Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу

«Операционные системы»

ДИАГНОСТИКА ПРОГРАММЫ

Студент: Суханов Е.А.
Группа: М8О–206Б–19
Вариант: 17
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2020.

Постановка задачи

Цель работы

Приобретение практических навыков диагностики работы программного обеспечения.

Задание

Необходимо продемонстрировать ключевые системные вызовы, которые используются в лабораторной работе №2. Для этого я буду использовать утилиту strace.

Вывод strace

```
reterer@serv:~/OS/os_exercise_02/buld$ echo "something string" |  strace ./parent less more
execve("./parent", ["./parent", "less", "more"], 0×7ffc2cdf17c0 /* 32 vars */) = 0
brk(NULL)                               = 0×556aa98c6000
arch_prctl(0×3001 /* ARCH_??? */, 0×7ffc04da47b0) = -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=28470, ...}) = 0
mmap(NULL, 28470, PROT_READ, MAP_PRIVATE, 3, 0) = 0×7f0e83086000
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\360q\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\363\377?\332\200\270\27\304d\245n\355Y\377\t\
334"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0×7f0e83084000
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\363\377?\332\200\270\27\304d\245n\355Y\377\t\
334"..., 68, 880) = 68
mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0×7f0e82e92000
mprotect(0×7f0e82eb7000, 1847296, PROT_NONE) = 0
mmap(0×7f0e82eb7000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0×25000) = 0×7f0e82eb7000
mmap(0×7f0e8302f000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0×19d000) =
0×7f0e8302f000
mmap(0×7f0e8307a000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0×1e7000) = 0×7f0e8307a000
mmap(0×7f0e83080000, 13528, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0)
= 0×7f0e83080000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0×7f0e83085540) = 0
mprotect(0×7f0e8307a000, 12288, PROT_READ) = 0
mprotect(0×556aa8ad4000, 4096, PROT_READ) = 0
mprotect(0×7f0e830ba000, 4096, PROT_READ) = 0
munmap(0×7f0e83086000, 28470)           = 0
pipe([3, 4])                            = 0
openat(AT_FDCWD, "less", O_WRONLY|O_CREAT|O_APPEND, 0600) = 5
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0×7f0e83085810) = 66595
close(5)                                = 0
close(3)                                = 0
pipe([3, 5])                            = 0
openat(AT_FDCWD, "more", O_WRONLY|O_CREAT|O_APPEND, 0600) = 6
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0×7f0e83085810) = 66596
close(6)                                = 0
close(3)                                = 0
brk(NULL)                               = 0×556aa98c6000
brk(0×556aa98e7000)                     = 0×556aa98e7000
read(0, "s", 1)                         = 1
read(0, "o", 1)                         = 1
read(0, "m", 1)                         = 1
read(0, "e", 1)                         = 1
read(0, "t", 1)                         = 1
read(0, "h", 1)                         = 1
read(0, "i", 1)                         = 1
read(0, "n", 1)                         = 1
read(0, "g", 1)                         = 1
read(0, " ", 1)                         = 1
read(0, "s", 1)                         = 1
read(0, "t", 1)                         = 1
read(0, "r", 1)                         = 1
read(0, "i", 1)                         = 1
read(0, "n", 1)                         = 1
read(0, "g", 1)                         = 1
read(0, "\n", 1)                        = 1
write(5, "something string\n", 17)      = 17
read(0, "", 1)                          = 0
close(5)                                = 0
```

```
wait4(66596, NULL, 0, NULL)             = 66596
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=66596, si_uid=1000, si_status=0,
si_utime=0, si_stime=0} ---
close(4)                                = 0
wait4(66595, NULL, 0, NULL)             = 66595
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=66595, si_uid=1000, si_status=0,
si_utime=0, si_stime=0} ---
exit_group(0)                           = ?
+++ exited with 0 +++
```

Комментирование вывода strace

```
execve("./parent", ["./parent", "less", "more"], 0×7ffc2cdf17c0 /* 32 vars */) = 0
```

Данный вызов заменяет выполняемый код другим. В данном случае из файла «./parent», во втором аргументе передается массив аргументов для данной программы, третьим аргументом передается массив переменных окружения. Execve Не должна возвращать значение, если она выполнена успешно. Но здесь стоит почему-то ноль.

```
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
```

Проверяет права доступа к файлу, который находится по указанному пути. R_OK – обозначает, что нужно проверить, что файл существует и есть права на чтение. В случае успеха возвращается 0, в случае ошибки - -1. При этом в errno устанавливается соответствующее значение.

```
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
```

Очень похож на open, который объясняется ниже. Но если задан относительный путь, то он считается не от рабочей директории, а от директории, на которгую ссылается дескриптор dirfd.

```
fstat(3, {st_mode=S_IFREG|0644, st_size=28470, ...}) = 0
```

Возвращает информацию об указанном файле. Сначала идет файловый дескриптор, затем указатель на структуру, в которую будет записана информация об файле. В случае успеха возвращается ноль. В случае неуспеха - -1, а код ошибки записывается в errno.

```
mmap(NULL, 28470, PROT_READ, MAP_PRIVATE, 3, 0) = 0×7f0e83086000
```

Отображает файл в оперативную память. Первый аргумент — указатель на область памяти, куда нужно отобразить файл. Затем идет размер области, которую нужно отобразить. Третьим аргументом идет режим защиты памяти, в данном случае, информацию можно только читать. Далее указывается будет ли данная область памяти общая для нескольких процессов или только для текущего. Затем указывается файловый дескриптор и отступ от начала файла. Возвращается указатель на область памяти, где была отображена облать файла.

```
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
```

Чтение из файлового дескриптора 3, в буффер ,указанный втором аргументе, затем идет кол-во байт, которые нужно считать. Возвращается кол-во успешно считанных байт.

```
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
```

Считывает максимум байтов (третий аргумент) из указанного файла (первый аргумент). Данные записываются в буфер (второй аргумент), четвертый аргумент обозначает отступ от начала файла. Возвращается кол-во считанных байт. При этом позиция файла не изменяется.

```
close(3)                                = 0
```
Закрывает указанный файловый дескриптор.

```
mprotect(0×7f0e8307a000, 12288, PROT_READ) = 0
```
Устанавливает права доступа к памяти. Сначала указывается начало области памяти, затем его длина и сам режим доступа. PROT_READ – память доступна только для чтения.

```
munmap(0×7f0e83086000, 28470)           = 0
```
Освобождает область памяти, в которой хранится отображенный фрагмент файла. Первый аргумент — указатель на область памяти. Второй — размер этой области.

```
pipe([3, 4])                            = 0
```
Создает канал. Первый аргумент — указатель на массив из двух элементов, куда запишутся файловые дескрипторы для чтения и записи.

```
clone(child_stack=NULL,flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0×7f0e83085810) = 66595
```
Создаёт новый процесс. Очищает TID для ребенка, но не для родителя, записывает TID ребенка в адрес 0x7f218bd63810. Создает сигнал для родителя SIGCHLD, вызываем при изменении статуса ребенка. Возвращает TID ребенка.

```
write(5, "something string\n", 17)      = 17
```
Записывает в файловый дескриптор содержимое буфера по указанному адресу, имеющий размер, который указывается третьим аргументом. Возвращается кол-во записанных байт.

```
wait4(66596, NULL, 0, NULL)             = 66596
```
Ожидает завершения работы процесса с PID, указанным в первом аргументе. Второй аргумент — указатель на структуру, которая хранит статус завершения процесса. Если он равняется NULL, то такая структура будет создана. Третий аргумент обозначает тип блокировки 0 — ожидание, пока указанный процесс не завершит свою работу. Четвертый аргумент — указатель на структуру rusage, которая хранит информацию об использовании памяти. Возвращается идентификатор завершенного процесса. Если произошла ошибка - -1. Или 0, если мы указали, что данный вызов не должен быть блокирующим, а процесс пока не завершился.

Вывод

Утилита strace позволяет отслеживать системные вызовы, выполняемые процессом. Таким образом можно выявить некоторые трудно-уловимые ошибки.