Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

**Приобретение практических навыков диагностики работы программного обеспечения**

Студент: Речинская Ангелина Юрьевна

Группа: М80-206Б-20

Преподаватель: Соколов Андрей Алексеевич

Дата: 21.12.2021

Оценка: 5

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**1. Постановка задачи**

При выполнении последующих лабораторных работ необходимо продемонстрировать ключевые системные вызовы, которые в них используются.

Используемые утилиты: strace.

**2. Описание используемых утилит**

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

Для удобства работы с протоколом утилиты можно использовать следующие ключи:

· -o file – Перенаправить протокол утилиты в файл file

· -e trace=filters – Указать выражения, по которым будут фильтроваться системные вызовы. Например -e trace=write,%process задаёт фильтрацию по системным вызовам write и по группе системных вызовов, связанных с межпроцессорным взаимодействием.

· -f – Отслеживать системные вызовы в дочерних процессах

· -y – Заменить в протоколе все файловые дескрипторы на имена соответствующих им файлов (где возможно).

· -p file – Отслеживать только обращения к файлу file

· -k – Отображать стек вызовов

**3. Пример использования утилит**

Лабораторная работа 2 – взаимодействие между процессами

303 execve("./var13", ["./var13"], 0x7ffff9f4bc38 /\* 19 vars \*/) = 0

303 arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffe7744fc0) = -1 EINVAL (Invalid argument)

303 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\0"..., 832) = 832

303 arch\_prctl(ARCH\_SET\_FS, 0x7ff188521380) = 0

303 pipe([3, 4]) = 0

303 pipe([5, 6]) = 0

303 pipe([7, 8]) = 0

303 pipe([9, 10]) = 0

303 pipe([11, 12]) = 0

303 pipe([13, 14]) = 0

303 clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7ff188521650) = 304

303 clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD <unfinished ...>

304 dup2(9, 0) = 0

304 dup2(12, 1) = 1

304 execve("child1", ["child1"], 0x7fffe77450a8 /\* 19 vars \*/ <unfinished ...>

303 <... clone resumed>, child\_tidptr=0x7ff188521650) = 305

305 dup2(11, 0) = 0

305 dup2(14, 1) = 1

305 execve("child2", ["child2"], 0x7fffe77450a8 /\* 19 vars \*/ <unfinished ...>

303 read(0, <unfinished ...>

304 <... execve resumed>) = 0

304 arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffc6b46630) = -1 EINVAL (Invalid argument)

304 read(10, "\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\0"..., 832) = 832

304 arch\_prctl(ARCH\_SET\_FS, 0x7fdb02ef1540) = 0

304 read(0, <unfinished ...>

305 <... execve resumed>) = 0

305 arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffec726750) = -1 EINVAL (Invalid argument)

305 read(9, "\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\0"..., 832) = 832

305 arch\_prctl(ARCH\_SET\_FS, 0x7f352ad51540) = 0

305 read(0, <unfinished ...>

303 <... read resumed>"H", 1) = 1

303 read(0, " ", 1) = 1

303 read(0, "L", 1) = 1

303 read(0, "L", 1) = 1

303 read(0, " ", 1) = 1

303 read(0, "O", 1) = 1

303 read(0, "\n", 1) = 1

303 write(10, "\6\0\0\0", 4) = 4

304 <... read resumed>"\6\0\0\0", 4) = 4

303 write(10, "H LL O", 6 <unfinished ...>

304 write(1, "\6\0\0\0", 4 <unfinished ...>

303 <... write resumed>) = 6

304 <... write resumed>) = 4

305 <... read resumed>"\6\0\0\0", 4) = 4

303 wait4(-1, <unfinished ...>

304 read(0, <unfinished ...>

305 write(1, "\6\0\0\0", 4 <unfinished ...>

304 <... read resumed>"H LL O", 6) = 6

305 <... write resumed>) = 4

304 write(1, "h ll o", 6 <unfinished ...>

305 read(0, <unfinished ...>

304 <... write resumed>) = 6

305 <... read resumed>"h ll o", 6) = 6

304 exit\_group(0 <unfinished ...>

305 write(1, "h\_ll\_o", 6 <unfinished ...>

304 <... exit\_group resumed>) = ?

305 <... write resumed>) = 6

304 +++ exited with 0 +++

305 exit\_group(0) = ?

305 +++ exited with 0 +++

303 <... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 304

303 --- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=304, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

303 wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 305

303 read(13, "\6\0\0\0", 4) = 4

303 read(13, "h\_ll\_o", 6) = 6

303 write(1, "h\_ll\_o\n", 7) = 7

303 exit\_group(0) = ?

303 +++ exited with 0 +++

**4. Выводы**

Утилиты strace и ltrace – простые и надежные инструменты для отслеживания системных вызовов. Они полезны при отладке и тестировании программ. Несмотря на то, что на первый взгляд протоколы этих утилит кажутся чересчур объёмными и непонятными, при помощи различных ключей можно отфильтровать их «на свой вкус».

Утилита strace позволяет отслеживать большинство системных вызовов. Но если требуется пронаблюдать за более узкими аспектами работы программ, то на помощь придёт ltrace (для отслеживания вызовов динамических библиотек), ftrace (для отслеживания работы операционной системы), perf, SystemTap и многие другие.

Данные утилиты пригодились мне для изучения принципов работы системных вызовов и поиска ошибок в моих лабораторных работах.