**Отчет**

**по лабораторной работе №1.**

**3 семестр**

|  |  |
| --- | --- |
| **Студент:** | **Фирфаров А.С.** |
| **Группа:** | **8O-208Б** |
| **Преподаватель** | **Миронов Е.С.** |

**Цель работы**

Приобретение практических навыков диагностики работы программного обеспечения.

**Задание**

При выполнении последующих лабораторных работ 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 должен содержать краткую сводку по исследованию последующих ЛР.

**Системные вызовы**

**Лабораторная работа 2**

pidt fork() – создание дочернего процесса. Если возвращаемое значение 0 – находимся в дочернем процессе, если > 0 – в родительском.

int pipe(int pipefd[2]) – создание канала для межпроцессорного взаимодействия. Возвращает 0 при успехе, -1 при ошибке.

void exit(int status) – обычное завершение работы программы.

ssize\_t write(int fd, const void \*buf, size\_t count) - производит запись в описатель файла . В случае успешного завершения возвращается количество байтов, которые были. В случае ошибки возвращается -1, а переменной errno присваивается соответствующее значение.

ssize\_t read(int fd, void \*buf, size\_t count)- cчитывает данные файлового описателя . При успешном завершении вызова возвращается количество байтов, которые были считаны, а позиция файла увеличивается на это значение. В случае ошибки возвращаемое значение равно -1, а переменной errno присваивается номер ошибки.

int close(int fd) - закрыть файловый дескриптор.  Возвращает ноль при успешном завершении или -1, если произошла ошибка.

**Лабораторная работа 3**

int open(const char \*pathname, int flags) – открыть файл.

int close(int fd) - закрыть файловый дескриптор.  Возвращает ноль при успешном завершении или -1, если произошла ошибка.

off\_t lseek(int fildes, off\_t offset, int whence) - установить смещение для позиционирования операций чтения/записи. При успешном выполнении возвращает получившееся в результате смещение в батах от начала файла. В противном случае, возвращается значение -1 и errno показывает ошибку.

int pthread\_create(pthread\_t \*restrict thread, const pthread\_attr\_t \*restrict attr, void \*(\*start\_routine)(void\*), void \*restrict arg) – создать поток. При успехе возвращает 0.

int pthread\_join(pthread\_t thread, void \*\*retval) – ожидание завершения потока. При успехе возвращает 0.

**Лабораторная работа 4**

int open(const char \*pathname, int flags) – открыть файл.

int close(int fd) - закрыть файловый дескриптор.  Возвращает ноль при успешном завершении или -1, если произошла ошибка.

int ftruncate(int fd, off\_t length) - устанавливают длину обычного файла с файловым дескриптором fd в length байт. При успешной работе возвращает 0. При ошибке возвращается -1, а переменной errno присваивается номер ошибки.

void \* mmap(void \*start, size\_t length, int prot , int flags, int fd, off\_t offset) - отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым описателем fd, в память, начиная с адреса start. При удачном выполнении mmap возвращает указатель на область с отраженными данными. При ошибке возвращается значение MAP\_FAILED (-1), а переменная errno приобретает соответствующее значение.

pidt fork() – создание дочернего процесса. Если возвращаемое значение 0 – находимся в дочернем процессе, если > 0 – в родительском.

pid\_t wait(int \*status) – ожидает завершения процесса.

void exit(int status) – обычное завершение работы программы.

int munmap(void \*start, size\_t length) – снять отражение. При удачном выполнении возвращаемое значение равно нулю. При ошибке возвращается -1, а переменная errno приобретает соответствующее значение.

**Лабораторная работа 5**

void \*dlopen(const char \*filename, int flag) - загружает динамическую библиотеку, имя которой указано в строке filename, и возвращает прямой указатель на начало динамической библиотеки.

void \*dlsym(void \*handle, char \*symbol) - использует указатель на динамическую библиотеку, возвращаемую dlopen, и оканчивающееся нулем символьное имя, а затем возвращает адрес, указывающий, откуда загружается этот символ. Если символ не найден, то возвращаемым значением является NULL.

const char \*dlerror(void) – просмотреть ошибки, произошедшие при выполнении функций dl.

int dlclose(void \*handle) - уменьшает на единицу счетчик ссылок на указатель динамической библиотеки handle. Если нет других загруженных библиотек, использующих ее символы и если счетчик ссылок принимает нулевое значение, то динамическая библиотека выгружается. Возвращает 0 при удачном завершении и ненулевой результат при ошибке.

**Лабораторная работа 6-8**

int zmq\_msg\_init\_size (zmq\_msg\_t \*msg, size\_t size) – инициализировать сообщение указанного размера. При успехе возвращает 0. В случае ошибки возвращаемое значение равно -1, а переменной errno присваивается номер ошибки.

int zmq\_msg\_send (zmq\_msg\_t \*msg, void \*socket, int flags) – отправить сообщение на сокет. При успехе возвращает количество байт сообщения. В случае ошибки возвращаемое значение равно -1, а переменной errno присваивается номер ошибки.

int zmq\_msg\_close (zmq\_msg\_t \*msg) –выпустить сообщение. При успехе возвращает 0. В случае ошибки возвращаемое значение равно -1, а переменной errno присваивается номер ошибки.

int zmq\_msg\_init (zmq\_msg\_t \*msg) – инициализировать пустое сообщение. При успехе возвращает 0. В случае ошибки возвращаемое значение равно -1, а переменной errno присваивается номер ошибки.

int zmq\_msg\_recv (zmq\_msg\_t \*msg, void \*socket, int flags) – получить сообщение из сокета. При успехе возвращает количество байт сообщения. В случае ошибки возвращаемое значение равно -1, а переменной errno присваивается номер ошибки.

void \*zmq\_ctx\_new () – создать новый контекст. Должен вернуть непрозрачный дескриптор вновь созданному контексту в случае успеха. В противном случае он должен вернуть NULL и присвоить переменной errno номер ошибки.

void \*zmq\_socket (void \*context, int type) – создать сокет из контекста context.

int zmq\_connect (void \*socket, const char \*endpoint) - создать исходящее соединение из сокета. При успехе возвращает 0. В случае ошибки возвращаемое значение равно -1, а переменной errno присваивается номер ошибки.

int zmq\_bind (void \*socket, const char \*endpoint) - принимать входящие соединения через сокет. При успехе возвращает 0. В случае ошибки возвращаемое значение равно -1, а переменной errno присваивается номер ошибки.

int zmq\_close (void \*socket) – закрыть сокет. При успехе возвращает 0. В случае ошибки возвращаемое значение равно -1, а переменной errno присваивается номер ошибки.

int zmq\_ctx\_destroy (void \*context) – уничтожить контекст. При успехе возвращает 0. В случае ошибки возвращаемое значение равно -1, а переменной errno присваивается номер ошибки.

**Выводы**

Использование strace для отслеживания системных вызовов позволяет понять, как программа взаимодействует с операционной системой, какие вызовы на самом деле производятся. Информация о том, какие системные вызовы производит программа, может помочь выяснить, что делает программа, узнать причину неправильной работы программы и исправить ошибки в коде. Strace позволяет отслеживать вызовы прямо во время выполнения программы, что позволяет понять поведение программы и найти возможные ошибки системных вызовов прямо во время их использования, не дожидаясь выполнения всего кода программы. Для эффективного использования strace необходимо знать, что делают различные системные вызовы, какие значения они должны возвращать. Также starce может отпугнуть новичков не слишком дружелюбным интерфейсом. Но изучив системные вызовы и научившись пользоваться strace, можно легко контролировать процесс выполнения программы и находить возникающие ошибки.