dirp->d_name, ".") == 0 || strcmp(dirp->d_name, "..") == 0)
            continue;  /* пропустить каталоги "." и ".." */
    strcpy(&fullpath[n], dirp->d_name); /* добавить имя после слеша */
     if ((ret = dopath(func)) != 0) /* рекурсия */
     break; /* выход по ошибке */
   }
fullpath[n-1] = 0; /* стереть часть строки от слеша и до конца */
if (closedir(dp) < 0)  err_ret("невозможно закрыть каталог %s", fullpath);
return(ret);
}
```

```
static int myfunc(const char *pathname, const struct stat *statptr, int type)
 {
switch (type) {
   case FTW_F:
    switch (statptr->st_mode & S_IFMT)
     {
        case S_IFREG: nreg++; break;
        case S_IFBLK: nblk++; break;
        case S_IFCHR: nchr++; break;
        case S_IFIFO: nfifo++; break;
        case S_IFLNK: nslink++; break;
        case S_IFSOCK: nsock++; break;
        case S_IFDIR: /* каталоги должны иметь type = FTW_D*/
                        err_dump("признак S_IFDIR для %s", pathname);
    } break;
  case FTW_D: ndir++; break;
  case FTW_DNR:  err_ret("закрыт доступ к каталогу %s", pathname); break;
  case FTW_NS: err_ret("ошибка вызова функции stat для %s", pathname); break;
  default:  err_dump("неизвестный тип %d для файла %s", type, pathname);
 }
 return(0);
}
```

Функция chdir

chdir(2) System Calls Manual chdir(2)

NAME top

chdir, fchdir - change working directory

LIBRARY top

Standard C library (libc, -lc)

SYNOPSIS top

```
#include <unistd.h>

int chdir(const char *path);
int fchdir(int fd);
```

Feature Test Macro Requirements for glibc (see

feature_test_macros(7)):

fchdir():

_XOPEN_SOURCE >= 500

|| /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L

|| /* glibc up to and including 2.19: */ _BSD_SOURCE

DESCRIPTION top

chdir() changes the current working directory of the calling process to the directory specified in path.

fchdir() is identical to chdir(); the only difference is that the directory is given as an open file descriptor.

RETURN VALUE top

On success, zero is returned. On error, -1 is returned, and

errno is set to indicate the error.

Задание:

Написать программу, выводящую на экран дерево каталогов.

Приведенная версия функции myftw действует только в текущем каталоге. Необходимо изменить функцию таким образом, чтобы всякий раз, когда встречается каталог функция myftw вызывала функцию **chdir() для перехода в этот каталог.** Это делается для того, чтобы передавать функции lstat() только короткое имя файла. После обработки всех файлов в каталоге необходимо вызвать chdir("..").