

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №8 по курсу

«Операционные системы»

Тема работы

“Утилита strace”

Студент: Мазепа Илья Алексеевич

Группа: М8О-209Б-23

Преподаватель: Миронов Евгений Сергеевич

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2024

GitHub репозиторий: https://github.com/Tyhyqo/mai_os

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.

Средства диагностики

ОС Unix: strace

Лабораторная работа №1

Описание программы

Программа состоит из трех частей: parent, child_1 и child_2. Родительский процесс создает два дочерних процесса и взаимодействует с ними через каналы (pipe).

Команда для выполнения strace:

```
strace -o strace_lab_1.txt ./parent
```

Анализ strace

1. **execve(...)** — запуск файла parent.
2. **openat(...)** / **mmap(...)** — загрузка и отображение libc.so.6.
3. **pipe2(...)** — создание нескольких каналов для обмена данными.
4. **clone(...)** — запуск двух дочерних процессов.
5. **close(...)** — закрытие неиспользуемых дескрипторов.
6. **read(...)** / **write(...)** — чтение строк из stdin и передача в каналы.
7. **getrandom(...)** — получение случайных данных (идентификаторы).
8. **wait4(...)** — ожидание завершения дочерних процессов, освобождение ресурсов.
9. **exit_group(0)** — завершение программы.

Выводы:

- Показано создание дочерних процессов и взаимодействие через каналы (pipe).
- Используются вызовы clone, read, write и wait4, обеспечивающие передачу данных и синхронизацию.

Лабораторная работа №2

Описание программы

Программа для лабораторной работы №2 состоит из одного файла main.c и создает несколько потоков с помощью pthread_create. Каждый поток обрабатывает данные, используя общую память и системные вызовы для управления сигналами. Передача данных и синхронизация осуществляются с помощью pthread_barrier_t.

Команда для выполнения strace:

```
strace -o strace_lab_2.txt ./main 8
```

Анализ strace

1. newfstatat(...):

- Получает информацию о файле или о связанном объекте по файловому дескриптору (здесь fd=0 или fd=3).
- Возвращает атрибуты файла (например, st_mode, st_size).
- Параметр AT_EMPTY_PATH вместе с пустым путем означает, что fstatat/newfstatat используют уже открытый дескриптор для получения метаданных.

2. rt_sigprocmask(...) / rt_sigaction(...):

- Управляют сигналами (установка или снятие блокировки, обработчиков сигналов).
- sigprocmask` задает/считывает маску блокируемых сигналов.
- sigaction (rt_sigaction) регистрирует обработчик для конкретного сигнала.

3. mmap(...) / mprotect(...) / munmap(...):

- mmap выделяет виртуальную память (обычно под стеки потоков или под другие буферы) или отображает файл в память.
- mprotect меняет права доступа (PROT_READ, PROT_WRITE, PROT_NONE) для уже отображенных страниц.
- munmap освобождает область памяти.

4. clone3(...):

- Создает новый поток или процесс (в зависимости от флагов).

– Здесь видны флаги `CLONE_VM`, `CLONE_FS`, `CLONE_FILES`, `CLONE_SIGHAND`, `CLONE_THREAD` и т. д. — они указывают, что создается поток (shared memory space, дескрипторы, обработчики сигналов и т.д.), а не отдельный процесс.

– `stack=...` (и `stack_size=...`) — новый поток получает собственный стек по адресу, возвращенном `mmap`.

– `child_tid=...` / `parent_tid=...` — механизмы для установки идентификаторов потока.

Выводы:

- Организовано создание нескольких потоков с помощью **`pthread_create`**.

- В `strace` заметны вызовы `clone3`, «`mmap`» и системные вызовы для сигналов, отражающие многопоточность и синхронизацию через `pthread_barrier_t`.

Лабораторная работа №3

Описание программы

Аналогично лабораторной работе №1, но используется `mmaping`, вместо `pipe`, для передачи данных.

Команда для выполнения `strace`:

```
strace -o strace_lab_3.txt ./parent
```

Анализ `strace`

1. `execve(...)`:

– Инициализирует запуск программы `parent`.

– Подгружает динамические библиотеки (`openat`, `mmap`) перед выполнением основного кода.

2. `unlink(...)`:

– Удаляет ранее созданные объекты (например, семафоры или файлы в `/dev/shm/sem.*`).

– Часто используется для очистки окружения перед повторным созданием необходимых ресурсов.

3. `openat(...)` + `ftruncate(...)`:

– Создает или открывает отображаемый файл (шареную память) для межпроцессного взаимодействия.

– `ftruncate` задает размер этого файла.

4. `mmap(...)` / `munmap(...)`:

- Отображает или освобождает разделяемую память, используемую для обмена данными.
- При MAP_SHARED изменения видны всем процессам, подключенным через ту же память.

5. **getrandom(...):**

- Получает случайные данные из генератора случайных чисел ядра (часто используется для генерации уникальных имен).

6. **link(...):**

- Создает новую привязку (имя для ресурса), например, для семафора или временного файла.

7. **clone(...):**

- Порождает дочерние процессы.
 - Настраивает общий адресный пространство, дескрипторы, сигналы в зависимости от флагов.
- #### 8. **futex(...):**
- Управляет блокировками/пробуждениями, часто используется внутри реализации семафоров.
- #### 9. **wait4(...):**
- Родитель ждет завершения дочерних процессов, затем освобождает ресурсы.

Выводы:

- Акцент на обмене через разделяемую память и семафоры.
- mmap, futex и clone подтверждают реализацию межпроцессного взаимодействия.

Лабораторная работа №4

Описание программы

Program1 использует линковку библиотек на этапе компиляции. Program2 динамически загружает библиотеки с помощью dlopen.

Команда для выполнения strace:

```
strace -o strace_lab_4_prog_1.txt ./Program1  
strace -o strace_lab_4_prog_2.txt ./Program2
```

Анализ strace

Program1 (линковка на этапе компиляции):

1. **execve(...)** — запуск исполняемого файла.
2. **openat(...)** — загрузка динамических библиотек (например, libDerivative1.so,

libPi1.so), на которые программа ссылается еще во время компоновки.

3. **mmap(...)** — отображение библиотек в память.

4. **close(...)** — освобождение дескрипторов, не требуемых в дальнейшем.

5. **read(...)** / **write(...)** — ввод команд, вывод результатов.

Program2 (динамическая загрузка с помощью dlopen):

1. **execve(...)** — запуск исполняемого файла.

2. **openat(...)** — поиск библиотек, необходимых для базового окружения (например, libc.so.6).

3. **read(...)** — чтение команд пользователя (например, «1 ...», «2 ...»).

4. **dlopen(...)** / **dlsym(...)** / **dlclose(...)** — динамическая загрузка заданной библиотеки (например, libDerivative1.so, libPi1.so), получение символов функций и выгрузка библиотеки после выполнения.

5. **mmap(...)** — отображение библиотек и других сегментов памяти.

6. **write(...)** — вывод результатов, завершение программы.

Выводы:

- Program1 использует статическую линковку библиотек, Program2 — динамическую.

- В strace видны execve, openat и mmap: загрузка библиотек и обработка команд.

Лабораторные работы №5-7

Описание программы

Программа для лабораторной работы №8 представляет собой расширенную версию управляющего узла, включающую обработку дополнительных типов сообщений и улучшенную устойчивость к сбоям вычислительных узлов. Она взаимодействует с вычислительными узлами через очереди сообщений, обеспечивает мониторинг состояния узлов и выполняет отложенные вычисления.

Команда для выполнения strace:

```
strace -o strace_lab_8.txt ./manager
```

Анализ strace

Анализ strace для лабораторной работы №5–7

1. **execve(...)**

- Запускает управляющий узел manager.

- Загружает динамические библиотеки, необходимые для работы системы.

2. **openat(...)**

– Открывает файлы конфигурации и библиотеки (например, libzmq.so.5, libsodium.so.23).

– Используется при загрузке ZeroMQ, обеспечивающего обмен сообщениями.

3. **mmap(...)**

– Отображает динамические библиотеки и анонимные области памяти.

– Используется для размещения кода и данных в виртуальной памяти процесса.

4. **read(...)**

– Считывает заголовки ELF и другие метаданные из библиотек.

– Применяется во время инициализации системы и при обработке команд пользователя.

5. **close(...)**

– Закрывает файлы библиотек после их отображения в память.

– Освобождает дескрипторы, когда файлы больше не требуются.

6. **mprotect(...)**

– Изменяет права доступа к ранее отображенным сегментам.

– Позволяет защитить код и данные от записи или выполнения.

7. **pread64 / newfstatat** — чтение из файла по смещению, получение метаданных файла.

8. **rseq / set_robust_list / set_tid_address** — используются для низкоуровневой синхронизации потоков (futex и проч.).

9. **getpid** — получение **PID** процесса.

10. **sched_getaffinity** — получение **CPU affinity** процесса.

11. **fcntl** — управление файловыми дескрипторами.

12. **epoll_* / poll** — механизмы многоплексирования ввода-вывода.

13. **rt_sigprocmask** — управление масками сигналов в реальном времени.

14. **futex** — базовая синхронизация потоков на уровне ядра.

15. **eventfd2** — механизм синхронного уведомления процессов.

Выводы:

- Управляющий узел manager взаимодействует с вычислительными узлами (ZeroMQ, libsodium).
- В логах strace преобладают openat, mmap и read (загрузка и чтение библиотек), а также futex, epoll_ctl и другие системные вызовы, отражающие сложную многопоточную среду.

Вывод

- Каждая лабораторная работа демонстрирует различные аспекты применения системных вызовов: от создания процессов и потоков до динамической загрузки библиотек и межпроцессной синхронизации.
- strace помогает проследить механизм работы программ и подтвердить корректность использования системных ресурсов.