**Федеральное государственное автономное образовательное учреждение**

**Высшего профессионального образования**

Санкт-Петербургский политехнический университет

Институт информационных технологий и управления

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №1**

«Журналирование в Windows и Linux»

Работу выполнил студент гр. № 53501/3 Цыганов А.А.

Работу принял преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Душутина Е.В.

Санкт-Петербург

2016

# Цель работы:

Рассмотреть способы журналирования в ОС Linux и Windows, в частности, утилиты strace и Журнал событий Windows. Провести анализ на примере собственных тестовых и встроенных программ.

# Журналирование в ОС Linux

## Утилита strace:

Strace — это утилита, отслеживающая системные вызовы, которые представляют собой механизм трансляции, обеспечивающий интерфейс между процессом и операционной системой (ядром). Эти вызовы могут быть перехвачены и прочитаны, что позволяет лучше понять, что процесс пытается сделать в заданное время.

Утилита strace предназначена для отслеживания системных вызовов. Утилиту strace используют для анализа поведения процессов, используя следующие флаги:

* **-o** указывает, в какой файл перенаправить вывод программы;
* **-s length** увеличивает длину строки до length байт (по умолчанию равна 32 байтам);
* **-c** выводит время, количество и продолжительность вызовов;
* **-p PID** позволяет отслеживать уже запущенный процесс, указав его PID;
* **-e option** отображения конкретного системного вызова;

Пример использования strace без параметров на программе ls:

|  |  |
| --- | --- |
|  | strace ls |

При использовании утилиты strace без флагов на консоль выводится список системных вызовов в порядке их появления.

Структура вывода **strace**: имя системного вызова (список параметров, переданных вызову) = код завершения.

Можно записать системных вызовы в отдельный файл. Флаг **-o** перенаправляет вывод утилиты в файл.

|  |  |
| --- | --- |
|  | strace -o output.txt ls |

alex@ubuntu:~$ strace ls

execve("/bin/ls", ["ls"], [/\* 59 vars \*/]) = 0

brk(0) = 0x2520000

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) = 0x7f27635fa000

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=86295, ...}) = 0

mmap(NULL, 86295, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f27635e4000

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\300[\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=138400, ...}) = 0

mmap(NULL, 2242312, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f27631b5000

mprotect(0x7f27631d6000, 2093056, PROT\_NONE) = 0

mmap(0x7f27633d5000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x20000) = 0x7f27633d5000

mmap(0x7f27633d7000, 5896, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f27633d7000

close(3) = 0

…

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f27635e1000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f27635df000

arch\_prctl(ARCH\_SET\_FS, 0x7f27635df800) = 0

mprotect(0x7f2762fa2000, 16384, PROT\_READ) = 0

mprotect(0x7f2762566000, 4096, PROT\_READ) = 0

mprotect(0x7f276276f000, 4096, PROT\_READ) = 0

mprotect(0x7f2762973000, 4096, PROT\_READ) = 0

mprotect(0x7f2762be0000, 4096, PROT\_READ) = 0

mprotect(0x7f27631b3000, 4096, PROT\_READ) = 0

mprotect(0x7f27633d5000, 4096, PROT\_READ) = 0

mprotect(0x61b000, 4096, PROT\_READ) = 0

mprotect(0x7f27635fc000, 4096, PROT\_READ) = 0

munmap(0x7f27635e4000, 86295) = 0

set\_tid\_address(0x7f27635dfad0) = 7792

set\_robust\_list(0x7f27635dfae0, 24) = 0

rt\_sigaction(SIGRTMIN, {0x7f2762353bb0, [], SA\_RESTORER|SA\_SIGINFO, 0x7f276235ed10}, NULL, 8) = 0

rt\_sigaction(SIGRT\_1, {0x7f2762353c40, [], SA\_RESTORER|SA\_RESTART|SA\_SIGINFO, 0x7f276235ed10}, NULL, 8) = 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", 0x7ffcedee2760) = -1 ENOENT (No such file or directory)

statfs("/selinux", 0x7ffcedee2760) = -1 ENOENT (No such file or directory)

brk(0) = 0x2520000

brk(0x2541000) = 0x2541000

open("/proc/filesystems", O\_RDONLY) = 3

fstat(3, {st\_mode=S\_IFREG|0444, st\_size=0, ...}) = 0

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f27635f9000

read(3, "nodev\tsysfs\nnodev\trootfs\nnodev\tr"..., 1024) = 324

read(3, "", 1024) = 0

close(3) = 0

munmap(0x7f27635f9000, 4096) = 0

open("/usr/lib/locale/locale-archive", O\_RDONLY|O\_CLOEXEC) = 3

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=7216688, ...}) = 0

mmap(NULL, 7216688, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f2761c6c000

close(3) = 0

ioctl(1, SNDCTL\_TMR\_TIMEBASE or SNDRV\_TIMER\_IOCTL\_NEXT\_DEVICE or TCGETS, {B38400 opost isig icanon echo ...}) = 0

ioctl(1, TIOCGWINSZ, {ws\_row=24, ws\_col=187, ws\_xpixel=0, ws\_ypixel=0}) = 0

openat(AT\_FDCWD, ".", O\_RDONLY|O\_NONBLOCK|O\_DIRECTORY|O\_CLOEXEC) = 3

getdents(3, /\* 30 entries \*/, 32768) = 1008

getdents(3, /\* 0 entries \*/, 32768) = 0

close(3) = 0

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(136, 5), ...}) = 0

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f27635f9000

write(1, "Desktop Documents Downloads D"..., 100Desktop Documents Downloads Dropbox examples.desktop Music Pictures Public Templates Videos

) = 100

close(1) = 0

munmap(0x7f27635f9000, 4096) = 0

close(2) = 0

exit\_group(0) = ?

+++ exited with 0 +++

Флаг **-c** позволяет выводить общую статистику в табличном виде по каждому из системных вызовов: время обработки, кол-во вызовов, ошибки и т.д.

|  |  |
| --- | --- |
|  | strace –с ls |

% time seconds usecs/call calls errors syscall

------ ----------- ----------- --------- --------- --------------

100.00 0.000723 362 2 getdents

0.00 0.000000 0 9 read

0.00 0.000000 0 1 write

0.00 0.000000 0 10 open

0.00 0.000000 0 13 close

0.00 0.000000 0 11 fstat

0.00 0.000000 0 26 mmap

0.00 0.000000 0 16 mprotect

0.00 0.000000 0 3 munmap

0.00 0.000000 0 3 brk

0.00 0.000000 0 2 rt\_sigaction

0.00 0.000000 0 1 rt\_sigprocmask

0.00 0.000000 0 2 ioctl

0.00 0.000000 0 9 9 access

0.00 0.000000 0 1 execve

0.00 0.000000 0 1 getrlimit

0.00 0.000000 0 2 2 statfs

0.00 0.000000 0 1 arch\_prctl

0.00 0.000000 0 1 set\_tid\_address

0.00 0.000000 0 1 openat

0.00 0.000000 0 1 set\_robust\_list

------ ----------- ----------- --------- --------- ----------------

100.00 0.000723 116 11 total

Пример использования ключ **–р** позволяет присоединиться к уже запущенному процессу, используя его PID. Ниже в примере используется работающее приложение htop c PID = 2322.

alex@ubuntu:~$ ps -ax | grep htop

8227 pts/16 S+ 0:00 htop

|  |  |
| --- | --- |
|  | strace -p 8227 |

open("/proc/74/stat", O\_RDONLY) = 4

read(4, "74 (ata\_sff) S 2 0 0 0 -1 692388"..., 2048) = 172

read(4, "", 1876) = 0

close(4) = 0

openat(AT\_FDCWD, "/proc/75/task", O\_RDONLY|O\_NONBLOCK|O\_DIRECTORY|O\_CLOEXEC) = 4

getdents(4, /\* 3 entries \*/, 32768) = 72

getdents(4, /\* 0 entries \*/, 32768) = 0

close(4) = 0

open("/proc/75/statm", O\_RDONLY) = 4

read(4, "0 0 0 0 0 0 0\n", 255) = 14

read(4, "", 241) = 0

close(4) = 0

open("/proc/75/stat", O\_RDONLY) = 4

read(4, "75 (md) S 2 0 0 0 -1 69238880 0 "..., 2048) = 167

read(4, "", 1881) = 0

close(4) = 0

openat(AT\_FDCWD, "/proc/76/task", O\_RDONLY|O\_NONBLOCK|O\_DIRECTORY|O\_CLOEXEC) = 4

getdents(4, /\* 3 entries \*/, 32768) = 72

getdents(4, /\* 0 entries \*/, 32768) = 0

close(4) = 0

open("/proc/76/statm", O\_RDONLY) = 4

read(4, "0 0 0 0 0 0 0\n", 255) = 14

read(4, "", 241) = 0

close(4) = 0

open("/proc/76/stat", O\_RDONLY) = 4

read(4, "76 (devfreq\_wq) S 2 0 0 0 -1 692"..., 2048) = 175

read(4, "", 1873) = 0

close(4) = 0

openat(AT\_FDCWD, "/proc/80/task", O\_RDONLY|O\_NONBLOCK|O\_DIRECTORY|O\_CLOEXEC) = 4

getdents(4, /\* 3 entries \*/, 32768) = 72

getdents(4, /\* 0 entries \*/, 32768) = 0

close(4) = 0

open("/proc/80/statm", O\_RDONLY) = 4

read(4, "0 0 0 0 0 0 0\n", 255) = 14

read(4, "", 241) = 0

close(4) = 0

Проведем запуск утилиты strace на примере простой программы, для отслеживания определенных системных вызовов. Ниже приведен текст программы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #include <stdio.h>  #include <unistd.h>  #include <signal.h>  #include <stdlib.h>  int main()  {  int pid = getpid();  printf("Initial pid: %d**\n**", pid);  int n=0;  int m=0;  printf("Please, enter n and m: ");  scanf("%d %d", &n, &m);  int p;  pid=fork();  **if**(pid==0)  {  p=getpid();  printf("Child pid=%d. n=%d**\n**",p, n);  } **else**  {  p=getpid();  printf("Father. pid=%d. m=%d**\n**",p, m);  kill(pid, SIGKILL);  }  } |

Ниже приведен список системных вызовов: strace -o output.txt ./a.out

execve("./a.out", ["./a.out"], [/\* 64 vars \*/]) = 0

brk(0) = 0x6a9000

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) = 0x7f34399dc000

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=86295, ...}) = 0

mmap(NULL, 86295, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f34399c6000

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\0`\v\2\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=1869392, ...}) = 0

mmap(NULL, 3972864, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f34393f1000

mprotect(0x7f34395b1000, 2097152, PROT\_NONE) = 0

mmap(0x7f34397b1000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1c0000) = 0x7f34397b1000

mmap(0x7f34397b7000, 16128, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f34397b7000

close(3) = 0

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f34399c5000

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f34399c4000

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f34399c3000

arch\_prctl(ARCH\_SET\_FS, 0x7f34399c4700) = 0

mprotect(0x7f34397b1000, 16384, PROT\_READ) = 0

mprotect(0x600000, 4096, PROT\_READ) = 0

mprotect(0x7f34399de000, 4096, PROT\_READ) = 0

munmap(0x7f34399c6000, 86295) = 0

getpid() = 11537

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(136, 19), ...}) = 0

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f34399db000

write(1, "Initial pid: 11537\n", 19) = 19

fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(136, 19), ...}) = 0

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f34399da000

write(1, "Please, enter n and m: ", 23) = 23

read(0, "1 2\n", 1024) = 4

clone(child\_stack=0, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f34399c49d0) = 11538

write(1, "Father. pid=11537. m=2\n", 23) = 23

kill(11538, SIGKILL) = 0

exit\_group(0) = ?

+++ exited with 0 +++

Вывод системных вызовов в табличном представлении: strace -с -o output.txt ./a.out

% time seconds usecs/call calls errors syscall

------ ----------- ----------- --------- --------- --------------

0.00 0.000000 0 2 read

0.00 0.000000 0 4 1 write

0.00 0.000000 0 2 open

0.00 0.000000 0 2 close

0.00 0.000000 0 4 fstat

0.00 0.000000 0 10 mmap

0.00 0.000000 0 4 mprotect

0.00 0.000000 0 1 munmap

0.00 0.000000 0 1 brk

0.00 0.000000 0 3 3 access

0.00 0.000000 0 1 getpid

0.00 0.000000 0 1 clone

0.00 0.000000 0 1 execve

0.00 0.000000 0 1 kill

0.00 0.000000 0 1 arch\_prctl

------ ----------- ----------- --------- --------- --------------

100.00 0.000000 38 4 total

## Утилита ltrace:

Программа ltrace предназначена для отладки динамически собранных программ. Отлаживаемый код запускается под управлением ltrace, при этом вызовы динамических библиотек, а также получаемые процессом сигналы перехватываются и регистрируются. Возможна также регистрация системных вызовов со стороны отлаживаемой программы (-s опция). Для отладки программы её не нужно перекомпилировать, поэтому возможно использование ltrace с программами, исходный текст которых не доступен.

Ниже приведен пример использования утилиты ltrace и список опций:

Usage: ltrace [option ...] [command [arg ...]]

Trace library calls of a given program.

-a, --align=COLUMN align return values in a secific column.

-A MAXELTS maximum number of array elements to print.

-b, --no-signals don't print signals.

-c count time and calls, and report a summary on exit.

-C, --demangle decode low-level symbol names into user-level names.

-D, --debug=MASK enable debugging (see -Dh or --debug=help).

-Dh, --debug=help show help on debugging.

-e FILTER modify which library calls to trace.

-f trace children (fork() and clone()).

-F, --config=FILE load alternate configuration file (may be repeated).

-h, --help display this help and exit.

-i print instruction pointer at time of library call.

-l, --library=LIBRARY\_PATTERN only trace symbols implemented by this library.

-L do NOT display library calls.

-n, --indent=NR indent output by NR spaces for each call level nesting.

-o, --output=FILENAME write the trace output to file with given name.

-p PID attach to the process with the process ID pid.

-r print relative timestamps.

-s STRSIZE specify the maximum string size to print.

-S trace system calls as well as library calls.

-t, -tt, -ttt print absolute timestamps.

-T show the time spent inside each call.

-u USERNAME run command with the userid, groupid of username.

-V, --version output version information and exit.

-x FILTER modify which static functions to trace.

Выполним утилиту ltrace (с опцией -S) на системной программе ls:

alex@ubuntu:~/Dropbox/политех/Системное программирование$ ltrace -S -o transfer ls

a.out app.cpp transfer Цыганова А. А. гр. №53501-3; Л. р. № 1.docx

SYS\_brk(0) = 0x19ec000

SYS\_access("/etc/ld.so.nohwcap", 00) = -2

SYS\_mmap(0, 8192, 3, 34) = 0x7f3280145000

SYS\_access("/etc/ld.so.preload", 04) = -2

SYS\_open("/etc/ld.so.cache", 524288, 01) = 3

SYS\_fstat(3, 0x7fff098f9930) = 0

SYS\_mmap(0, 0x15117, 1, 2) = 0x7f328012f000

SYS\_close(3) = 0

…

malloc(32) = 0x19f1bc0

strlen(".") = 1

malloc(2) = 0x19f1bf0

memcpy(0x19f1bf0, ".\0", 2) = 0x19f1bf0

\_\_errno\_location() = 0x7f328012a6b0

opendir("." <unfinished ...>

SYS\_openat(-100, 0x19f1bf0, 0x90800, 0) = 3

<... opendir resumed> ) = 0x19f1c10

readdir(0x19f1c10 <unfinished ...>

SYS\_getdents(3, 0x19f1c40, 0x8000, 142) = 232

<... readdir resumed> ) = 0x19f1c40

strlen("transfer") = 8

malloc(9) = 0x19f9c50

memcpy(0x19f9c50, "transfer\0", 9) = 0x19f9c50

readdir(0x19f1c10) = 0x19f1c60

readdir(0x19f1c10) = 0x19f1c78

strlen("app.cpp") = 7

malloc(8) = 0x19f9c70

memcpy(0x19f9c70, "app.cpp\0", 8) = 0x19f9c70

readdir(0x19f1c10) = 0x19f1c98

strlen("a.out") = 5

malloc(6) = 0x19f9c90

memcpy(0x19f9c90, "a.out\0", 6) = 0x19f9c90

readdir(0x19f1c10) = 0x19f1cb8

strlen("\320\246\321\213\320\263\320\260\320\275\320\276\320\262\320\260 \320\220. \320\220. \320\263\321\200. \342"...) = 62

malloc(63) = 0x19f9cb0

memcpy(0x19f9cb0, "\320\246\321\213\320\263\320\260\320\275\320\276\320\262\320\260 \320\220. \320\220. \320\263\321\200. \342"..., 63) = 0x19f9cb0

readdir(0x19f1c10) = 0x19f1d10

readdir(0x19f1c10 <unfinished ...>

SYS\_getdents(3, 0x19f1c40, 0x8000, 0x19f1cc8) = 0

<... readdir resumed> ) = 0

closedir(0x19f1c10 <unfinished ...>

SYS\_close(3) = 0

<... closedir resumed> ) = 0

Ниже приведен табличный результат выполнения ltrace (с опцией -S) на утилите ls:

alex@ubuntu:~/Dropbox/политех/Системное программирование$ ltrace -c -S -o transfer ls

a.out app.cpp transfer Цыганова А. А. гр. №53501-3; Л. р. № 1.docx

% time seconds usecs/call calls function

------ ----------- ----------- --------- --------------------

18.85 0.027030 281 96 mbrtowc

18.56 0.026612 277 96 mbsinit

9.27 0.013288 276 48 iswprint

8.89 0.012743 265 48 wcwidth

6.06 0.008694 322 27 SYS\_mmap

4.63 0.006631 276 24 \_\_ctype\_get\_mb\_cur\_max

4.07 0.005837 291 20 \_\_errno\_location

2.17 0.003107 282 11 malloc

1.99 0.002857 357 8 strlen

1.88 0.002700 337 8 getenv

1.73 0.002486 155 16 SYS\_mprotect

1.67 0.002388 341 7 readdir

1.53 0.002200 275 8 memcpy

1.33 0.001906 317 6 \_\_overflow

1.21 0.001728 123 14 SYS\_close

1.11 0.001593 1593 1 setlocale

1.02 0.001465 366 4 fwrite\_unlocked

0.99 0.001414 282 5 strcoll

0.93 0.001333 121 11 SYS\_open

0.92 0.001317 329 4 free

0.84 0.001210 100 12 SYS\_fstat

0.81 0.001163 129 9 SYS\_read

0.79 0.001134 567 2 fclose

0.76 0.001091 121 9 SYS\_access

0.75 0.001075 1075 1 strrchr

0.72 0.001034 258 4 \_\_freading

0.50 0.000714 238 3 SYS\_brk

0.43 0.000615 615 1 isatty

0.39 0.000560 560 1 opendir

0.37 0.000530 265 2 fileno

0.37 0.000526 263 2 fflush

0.36 0.000520 260 2 \_\_fpending

0.35 0.000498 249 2 SYS\_statfs

0.33 0.000480 480 1 ioctl

0.33 0.000478 478 1 closedir

0.33 0.000468 156 3 SYS\_munmap

0.27 0.000383 383 1 \_\_cxa\_atexit

0.25 0.000364 182 2 SYS\_rt\_sigaction

0.24 0.000349 349 1 textdomain

0.21 0.000296 296 1 getopt\_long

0.20 0.000286 286 1 \_setjmp

0.18 0.000264 264 1 bindtextdomain

0.18 0.000263 131 2 SYS\_ioctl

0.17 0.000249 249 1 realloc

0.16 0.000235 235 1 SYS\_set\_tid\_address

0.15 0.000209 209 1 SYS\_rt\_sigprocmask

0.14 0.000204 204 1 SYS\_getrlimit

0.14 0.000195 97 2 SYS\_getdents

0.13 0.000185 185 1 SYS\_write

0.11 0.000154 154 1 SYS\_set\_robust\_list

0.09 0.000128 128 1 SYS\_arch\_prctl

0.07 0.000098 98 1 SYS\_openat

0.06 0.000082 82 1 SYS\_futex

------ ----------- ----------- --------- --------------------

100.00 0.143369 537 total

Проверим работу ltrace (с опцией -S) на программе приведенной в предыдущем пункте:

alex@ubuntu:~/Dropbox/политех/Системное программирование$ ltrace -S -o transfer ./a.out

a.out app.cpp transfer Цыганова А. А. гр. №53501-3; Л. р. № 1.docx

SYS\_brk(0) = 0x1a4e000

SYS\_access("/etc/ld.so.nohwcap", 00) = -2

SYS\_mmap(0, 8192, 3, 34) = 0x7f6b96b8e000

SYS\_access("/etc/ld.so.preload", 04) = -2

SYS\_open("/etc/ld.so.cache", 524288, 01) = 3

SYS\_fstat(3, 0x7ffe95a24430) = 0

SYS\_mmap(0, 0x15117, 1, 2) = 0x7f6b96b78000

SYS\_close(3) = 0

SYS\_access("/etc/ld.so.nohwcap", 00) = -2

SYS\_open("/lib/x86\_64-linux-gnu/libc.so.6", 524288, 022656220610) = 3

SYS\_read(3, "\177ELF\002\001\001\003", 832) = 832

SYS\_fstat(3, 0x7ffe95a24470) = 0

SYS\_mmap(0, 0x3c9f00, 5, 2050) = 0x7f6b965a3000

SYS\_mprotect(0x7f6b96763000, 2097152, 0) = 0

SYS\_mmap(0x7f6b96963000, 0x6000, 3, 2066) = 0x7f6b96963000

SYS\_mmap(0x7f6b96969000, 0x3f00, 3, 50) = 0x7f6b96969000

SYS\_close(3) = 0

SYS\_mmap(0, 4096, 3, 34) = 0x7f6b96b77000

SYS\_mmap(0, 4096, 3, 34) = 0x7f6b96b76000

SYS\_mmap(0, 4096, 3, 34) = 0x7f6b96b75000

SYS\_arch\_prctl(4098, 0x7f6b96b76700, 0x7f6b96b75010, 34) = 0

SYS\_mprotect(0x7f6b96963000, 16384, 1) = 0

SYS\_mprotect(0x600000, 4096, 1) = 0

SYS\_mprotect(0x7f6b96b90000, 4096, 1) = 0

SYS\_munmap(0x7f6b96b78000, 86295) = 0

\_\_libc\_start\_main(0x4006a6, 1, 0x7ffe95a24db8, 0x400780 <unfinished ...>

getpid( <unfinished ...>

SYS\_getpid() = 15443

<... getpid resumed> ) = 15443

printf("Initial pid: %d\n", 15443 <unfinished ...>

SYS\_fstat(1, 0x7ffe95a244d0) = 0

SYS\_mmap(0, 4096, 3, 34) = 0x7f6b96b8d000

SYS\_write(1, "Initial pid: 15443\n", 19) = 19

<... printf resumed> ) = 19

printf("Please, enter n and m: ") = 23

scanf(0x40082d, 0x7ffe95a24cb8, 0x7ffe95a24cbc, 0x6e207265746e6520 <unfinished ...>

SYS\_fstat(0, 0x7ffe95a24800) = 0

SYS\_mmap(0, 4096, 3, 34) = 0x7f6b96b8c000

SYS\_write(1, "Please, enter n and m: ", 23) = 23

# Журналирование в Windows

## ProcessMonitor:

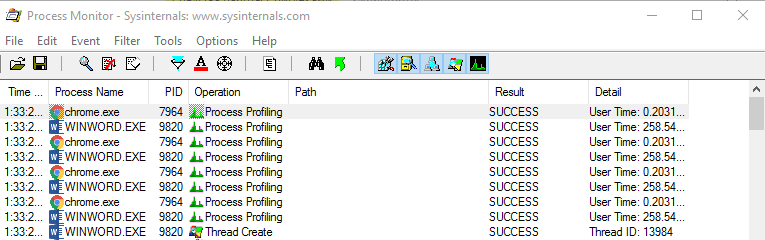
Приложение с графическим интерфейсом, позволяет получить информацию различного рода об исполняющихся процессах. Список процессов может быть сортирован, фильтрован или быть представлен в иерархической структуре.

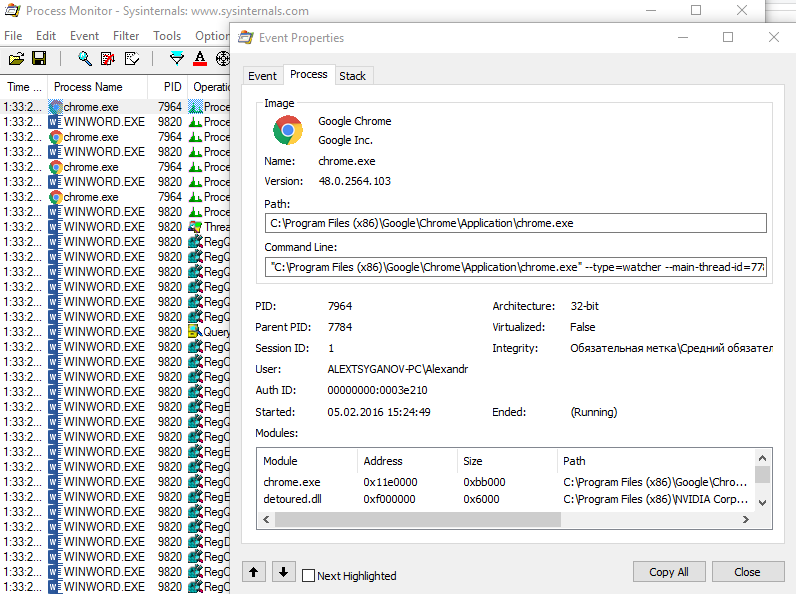
Process Monitor контролирует и следит за всей работой [операционной системы](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) и отображает все происходящие [процессы](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), работающие [библиотеки](https://ru.wikipedia.org/wiki/DLL), различные [драйвера](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B0%D0%B9%D0%B2%D0%B5%D1%80) устройств, а также все изменения, происходящие с файлами, и выводит сообщение об их удалении или открытии.

Включает в себя инструмент для мониторинга [системного реестра](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B5%D1%81%D1%82%D1%80_Windows) и показывает, какие программы обращаются к нему (какие ключи читают и пытаются в них что-либо записать).

С помощью Process Monitor можно искать и отслеживать неполадки в системе, а также [вредоносные программы](https://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%B4%D0%BE%D0%BD%D0%BE%D1%81%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0). Ко всему прочему, [утилита](https://ru.wikipedia.org/wiki/%D0%A3%D1%82%D0%B8%D0%BB%D0%B8%D1%82%D0%B0) может предоставить детальную информацию о каждом [процессе](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), какой [пользователь](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C) её запустил или полный путь к [каталогу](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3_(%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)). Process Monitor может обрабатывать миллионы процессов, а весь [журнал событий](https://ru.wikipedia.org/wiki/%D0%96%D1%83%D1%80%D0%BD%D0%B0%D0%BB_%D1%81%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B9) сохранить в файл до 1 [гигабайта](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%B3%D0%B0%D0%B1%D0%B0%D0%B9%D1%82).

Ниже приведены примеры программы:





## Системный журнал Windows:

В Windows, системный журнал используется для централизованного хранения информации о важных программных событиях в системе. Появление каждого важного события заносится в файл. К таким событиям относятся, например, вход пользователя в систему, программные ошибки.

Все события в ОС Windows разделены на следующие категории:

* Приложение — журнал, в котором фиксируются сообщения, поступившие от пользовательских приложений. В зависимости от важности, события делятся на три категории: ошибка, предупреждение и уведомление.
* Безопасность — журнал, в котором операционная система фиксирует различные сообщения, связанные с безопасностью, например, успешный или неуспешный вход пользователя.
* Установка — журнал, в котором протоколируются все события, связанные с обновлением операционной системы
* Система — журнал, в которых попадают сообщения о системных событиях, как, например, состояние различных служб.
* Перенаправленные события — журнал, в который пересылаются данный из других компьютеров.

Ниже приведен интерфейс системного журнала:

