**Московский авиационный институт**
**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Операционные системы»

# Лабораторная работа № 1

## Тема: Утилита диагностики strace

Студент: Туманов Георгий
Группа: 80-201
Преподаватель: Соколов А.А.
Дата:
Оценка:

Москва, 2019

## 1. Постановка задачи

Освоить утилиту диагностики strace и продемонстрировать её вывод на различных программах.

## 2. Описание

Продемонстрируем вывод strace на максимально простой программе:

```
int main() { return 0; }
```

Получаем:

```
$ strace ./1.out
execve("./1.out", ["./1.out"], 0x7fff1cc98570 /* 51 vars */) =
0
brk(NULL)                                = 0x55fb53b34000
access("/etc/ld.so.nohwcap", F_OK)       = -1 ENOENT (No such
file or directory)
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=150589, ...}) = 0
mmap(NULL,   150589,   PROT_READ,   MAP_PRIVATE,   3,   0)   =
0x7f9d16b54000
close(3)                                 = 0
access("/etc/ld.so.nohwcap", F_OK)       = -1 ENOENT (No such
file or directory)
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\260\34\2\0\0\0
\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
mmap(NULL,             8192,             PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9d16b52000
mmap(NULL,            4131552,            PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f9d16561000
mprotect(0x7f9d16748000, 2097152, PROT_NONE) = 0
mmap(0x7f9d16948000,        24576,        PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,     3,     0x1e7000)     =
0x7f9d16948000
mmap(0x7f9d1694e000,        15072,        PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f9d1694e000
close(3)                                 = 0
arch_prctl(ARCH_SET_FS, 0x7f9d16b534c0) = 0
```

```
mprotect(0x7f9d16948000, 16384, PROT_READ) = 0
mprotect(0x55fb53483000, 4096, PROT_READ) = 0
mprotect(0x7f9d16b79000, 4096, PROT_READ) = 0
munmap(0x7f9d16b54000, 150589)          = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Как оказывается, даже пустая программа на Си делает большое количество системных вызовов. Эти системные вызовы есть ни что иное как инициализация glibc. На остальных примерах эти системные вызовы мы будем опускать.

Посмотрим, какие системные вызовы происходят при выделении памяти:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

#define LINE "\n\n================================\n\n"

int main()
{
     write(STDOUT_FILENO, LINE, sizeof(LINE) / sizeof(char));
     void *a = malloc(10);
     write(STDOUT_FILENO, LINE, sizeof(LINE) / sizeof(char));
     free(a);
     write(STDOUT_FILENO, LINE, sizeof(LINE) / sizeof(char));
     return 0;
}
```

Получаем:

```
$ strace ./2.out
...
write(1, "\n\n================================"..., 38

================================

) = 38
brk(NULL)                               = 0x55c08150c000
brk(0x55c08152d000)                     = 0x55c08152d000
write(1, "\n\n================================"..., 38

================================
```

```
) = 38
write(1, "\n\n==============================", 38

================================

) = 38
exit_group(0)                           = ?
+++ exited with 0 +++
```

Как видно, free не делает каких-либо системных вызовов, так как он просто очищает одну из внутренних структур аллокатора glibc.

Посмотрим, какие системные вызовы делают printf и scanf:

```
#include <stdio.h>

int main()
{
     printf("Hello world!\n");
     printf("Hello world!\n");
     printf("Hello world!\n");
     return 0;
}
```

Получаем:

```
$ strace ./3.out
...
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...})
= 0
brk(NULL)                               = 0x55654fd20000
brk(0x55654fd41000)                     = 0x55654fd41000
write(1, "Hello world!\n", 13Hello world!
)         = 13
write(1, "Hello world!\n", 13Hello world!
)         = 13
write(1, "Hello world!\n", 13Hello world!
)         = 13
exit_group(0)                           = ?
+++ exited with 0 +++
```

Как видно, при первом вызове printf происходят дополнительные системные вызовы. Судя по brk, он выделяет память для хранения промежуточных вычислений (например, для преобразования числа в строку).

```
#include <stdio.h>

int main()
{
     int a;
     scanf("%i", &a);
     return a;
}
```
Получаем:
```
$ strace ./4.out
...
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...})
= 0
brk(NULL)                                = 0x559ea51f2000
brk(0x559ea5213000)                      = 0x559ea5213000
read(0, 99
"99\n", 1024)                  = 3
lseek(0, -1, SEEK_CUR)                   =  -1  ESPIPE  (Illegal
seek)
exit_group(99)                           = ?
+++ exited with 99 +++
```
scanf делает аналогичные вызовы, что и printf. Однако, в конце программы вызывается lseek, необходимый для деинициализации scanf.

Посмотрим системные вызовы при совместном использовании printf и scanf:
```
#include <stdio.h>

int main()
{
     int a, b;
     printf("Enter the number A: ");
     scanf("%i", &a);
     printf("Enter the number B: ");
     scanf("%i", &b);
     printf("Sum of A and B is %i\n",  a + b);
     return 0;
}
```
Получаем:
```
$ strace ./5.out
...
```

```
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...})
= 0
brk(NULL)                                = 0x55799bcb8000
brk(0x55799bcd9000)                      = 0x55799bcd9000
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...})
= 0
write(1, "Enter the number A: ", 20Enter the number A: )    =
20
read(0, 7
"7\n", 1024)                  = 2
write(1, "Enter the number B: ", 20Enter the number B: )    =
20
read(0, 3
"3\n", 1024)                  = 2
write(1, "Sum of A and B is 10\n", 21Sum of A and B is 10
)  = 21
lseek(0, -1, SEEK_CUR)                   =  -1  ESPIPE  (Illegal
seek)
exit_group(0)                            = ?
+++ exited with 0 +++
```

Как видно, инициализация у printf и scanf общая, а lseek также вызывается в конце программы.

### 3. Выводы:

Освоил работу с утилитой диагностики strace и научился использовать её для отладки системных вызовов в программах.