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: Климов Иван Павлович

Группа: М8О–212Б–22

Вариант: 10

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)**

ivanklimov@MacBook-Air-Ivan-2 src % sudo dtruss /Users/ivanklimov/Desktop/os\_lab\_1/src/main

SYSCALL(args) = return

Создайте 2 файла

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

bsdthread\_register(0x1A028401C, 0x1A0284010, 0x4000) = 1073742303 0

shm\_open(0x1A014CF55, 0x0, 0x2BA0000) = 3 0

fstat64(0x3, 0x16D25E0E0, 0x0) = 0 0

mmap(0x0, 0x4000, 0x1, 0x40001, 0x3, 0x0) = 0x102CB4000 0

close(0x3) = 0 0

ioctl(0x2, 0x4004667A, 0x16D25E18C) = 0 0

mprotect(0x102CC0000, 0x4000, 0x0) = 0 0

mprotect(0x102CCC000, 0x4000, 0x0) = 0 0

mprotect(0x102CD0000, 0x4000, 0x0) = 0 0

mprotect(0x102CDC000, 0x4000, 0x0) = 0 0

mprotect(0x102CE0000, 0x4000, 0x0) = 0 0

mprotect(0x102CEC000, 0x4000, 0x0) = 0 0

mprotect(0x102CB8000, 0x90, 0x1) = 0 0

mprotect(0x102CB8000, 0x90, 0x3) = 0 0

mprotect(0x102CB8000, 0x90, 0x1) = 0 0

mprotect(0x102CF0000, 0x4000, 0x1) = 0 0

mprotect(0x102CF4000, 0x90, 0x1) = 0 0

mprotect(0x102CF4000, 0x90, 0x3) = 0 0

mprotect(0x102CF4000, 0x90, 0x1) = 0 0

mprotect(0x102CB8000, 0x90, 0x3) = 0 0

mprotect(0x102CB8000, 0x90, 0x1) = 0 0

mprotect(0x102CF0000, 0x4000, 0x3) = 0 0

mprotect(0x102CF0000, 0x4000, 0x1) = 0 0

objc\_bp\_assist\_cfg\_np(0x1A0114300, 0x8000000000201048, 0x0) = -1 Err#5

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

getentropy(0x16D25DFA8, 0x20, 0x0) = 0 0

getentropy(0x16D25DFF8, 0x40, 0x0) = 0 0

getpid(0x0, 0x0, 0x0) = 1660 0

stat64("/AppleInternal\0", 0x16D25E6F0, 0x0) = -1 Err#2

csops\_audittoken(0x67C, 0x7, 0x16D25E220) = 0 0

proc\_info(0x2, 0x67C, 0xD) = 64 0

csops\_audittoken(0x67C, 0x7, 0x16D25E2E0) = 0 0

sysctlbyname(kern.osvariant\_status, 0x15, 0x16D25E758, 0x16D25E750, 0x0) = 0 0

csops(0x67C, 0x0, 0x16D25E77C) = 0 0

mprotect(0x102BAC000, 0x100000, 0x1) = 0 0

write(0x1, "\320\241\320\276\320\267\320\264\320\260\320\271\321\202\320\265 2 \321\204\320\260\320\271\320\273\320\260\n\0", 0x1E) = 30 0

one

read(0x0, "one\n\0", 0x64) = 4 0

two

Введите строки

read(0x0, "two\n\0", 0x64) = 4 0

open("one\n\0", 0x201, 0x180) = 3 0

open("two\n\0", 0x201, 0x180) = 4 0

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

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

fork() = 1675 0

fork() = 1676 0

close(0x5) = 0 0

close(0x7) = 0 0

write(0x1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\320\270\n\0", 0x1C) = 28 0

one

read(0x0, "one\n\210\0", 0x64) = 4 0

write(0x6, "one\n\0", 0x4) = 4 0

two

read(0x0, "two\n\223\0", 0x64) = 4 0

write(0x6, "two\n\223\0", 0x4) = 4 0

three

read(0x0, "three\n\0", 0x64) = 6 0

write(0x6, "three\n\0", 0x6) = 6 0

four

read(0x0, "four\n\0", 0x64) = 5 0

write(0x6, "four\n\0", 0x5) = 5 0

five

read(0x0, "five\n\0", 0x64) = 5 0

write(0x6, "five\n\0", 0x5) = 5 0

six

read(0x0, "six\n\0", 0x64) = 4 0

write(0x6, "six\n\0", 0x4) = 4 0

seven

read(0x0, "seven\n\0", 0x64) = 6 0

write(0x6, "seven\n\0", 0x6) = 6 0

eight

read(0x0, "eight\n\0", 0x64) = 6 0

write(0x6, "eight\n\0", 0x6) = 6 0

nine

read(0x0, "nine\n\0", 0x64) = 5 0

write(0x8, "nine\n\0", 0x5) = 5 0

ten

read(0x0, "ten\n\0", 0x64) = 4 0

write(0x6, "ten\n\0", 0x4) = 4 0

^D

конец работы :)

read(0x0, "\0", 0x64) = 0 0

write(0x6, "\0", 0x0) = 0 0

read(0x0, "\n\0", 0x64) = 1 0

close(0x6) = 0 0

close(0x8) = 0 0

wait4(0xFFFFFFFFFFFFFFFF, 0x0, 0x0) = 1676 0

write(0x1, "\320\272\320\276\320\275\320\265\321\206 \321\200\320\260\320\261\320\276\321\202\321\213 :)\n\0", 0x1B) = 27 0

close(0x3) = 0 0

close(0x4) = 0 0

thread\_selfid(0x0, 0x0, 0x0) = 20794 0

**Для отслеживания дочерних процессов:**

ivanklimov@MacBook-Air-Ivan-2 ~ % sudo dtruss -p 1676

Password:

SYSCALL(args) = return

write(0x1, "n\0", 0x1) = 1 0

write(0x1, "n\0", 0x1) = 1 0

write(0x1, "\n\0", 0x1) = 1 0

read(0x0, "\0", 0x64) = 0 0

thread\_selfid(0x0, 0x0, 0x0) = 20809 0

ivanklimov@MacBook-Air-Ivan-2 ~ % sudo dtruss -p 1675

SYSCALL(args) = return

write(0x1, "n\0", 0x1) = 1 0

write(0x1, "\n\0", 0x1) = 1 0

read(0x0, "two\n\0", 0x64) = 4 0

write(0x1, "t\0", 0x1) = 1 0

write(0x1, "w\0", 0x1) = 1 0

write(0x1, "\n\0", 0x1) = 1 0

read(0x0, "three\n\0", 0x64) = 6 0

write(0x1, "t\001\0", 0x1) = 1 0

write(0x1, "h\0", 0x1) = 1 0

write(0x1, "r\0", 0x1) = 1 0

write(0x1, "\n\0", 0x1) = 1 0

read(0x0, "four\n\0", 0x64) = 5 0

write(0x1, "f\0", 0x1) = 1 0

write(0x1, "r\0", 0x1) = 1 0

write(0x1, "\n\0", 0x1) = 1 0

read(0x0, "five\n\0", 0x64) = 5 0

write(0x1, "f\0", 0x1) = 1 0

write(0x1, "v\0", 0x1) = 1 0

write(0x1, "\n\0", 0x1) = 1 0

read(0x0, "six\n\230\0", 0x64) = 4 0

write(0x1, "s\006\0", 0x1) = 1 0

write(0x1, "x\006\0", 0x1) = 1 0

write(0x1, "\n\006\0", 0x1) = 1 0

read(0x0, "seven\n\0", 0x64) = 6 0

write(0x1, "s\006\0", 0x1) = 1 0

write(0x1, "v\006\0", 0x1) = 1 0

write(0x1, "n\006\0", 0x1) = 1 0

write(0x1, "\n\006\0", 0x1) = 1 0

read(0x0, "eight\n\0", 0x64) = 6 0

write(0x1, "g\006\0", 0x1) = 1 0

write(0x1, "h\006\0", 0x1) = 1 0

write(0x1, "t\006\0", 0x1) = 1 0

write(0x1, "\n\0", 0x1) = 1 0

read(0x0, "ten\n\233\0", 0x64) = 4 0

write(0x1, "t\0", 0x1) = 1 0

write(0x1, "n\0", 0x1) = 1 0

write(0x1, "\n\0", 0x1) = 1 0

read(0x0, "\227\006\0", 0x64) = 0 0

thread\_selfid(0x0, 0x0, 0x0) = 20808 0

**Найденные мной системные вызовы внутри strace**

1) **bsdthread\_register(0x1A028401C, 0x1A0284010, 0x4000)** - это системный вызов, выполняемый в процессе, который зарегистрировал поток в ядре операционной системы. Этот вызов, вероятно, связан с созданием нового потока.

**Аргументы вызова:**

* 0x1A028401C: Это может быть идентификатор потока (thread ID) или какой-то другой уникальный идентификатор для идентификации потока.
* 0x1A0284010: Это, вероятно, указатель на стек потока.
* 0x4000: Это размер стека потока в байтах.

**Результат вызова:** 1073742303: Это, вероятно, идентификатор (или код) вновь созданного потока.

0: Это, вероятно, код ошибки, и в данном случае, он указывает на успешное завершение вызова.

2)**fstat64(0x3, 0x16D25E0E0, 0x0)** - это вызов fstat64, который используется для получения информации о файле, на который указывает файловый дескриптор.

**Аргументы вызова:**

* 0x3: Это файловый дескриптор, который указывает на открытый файл. В данном случае, 0x3 - это файловый дескриптор №3.
* 0x16D25E0E0: Это адрес структуры stat64, в которую будет сохранена информация о файле.
* 0x0: Этот аргумент обычно игнорируется в вызове fstat64. Ваш 0x0 означает нулевой указатель.

Результат вызова fstat64 сохраняется в структуре stat64. Эта структура содержит различные атрибуты файла, такие как размер, права доступа, время последнего доступа и другие.

3) **mprotect** контролирует доступ к области памяти. Если программой производится запрещенный этой функцией доступ к памяти, то такая программа получает сигнал SIGSEGV. При удачном завершении вызова возвращаемое значение равно нулю. При ошибке оно равно -1, а переменной errno присваивается номер ошибки.

4)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.

5) **thread\_selfid(0x0, 0x0, 0x0)** идентификатор (ID) текущего потока.

6) 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).

**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.

Когда мы начинаем использовать strace, наблюдаем определенные шаги в последовательности событий. Видим большое количество системных вызовов, которые не запускались в коде нашей программы.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.

Однако среди всех этих системных вызовов можно найти и те, что были инициированы мной непосредственно в коде. Используя strace, я убедился, что моя программа функционирует корректно, и все действия выполняются в ней так, как и было задумано.