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 ПРОГРАМММНОГО ОБЕСПЕЧЕНИЯ**

Студент: Салихов Тимур Русланович

Группа: М8О–212Б–22

Вариант: 1

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 курсу ОС необходимо продемонстрировать ключевые системные вызовы, которые в них используются и то, что их использование соответствует варианту ЛР.*

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

**Общие сведения о программе**

Для диагностики используется лабораторная работа № 3. Программа компилируется из файла main.cpp. Помимо этого, есть программа calculator.cpp, которая запускается при помощи exec. Обе программы собираются при помощи системы сборки CMake. В программе используются следующие системные вызовы:

1. **fork** - создает дочерний процесс, повторяющий программу родительского.
2. **exec** - позволяет запустить исполняемый файл изнутри программы.
3. **open** - открытие файла
4. **сlose** - закрытие файла
5. **mmap -** создать маппинг файла
6. **munmap -** удалить маппинг файла
7. **sem\_open -** открыть семафор
8. **sem\_close -** закрыть семафор

**Общий метод и алгоритм решения**.

Диагностика программы производится в операционной системе macOS Monterey при помощи утилиты dtrace. Утилита отображает все системные вызовы, сгенерированные программой.

**Основные файлы программы**

**Вывод dtrace:**

SYSCALL(args) = return

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

bsdthread\_register(0x7FF8174D2F5C, 0x7FF8174D2F48, 0x2000) = 1073742303 0

shm\_open(0x7FF8173A0F5D, 0x0, 0x1739F78A) = 3 0

fstat64(0x3, 0x7FF7B1DDD280, 0x0) = 0 0

mmap(0x0, 0x2000, 0x1, 0x40001, 0x3, 0x0) = 0x10E228000 0

close(0x3) = 0 0

ioctl(0x2, 0x4004667A, 0x7FF7B1DDD334) = 0 0

mprotect(0x10E22F000, 0x1000, 0x0) = 0 0

mprotect(0x10E234000, 0x1000, 0x0) = 0 0

mprotect(0x10E235000, 0x1000, 0x0) = 0 0

mprotect(0x10E23A000, 0x1000, 0x0) = 0 0

mprotect(0x10E22A000, 0x90, 0x1) = 0 0

mprotect(0x10E22A000, 0x90, 0x3) = 0 0

mprotect(0x10E22A000, 0x90, 0x1) = 0 0

mprotect(0x10E23B000, 0x1000, 0x1) = 0 0

mprotect(0x10E23C000, 0x90, 0x1) = 0 0

mprotect(0x10E23C000, 0x90, 0x3) = 0 0

mprotect(0x10E23C000, 0x90, 0x1) = 0 0

mprotect(0x10E22A000, 0x90, 0x3) = 0 0

mprotect(0x10E22A000, 0x90, 0x1) = 0 0

mprotect(0x10E23B000, 0x1000, 0x3) = 0 0

mprotect(0x10E23B000, 0x1000, 0x1) = 0 0

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

getentropy(0x7FF7B1DDD210, 0x20, 0x0) = 0 0

getentropy(0x7FF7B1DDD270, 0x40, 0x0) = 0 0

getpid(0x0, 0x0, 0x0) = 1161 0

stat64("/AppleInternal\0", 0x7FF7B1DDD850, 0x0) = -1 Err#2

csops\_audittoken(0x489, 0x10, 0x7FF7B1DDD390) = -1 Err#22

proc\_info(0x2, 0x489, 0xD) = 64 0

csops\_audittoken(0x489, 0x10, 0x7FF7B1DDD460) = -1 Err#22

sysctlbyname(kern.osvariant\_status, 0x15, 0x7FF7B1DDD898, 0x7FF7B1DDD8A0, 0x0) = 0 0

csops(0x489, 0x0, 0x7FF7B1DDD8D4) = 0 0

sysctlbyname(kern.system\_version\_compat, 0x1A, 0x0, 0x0, 0x7FF7B1DDD904) = 0 0

mprotect(0x10E126000, 0x100000, 0x1) = 0 0

open("memory.txt\0", 0x602, 0x1B6) = 3 0

ftruncate(0x3, 0x400, 0x0) = 0 0

mmap(0x0, 0x400, 0x3, 0x40001, 0x3, 0x0) = 0x10E244000 0

close(0x3) = 0 0

sem\_open(0x10E122F65, 0x200, 0x1FF) = 3 0

fork() = 1164 0

getrlimit(0x1008, 0x7FF7B1DDE890, 0x0) = 0 0

fstat64(0x0, 0x7FF7B1DDE878, 0x0) = 0 0

ioctl(0x0, 0x4004667A, 0x7FF7B1DDE8C4) = 0 0

12 13 41

read\_nocancel(0x0, "12 13 41\n\0", 0x1000) = 9 0

1 2 3 4

read\_nocancel(0x0, "1 2 3 4\n\0", 0x1000) = 8 0

read\_nocancel(0x0, "\0", 0x1000) = 0 0

sem\_post(0x3, 0x0, 0x0) = 0 0

sem\_close(0x3, 0x0, 0x0) = 0 0

munmap(0x10E244000, 0x400) = 0 0

wait4(0x0, 0x7FF7B1DDEA34, 0x0) = 1164 0

**Пояснение**

1). open("memory.txt\0", 0x602, 0x1B6) = 3 0

Открытие файла для обмена данными

2). ftruncate(0x3, 0x400, 0x0) = 0 0

Изменение длины файла

3). mmap(0x0, 0x400, 0x3, 0x40001, 0x3, 0x0) = 0x10E244000 0

Маппирование файла

4). close(0x3) = 0 0

Закрытие файла

5). sem\_open(0x10E122F65, 0x200, 0x1FF) = 3 0

Создание семафора

6). fork() = 1164 0

Форк процесса

7). 12 13 41

read\_nocancel(0x0, "12 13 41\n\0", 0x1000) = 9 0

1 2 3 4

read\_nocancel(0x0, "1 2 3 4\n\0", 0x1000) = 8 0

read\_nocancel(0x0, "\0", 0x1000) = 0 0

Чтение данных из терминала

8). sem\_post(0x3, 0x0, 0x0) = 0 0

Пост семафора

9). sem\_close(0x3, 0x0, 0x0) = 0 0

Закрытие семафора

10). munmap(0x10E244000, 0x400) = 0 0

Закрытие мапа

11). wait4(0x0, 0x7FF7B1DDEA34, 0x0) = 1164 0

Ожидание завершения дочернего процесса

**Вывод**

Было диагностировано выполнение программы из лабораторной работы 3. В результате диагностики было определено, что происходят именно те вызовы, которые и требуются. В процессе выполнения работы мне пришлось изучить аналог strace для macOS. Кроме того выяснилось, что для корректной работы утилиты, необходимо было отключить System Integrity Protection.