Московский авиационный институт
(национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу «Операционные системы»**

Студентка: А. Довженко
Преподаватель: Е. С. Миронов
Группа: 08-207
Дата:
Оценка:
Подпись:

**Москва, 2017**

# Лабораторная работа №1

**Задача:** Часть 1. Написать собственную программу, которая демонстрирует работу с различными вызовами (8-15) операционной системы. Произвести диагностику работы написанной программы с помощью утилит ОС, изучив основные принципы применения используемых утилит.

Часть 2. Выбрать стороннее программное обеспечение. Произвести диагностику ПО. Выявить ключевые особенности работы. Выявить предполагаемые ключевые системные вызовы, которые используются в стороннем программном обеспечении.

**Операционная система:** Unix.

# 1 Описание

Системные вызовы – один из уровней абстракции, предоставляемый пользовательским программам. Само управление ресурсами проходит незаметно для пользователя и осуществляется в автоматическом режиме. Любой однопроцессорный компьютер одномоментно может выполнить только одну команду. Когда процесс выполняет пользовательскую программу в режиме пользователя и нуждается в какой-нибудь услуге ОС, он должен выполнить команду системного прерывания, чтобы передать управление ОС. Затем ОС по параметрам вызова определяет, что именно требуется вызывающему процессу. После этого она обрабатывает системный вызов и возвращает управление той команде, которая следует за системным вызовом. В некотором смысле выполнение системного вызова похоже на выполнение особой разновидности вызова процедуры, с той лишь разнцией, что системные вызовы входят в ядро, а процедурные – нет.
Использованные системные вызовы:

- **int creat(const char *pathname, modet mode);** – создает и открывает файловый дескриптор в соответствии с флагами открытия.
- **int open(const char *path, int oflag, ...);** – открытие файлового дескриптора: первый аргумент – путь до файла, второй – флаги открытия. */
- **int read(int fd, void *buffer, int nbyte);** – читает nbyte из файлового дескриптора fd в буффер buffer.
- **int write(int fd, void *buffer, int nbyte);** – записывает количество байтов в 3 аргументе из буфера в файл с дискриптором fd, возвращает количество записанных байтов или -1 в случае ошибки.
- **int close(int fd);** – закрывает файловый дескриптор.
- **int fsync(int fd);** – синхронизирует состояние файла в памяти с его состоянием на диске
- **int truncate(int fd, offt lenght)** – устанавливает длину файла с файловым дескриптором fd в lenght байт. Если файл до этой операции был длинее, то отсеченные данные теряются. Если файл был короче, то он увеличивается, а добавленная часть заполняется нулевыми байтами.
- **pidt fork(void);** – создает дочерний процесс. Если возвращает 0, то созданный процесс – ребенок, если >0, то – родитель.

- **pidt wait(int *status);** – приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик.
- **offt lseek(int fd, offt offset, int whence)** – устанавливает смещение для файлового дескриптора в значение аргумента offset в соответствии с директивой whence, которая может принимать одно из следующих значений: SEEKSET(смещение устанавливается в offset байт от начала файла), SEEKCUR(смещение устанавливается как текущее смещение плюс offset байт), SEEKEND(смещение устанавливается как размер файла плюс offset байт).
- **int execl(const char *path, const char *arg);** – передаем аргументы в командную строку (когда их количество известно)
- **exit(int status);** –выходит из процесса с заданным статусом.

Средства диагностики программы: Для диагностики была использована утилита strace. Она отслеживает системные вызовы, которые представляют собой механизм трансляции, обеспечивающий интерфейс между процессом и операционной системой (ядром). Вызовы могут быть перехвачены и прочитаны. Это позволяет лучше понять, что процесс пытается сделать в заданное время. Перехватывая эти вызовы, мы можем добиться лучшего понимания поведения процессов, особенно если что-то идет не так.

# 2 Исходный код

```c
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h> // options flags
#include <inttypes.h>
#include <errno.h>

#define BUFSIZE 256
#define STRINGCUT 4

int main(void)
{
    const char *testSting = "easyeasyeasy";
    char buf[BUFSIZE];
    char bufTrunc[BUFSIZE];
    int fd;
    ssize_t rd, wr;

    //create file "test"
    fd = creat("test", S_IREAD | S_IWRITE);
    if (fd == -1) {
        perror("Error creat");
    } else {
        fprintf(stdout, "test opened for read/write access\n");
        fprintf(stdout, "test opened is empty\n");
    }

    if (fsync(fd) == -1) {
        perror("Error fsync");
    }

    // write test string in file "test"
    if (wr = write(fd, testSting, strlen(testSting)) == -1) {
        perror("Error write");
    }
    if (lseek(fd, 0, SEEK_SET) == -1) {
        perror("Error lseek");
    }

    if (close(fd) == -1) {
        perror("Error close");
    }

```

```
48      // read from file in buffer
49      fd = open("test", O_RDWR);
50      if (read(fd, buf, strlen(testSting)) == strlen(testSting)) {
51          fprintf(stdout, " \backslash %s", buf);
52          fprintf(stdout, " was written to test\n");
53      } else {
54          perror("Error read");
55      }
56
57      if (close(fd) == -1) {
58          perror("Error close");
59      }
60      if (truncate("test", STRINGCUT) == -1) {
61          perror("Error truncate");
62      }
63
64      // open file for reading
65      fd = open("test", O_RDONLY);
66      rd = read(fd, bufTrunc, BUFSIZE);
67      if (rd == -1 && errno == EINTR) {
68          perror("Error read");
69      }
70
71      fprintf(stdout, "String after trunc: ");
72      fprintf(stdout, " \%s\n", bufTrunc);
73      if (close(fd) == -1) {
74          perror("Error close");
75      }
76
77
78      //create new directory, called newdir, using fork() and exec()
79      if (fork() != 0) {
80          wait(NULL);
81      } else {
82          execl("/bin/mkdir", "mkdir", "newdir", (char *) NULL);
83          perror("Error exec");
84      }
85
86      // use new directory
87      const char *msg = "I'm alive...\n";
88      if ((fd = open("newdir/newfile", O_RDWR | O_CREAT, 0644)) == -1) {
89          perror("Error opening");
90      }
91      write(fd, msg, strlen(msg));
92      close(fd);
93
94      pid_t pid = fork();
95      if (pid == 0) {
96          fprintf(stdout, "It's child process, pid = ");
```

```
97          fprintf(stdout, " \%ld\n", (intmax_t)getpid());
98          fprintf(stdout, "Parent's pid = ");
99          fprintf(stdout, " \%ld\n", (intmax_t)getppid());
100     } else if (pid > 0) {
101         fprintf(stdout, "It's parent process, pid = ");
102         fprintf(stdout, " \%ld\n", (intmax_t)getpid());
103         fprintf(stdout, "Parent's pid = ");
104         fprintf(stdout, " \%ld\n", (intmax_t)getppid());
105         wait(NULL);
106     } else if (pid == -1) {
107         perror("Error fork");
108     }
109     return 0;
110 }
```

# 3 Диагностика программы с помощью strace

```
karma@karma:~/mai_study/OS/lab1$ sudo strace ./run
[sudo] password for karma:
execve("./run",["./run"],[/* 26 vars */]) = 0
brk(NULL)                                = 0x1435000
access("/etc/ld.so.nohwcap",F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7efeb7af7000
access("/etc/ld.so.preload",R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache",O_RDONLY|O_CLOEXEC) = 3
fstat(3,st_mode=S_IFREG|0644,st_size=88446,...) = 0
mmap(NULL,88446,PROT_READ,MAP_PRIVATE,3,0) = 0x7efeb7ae1000
close(3)                                 = 0
access("/etc/ld.so.nohwcap",F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6",O_RDONLY|O_CLOEXEC) = 3
read(3,"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0 > \0\1\0\0\0P\t\2\0\0\0\0\0"..., 832) =
832
fstat(3,st_mode=S_IFREG|0755,st_size=1868984,...) = 0
mmap(NULL,3971488,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7efeb750a000
mprotect(0x7efeb76ca000,2097152,PROT_NONE) = 0
mmap(0x7efeb78ca000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x7efeb78ca000
mmap(0x7efeb78d0000,14752,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1,
= 0x7efeb78d0000
close(3)                                 = 0
mmap(NULL,4096,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7efeb7ae0000
mmap(NULL,4096,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7efeb7adf000
mmap(NULL,4096,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7efeb7ade000
arch_prctl(ARCH_SET_FS,0x7efeb7adf700) = 0
mprotect(0x7efeb78ca000,16384,PROT_READ) = 0
mprotect(0x601000,4096,PROT_READ)     = 0
mprotect(0x7efeb7af9000,4096,PROT_READ) = 0
munmap(0x7efeb7ae1000,88446)           = 0
// создание и открытие файла
creat("test",02)                         = 3
fstat(1,st_mode=S_IFCHR|0620,st_rdev=makedev(136,4),...) = 0
brk(NULL)                                = 0x1435000
brk(0x1456000)                           = 0x1456000
// вывод строк в стандартный поток вывода (дескриптор --1)
write(1,"test opened for read/write acces"...,34test opened for read/write
access
```

```
) = 34
write(1,"test opened is empty\backslashn", 21testopenedisempty
)  = 21
// синхронизация его данных и мета-данных
fsync(3)                                = 0
// записывает строку в файл
write(3,"easyeasyeasy",12)            = 12
// смещение дескриптора в начало файла
lseek(3,0,SEEK_SET)                    = 0
// закрытие файла
close(3)                                = 0
// открытие файла
open("test",O_RDWR)                     = 3
// считывание строки из файла и вывод считанной строки в стандартный поток
вывода,закрытие файла
read(3,"easyeasyeasy",12)              = 12
write(1,"easyeasyeasy was written to test"...,33easyeasyeasy was written to
test
) = 33
close(3)                                = 0
// укорачивание файла до 4 байт
truncate("test",4)                      = 0
// открытие файла и считывание укороченной строки
open("test",O_RDONLY)                   = 3
read(3,"easy",256)                     = 4
// вывод укороченной строки в стандартный поток вывода,закрытие файла
write(1,"String after trunc: easy\n", 25Stringaftertrunc : easy
) = 25
close(3)                                = 0
// создается процесс-потомок,в котором создается новая директория
clone(child_stack=0,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_tidptr
= 15566
wait4(-1,NULL,0,NULL)                  = 15566
---SIGCHLD si_signo=SIGCHLD,si_code=CLD_EXITED,si_pid=15566,si_uid=0,si_status=0,si_ut
---
// создание и открытие в новой директории нового файла
open("newdir/newfile",O_RDWR|O_CREAT,0644) = 3
// записывается строка в созданный файл и файл закрывается
write(3,"I'm alive...\n", 13) = 13
close(3)                                = 0
// создается процесс-потомок
```

```
clone(child_stack=0,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,child_tidptr
= 15569
// получение pid'а процесса
getpid()                                = 15565
write(1,"It's parent process,pid = 15565"...,33It's parent process,pid = 15565
) = 33
//получение pid'а родительского процесса
getppid()                               = 15563
write(1,"Parent's pid = 15563\n",21Parent'spid = 15563
)  = 21
// ожидание завершения процесса-потомка
wait4(-1,It's child process,pid = 15569
Parent's pid = 15565
NULL,0,NULL)                = 15569
---SIGCHLD si_signo=SIGCHLD,si_code=CLD_EXITED,si_pid=15569,si_uid=0,si_status=0,si_ut
---
exit_group(0)                           = ?
+++ exited with 0 +++
```

# 4 Диагностика стороннего ПО (утилита cp)

```
karma@karma:~/mai_study/OS/lab1$ strace cp test newtest.txt
// выполнить программу "/bin/cp"
execve("/bin/cp",["cp","test","newtest.txt"],[/* 73 vars */]) = 0
brk(NULL)                               = 0x2131000
// далее множественные вызовы для работы с памятью,проверяющие права доступа,отражение
файлов в памяти,контроль доступа к областям памяти
access("/etc/ld.so.nohwcap",F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7ff617572000
access("/etc/ld.so.preload",R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache",O_RDONLY|O_CLOEXEC) = 3
fstat(3,st_mode=S_IFREG|0644,st_size=88446,...) = 0
mmap(NULL,88446,PROT_READ,MAP_PRIVATE,3,0) = 0x7ff61755c000
close(3)                                = 0
access("/etc/ld.so.nohwcap",F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libselinux.so.1",O_RDONLY|O_CLOEXEC) = 3
read(3,"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0 > \0\1\0\0\0\260Z\0\0\0\0\0\0"..., 832) =
832
fstat(3,st_mode=S_IFREG|0644,st_size=130224,...) = 0
```

```
mmap(NULL,2234080,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7ff61712d000
mprotect(0x7ff61714c000,2093056,PROT_NONE) = 0
mmap(0x7ff61734b000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x
= 0x7ff61734b000
mmap(0x7ff61734d000,5856,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1,0
= 0x7ff61734d000
close(3)                                = 0
access("/etc/ld.so.nohwcap",F_OK)       = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libacl.so.1",O_RDONLY|O_CLOEXEC) = 3
read(3,"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0 > \0\1\0\0\0\20\34\0\0\0\0\0\0"..., 832) =
832
fstat(3,st_mode=S_IFREG|0644,st_size=31232,...) = 0
mmap(NULL,2126336,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7ff616f25000
mprotect(0x7ff616f2c000,2093056,PROT_NONE) = 0
mmap(0x7ff61712b000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x
= 0x7ff61712b000
close(3)                                = 0
access("/etc/ld.so.nohwcap",F_OK)       = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libattr.so.1",O_RDONLY|O_CLOEXEC) = 3
read(3,"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0 > \0\1\0\0\0\300\20\0\0\0\0\0\0"..., 832) =
832
fstat(3,st_mode=S_IFREG|0644,st_size=18624,...) = 0
mmap(NULL,4096,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7ff61755b000
mmap(NULL,2113760,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7ff616d20000
mprotect(0x7ff616d24000,2093056,PROT_NONE) = 0
mmap(0x7ff616f23000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x
= 0x7ff616f23000
close(3)                                = 0
access("/etc/ld.so.nohwcap",F_OK)       = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6",O_RDONLY|O_CLOEXEC) = 3
read(3,"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0 > \0\1\0\0\0P\t\2\0\0\0\0\0"..., 832) =
832
fstat(3,st_mode=S_IFREG|0755,st_size=1868984,...) = 0
mmap(NULL,3971488,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7ff616956000
mprotect(0x7ff616b16000,2097152,PROT_NONE) = 0
mmap(0x7ff616d16000,24576,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0
= 0x7ff616d16000
mmap(0x7ff616d1c000,14752,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1,
= 0x7ff616d1c000
close(3)                                = 0
access("/etc/ld.so.nohwcap",F_OK)       = -1 ENOENT (No such file or directory)
```

```
open("/lib/x86_64-linux-gnu/libpcre.so.3",O_RDONLY|O_CLOEXEC) = 3
read(3,"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0 > \0\1\0\0\0000\25\0\0\0\0\0\0"..., 832) =
832
fstat(3,st_mode=S_IFREG|0644,st_size=456632,...) = 0
mmap(NULL,2552072,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7ff6166e6000
mprotect(0x7ff616754000,2097152,PROT_NONE) = 0
mmap(0x7ff616954000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x
= 0x7ff616954000
close(3)                                = 0
access("/etc/ld.so.nohwcap",F_OK)       = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libdl.so.2",O_RDONLY|O_CLOEXEC) = 3
read(3,"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0 > \0\1\0\0\0240\r\0\0\0\0\0\0"..., 832) =
832
fstat(3,st_mode=S_IFREG|0644,st_size=14608,...) = 0
mmap(NULL,4096,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7ff61755a000
mmap(NULL,2109680,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7ff6164e2000
mprotect(0x7ff6164e5000,2093056,PROT_NONE) = 0
mmap(0x7ff6166e4000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x
= 0x7ff6166e4000
close(3)                                = 0
access("/etc/ld.so.nohwcap",F_OK)       = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libpthread.so.0",O_RDONLY|O_CLOEXEC) = 3
read(3,"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0 > \0\1\0\0\0260`\0\0\0\0\0\0"..., 832) =
832
fstat(3,st_mode=S_IFREG|0755,st_size=138696,...) = 0
mmap(NULL,2212904,PROT_READ|PROT_EXEC,MAP_PRIVATE|MAP_DENYWRITE,3,0) = 0x7ff6162c5000
mprotect(0x7ff6162dd000,2093056,PROT_NONE) = 0
mmap(0x7ff6164dc000,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,3,0x
= 0x7ff6164dc000
mmap(0x7ff6164de000,13352,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,-1
= 0x7ff6164de000
close(3)                                = 0
mmap(NULL,4096,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7ff617559000
mmap(NULL,8192,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7ff617557000
arch_prctl(ARCH_SET_FS,0x7ff617557800) = 0
mprotect(0x7ff616d16000,16384,PROT_READ) = 0
mprotect(0x7ff6164dc000,4096,PROT_READ) = 0
mprotect(0x7ff6166e4000,4096,PROT_READ) = 0
mprotect(0x7ff616954000,4096,PROT_READ) = 0
mprotect(0x7ff616f23000,4096,PROT_READ) = 0
mprotect(0x7ff61712b000,4096,PROT_READ) = 0
```

```
mprotect(0x7ff61734b000,4096,PROT_READ) = 0
mprotect(0x623000,4096,PROT_READ)     = 0
mprotect(0x7ff617574000,4096,PROT_READ) = 0
munmap(0x7ff61755c000,88446)           = 0
set_tid_address(0x7ff617557ad0)         = 15755
set_robust_list(0x7ff617557ae0,24)     = 0
rt_sigaction(SIGRTMIN,0x7ff6162cab50,[],SA_RESTORER|SA_SIGINFO,0x7ff6162d6390,NULL,8)
= 0
rt_sigaction(SIGRT_1,0x7ff6162cabe0,[],SA_RESTORER|SA_RESTART|SA_SIGINFO,0x7ff6162d63
= 0
rt_sigprocmask(SIG_UNBLOCK,[RTMIN RT_1],NULL,8) = 0
getrlimit(RLIMIT_STACK,rlim_cur=8192*1024,rlim_max=RLIM64_INFINITY) = 0
statfs("/sys/fs/selinux",0x7fff70799bb0) = -1 ENOENT (No such file or directory)
statfs("/selinux",0x7fff70799bb0)      = -1 ENOENT (No such file or directory)
brk(NULL)                               = 0x2131000
brk(0x2152000)                          = 0x2152000
open("/proc/filesystems",O_RDONLY)     = 3
fstat(3,st_mode=S_IFREG|0444,st_size=0,...) = 0
read(3,"nodev\tsysfs\nnodev\trootfs\nnodev\tr"..., 1024) = 358
read(3,"",1024)                        = 0
close(3)                                = 0
open("/usr/lib/locale/locale-archive",O_RDONLY|O_CLOEXEC) = 3
fstat(3,st_mode=S_IFREG|0644,st_size=4228720,...) = 0
mmap(NULL,4228720,PROT_READ,MAP_PRIVATE,3,0) = 0x7ff615ebc000
close(3)                                = 0
geteuid()                               = 1000
//считывается информация о файлах
stat("newtest.txt",0x7fff70799a30)     = -1 ENOENT (No such file or directory)
stat("test",st_mode=S_IFREG|0777,st_size=4,...) = 0
stat("newtest.txt",0x7fff707997c0)     = -1 ENOENT (No such file or directory)
// открытие файла для чтения
open("test",O_RDONLY)                  = 3
fstat(3,st_mode=S_IFREG|0777,st_size=4,...) = 0
// открытие файла для записи,если файл не создан,то создается новый файл
open("newtest.txt",O_WRONLY|O_CREAT|O_EXCL,0777) = 4
fstat(4,st_mode=S_IFREG|0775,st_size=0,...) = 0
fadvise64(3,0,0,POSIX_FADV_SEQUENTIAL) = 0
mmap(NULL,139264,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7ff617535000
// зачитывается строка из первого файла (дескриптор 3)
read(3,"easy",131072)                  = 4
// записывается строка во второй файл (дескриптор 4)
```

```
write(4,"easy",4)                        = 4
read(3,"",131072)                        = 0
// файлы закрываются
close(4)                                   = 0
close(3)                                   = 0
munmap(0x7ff617535000,139264)             = 0
lseek(0,0,SEEK_CUR)                      = -1 ESPIPE (Illegal seek)
close(0)                                   = 0
close(1)                                   = 0
close(2)                                   = 0
exit_group(0)                              = ?
+++ exited with 0 +++
```

Данная утилита копирует файлы. Исходный файл остается неизменным, имя созданного файла может быть таким же, как и у исходного, или измениться. В моем случае нужно было скопировать содержимое файла test в новый, несуществующий файл newtest. После создания оболочкой дочернего процесса последний находит и выполняет файл cp и передает ему имена исходного и целевого файлов. Сначала утилита производит большое количество вызовов, контролирующих память и только потом производится копирование файла с помощью вызовов open, read, write, close.

# 5 Выводы

Узучение системных вызовов позволило по-настоящему понять, что делает операционная система в момент их исполнения. Важно правильно обрабатывать системные вызовы, это поможет избежать ошибок при некорректном исполнении программы и поможет в ее отладке. Идея такого интерфейса довольно проста: получил команду – выполняй или отчитывайся, почему не смог. Система ЭВМ-Человек, словно ее живой аналог Человек-Человек, способна достичь высокой производительности в ходе диалога: решения принимаются по ходу дела, а не планируются за сотню ходов. Если что-то пошло не так, при правильной обработке ошибок мы сразу получим сообщение о ней в этом месте программы.

Утилиты диагностики позволяют быстро выяснить, как программа взаимодействует с операционной системой. Это происходит путем мониторинга системных вызовов и сигналов. Становится понятнее работа собственного и стороннего программного обеспечения. Можно узнать, что делают конкретные системные вызовы, подсчитать их, узнать время их работы и ошибки при их использовании, если они есть, отследить запущенные процессы.

В случае, если у нас нет доступа к исходному коду, мы можем воспользоваться этими утилитами и узнать, что действительно происходит в программе. Отчаявшийся пользователь может использовать утилиты диагностики для отладки своего кода. Они позволяют увидеть не только, где программа "упала но и почему это произошло. Для эффективного использования трассировки необходимо знать системные вызовы и понимать их работу. Этому мы и учились в этой лабораторной работе.