Московский Авиационный Интститут
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Операционные системы»

Диагностика программного обеспечения

Студент: Артемьев Дмитрий Иванович
Группа: М8О-206Б-18
Вариант: 1
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2019

Цель работы

Приобретение практических навыков диагностики работы программного обеспечения.

Задание

При выполнении последующих лабораторных работ необходимо продемонстрировать ключевые системные вызовы, которые в них используются и то, что их использование соответствует варианту ЛР.

По итогам выполнения всех лабораторных работ отчет по данной должен содержать краткую сводку по исследованию последующих ЛР.

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. Это позволяет лучше понять, что процесс пытается сделать в заданное время.

Некоторые ключи:

- -i - выводить указатель на инструкцию во время выполнения системного вызова;
- -k - выводить стек вызовов для отслеживаемого процесса после каждого системного вызова;
- -o - выводить всю информацию о системных вызовах не в стандартный поток ошибок, а в файл;
- -q - не выводить сообщения о подключении о отключении от процесса;
- -r - выводить временную метку для каждого системного вызова;
- -s - указать максимальный размер выводимой строки, по умолчанию 32;
- -t - выводить время суток для каждого вызова;
- -T - выводить длительность выполнения системного вызова;
- -x - выводить все не ASCI-строки в шестнадцатеричном виде;
- -xx - выводить все строки в шестнадцатеричном виде;
- -c - подсчитывать количество ошибок, вызовов и время выполнения для каждого системного вызова;
- -S - сортировать информацию выводимую при опции -c. Доступны поля time, calls, name и nothing. По умолчанию используется time;

- -e - позволяет отфильтровать только нужные системные вызовы или события;
- -v - позволяет выводить дополнительную информацию, такую как версии окружения, статистику и так далее;
- -b - если указанный системный вызов обнаружен, трассировка прекращается;
- -f - отслеживать также дочерние процессы, если они будут созданы;
- -ff - если задана опция -o, то для каждого дочернего процесса будет создан отдельный файл с именем имя_файла.pid.
- -I - позволяет блокировать реакцию на нажатия Ctrl+C и Ctrl+Z;
- -u - запустить программу, от имени указанного пользователя.

Консоль

Рассмотрим вывод системных вызовов для второй лабораторной:

```
amder@amder-pc $ strace ./a.out < test.txt
execve("./a.out", ["./a.out"], 0x7ffe3d282790 /* 56 vars */) = 0
brk(NULL)                               = 0x561c48daf000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc4668f2d0) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, st_mode=S_IFREG|0644, st_size=282589, ...) = 0
mmap(NULL, 282589, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd99a004000
close(3)                                = 0
openat(AT_FDCWD, "/usr/lib/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`r\ 2\ 0\ 0
lseek(3, 64, SEEK_SET)                  = 64
read(3, "\ 6\ 0\ 0\ 0\ 4\ 0\ 0\ 0@\ 0\ 0\ 0\ 0\ 0\ 0\ 0@\ 0\ 0\ 0\ 0\ 0\ 0\ 0@\ 0\ 0\
lseek(3, 848, SEEK_SET)                 = 848
read(3, "\ 4\ 0\ 0\ 0\ 20\ 0\ 0\ 0\ 5\ 0\ 0\ 0GNU\ 0\ 2\ 0\ 0\ 300\ 4\ 0\ 0\ 0\ 3\ 0\
lseek(3, 880, SEEK_SET)                 = 880
read(3, "\ 4\ 0\ 0\ 0\ 24\ 0\ 0\ 0\ 3\ 0\ 0\ 0GNU\ 0003\ 321\ 363P\ 3617(e\ 35t\ 335*V
fstat(3, st_mode=S_IFREG|0755, st_size=2149496, ...) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd99a002
lseek(3, 64, SEEK_SET)                  = 64
read(3, "\ 6\ 0\ 0\ 0\ 4\ 0\ 0\ 0@\ 0\ 0\ 0\ 0\ 0\ 0\ 0@\ 0\ 0\ 0\ 0\ 0\ 0\ 0@\ 0\ 0\
lseek(3, 848, SEEK_SET)                 = 848
read(3, "\ 4\ 0\ 0\ 0\ 20\ 0\ 0\ 0\ 5\ 0\ 0\ 0GNU\ 0\ 2\ 0\ 0\ 300\ 4\ 0\ 0\ 0\ 3\ 0\
lseek(3, 880, SEEK_SET)                 = 880
read(3, "\ 4\ 0\ 0\ 0\ 24\ 0\ 0\ 0\ 3\ 0\ 0\ 0GNU\ 0003\ 321\ 363P\ 3617(e\ 35t\ 335*V
```

```
mmap(NULL, 1860536, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd999e3b000
mprotect(0x7fd999e60000, 1671168, PROT_NONE) = 0
mmap(0x7fd999e60000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7fd999fad000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1720
mmap(0x7fd999ff8000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
mmap(0x7fd999ffe000, 13240, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fd99a003540) = 0
mprotect(0x7fd999ff8000, 12288, PROT_READ) = 0
mprotect(0x561c48932000, 4096, PROT_READ) = 0
mprotect(0x7fd99a073000, 4096, PROT_READ) = 0
munmap(0x7fd99a004000, 282589)          = 0
pipe([3, 4])                            = 0
pipe([5, 6])                            = 0
pipe([7, 8])                            = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_t
read(0, "123 -567\ n", 9)                = 9
read(0, "", 9)                          = 0
close(5)                                = 0
close(3)                                = 0
write(4, "123 -567\ n", 9)               = 9
write(6, "\ 0\ 240\ 363hF\ 374\ 177\ 0\ 0", 9) = 9
--- SIGCHLD si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=29545, si_uid=1000, si_status
close(6)                                = 0
close(4)                                = 0
wait4(-1, [WIFEXITED(s) && WEXITSTATUS(s) == 0], 0, NULL) = 29545
close(8)                                = 0
read(7, "1\ 0", 2)                       = 2
write(1, "1\ 0", 21)                     = 2
close(7)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

В первых строчках видны какие-то ошибки. Хочется, не просматривая всего вывода, установить, нет ли других ошибок.

```
amder@amder-pc ~/Documents/MAI/3_sem/OSs/lab2/src (master*) $ strace -c ./a.out < test
1% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
18,89    0,000119         119         1           clone
13,97    0,000088          12         7           mmap
12,54    0,000079           9         8           close
12,38    0,000078           7        10           read
```

```
10,00    0,000063          21         3           write
8,57    0,000054          13         4           mprotect
6,98    0,000044          14         3           pipe
5,08    0,000032          32         1           munmap
4,13    0,000026          26         1           wait4
3,65    0,000023           3         6           lseek
2,22    0,000014           7         2           openat
0,79    0,000005           2         2           fstat
0,79    0,000005           2         2         1 arch_prctl
0,00    0,000000           0         1           brk
0,00    0,000000           0         1         1 access
0,00    0,000000           0         1           execve
------ ----------- ----------- --------- --------- ----------------
100.00    0,000630                    53         2 total
```

Как видно из вывода, ошибки присутствуют только в двух системных вызовах. Первая - в вызове arch_prtcl(code, addr), который устанавливает специфичное для данной архитектуры состояние процесса или треда. Тип ошибки - EINVAL (code не является допустимым кодом подфункции). Данная ошибка не критична для работы программы.

Вторая ошибка в вызове access(pathname, mode). access проверяет, имеет ли процесс права на чтение или запись, или же просто проверяет, существует ли файл (или другой объект файловой системы), с именем pathname. Если pathname является символьной ссылкой, то проверяются права доступа к файлу, на который она ссылается. Тип ошибки - ENOENT (компонент пути pathname не существует или является "висячей"символической ссылкой). Также не критично.

Демонстрация работы для 4-й лабораторной. Так как вывод системных вызовов достаточно большой, посмотрим только на вызываемые mmap. Интересно, что количество вызовов mmap не многим больше, чем во второй лабораторной.

```
amder@amder-pc $ strace -T -i -e mmap ./a.out <test.txt
[00007f7349136706] mmap(NULL, 282589, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f73490d5000 <
[00007f7349136706] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
[00007f7349136706] mmap(NULL, 135592, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x
[00007f7349136706] mmap(0x7f73490b8000, 65536, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FI
[00007f7349136706] mmap(0x7f73490c8000, 20480, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DE
[00007f7349136706] mmap(0x7f73490cd000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FI
[00007f7349136706] mmap(0x7f73490cf000, 12712, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_F
[00007f7349136706] mmap(NULL, 1860536, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0
[00007f7349136706] mmap(0x7f7348f0f000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_
[00007f7349136706] mmap(0x7f734905c000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_D
[00007f7349136706] mmap(0x7f73490a7000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_F
[00007f7349136706] mmap(0x7f73490ad000, 13240, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_F
[00007f7349136706] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,
[00007f7348fe4046] mmap(NULL, 100, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f73491
[00007f7348fe4046] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f734911
[00007f7348fe4046] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f734911
[00007f7348fd9939] --- SIGCHLD si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=29429, si_
4
[????????????????] +++ exited with 0 +++
```

Ну, и, конечно, сводка и ошибки системных вызовов.

```
amder@amder-pc $ strace -c ./a.out < test.txt
4
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
0,00    0,000000           0        11           read
0,00    0,000000           0         4           write
0,00    0,000000           0         6           close
0,00    0,000000           0         7           fstat
0,00    0,000000           0         2         2 lstat
0,00    0,000000           0         9           lseek
0,00    0,000000           0        16           mmap
0,00    0,000000           0         5           mprotect
0,00    0,000000           0         4           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         1         1 access
0,00    0,000000           0         3           getpid
0,00    0,000000           0         1           clone
0,00    0,000000           0         1           execve
0,00    0,000000           0         2           link
0,00    0,000000           0         5           unlink
0,00    0,000000           0         1           statfs
0,00    0,000000           0         2         1 arch_prctl
0,00    0,000000           0         3         1 futex
0,00    0,000000           0         1           set_tid_address
0,00    0,000000           0         8         2 openat
0,00    0,000000           0         1           set_robust_list
0,00    0,000000           0         1           prlimit64
------ ----------- ----------- --------- --------- ----------------
100.00    0,000000                   100         7 total
```

Выводы

Выполняя данную лабораторную, я научился использовать мощную утилиту strace для анализа поведения программы, узнал для себя несколько новых системных вызовов.