МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»
Кафедра 43

КУРСОВОЙ ПРОЕКТ
ЗАЩИЩЕН С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ
ст.преп _____ М.Д.Поляк

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ

Резервное копирование

по дисциплине: ОПЕРАЦИОННЫЕ СИСТЕМЫ

РАБОТУ ВЫПОЛНИЛ
СТУДЕНТК ГР. 4331 _____ Князян.Р.А.

Санкт-Петербург, 2016 г.

Содержание

1. Цель работы

Цель работы: реализовать демон для flash накопителей, осуществляющий резервное копирование данных под ОС Linux

2. Задание

Реализовать демон для flash-накопителя, работающего через интерфейс USB, реализующий копирование файлов между двумя flash-накопителями. При подключении двух flash-накопителей с заданными серийными номерами, демон должен запускать копирование файлов с одного накопителя на другой и сохранять текстовый лог успешно завершенных операций копирования на обоих flash-накопителях.

3. Техническая документация

3.1. Установка

Склонировать репозиторий с github при помощи команды:

```
git clone https://github.com/RubenKnyazyan/Kurs_project.git
```

Для работы демона необходима версия ядра не менее 3.19.

3.2. Использование

1) Сборка проекта:
   Необходимо перейти в корневой каталог репозитория и вызвать команду make. Демон будет скомпилирован.

2) Запуск проекта:
   После того, как демон был скомпилирован, он будет и автоматически запущен

3) Использование демона:
   Перед использованием демона необходимо указать серийные номера флеш-накопителей, путь к каталогу флеш-накопителей. Задаются они в файле deamon.c, (пример)

   ```
   //указываем серийный номер накопителей
   serial1 = (char*)"07AB1608151B5B72";
   serial2 = (char*)"c3f41ad10ff131";
   dir_serial1 = (char*)"/media/ruben/STORE/";
   dir_serial2 = (char*)"/media/ruben/B480-C68C/";;
   ```

   После того, как мы подключили накопитель, в log-файле появится запись о том, что демон начал свою работу. Так же, мы увидим, что в log'е отображаются все скопированные на диск файлы.

4) Выключение и удаление проекта:
   Чтобы выключить демона нам понадобиться диспетчер задач, к примеру htop. Запускаем htop, нажимаем F4, выбираем command, далее вводим ./deamon.o, и мы увидим запущенный процесс нашего демона. Нажимая F9 нам будет предложено "убить"процесс, соглашаемся и жмем Enter. Удаление исполняемого файла демона можно произвести вызвав команду make clean. в корневой директории репозитория.

4. Выводы

В процессе выполнения данной курсовой работы мною были получены знания и навыки, необходимые для работы с USB носителями, потоками, файлами и папками, в ОС семейства Linux, а так же знания и навыки в написании демонов.

5. Приложения

deamon.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/time.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <syslog.h>
#include <fstream>  // to copy files
#include <iostream>
#include <dirent.h> //for searching files
#include <sys/stat.h> // для создания папки
#include <pthread.h>
#include <semaphore.h>


static char* path_log_file = (char*)"/home/ruben/try2/daemon.log";
static char* serial1 = (char*)"c3f41ad10ff131";
static char* dir_serial1 = (char*)"/media/ruben/B480-C68C/";
static char* serial2 = (char*)"07AB1608151B5B72";
static char* dir_serial2 = (char*)"/media/ruben/STORE/";

/*static char* serial1 = (char*)"COUF9LAZ";
static char* dir_serial1 = (char*)"/run/media/alim/5B87-AB6D/";
static char* serial2 = (char*)"07AB1608151B5B72";
static char* dir_serial2 = (char*)"/home/alim/kp1/";*/
// const int NUM_THREADS = 10;1
// pthread_t threads[NUM_THREADS];

pthread_t th;
sem_t sem_name;
sem_t mutex;
const int countSemaphore = 10; //отслеживаем склько потоков созданно

struct structFile {
    std::string fileName;
    std::string pathFrom;
    std::string pathTo;
};

int Daemon(void);
// char* getTime();
int writeLog(char msg[256]);
char* getCommand(char command[128]);
char* getSerial( char *str );
int isSerial(char* ser);
void* copyFile(void *args);
```

```
void copyDir(std::string pathFrom, std::string pathTo);
long fileSize( std::fstream &f );
void createThread(structFile* sFile);
void* funcThread(void *args);

char* getSerial( char *str )
{
    ssize_t len;
    char buf[256];
    char *p;
    char buf2[256];
    int i;
    static char comText[256];
    bzero(comText, 256);
    strcpy(comText, "");

    len = readlink(str, buf, 256);
    if (len <= 0) {
        return (char*)"-";
    }
    buf[len] = '\0';
    // printf("%s\n", buf);/sys/block/sys/block/sdb
    sprintf(buf2, "%s/%s", "/sys/block/", buf);
    // printf("до %s\n", buf2);
    for (i=0; i<6; i++) {
        p = strrchr(buf2, '/');
        *p = 0;
    }
    // printf("после %s\n", buf2);
    strcat(buf2, "/serial");
    // printf("opening %s\n", buf2);

    int f = open(buf2, 0);
    if (f == -1) return (char*)"-";
    len = read(f, buf, 256);
    if (len <= 0) {
        return (char*)"-";
    }
    buf[len-1] = '\0';
    //printf("serial: %s\n", buf);
    strcat(comText, buf);

    return comText;
}

int isSerial(char* ser)
{
    if (strcmp(getSerial((char*)"/sys/block/sdb"), ser) == 0
        || strcmp(getSerial((char*)"/sys/block/sdc"), ser) == 0
        || strcmp(getSerial((char*)"/sys/block/sdd"), ser) == 0
        || strcmp(getSerial((char*)"/sys/block/sdf"), ser) == 0
        || strcmp(getSerial((char*)"/sys/block/sdg"), ser) == 0)
```

```
    {
        return 1;
    }
    return 0;
}

char* getCommand(char command[128]) { //функция возвращает результат выполнения linux ко
    FILE *pCom;
    static char comText[256];
    bzero(comText, 256);
    char  buf[64];
    pCom = popen(command, "r"); //выполняем
    if(pCom == NULL) {
        writeLog((char*)"Error Command");
        return (char*)"";
    }
    strcpy(comText, "");
    while(fgets(buf, 64, pCom) != NULL) { //читаем результат
        strcat(comText, buf);
    }
    pclose(pCom);
    return comText;
}

int writeLog(char* msg)
{ //функция записи строки в лог
    sem_wait(&mutex); //ждем пока семафор освободится
    FILE * pLog;
    pLog = fopen(path_log_file, "a");
    if(pLog == NULL) {
        return 1;
    }
    char str[1024];
    bzero(str, 1024);
    strcat(str, msg);
    strcat(str, (char*)"\n");
    fputs(str, pLog);
    fclose(pLog);
    sem_post(&mutex);
    return 0;
}

int main(int argc, char* argv[]) {
    pid_t parpid, sid;

    parpid = fork(); //создаем дочерний процесс
    if(parpid < 0) {
        exit(1);
    } else if(parpid != 0) {
        exit(0);
    }
    umask(0);//даем права на работу с фс
```

```
    sid = setsid();//генерируем уникальный индекс процесса
    if(sid < 0) {
        exit(1);
    }
    if((chdir("/")) < 0) {//выходим в корень фс
        exit(1);
    }
    close(STDIN_FILENO);//закрываем доступ к стандартным потокам ввода-вывода
    close(STDOUT_FILENO);
    close(STDERR_FILENO);

    return Daemon();
}

int Daemon(void)
{
    int v = 0;
    sem_init(&sem_name, 0, countSemaphore);
    sem_init(&mutex, 0, 1);
    // char *log;
    while(1) //собственно наш бесконечный цикл демона
    {
        // log = getCommand((char*)"who");
        if(isSerial(serial1) && isSerial(serial2)) // && isSerial(serial2)
        {
            copyDir(dir_serial2, dir_serial1);
            writeLog((char*)"\n#############################################################
            while (v != countSemaphore) //ждем пока все семафоры освободятся
            {
                sleep(1);
                sem_getvalue(&sem_name, &v);
            }
        }

        sleep(3);
    }
    return 0;
}

void* copyFile(void *args)
{
    structFile* sFile = (structFile*) args;
    // std::string pathFrom, std::string pathTo, std::string fileName

    std::string way1 = sFile->pathFrom;
    std::string way2 = sFile->pathTo;

    way1 = way1+sFile->fileName;
    way2 = way2+sFile->fileName;

    char myA[sFile->fileName.size()+1];
    strcpy(myA, sFile->fileName.c_str());
```

```
if (strcmp(myA,".") == 0 || strcmp(myA,"..") == 0)
{
    std::cout << "New Branch\n";
}
else
{
    const char * c1 = way1.c_str();
    const char * c2 = way2.c_str();

    char * buffer = new char;

    std::fstream infile(c1, std::ios::in | std::ios::binary);
    if (!infile.is_open()) // если файл не открыт
    {
        std::cout << "Файл не может быть открыт!\n"; // сообщить об этом
    }
    else
    {
        infile.read(buffer, sizeof(char)); //читаем первый сивол
        std::string log = "";

        if (infile.fail() && !infile.eof()) //проверка на папку
        {
            infile.close();
            remove(way2.c_str());
            log = c2;
            log = "dir: " + log;
            mkdir(way2.c_str(), 0755);
            std::cout << way2 << " -- Это папка.\n";
            copyDir(way1 + "/", way2 + "/");
        }
        else { //иначе файл
            std::fstream outfile(c2, std::ios::out | std::ios::app | std::ios::binar
            outfile.seekg(0, std::ios::beg);

            if (fileSize(outfile) != fileSize(infile) && fileSize(infile) > 0) //ecл
            {
                infile.close();
                outfile.close();
                // outfile.open(c2, std::ios::out | std::ios::binary);
                log = c2;
                log = "new file: " + log;
                sem_wait(&sem_name); //ждем пока семафор освободится
                createThread(sFile);
                // funcThread(sFile);
                // while (!infile.eof() && infile.good())
                // {
                //     outfile.write(buffer, sizeof(char));
                //     infile.read(buffer, sizeof(char));
                // }
            } else
```

```
                {
                    log = c2;
                    log = "file exists: " + log;
                }
                infile.close();
                outfile.close();
            }
            writeLog((char *) log.c_str());
        }
        delete buffer;
        std::cout << way1 << "->" << way2 << std::endl;
    }

    return 0;
}

void copyDir(std::string pathFrom, std::string pathTo)
{
    struct dirent **namelist;
    int n;
    n = scandir(pathFrom.c_str(), &namelist, 0, alphasort);
    if (n >= 0)
    {
        while (n--)
        {
            structFile* sFile = new structFile;
            sFile->pathFrom = pathFrom;
            sFile->pathTo = pathTo;
            sFile->fileName = namelist[n]->d_name;
            // createThread(sFile);
            copyFile((void*)sFile);
            free(namelist[n]);
        }
        free(namelist);
    }
}

long fileSize( std::fstream &f )
{
    long size, saveTellg;
    saveTellg = f.tellg();
    f.seekg(0, std::ios::end);
    size = f.tellg();
    f.seekg(saveTellg, std::ios::beg);
    return size;
}

void createThread(structFile* sFile)
{
    int status, status_addr;

    status = pthread_create(&th, NULL, funcThread, (void*) sFile);
```

```
    if (status != 0)
    {
        printf("main error: can't create thread, status = %d\n", status);
        sem_post(&sem_name);
        delete sFile;
        return;
    }

    // status = pthread_join(th, (void**)&status_addr);
    // if (status != 0)
    // {
    //     printf("main error: can't join thread, status = %d\n", status);
    //     return;
    // }
}

void* funcThread(void *args)
{
    structFile* sFile = (structFile*) args;

    std::string way1 = sFile->pathFrom;
    std::string way2 = sFile->pathTo;

    way1 = way1+sFile->fileName;
    way2 = way2+sFile->fileName;

    char myA[sFile->fileName.size()+1];
    strcpy(myA, sFile->fileName.c_str());

    const char * c1 = way1.c_str();
    const char * c2 = way2.c_str();

    char * buffer = new char;

    std::fstream infile(c1, std::ios::in | std::ios::binary);
    std::fstream outfile(c2, std::ios::out | std::ios::binary); //открываем файл для зап

    infile.read(buffer, sizeof(char)); //читаем первый сивол
    while (!infile.eof() && infile.good())
    {
        outfile.write(buffer, sizeof(char));
        infile.read(buffer, sizeof(char));
    }
    delete buffer;
    delete sFile;
    infile.close();
    outfile.close();
    sem_post(&sem_name);
    return 0;
}
```