МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 "Компьютерные науки и прикладная математика"

Кафедра 806 "Вычислительная математика и программирование"

Лабораторная работа №8

По курсу «Операционные системы»

Студент: Никольский К.Г.

Группа: М8О-208Б-23

Преподаватель: Миронов Е. С.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Что такое strace?**

**strace —** это мощный диагностический и отладочный инструмент в Unix-подобных операционных системах, включая Linux. Он позволяет отслеживать и записывать системные вызовы, которые выполняет процесс, а также сигналы, которые он получает. Системные вызовы — это точки взаимодействия между пользовательским программным обеспечением и ядром операционной системы. Используя **strace**, можно получить подробную информацию о том, какие файлы открывает программа, какие сетевые соединения устанавливает, какие ошибки возникают при выполнении и многое другое. Это особенно полезно для отладки программ, диагностики проблем с производительностью и анализа поведения процессов в реальном времени.

**Листинг работы strace и комментарии**

*execve("./OS\_LABS", ["./OS\_LABS"], 0x7ffc51d4e160 /\* 48 vars \*/) = 0*

Программа OS\_LABS запускается с помощью системного вызова execve. Здесь передаются аргументы командной строки (["./OS\_LABS"]) и переменные окружения (48 vars).

*brk(NULL) = 0x564808390000*

Системный вызов brk используется для управления кучей (heap). Здесь он возвращает текущее значение указателя на конец кучи.

*arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffa0902c60) = -1 EINVAL (Недопустимый аргумент)*

arch\_prctl пытается установить архитектуру-специфичные параметры, но возвращает ошибку EINVAL

*mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fa569bca000*Выделяется область памяти размером 8192 байта с помощью mmap для внутренних нужд программы.

*access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (Нет такого файла или каталога)*

Программа проверяет наличие файла /etc/ld.so.preload, который может содержать список библиотек для предварительной загрузки.

*openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3*

Открывается кэш динамических библиотек /etc/ld.so.cache. Этот файл используется для ускорения поиска библиотек.

*newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=77843, ...}, AT\_EMPTY\_PATH) = 0*

Получается информация о файле /etc/ld.so.cache с помощью newfstatat.

*mmap(NULL, 77843, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fa569bb6000*

Содержимое кэша динамических библиотек отображается в память с помощью mmap.

*close(3) = 0*

Файл /etc/ld.so.cache закрывается, так как он больше не нужен.

*openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3*

Открывается основная библиотека C (libc.so.6), которая необходима для работы программы.

*read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832*

*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*

*pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48*

*pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68*

*Считываются заголовки и дополнительная информация о библиотеке.*

*newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2220400, ...}, AT\_EMPTY\_PATH) = 0*

Получается информация о файле библиотеки libc.so.6.

*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*

*mmap(NULL, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fa56998d000*

Библиотека libc.so.6 отображается в память с помощью mmap. Флаг MAP\_DENYWRITE защищает файл от записи.

*mprotect(0x7fa5699b5000, 2023424, PROT\_NONE) = 0*

Область памяти, соответствующая библиотеке, защищается от доступа с помощью mprotect.

*mmap(0x7fa5699b5000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7fa5699b5000*

Область памяти, соответствующая коду библиотеки, отображается с правами на чтение и выполнение.

*mmap(0x7fa569b4a000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7fa569b4a000*

Область памяти, соответствующая данным библиотеки, отображается с правами на чтение.

*mmap(0x7fa569ba3000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7fa569ba3000*

Область памяти, соответствующая данным библиотеки, отображается с правами на чтение и запись.

*mmap(0x7fa569ba9000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fa569ba9000*

Выделяется дополнительная память для библиотеки.

*close(3) = 0*

Файл библиотеки libc.so.6 закрывается, так как она уже отображена в память.

*mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fa56998a000*

Выделяется дополнительная память для внутренних нужд программы.

*arch\_prctl(ARCH\_SET\_FS, 0x7fa56998a740) = 0*

Устанавливается базовый адрес сегмента FS для использования в программе.

*set\_tid\_address(0x7fa56998aa10) = 38930*

Устанавливается адрес для хранения идентификатора потока.

*set\_robust\_list(0x7fa56998aa20, 24) = 0*

*rseq(0x7fa56998b0e0, 0x20, 0, 0x53053053) = 0*

*mprotect(0x7fa569ba3000, 16384, PROT\_READ) = 0*

*mprotect(0x564807601000, 4096, PROT\_READ) = 0*

*mprotect(0x7fa569c04000, 8192, PROT\_READ) = 0*

Области память защищаются от записи

*prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0*

Устанавливается лимит размера стека для процесса.

*munmap(0x7fa569bb6000, 77843) = 0*

Освобождается память, отображенная для кэша динамических библиотек.

*getrandom("\xbc\xa1\x77\x71\x08\x2e\x32\xa8", 8, GRND\_NONBLOCK) = 8*

*brk(NULL) = 0x564808390000*

*brk(0x5648083b1000) = 0x5648083b1000*

Увеличивается размер кучи.

*newfstatat(0, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x1), ...}, AT\_EMPTY\_PATH) = 0*

**Начало блока взаимодействия с пользователем**

*read(0, 1.txt*

*"1.txt\n", 1024) = 6*

Считывается строка из стандартного ввода

*unlink("/dev/shm/sem.semaphore\_write") = -1 ENOENT (Нет такого файла или каталога)*

*unlink("/dev/shm/sem.semaphore\_read") = -1 ENOENT (Нет такого файла или каталога)*

Программа пытается удалить семафоры

*openat(AT\_FDCWD, "/dev/shm/sem.semaphore\_write", O\_RDWR|O\_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)*

*getrandom("\x4a\x00\x6e\xe9\x7a\x9a\x26\xee", 8, GRND\_NONBLOCK) = 8*

*newfstatat(AT\_FDCWD, "/dev/shm/sem.Yuua78", 0x7fffa09028f0, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)*

*openat(AT\_FDCWD, "/dev/shm/sem.Yuua78", O\_RDWR|O\_CREAT|O\_EXCL, 0666) = 3*

*write(3, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32*

Создается новый семафор с именем sem.Yuua78 в директории /dev/shm и записываются начальные данные

*mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7fa569c03000*

*link("/dev/shm/sem.Yuua78", "/dev/shm/sem.semaphore\_write") = 0*

Создается символическая ссылка на семафор с именем sem.semaphore\_write.

*newfstatat(3, "", {st\_mode=S\_IFREG|0664, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0*

*unlink("/dev/shm/sem.Yuua78") = 0*

Удаляется временный файл семафора sem.*Yuua78.*

*close(3) = 0*

*openat(AT\_FDCWD, "/dev/shm/sem.semaphore\_read", O\_RDWR|O\_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)*

*getrandom("\xda\xf7\x8c\x56\x0c\xd9\xff\x7c", 8, GRND\_NONBLOCK) = 8*

*newfstatat(AT\_FDCWD, "/dev/shm/sem.W36GF6", 0x7fffa09028f0, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)*

*openat(AT\_FDCWD, "/dev/shm/sem.W36GF6", O\_RDWR|O\_CREAT|O\_EXCL, 0666) = 3*

*write(3, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32*

*mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7fa569bc9000*

*link("/dev/shm/sem.W36GF6", "/dev/shm/sem.semaphore\_read") = 0*

*newfstatat(3, "", {st\_mode=S\_IFREG|0664, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0*

*unlink("/dev/shm/sem.W36GF6") = 0*

*close(3) = 0*

*openat(AT\_FDCWD, "/dev/shm/virtual\_memory", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 3*

*ftruncate(3, 10240) = 0*

*Создается файл разделяемой памяти /dev/shm/virtual\_memory и устанавливается размер*

*openat(AT\_FDCWD, "1.txt", O\_RDONLY) = 4*

Открывается файл 1.txt для чтения.

*clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fa56998aa10) = 38934*

**Создается дочерний процесс с помощью clone.**

*mmap(NULL, 10240, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7fa569bc6000*

*futex(0x7fa569bc9000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANY) = 0*

*newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x1), ...}, AT\_EMPTY\_PATH) = 0*

*write(1, "10\n", 310) = 3*

*futex(0x7fa569c03000, FUTEX\_WAKE, 1) = 1*

*write(1, "15\n", 315) = 3*

*futex(0x7fa569c03000, FUTEX\_WAKE, 1) = 1*

*write(1, "20\n", 320) = 3*

*futex(0x7fa569c03000, FUTEX\_WAKE, 1) = 1*

*write(1, "30\n", 330) = 3*

*futex(0x7fa569c03000, FUTEX\_WAKE, 1) = 1*

*write(1, "40\n", 340) = 3*

*futex(0x7fa569c03000, FUTEX\_WAKE, 1) = 1*

*munmap(0x7fa569bc6000, 1024) = 0*

*wait4(38934, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 38934*

Родительский процесс ожидает завершения дочернего процесса.

*--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=38934, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---*

*munmap(0x7fa569bc9000, 32) = 0*

*munmap(0x7fa569c03000, 32) = 0*

*unlink("/dev/shm/sem.semaphore\_write") = 0*

*unlink("/dev/shm/sem.semaphore\_read") = 0*

*close(3) = 0*

*unlink("/dev/shm/virtual\_memory") = 0*

*close(4) = 0*

Удаляются все семафоры, разделяемая память, освобождается отображённая память

*exit\_group(0) = ?*

*+++ exited with 0 +++*

**Комментарии к листингу**

Если обобщить вывод, то он разделён на 3 блока: запуск, работа и завершение работы.

1. Сначала с помощью различных системных вызовов инициализируется программа, подгружаются библиотеки, устанавливаются нужные значения.
2. Затем программа начинает выполняться. Происходит первый явный вызов кода из программы — считывается название файла. Затем происходит создание семафоров, открытые файла, создание дочернего процесса и вывод значений
3. И наконец, программа завершает работу: родительский процесс ждёт завершения дочернего, удаляются все семафоры, разделяемая память, освобождается отображенная память и программа выходит.

**Вывод**

В данной лабораторной работе **strace** был использован для анализа работы конкретного процесса. Были проанализированы все системные вызовы, которые были сделаны процессом, а также их аргументы и возвращаемые значения. Это позволило получить глубокое понимание того, как процесс взаимодействует с операционной системой, и выявить потенциальные области для оптимизации или устранения ошибок.