ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ 8 ЛАБОРАТОРНОЙ РАБОТЫ**

**ПО ДИСЦИПЛИНЕ «ОПЕРАЦИОННЫЕ СИСТЕМЫ»**

Выполнил(а) студент

группы М8О-208Б-23

Романов Вадим Михайлович

Проверили и приняли:

Живалев Е.А.

Катаев Ю. И.

Москва, 2024

**Тема:** «Диагностика ПО»

**Цели работы:**

* Приобретение практических навыков диагностики работы программного обеспечения

**Задание:**

Провести анализ системных вызовов во всех выполненных лабораторных работах по курсу ОС с целью:

* Выявления ключевых системных вызовов
* Подтверждения соответствия использованных вызовов заданиям работ
* Составления итогового отчета с результатами исследования

**Инструменты для диагностики:**

В зависимости от используемой операционной системы, доступны следующие средства анализа:

Для Windows:

* Отладчик WinDbg
* Набор утилит Sysinternals Suite:
  + Handle.exe - анализ открытых дескрипторов
  + Procmon.exe - мониторинг активности процессов
  + Procexp.exe - расширенный диспетчер процессов

Для Unix-подобных систем:

* strace - основной инструмент трассировки системных вызовов

Strace является мощным инструментом диагностики в Linux-системах, который позволяет:

* Отслеживать взаимодействие программ с ядром через системные вызовы
* Анализировать поведение программы на низком уровне
* Выявлять проблемы производительности и ошибки в работе приложений

Основные опции strace для эффективной диагностики:

* «-o файл» - сохранение вывода в указанный файл
* «-e выражение» - фильтрация системных вызовов по заданному шаблону
* «-f» - отслеживание дочерних процессов
* «-t» - добавление временных меток к каждому вызову
* «-y» - отображение путей для файловых дескрипторов
* «-p pid» - присоединение к работающему процессу
* «-k» - трассировка стека вызовов

**Анализ lab1\_log.txt:**

execve("./lab1", ["./lab1"], 0x7fffe2703ca0 /\* 48 vars \*/) = 0

Здесь запускается исполняемая программа lab1. Первый аргумент – это имя исполняемого файла, второй – массив аргументов (в данном случае только один аргумент, это сам исполняемый файл), третий – окружение.

brk(NULL) = 0x5ee11494b000

Этот вызов возвращает адрес конца сегмента данных программы. Он показывает текущее значение brk, которое указывает на конец памяти, выделенной для программы.

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x76de354d8000

Вызов mmap используется для отображения фрагмента файла или анонимной памяти. Здесь выделяется 8192 байт анонимной памяти.

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (Нет такого файла или каталога)

Программа проверяет, доступен ли файл /etc/ld.so.preload, но он отсутствует.

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

Программа открывает файл /etc/ld.so.cache, который используется загрузчиком динамических библиотек, чтобы кэшировать информацию о загруженных библиотеках.

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

...

mmap(...) = 0x76de35200000

Тут программа открывает библиотеку libc.so.6, содержащую стандартные функции C.

pipe2([3, 4], 0) = 0

Этот вызов создаёт пару связанных файловых дескрипторов (обычно используется для межпроцессного взаимодействия).

clone(child\_stack=NULL, flags=..., child\_tidptr=0x76de354c4a10) = 8599

Вызывается clone, чтобы создать новый поток выполнения (в данном случае, скорее всего, для обработки дочернего процесса).

write(1, "\320\222\320\262\320\265...\n", 47) = 47

Программа пишет пользователю сообщение, в данном случае на русском: "Введите строку с числами:" (последовательность кодов символов указывает на UTF-8).

read(0, 1 2 3 4 5)

Программа ожидает ввода данных с клавиатуры, где пользователь вводит числа.

wait4(-1, Дочерний процесс: Введенная строка: 1 2 3 4 5)

Программа ждёт завершения дочернего процесса и получает информацию о завершении.

exit\_group(0) = ?

Программа завершает выполнение с кодом 0, что указывает на успешное завершение.

**Вывод:**

В процессе выполнения лабораторной работы были успешно освоены инструменты диагностики strace и ltrace. Эти утилиты показали себя как эффективные инструменты для отслеживания системных вызовов и анализа работы программ. Хотя изначально вывод утилит может показаться сложным для понимания из-за большого объема информации, использование различных ключей фильтрации (например, -e trace для отбора конкретных системных вызовов) позволяет получить именно те данные, которые необходимы для анализа.