Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

Студент: Чирикова П. С.

Группа: М8О–201Б–21

Вариант: -

Преподаватель: Миронов Е. С.

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Постановка задачи**

**Цель работы**

Приобретение практических навыков диагностики работы программного обеспечения на примере 3 лабораторной работы.

**Задание**

Провести диагностику работы 3 лабораторной работы при помощи strace, объяснить результат работы strace.

**Вывод strace**

polina@polina-Vostro-3400:~/OS/cmake-build-debug/lab3$ strace ./lab3

execve("./lab3", ["./lab3"], 0x7ffdf18a1400 /\* 56 vars \*/) = 0

brk(NULL) = 0x56493cc67000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffdfb1f1d30) = -1 EINVAL (Недопустимый аргумент)

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

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

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=80266, ...}) = 0

mmap(NULL, 80266, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f35a9b66000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libpthread.so.0", 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\220q\0\0\0\0\0\0"..., 832) = 832

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0{E6\364\34\332\245\210\204\10\350-\0106\343="..., 68, 824) = 68

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=157224, ...}) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f35a9b64000

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0{E6\364\34\332\245\210\204\10\350-\0106\343="..., 68, 824) = 68

mmap(NULL, 140408, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f35a9b41000

mmap(0x7f35a9b47000, 69632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x6000) = 0x7f35a9b47000

mmap(0x7f35a9b58000, 24576, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x17000) = 0x7f35a9b58000

mmap(0x7f35a9b5e000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1c000) = 0x7f35a9b5e000

mmap(0x7f35a9b60000, 13432, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f35a9b60000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libstdc++.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`\341\t\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=1956992, ...}) = 0

mmap(NULL, 1972224, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f35a995f000

mprotect(0x7f35a99f5000, 1290240, PROT\_NONE) = 0

mmap(0x7f35a99f5000, 987136, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x96000) = 0x7f35a99f5000

mmap(0x7f35a9ae6000, 299008, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x187000) = 0x7f35a9ae6000

mmap(0x7f35a9b30000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1d0000) = 0x7f35a9b30000

mmap(0x7f35a9b3e000, 10240, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f35a9b3e000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libgcc\_s.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\3405\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=104984, ...}) = 0

mmap(NULL, 107592, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f35a9944000

mmap(0x7f35a9947000, 73728, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x3000) = 0x7f35a9947000

mmap(0x7f35a9959000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x15000) = 0x7f35a9959000

mmap(0x7f35a995d000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x18000) = 0x7f35a995d000

close(3) = 0

openat(AT\_FDCWD, "/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\300A\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\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

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\30x\346\264ur\f|Q\226\236i\253-'o"..., 68, 880) = 68

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2029592, ...}) = 0

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\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

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\30x\346\264ur\f|Q\226\236i\253-'o"..., 68, 880) = 68

mmap(NULL, 2037344, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f35a9752000

mmap(0x7f35a9774000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x22000) = 0x7f35a9774000

mmap(0x7f35a98ec000, 319488, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19a000) = 0x7f35a98ec000

mmap(0x7f35a993a000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f35a993a000

mmap(0x7f35a9940000, 13920, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f35a9940000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.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\300\323\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=1369384, ...}) = 0

mmap(NULL, 1368336, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f35a9603000

mmap(0x7f35a9610000, 684032, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xd000) = 0x7f35a9610000

mmap(0x7f35a96b7000, 626688, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xb4000) = 0x7f35a96b7000

mmap(0x7f35a9750000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x14c000) = 0x7f35a9750000

close(3) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f35a9601000

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f35a95fe000

arch\_prctl(ARCH\_SET\_FS, 0x7f35a95fe740) = 0

mprotect(0x7f35a993a000, 16384, PROT\_READ) = 0

mprotect(0x7f35a9750000, 4096, PROT\_READ) = 0

mprotect(0x7f35a995d000, 4096, PROT\_READ) = 0

mprotect(0x7f35a9b30000, 45056, PROT\_READ) = 0

mprotect(0x7f35a9b5e000, 4096, PROT\_READ) = 0

mprotect(0x56493b3d4000, 4096, PROT\_READ) = 0

mprotect(0x7f35a9ba7000, 4096, PROT\_READ) = 0

munmap(0x7f35a9b66000, 80266) = 0

set\_tid\_address(0x7f35a95fea10) = 38012

set\_robust\_list(0x7f35a95fea20, 24) = 0

rt\_sigaction(SIGRTMIN, {sa\_handler=0x7f35a9b47bf0, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_SIGINFO, sa\_restorer=0x7f35a9b55420}, NULL, 8) = 0

rt\_sigaction(SIGRT\_1, {sa\_handler=0x7f35a9b47c90, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_RESTART|SA\_SIGINFO, sa\_restorer=0x7f35a9b55420}, NULL, 8) = 0

rt\_sigprocmask(SIG\_UNBLOCK, [RTMIN RT\_1], NULL, 8) = 0

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

brk(NULL) = 0x56493cc67000

brk(0x56493cc88000) = 0x56493cc88000

futex(0x7f35a9b3e6bc, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

futex(0x7f35a9b3e6c8, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

read(0,

**Описание работы**

Polina, [19.03.2023 13:50]

execve(const char \*filename, char \*const argv [], char \*const envp[]);

выполняет программу, заданную параметром filename. Программа должна быть или двоичным исполняемым файлом, или скриптом, начинающимся со строки вида "#! интерпретатор [аргументы]". argv -- это массив строк, аргументов новой программы. envp -- это массив строк в формате key=value, которые передаются новой программе в качестве окружения (environment). Как argv, так и envp завершаются нулевым указателем. При успешном завершении execve() не возвращает управление, при ошибке возвращается -1

✅ brk(void \*end\_data\_segment);

brk устанавливает конец сегмента данных в значение, указанное в аргументе end\_data\_segment, когда это значение является приемлимым, система симулирует нехватку памяти и процесс не достигает своего максимально возможного размера сегмента данных (см. setrlimit(2)). (https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=setrlimit&category=2)

В случае успеха brk возвращает ноль. В случае ошибки возвращается -1

✅ arch\_prctl(int code, unsigned long addr)

arch\_prctl устанавливает специфичное для данной архитектуры состояние процесса или треда. Параметр code выбирает подфункцию и передаёт ей аргумент addr .

✅ access(const char \*pathname, int 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. mode -- это маска, состоящая из одного или более флагов

В случае успеха (есть все запрошенные права) возвращается нуль. При ошибке (по крайней мере один запрос прав из mode был неудовлетворен, или случилась другая ошибка), возвращается -1

✅ openat(int dirfd, const char \*pathname, int flags);

openat() работает точно так же, как open(2), за исключением

Если путь, указанный в pathname, является относительным, то он интерпретируется относительно каталога, на который ссылается файловый дескриптор dirfd (а не относительно текущего рабочего каталога вызывающего процесса, как это делается с помощью open(2) для относительного пути). возвращает новый дескриптор файла.

✅ fstat(int filedes, struct stat \*buf);

fstat возвращают информацию об указанном файле. Для этого не требуется иметь права доступа к файлу, хотя потребуются права поиска во всех каталогах, указанных в полном имени файла. возвращается информация об открытом файле, на который указывает filedes (1 арг) и заполняет буфер buf (второй арг)

В случае успеха возвращается ноль. При ошибке возвращается -1, а переменной errno присваивается номер ошибки.

✅ mmap(void \*start, size\_t length, int prot , int flags, int fd, off\_t offset); отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым описателем fd, в память, начиная с адреса start. Последний параметр (адрес) необязателен, и обычно бывает равен 0. Настоящее местоположение отраженных данных возвращается самой функцией mmap, и никогда не бывает равным 0.

✅ close(int fd);

close закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно. Если fd является последней копией какого-либо файлового дескриптора, то ресурсы, связанные с ним, освобождаются;

close возвращает ноль при успешном завершении или -1, если произошла ошибка.

✅ read(int fd, void \*buf, size\_t count);

read() пытается записать count байтов файлового описателя fd в буфер, адрес которого начинается с buf. При успешном завершении вызова возвращается количество байтов, которые были считаны (нулевое значение означает конец файла), а позиция файла увеличивается на это значение. В случае ошибки возвращаемое значение равно -1, а переменной errno присваивается номер ошибки.

✅ pread(int fd, void \*buf, size\_t count, off\_t offset);

pread() записывает максимум count байтов из описателя файлов fd, начиная со смещения offset (от начала файла), в буфер buf. Текущая позиция файла не изменяется.

При удачном завершении вызова возвращается количество прочитанных или записанных байтов. При ошибке возвращается -1.

Polina, [19.03.2023 13:50]

✅ mprotect(const void \*addr, size\_t len, int prot);

mprotect контролирует доступ к области памяти. Если программой производится запрещенный этой функцией доступ к памяти, то такая программа получает сигнал SIGSEGV.

Новые установки защиты заменяют предыдущие. Например, если память была ранее помечена PROT\_READ, а mprotect вызывается с помощью параметра prot

При удачном завершении вызова возвращаемое значение равно нулю. При ошибке оно равно -1

✅ set\_tid\_address(int \*tidptr);

set\_tid\_address() устанавливает у вызывающей нити значение clear\_child\_tid равным tidptr.

Если нить, чьё значение clear\_child\_tid не равно NULL, завершается и если нить использовала общую память с другими нитями, то по адресу, указанному в clear\_child\_tid, записывается 0 и ядро выполняет следующую операцию:

Вызов set\_tid\_address() всегда возвращает ID вызывающей нити.

✅ set\_robust\_list(struct robust\_list\_head \*head, size\_t len);

служат для ведения понетевых списков надёжных фьютексов. Данные списки управляются из пользовательского пространства: ядро знает только расположение начала списка. Нить может информировать ядро о расположении своего списка надёжных фьютексов с помощью set\_robust\_list(). Адрес списка надёжных фьютексов нити можно получить с помощью get\_robust\_list().

возвращают ноль при успешном выполнении и код ошибки в противном случае.

✅ futex (void \*futex, int op, int val, const struct timespec \*timeout);

Системный вызов sys\_futex обеспечивает программный метод для ожидания изменения значения указанного адреса памяти и метод пробуждения всех ожидающих на определенном адресе (хотя адреса для одного и того же участка памяти в разных процессах могут быть не идентичны, ядро распределяет их внутренне так, что один участок памяти, распределенный разными методами, будет соответствовать одинм вызовам sys\_futex). Когда операции futex(4) (https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=futex&category=4) заканчиваются без завершения спора в пространстве пользователя, должен быть сделан вызов к ядру для выноса решения.

В зависимости от исполняемой операции возвращаемые значения могут иметь разные смысловые значения.

FUTEX\_WAIT

Возвращает 0, если процесс был пробужден вызовом FUTEX\_WAKE. В случаем истечения срока таймера возвращается ETIMEDOUT. Если футекс не был эквивалентен ожидаемому значению, то операция возвращает EWOULDBLOCK. Сигналы (или другие ложные срабатывания) приводят FUTEX\_WAIT к возврату EINTR.

FUTEX\_WAKE

Возвращает число пробужденных процессов.

FUTEX\_FD

Возвращает новый описатель файлов, ассоциированный с футексом.

✅ prlimit(pid\_t pid, int resource,

const struct rlimit \*\_Nullable new\_limit,

struct rlimit \*\_Nullable old\_limit);

Системный вызов prlimit(), который есть только в Linux объединяет и расширяет функции setrlimit() и getrlimit(). Он может использоваться для задания и получения ограничений ресурсов произвольного процесса.

Аргумент resource имеет тот же смысл что и в setrlimit() и getrlimit().

Если значение аргумента new\_limit не равно NULL, то структура rlimit, на которую он указывает, используется для задания новых значений мягкий и жёстких ограничений для resource. Если значение аргумента old\_limit не равно NULL, то успешный вызов prlimit() помещает текущие значения мягких и жёстких ограничений для resource в структуру rlimit, на которую указывает old\_limit.

В аргументе pid задаётся идентификатор процесса с которым работает вызов. Если pid равно 0, то вызов применяется к вызывающему процессу.

Возврат 0 успех, -1 ошибка

✅ read(int fd, void \*buf, size\_t count);

read() пытается записать count байтов файлового описателя fd в буфер, адрес которого начинается с buf.

Если количество count равно нулю, то read() возвращает это нулевое значение и завершает свою работу. Если count больше, чем SSIZE\_MAX, то результат не может быть определен.

Возвращает При успешном завершении вызова возвращается количество байтов, которые были считаны (нулевое значение означает конец файла), а позиция файла увеличивается на это значение. Ошибка -1

**Вывод**

В результате данной лабораторной работы я выяснила, что при помощи strace можно анализировать работу программы, смотреть на различные системные вызовы их параметры, также можно смотреть системные вызовы по процессам, все это помогает искать неполадки в работе программы и устранять их.