Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный политехнический университет»

Институт информационных технологий и управления
Кафедра «Распределенные вычисления и компьютерные сети»

КУРСОВАЯ РАБОТА

Разработка программ для управления файлами и каталогами по дисциплине «Операционные системы»

Выполнил _____ В.Б. Борисов
студент гр. 23507/1

Руководитель _____ Д.А. Тимофеев
ст. преп.

Санкт-Петербург
2015

Оглавление

Введение

Файлы в Linux играют ключевую роль. Все данные пользователей хранятся в виде файлов. Классическая файловая система представляет данные в виде вложенных друг в друга каталогов, в которых содержатся файлы. Один из каталогов является «вершиной» файловой системы («корень»), в нем содержатся все остальные каталоги и файлы. Помимо этого, файлы в Linux определяют привилегии пользователей, поскольку права пользователя в большинстве случаев контролируются с помощью прав доступа к файлам. Файлы обеспечивают доступ к периферийным устройствам компьютера, включая диски, CD-ROM, принтеры, терминалы, сетевые адаптеры и даже память. Все программы, которые выполняются в системе, включая прикладные задачи пользователей, системные процессы и даже ядро Linux, являются исполняемыми файлами. В Linux все доступное пользователям файловое пространство объединено в единое дерево каталогов, корнем которого является каталог "/". В большинстве случаев единое дерево, такое каким его видит пользователь системы, составлено из нескольких отдельных файловых систем, которые могут иметь различную внутреннюю структуру, а файлы, принадлежащие этим файловым системам, могут быть расположены на различных устройствах.

Каждый файл имеет связанные с ним метаданные (хранящиеся в индексных дескрипторах - inode), содержащие все характеристики файла и позволяющие операционной системе выполнять операции, заказанные прикладной задачей: открыть файл, прочитать или записать данные, создать, удалить или копировать (в том числе рекурсивно) файл. Например, для копирования файлов используют команду «cp», а для получения метаданных команду «stat». В частности, метаданные содержат указатели на дисковые блоки хранения данных файла. Имя файла в файловой системе является указателем на его метаданные, в то время как метаданные не содержат указателя на имя файла.

Обычный файл представляет собой наиболее общий тип файлов, содержащий данные в некотором формате. Для операционной системы такие файлы представляют собой просто последовательность байтов. Вся интерпретация содержимого файла производится прикладной программой, обрабатывающей файл. К этим файлам относятся текстовые файлы, бинарные данные, исполняемые программы и т.п. Каталог - это файл, содержащий имена находящихся в нем файлов, а также указатели на дополнительную информацию - метаданные, позволяющие операционной системе

производить операции над этими файлами. Каталоги определяют положение файла в дереве файловой системы, поскольку сам файл не содержит информации о своем местонахождении. Любая задача, имеющая право на чтение каталога, может прочесть его содержимое, но только ядро имеет право на запись в каталог. По существу, каталог представляет собой таблицу, каждая запись которой соответствует некоторому файлу. Первое поле каждой записи содержит указатель на метаданные (номер inode), а второе определяет имя файла. Специальный файл устройства обеспечивает доступ к физическому устройству. В Linux различают символьные (character) и блочные (block) файлы устройств. Доступ к устройствам осуществляется путем открытия, чтения и записи в специальный файл устройства.

Основная часть

Постановка задачи

В данной курсовой работе необходимо следующее:

- Изучить структуру проекта Xv6, научиться компилировать и запускать операционную систему в виртуальной машине QEMU, модифицировать примеры прикладных программ и выполнять их в среде Xv6.
- Научиться отлаживать ядро Xv6 с помощью отладчика GDB и виртуальной машины QEMU.
- Написать новые пользовательские программы для управления файлами и каталогами:
    1. **cp** – копирование файлов с поддержкой рекурсивного копирования каталогов;
    2. **stat** – печать метаданных файла (тип, размер, количество ссылок).

Описание реализации

I. Установка

Перед тем как приступить к выполнению задания мной была установлена свободная программа для эмуляции аппаратного обеспечения различных платформ - QEMU в среде GNU/Linux. Сделано это было с помощью команды: «sudo apt-get install qemu».

Потом была скачана операционная система Xv6 с помощью git командой: «git clone git://pdos.csail.mit.edu/xv6/xv6.git».

Для компилирования Xv6 я использовал стандартную команду «make», а для сборки ядра «make qemu».

Таким образом, проект скомпилировался и появилось диалоговое окно QEMU, ожидающее команды для Xv6.

II. Написание программ

Далее мной были написаны программы **stat** и **cp**, которые были перенесены в папку Xv6
a) stat — программа отображения характеристики необходимого файла.
Программа **stat** должна получать на вход название одного файла, доступного для системы.

- В случае ввода нескольких названий файлов, либо наоборот, не ввода названия программа выдаст ошибку: «`stat: incorrectly set parameters`»
- В случае, если программа не сможет открыть файл, она выдаст ошибку: «`stat: cannot open`»
- Если по какой-либо причине программа не сможет получить данные файла, она выдаст ошибку: «`stat: cannot stat`»

В случае правильных входных параметров программа выведет на экран все данные файла, описанные структуре stat в файле stat.h, а именно:

- Type (file, directory or special device)
- File system's disk device
- Inode number

- Number of links fo file
- Size (bytes)

б) cp — программа копирования файлов с поддержкой рекурсивного копирования каталогов. Программа должна получать на вход название копируемого файла и название нового файла для сохранения копии.
Функции, реализованные в **cp**:

- *void copy(char *frst, char *scnd)* — функция копирования файлов и каталогов, включающая в себя обработку ошибок, связанных с открытием файлов.
- *int copyFile(int fdr, int fdw)* — функция копирования файла, с использованием функций *read(int a, char b, int c)* и *write(int a, char b, int c)*, которая вызывается через функцию *copy(char *frst, char *scnd).*

Обработка ошибок в программе **cp:**

- при получении неверно заданных входных параметров программа выведет на экран следующее сообщение об ошибке: «cp: incorrectly set parameters»
- при ошибке открытия любого из двух файлов программа выведет на экран следующее сообщение об ошибке: «cp: fail of the opening» [Название файла]
- при ошибке получения характеристики о первом входном параметре программа выведет на экран следующее сообщение об ошибке: «cp: fail of the review statistic»
- в случае, если копируемый файл не является директорией, а второй входной параметр — это директория, программы выведет следующее сообщение: [название файла] is not a directory
- в случае, если при копировании файла произошла ошибка, программа выведет следующее сообщение об ошибке: «cp: fail of the opening»

Команды, использованные для компиляции файлов stat.c и cp.c:

```
gcc stat.c -o stat

gcc cp.c -o cp
```

Для их подключения к операционной системе необходимо в файле Makefile в пункте UPROGS=\ прописать *_stat* и *_cp.*

Тестирование

Вначале мы в консоли с помощью команды «**cd**» xv6-r1 заходим в папку где расположены различные объекты, которые мы можем увидеть с помощью «**ls**».

Затем делаем компиляцию «**make**», потом отладку ядра «**make qemu**» и просматриваем содержимое директории «**ls**». Отныне все тестирование пройдет здесь.

Тест 1.

Скопируем файл **cat** в новый файл **cat1**. Для этого используем команду

```
cp cat cat1
```

Затем сравним их параметры с помощью команд

```
stat cat
stat cat1
```

Из сравнения характеристик видно, что файлы отличаются только по номеру в системе, а значит функция сработала верно. Тест 1 пройден.

Тест 2.

Скопируем файл **wc** в уже существующий файл **cat1**. Для этого используем команду

```
cp wc cat1
```

Затем вновь сравним параметры с помощью команд

```
stat wc
stat cat1
```

Из результатов просмотра характеристики видно, что программа вновь сработала верно. Тест 2 пройден.

Тест 3.

Попробуем ввести неверные параметры cp и stat

Видно, что программы сработали верно, в соответствии с их спецификацией. Тест 3 пройден.

Тест 4.

Протестируем рекурсивное копирование директорий: создадим директорию командами

```
mkdir dirA
mkdir dirA/dirB
mkdir dirA/dirC
mkdir dirA/dirB/dirD
```

Затем скопируем **dirA** в ещё не созданную **dirZ** с помощью команды

```
cp dirA dirZ
```

А затем сравним их статистику и вложенные папки в **dirA** и **dirZ**:

Как видно, по характеристике обе директории одинаковые, а также вложенные в них поддиректории совпадают. Тест 4 пройден

Тест 5.

Попробуем скопировать директорию в файл. Для этого создадим директорию **dir** и файл **file** путем копирования в него файла **cat**. После этого выполним команду

```
cp dir file
```

Из результатов теста видно, что программа работает верно, в соответствии со своей спецификацией. Тест 5 пройден.

Заключение

В результате проделанной работы можно сделать вывод. Я немного научился работать с Unix-подобной операционной системой, именуемой себя Xv6. Освоил работу с файловой системой и открыл для себя новую операционную систему. Реализовал основные команды для работы с файлами и каталогами. Поставленная задача решена.

Приложение 1 (код программы)

- Программа stat:

```
#include "fcntl.h"
#include "types.h"
#include "stat.h"
#include "user.h"
int main(int argc, char *argv[]){
   struct stat s; // с т р у к т у р а  о п и с а н н а я  в stat.h
   if (argc != 2) // п р о в е р к а  а р г у м е н т о в  д л я stat: stat
[н а з в а н и е]
      printf(0,"stat: incorrectly set parameters\n");
   else{
      int i = open(argv[1], O_RDONLY); // о т к р ы т и е  ф а й л а
 // д а л е е  о б р а б о т к а  о ш и б о к
if (i < 0)
         printf(0,"stat: cannot open %s\n",argv[1]);
      else
         if (fstat(i, &s) < 0)
            printf(2,"stat: cannot stat %s\n",argv[1]);
         else{
            printf(0," Type: ");
            if (s.type == T_DEV)
               printf(0,"special device: %d", s.type );
            else
               if (s.type == T_DIR)
                  printf(0,"directory: %d", s.type );
               else
                    printf(0,"file: %d", s.type );
                 // в ы в о д  м е т а д а н н ы х
               printf(0,"\n File system's disk device: %d\n",s.dev);
               printf(0," Inode number: %d\n",s.ino);
               printf(0," Number of links fo file: %d\n",s.nlink);
               printf(0," Size: %d",s.size);
               printf(0," bytes\n");
      }
   close(i);
   }
exit();
}
```

- Программа cp:

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
#include "fs.h"
int copyFile(int fdr, int fdw) // функция копирования с исп-м
read and write

{
char buf[512];
   int r,w;
   while ((r = read(fdr, buf, sizeof(buf))) > 0)
   {
      w = write(fdw, buf, r);
      if (w != r || w < 0)
         return -1;
   }
   if (r < 0)
      return -1;
   else
      return 0;
}
void copy(char *frst, char *scnd) // функция копирование файлов
и каталогов с обработкой ошибок
{ // fdr - file directory read; fdw - file directory write
   struct stat temp;
   int fdr = open(frst, O_RDONLY), fdw;
   if (fdr < 0)
      printf(0,"cp: fail of the opening %s\n", frst);
   else
   {
      if (fstat(fdr, &temp) < 0)
         printf(0,"cp: fail of the review statistic %s\n", frst);
      else
      {
         if (temp.type != T_DIR)
         {

            fdw = open(scnd, O_CREATE | O_WRONLY);
```

```
   if (fdw < 0)
      printf(1,"cp: fail of the opening %s\n", scnd);
   else
      if (copyFile(fdr,fdw) < 0)
         printf(0,"cp: copy error %s to %s\n", frst, scnd);
      else
         return;
      close(fdw);
}
else
{
   if (mkdir(scnd) < 0)
   {
      fdw = open(scnd, O_RDONLY);
      if (fdw < 0)
         printf(1,"cp: fail of the opening or creating %s\n", scnd);
      else
      {
         fstat(fdw, &temp);
         if(temp.type != T_DIR)
            printf(1,"cp: %s is not a directory\n", scnd);
      }
      close(fdw);
   }
   else
   {
      if(strlen(frst) + 1 + DIRSIZ + 1 > 512)
         printf(1, "(cp: copied path is too long\n");
      else
      {
         if(strlen(scnd) + 1 + DIRSIZ + 1 > 512)
            printf(1, "cp: created path is too long\n");
         else
         {
            char *r, *w;
            struct dirent dirr;
            r = frst + strlen(frst);
            w = scnd + strlen(scnd);
            *r++ = '/';
            *w++ = '/';
            while(read(fdr, &dirr, sizeof(dirr)) == sizeof(dirr))
            {
```

```
                        if(dirr.inum == 0 || strcmp(dirr.name,".") == 0 ||
strcmp(dirr.name,"..") == 0)
                            continue;
                        memmove(r, dirr.name, DIRSIZ);
                        memmove(w, dirr.name, DIRSIZ);
                        r[DIRSIZ] = 0;
                        w[DIRSIZ] = 0;
                        copy(frst, scnd);
                    }
                    close(fdr);
                    return;
                }
            }
        }
    }
  }
  close(fdr);
  exit();
}


int main(int argc, char *argv[])
{
  if(argc != 3) // п р о в е р к а  п а р а м е т р о в  д л я  cp: cp
[н а з в а н и е] [н а з в а н и е]
    printf(0,"cp: incorrectly set parameters\n");
  else
  {
    char frst[512], scnd[512];
    strcpy(frst, argv[1]);
    strcpy(scnd, argv[2]);
    copy(frst, scnd);
  }
  exit();
}
```