Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №8 по курсу**

**«Операционные системы»**

**ДИАГНОСТИКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Студент: Баталин Дмитрий Андреевич

Группа: М8О–206Б–20

Вариант: 4

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

* Приобретение практических навыков диагностики работы программного обеспечения.

## Задание

При выполнении лабораторных работ по курсе ОС необходимо продемонстрировать системные вызовы, которые в них используются и то, что их использование соответствует варианту ЛР.

По итогам выполнения всех лабораторных работа отчет по данной ЛР должен содержать краткую сводку по исследованию написанных программ.

**Общие сведения о программе**

Для анализа я буду использовать лабораторную работу номер 1. В ней мною были использованы следующие системные вызовы:

1. **wait** – приостанавливает выполнение вызвавшего процесса до тех пор, пока не прекратит выполнение один из его потомков.
2. **pipe** – создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.
3. **close** – закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно.
4. **write -**  записывает до count байтов из буфера buf в файл, на который ссылается файловый описатель fd.
5. **read -** пытается записать count байтов файлового описателя fd в буфер, адрес которого начинается с buf.
6. **fork –** создаёт новый процесс (потомок), который является практически полной копией процесса-родителя, выполняющего этот вызов.
7. **exit -** "немедленно" завершает работу программы.

**Пример работы strace (dtrace)**

moses@MBP-Dmitrij src % sudo dtruss /Users/moses/Documents/cs/OS/build/os\_lab\_1/src/lab1

SYSCALL(args) = return

munmap(0x10488C000, 0x98000) = 0 0

munmap(0x104924000, 0x8000) = 0 0

munmap(0x10492C000, 0x4000) = 0 0

munmap(0x104930000, 0x4000) = 0 0

munmap(0x104934000, 0x5C000) = 0 0

crossarch\_trap(0x0, 0x0, 0x0) = -1 Err#45

fsgetpath(0x16B71F278, 0x400, 0x16B71F258) = 53 0

fsgetpath(0x16B71F288, 0x400, 0x16B71F268) = 14 0

csrctl(0x0, 0x16B71F68C, 0x4) = -1 Err#1

\_\_mac\_syscall(0x18B482908, 0x2, 0x16B71F5D0) = 0 0

csrctl(0x0, 0x16B71F6AC, 0x4) = -1 Err#1

\_\_mac\_syscall(0x18B47F75E, 0x5A, 0x16B71F640) = 0 0

sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16B71EBA8, 0x16B71EBA0, 0x18B4813AF, 0xD) = 0 0

sysctl([CTL\_KERN, 137, 0, 0, 0, 0] (2), 0x16B71EC58, 0x16B71EC50, 0x0, 0x0) = 0 0

open("/\0", 0x20100000, 0x0) = 3 0

openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0) = 4 0

dup(0x4, 0x0, 0x0) = 5 0

fstatat64(0x4, 0x16B71E731, 0x16B71E6A0) = 0 0

openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0) = 6 0

fcntl(0x6, 0x32, 0x16B71E730) = 0 0

dup(0x6, 0x0, 0x0) = 7 0

dup(0x5, 0x0, 0x0) = 8 0

close(0x3) = 0 0

close(0x5) = 0 0

close(0x4) = 0 0

close(0x6) = 0 0

shared\_region\_check\_np(0x16B71ED40, 0x0, 0x0) = 0 0

fsgetpath(0x16B71F290, 0x400, 0x16B71F1E8) = 82 0

fcntl(0x8, 0x32, 0x16B71F290) = 0 0

close(0x8) = 0 0

close(0x7) = 0 0

getfsstat64(0x0, 0x0, 0x2) = 11 0

getfsstat64(0x104AF2040, 0x5D28, 0x2) = 11 0

getattrlist("/\0", 0x16B71F1C0, 0x16B71F130) = 0 0

stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld\_shared\_cache\_arm64e\0", 0x16B71F520, 0x0) = 0 0

dtrace: error on enabled probe ID 1690 (ID 845: syscall::stat64:return): invalid address (0x0) in action #11 at DIF offset 12

stat64("/Users/moses/Documents/cs/OS/build/os\_lab\_1/src/lab1\0", 0x16B71E9D0, 0x0) = 0 0

open("/Users/moses/Documents/cs/OS/build/os\_lab\_1/src/lab1\0", 0x0, 0x0) = 3 0

mmap(0x0, 0x8778, 0x1, 0x40002, 0x3, 0x0) = 0x104B34000 0

fcntl(0x3, 0x32, 0x16B71EAE8) = 0 0

close(0x3) = 0 0

munmap(0x104B34000, 0x8778) = 0 0

stat64("/Users/moses/Documents/cs/OS/build/os\_lab\_1/src/lab1\0", 0x16B71EF40, 0x0) = 0 0

stat64("/usr/lib/libSystem.B.dylib\0", 0x16B71DED0, 0x0) = -1 Err#2

stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0", 0x16B71DE80, 0x0) = -1 Err#2

stat64("/usr/lib/system/libdispatch.dylib\0", 0x16B71BAE0, 0x0) = -1 Err#2

stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/system/libdispatch.dylib\0", 0x16B71BA90, 0x0) = -1 Err#2

stat64("/usr/lib/system/libdispatch.dylib\0", 0x16B71BAE0, 0x0) = -1 Err#2

open("/dev/dtracehelper\0", 0x2, 0x0) = 3 0

ioctl(0x3, 0x80086804, 0x16B71DB28) = 0 0

close(0x3) = 0 0

open("/Users/moses/Documents/cs/OS/build/os\_lab\_1/src/lab1\0", 0x0, 0x0) = 3 0

\_\_mac\_syscall(0x18B482908, 0x2, 0x16B71D1A0) = 0 0

map\_with\_linking\_np(0x16B71D070, 0x1, 0x16B71D0A0) = 0 0

close(0x3) = 0 0

mprotect(0x1046E4000, 0x4000, 0x1) = 0 0

shared\_region\_check\_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0) = 0 0

mprotect(0x104AF0000, 0x40000, 0x1) = 0 0

access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0) = -1 Err#2

bsdthread\_register(0x18B770E34, 0x18B770E28, 0x4000) = 1073742303 0

getpid(0x0, 0x0, 0x0) = 1858 0

shm\_open(0x18B60AF51, 0x0, 0x636A626F) = 3 0

fstat64(0x3, 0x16B71DFB0, 0x0) = 0 0

mmap(0x0, 0x4000, 0x1, 0x40001, 0x3, 0x0) = 0x104B3C000 0

close(0x3) = 0 0

ioctl(0x2, 0x4004667A, 0x16B71E05C) = 0 0

mprotect(0x104B48000, 0x4000, 0x0) = 0 0

mprotect(0x104B54000, 0x4000, 0x0) = 0 0

mprotect(0x104B58000, 0x4000, 0x0) = 0 0

mprotect(0x104B64000, 0x4000, 0x0) = 0 0

mprotect(0x104B68000, 0x4000, 0x0) = 0 0

mprotect(0x104B74000, 0x4000, 0x0) = 0 0

mprotect(0x104B40000, 0x98, 0x1) = 0 0

mprotect(0x104B40000, 0x98, 0x3) = 0 0

mprotect(0x104B40000, 0x98, 0x1) = 0 0

mprotect(0x104B78000, 0x4000, 0x1) = 0 0

mprotect(0x104B7C000, 0x98, 0x1) = 0 0

mprotect(0x104B7C000, 0x98, 0x3) = 0 0

mprotect(0x104B7C000, 0x98, 0x1) = 0 0

mprotect(0x104B40000, 0x98, 0x3) = 0 0

mprotect(0x104B40000, 0x98, 0x1) = 0 0

mprotect(0x104B78000, 0x4000, 0x3) = 0 0

mprotect(0x104B78000, 0x4000, 0x1) = 0 0

mprotect(0x104AF0000, 0x40000, 0x3) = 0 0

mprotect(0x104AF0000, 0x40000, 0x1) = 0 0

objc\_bp\_assist\_cfg\_np(0x18B3A9400, 0x80000018001C1048, 0x0) = -1 Err#5

issetugid(0x0, 0x0, 0x0) = 0 0

mprotect(0x104AF0000, 0x40000, 0x3) = 0 0

getentropy(0x16B71D7C8, 0x20, 0x0) = 0 0

mprotect(0x104AF0000, 0x40000, 0x1) = 0 0

mprotect(0x104AF0000, 0x40000, 0x3) = 0 0

mprotect(0x104AF0000, 0x40000, 0x1) = 0 0

getattrlist("/Users/moses/Documents/cs/OS/build/os\_lab\_1/src/lab1\0", 0x16B71DF40, 0x16B71DF58) = 0 0

access("/Users/moses/Documents/cs/OS/build/os\_lab\_1/src\0", 0x4, 0x0) = 0 0

open("/Users/moses/Documents/cs/OS/build/os\_lab\_1/src\0", 0x0, 0x0) = 3 0

fstat64(0x3, 0x12F604470, 0x0) = 0 0

csrctl(0x0, 0x16B71E16C, 0x4) = -1 Err#1

fgetattrlist(0x3, 0x16B71E210, 0x16B71E190) = 0 0

\_\_mac\_syscall(0x19681064C, 0x2, 0x16B71E190) = 0 0

fcntl(0x3, 0x32, 0x16B71DE28) = 0 0

close(0x3) = 0 0

open("/Users/moses/Documents/cs/OS/build/os\_lab\_1/src/Info.plist\0", 0x0, 0x0) = -1 Err#2

proc\_info(0x2, 0x742, 0xD) = 64 0

csops\_audittoken(0x742, 0x10, 0x16B71E1B0) = 0 0

sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16B71E508, 0x16B71E500, 0x18EB22D3D, 0x15) = 0 0

sysctl([CTL\_KERN, 135, 0, 0, 0, 0] (2), 0x16B71E598, 0x16B71E590, 0x0, 0x0) = 0 0

csops(0x742, 0x0, 0x16B71E63C) = 0 0

mprotect(0x104AF0000, 0x40000, 0x3) = 0 0

pipe(0x0, 0x0, 0x0) = 3 0

pipe(0x0, 0x0, 0x0) = 5 0

pipe(0x0, 0x0, 0x0) = 7 0

fork() = 1861 0

fork() = 1862 0

close(0x3) = 0 0

close(0x8) = 0 0

Hello there IAM TORIEL

hello there iam toriel

read(0x0, "Hello there IAM TORIEL\n\0", 0x63) = 27 0

write(0x4, "Hello there IAM TORIEL\n\0", 0x64) = 100 0

wait4(0xFFFFFFFFFFFFFFFF, 0x0, 0x0) = 1862 0

read(0x7, "hello there iam toriel\n\0", 0x64) = 100 0

write(0x1, "hello there iam toriel\n\0", 0x64) = 100 0

close(0x4) = 0 0

close(0x7) = 0 0

**После начала ожидания программой пользовательского ввода я открыл еще два трассировщика на два дочерних узла:**

moses@MBP-Dmitrij ~ % sudo dtruss -p 1861

SYSCALL(args) = return

write(0x6, "hello there iam toriel\n\0", 0x64) = 100 0

close(0x3) = 0 0

close(0x6) = 0 0

moses@MBP-Dmitrij ~ % sudo dtruss -p 1862

SYSCALL(args) = return

write(0x8, "hello there iam toriel\n\0", 0x64) = 100 0

close(0x5) = 0 0

**Найденные мной системные вызовы внутри strace**

1. **openat**(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0) – работает так же как и open, но получает на вход файловый дескриптор, чтобы искать файлы по локальному пути относительно заданного файлового дескриптора.

2. В **fsgetpath**() возвращает путь в предоставленном вызывающей стороной буфере strict\_buf длины, указанной buflen, связанном с объектом файловой системы, идентифицируемым fsid и obj\_id. fsid — это указатель на структуру, которая идентифицирует файловую систему, которой принадлежит объект.

3. **sysctl** используется для изменения параметров ядра во время выполнения.

4. **dup** – дублирует файловый дескриптор

5. **proc**\_**info** – получает некоторую нужную программе информацию о процессе.

6. Вызов **mmap** отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым описателем fd, в память, начиная с адреса start.

7. Системный вызов **munmap** удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти" (invalid memory reference).

8. **fcntl** выполняет одну из различных дополнительных операций над файловым дескриптором fd.

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

10. **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.

11. **pipe** создает пару файловых описателей

**Вывод**

Процесс выполнения программы на С++ обычно включает следующие шаги:

1. Загрузка исполняемого файла в память.
2. Загрузка динамических библиотек (если они используются).
3. Инициализация статических переменных.
4. Выделение памяти для динамических переменных и структур данных.
5. Выполнение кода программы.
6. Освобождение выделенных ресурсов и завершение программы.

Примерно такую последовательность мы и видим, когда запускаем strace. Мы видим, что есть огромное множество системных вызовов, которые мы внутри своей программы не вызывали. Программа сама очищает память, занимает ее, использует файловые дескрипторы. Это и есть процессы загрузки и инициализации, которые происходят внутри системных библиотеку и динамических библиотек.

Но среди огромного количества системных вызовов есть и те, которые я действительно вызывал внутри кода. Благодаря strace я удостоверился, что моя программа работает правильно и все действия в ней совершаются так, как я и задумывал.