Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу

«Операционные системы»

Использование утилиты strace

Студент: Шевлякова София Сергеевна
Группа: М8О–208Б–21
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Постановка задачи

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.

Описание утилиты strace

Strace – это утилита Linux, отслеживающая системные вызовы, которые представляют собой механизм трансляции, обеспечивающий интерфейс между процессором и операционной системой. Использование данной утилиты позволяет понять, что процесс пытается сделать в данное время. Strace может быть очень полезен при отладке программ.

$ strace опции команда аргументы – синтаксис утилиты.

Для удобства работы с протоколом утилиты можно использовать следующие ключи:

- **-o file** – выводит всю информацию о системных вызовах не в стандартный поток ошибок, а в file;
- **-i** – выводит указатель на инструкцию во время выполнения системного вызова;
- **-p PID** – указывает PID процесса, к которому следует подключиться для отслеживания системных вызовов;
- **-k** – выводит стек вызовов для отслеживаемого процесса после каждого системного вызова;
- **-T** – выводит длительность выполнения системного вызова;

- **-b** – если указанный системный вызов обнаружен, трассировка прекращается;
- **-c** – подсчитывает количество ошибок, вызовов и время выполнения для каждого системного вызова;
- **-f** – отслеживание системных вызовов в дочерних процессах, если они были созданы;
- **-y** – заменит в протоколе все файловые дескрипторы на имена соответствующих им файлов, если это возможно;
- **-e trace=filters** – указываем выражение, по которым будут фильтроваться системные вызовы;

Протокол утилиты strace

```
1.  execve("./a.out", ["./a.out"], 0x7ffd6e706ba8 /* 18 vars */) = 0
2.  brk(NULL)                           = 0x564e0e721000
3.  arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc8cadcbe0) = -1 EINVAL (Invalid argument)
4.  access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
5.  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
6.  fstat(3, {st_mode=S_IFREG|0644, st_size=57643, ...}) = 0
7.  mmap(NULL, 57643, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f0d3feb6000
8.  close(3)                            = 0
9.  openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
10. read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0"..., 832) = 832
11. pread64(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\0\0\0\0\0"..., 784, 64) = 784
12. pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) = 32
13. pread64(3,
    "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68,
    880) = 68
14. fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0
15. mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
    0x7f0d3feb4000
16. pread64(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\0\0\0\0\0"..., 784, 64) = 784
17. pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) = 32
18. pread64(3,
    "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68,
    880) = 68
19. mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f0d3fcc2000
20. mprotect(0x7f0d3fce7000, 1847296, PROT_NONE) = 0
21. mmap(0x7f0d3fce7000, 1540096, PROT_READ|PROT_EXEC,
    MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7f0d3fce7000
22. mmap(0x7f0d3fe5f000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
    0x19d000) = 0x7f0d3fe5f000
23. mmap(0x7f0d3feaa000, 24576, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f0d3feaa000
24. mmap(0x7f0d3feb0000, 13528, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f0d3feb0000
25. close(3)                            = 0
26. arch_prctl(ARCH_SET_FS, 0x7f0d3feb5540) = 0
27. mprotect(0x7f0d3feaa000, 12288, PROT_READ) = 0
28. mprotect(0x564e0ccf8000, 4096, PROT_READ) = 0
```

29. mprotect(0x7f0d3fef2000, 4096, PROT_READ) = 0
30. munmap(0x7f0d3feb6000, 57643) = 0
31. brk(NULL) = 0x564e0e721000
32. brk(0x564e0e742000) = 0x564e0e742000
33. read(0, test
34. "t", 1) = 1
35. read(0, "e", 1) = 1
36. read(0, "s", 1) = 1
37. read(0, "t", 1) = 1
38. read(0, "\n", 1) = 1
39. pipe([3, 4]) = 0
40. pipe([5, 6]) = 0
41. openat(AT_FDCWD, "test", O_WRONLY|O_CREAT, 0600) = 7
42. clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 174 attached
43. , child_tidptr=0x7f0d3feb5810) = 174
44. [pid 174] close(4 <unfinished ...>
45. [pid 173] close(3 <unfinished ...>
46. [pid 174] <... close resumed>) = 0
47. [pid 173] <... close resumed>) = 0
48. [pid 174] close(5 <unfinished ...>
49. [pid 173] close(6 <unfinished ...>
50. [pid 174] <... close resumed>) = 0
51. [pid 173] <... close resumed>) = 0
52. [pid 174] execve("child", ["child", "7", "3", "6"], 0x7ffc8cadccc8 /* 18 vars */ <unfinished ...>
53. [pid 173] read(0, <unfinished ...>
54. [pid 174] <... execve resumed>) = 0
55. [pid 174] brk(NULL) = 0x562b2680d000
56. [pid 174] arch_prctl(0x3001 /* ARCH_??? */, 0x7fffe0e280c0) = -1 EINVAL (Invalid argument)
57. [pid 174] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
58. [pid 174] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 4
59. [pid 174] fstat(4, {st_mode=S_IFREG|0644, st_size=57643, ...}) = 0
60. [pid 174] mmap(NULL, 57643, PROT_READ, MAP_PRIVATE, 4, 0) = 0x7fc92851d000
61. [pid 174] close(4) = 0
62. [pid 174] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 4
63. [pid 174] read(4, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0"..., 832) = 832
64. [pid 174] pread64(4, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
65. [pid 174] pread64(4, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
66. [pid 174] pread64(4, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68
67. [pid 174] fstat(4, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0
68. [pid 174] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc92851b000
69. [pid 174] pread64(4, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
70. [pid 174] pread64(4, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
71. [pid 174] pread64(4, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68
72. [pid 174] mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 4, 0) = 0x7fc928329000
73. [pid 174] mprotect(0x7fc92834e000, 1847296, PROT_NONE) = 0
74. [pid 174] mmap(0x7fc92834e000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x25000) = 0x7fc92834e000
75. [pid 174] mmap(0x7fc9284c6000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x19d000) = 0x7fc9284c6000
76. [pid 174] mmap(0x7fc928511000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x1e7000) = 0x7fc928511000

```
77. [pid  174] mmap(0x7fc928517000, 13528, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fc928517000
78. [pid  174] close(4)                 = 0
79. [pid  174] arch_prctl(ARCH_SET_FS, 0x7fc92851c540) = 0
80. [pid  174] mprotect(0x7fc928511000, 12288, PROT_READ) = 0
81. [pid  174] mprotect(0x562b260d7000, 4096, PROT_READ) = 0
82. [pid  174] mprotect(0x7fc928559000, 4096, PROT_READ) = 0
83. [pid  174] munmap(0x7fc92851d000, 57643) = 0
84. [pid  174] dup2(7, 1)               = 1
85. [pid  174] read(3, meow
86. <unfinished ...>
87. [pid  173] <... read resumed>"m", 1)   = 1
88. [pid  173] read(0, "e", 1)          = 1
89. [pid  173] read(0, "o", 1)          = 1
90. [pid  173] read(0, "w", 1)          = 1
91. [pid  173] read(0, "\n", 1)         = 1
92. [pid  173] write(4, "\5\0\0\0", 4)    = 4
93. [pid  174] <... read resumed>"\5\0\0\0", 4) = 4
94. [pid  173] write(4, "meow\0", 5)      = 5
95. [pid  174] brk(NULL <unfinished ...>
96. [pid  173] read(5,  <unfinished ...>
97. [pid  174] <... brk resumed>)        = 0x562b2680d000
98. [pid  174] brk(0x562b2682e000)       = 0x562b2682e000
99. [pid  174] read(3, "meow\0", 5)      = 5
100.[pid  174] write(6, "\377\377\377\377", 4) = 4
101.[pid  173] <... read resumed>"\377\377\377\377", 4) = 4
102.[pid  174] read(3,  <unfinished ...>
103.[pid  173] fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
104.[pid  173] write(1, "meow has no ';' or '.' in the en"..., 34meow has no ';' or '.' in the end
105.) = 34
106.[pid  173] read(0,
```

Описание системных вызовов

execve() – исполняет программу с ключами, возвращает 0 при успешном выполнении и -1 в случае ошибки

brk() – устанавливает конец сегмента данных в значение NULL, возвращает указатель на начало новой области памяти.

openat() – открывает файл относительно дескриптора указанного каталога с правами доступа, возвращает новый файловый дескриптор, иначе -1.

fstat() – заполняет структуру, указанную вторым аргументом информацией об файле с заданным файловым дескриптором. Возвращает 0 в случае успешного выполнения.

close() – закрывает файл с переданным файловым дескриптором, возвращает 0 в случае успешного выполнения.

read() – читает байты данных из файла с заданным файловым дескриптором в буффер указанный вторым аргументом, возвращает число успешно считанных байт.

mprotect() – контролирует доступ к области памяти, начинающейся с заданного адреса, длины переданного количества байт.

arch_prctl() – устанавливает специфичное для архитектуры состояние, возвращает 0 в случае успешного выполнения.

access() – проверяет права доступа к файлу, или же просто проверяет существует ли файл, возвращает 0 в случае успешного выполнения, иначе -1.

pread64() – выполняет чтение из файла, начиная с заданной позиции, возвращает количество прочитанных байт или -1 в случае ошибки.

clone() – создает новый процесс, позволяет процессу-потомку разделять части их контекста выполнения с вызывающим процессом. В случае успеха возвращает файловый дескриптор процесса-потомка, иначе -1.

dup2() – создает дубликат файлового дескриптора, возвращает новый дескриптор или -1 в случае ошибки.

mmap() – отображает файл на память, возвращает указатель на область с отраженными данными или -1 в случае ошибки.

munmap() – снимает отражение из заданной области памяти, с указателем на начало памяти и заданной длиной байт. Возвращает 0 в случае успешного выполнения

write() – записывает байты из буффера (второй аргумент) в файл с файловым дескриптором. Возвращает число успешно записанных байт.

Вывод

В результате выполнения данной лабораторной работы я приобрела важные практические навыки диагностики программного обеспечения. Я поняла, что утилита strace – хороший инструмент для отслеживания системных вызовов. Она очень полезна при отладке и тестировании программ, что пригодится мне в будущем. Протокол этой утилиты сначала кажется громоздкой и запутанной, но на деле, разобравшись в системных вызовах, он оказывается весьма хорошо читаемым, причем протокол утилиты можно заранее отредактировать с помощью различных ключей и подать на вывод только интересующую информацию.