Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

**СРЕДСТВА ДИАГНОСТИКИ В ОС**

Студент: Киреев Александр Константинович

Группа: М8О–206Б–19

Вариант: 4

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## 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.

**Общие сведения о программе**

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода. Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float.

Программа компилируется из файла main.c вместе с файлов io.c. Также используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, stdbool.h, sys/types.h, sys/stat.h, fcntl.h, sys/mman.h. В программе используются следующие системные вызовы:

1. **mmap** ­­–– отражает *length* байтов, начиная со смещения *offset*файла (или другого объекта), определенного файловым дескриптором *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.
2. **munmap** ­­–– удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти". При удачном выполнении munmap возвращаемое значение равно нулю. При ошибке возвращается -1, а переменная *errno* приобретает соответствующее значение.
3. **fork ––** создает новый процесс, который является копией родительского процесса, за исключением разных process ID и parent process ID. В случае успеха fork() возвращает 0 для ребенка, число больше 0 для родителя – child ID, в случае ошибки возвращает -1.
4. **read ––** предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернет число прочитанных байт, иначе -1.
5. **write ––** предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернет число записанных байт, иначе -1.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Написать ЛР4.
2. Изучить работу утилиты strace на ОС Ubuntu.
3. Запустить strace с ключом -f и небольшим тестом.

Давайте теперь проанализируем вывод strace.

|  |
| --- |
| В родительском процессе мы отражаем разделяемую память для процесса-отца и процесса-ребенка для буфера данных размера 4096 байт и для буфера сигналов размера 4 байта. Адреса, по которым будут располагаться данные возвращаются из функций mmap. |
| mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED|MAP\_ANONYMOUS, -1, 0) = 0x7fc9a9d95000  mmap(NULL, 4, PROT\_READ|PROT\_WRITE, MAP\_SHARED|MAP\_ANONYMOUS, -1, 0) = 0x7fc9a9d94000 |
| Системный вызов для создания дочернего процесса или потока. |
| clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLDstrace: Process 5404 attached  , child\_tidptr=0x7fc9a9d84850) = 5404 |
| Открыть файл с именем “test\_file\_name” с правом на запись, если он существует, либо создать файл с правами на чтение и запись, если его нет. Данному файлу присвоен файловый дескриптор равный 3. |
| [pid 5404] openat(AT\_FDCWD, "test\_file\_name", O\_WRONLY|O\_CREAT|O\_TRUNC, 0666) = 3 |
| Очистка отраженной разделяемой памяти по указанному адресу. |
| [pid 5403] munmap(0x7fc9a9d95000, 4096 <unfinished ...>  [pid 5403] <... munmap resumed>) = 0  [pid 5403] munmap(0x7fc9a9d94000, 4 <unfinished ...>  [pid 5403] <... munmap resumed>) = 0 |
| Завершение работы процесса-отца и процесса-сына без ошибок. |
| [pid 5403] exit\_group(0) = ?  [pid 5404] exit\_group(0) = ? |
| Загрузка и запуск исполняемой программы по указанному пути с передачей одного аргумента и переменных окружения. |
| execve("./lab4", ["./lab4"], 0x7ffcb89522d8 /\* 44 vars \*/) = 0 |

**Основные файлы программы**

**io.h:**

#pragma once

#include <stdio.h>

#include <ctype.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdbool.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#define BUF\_SIZE 64

#define RF\_ENDL 3

#define RF\_EOF 1

#define RF\_INVALID 2

#define RF\_VALID 0

// проигнорировать невалидную строку

void skip\_str();

// проверка числа на валидность

bool is\_valid(char\* buf);

// считать float с клавиатуры при помощи read

int read\_float(float\* n);

// считать имя файла с клавиатуры при помощи read

void read\_file\_name(char file\_name[BUF\_SIZE]);

**main.c:**

#include <stdio.h>

#include <ctype.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdbool.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <fcntl.h>

#include "io.h"

#define PAGESIZE 4096

#define CHILD\_BLOCKED 0

#define PARENT\_BLOCKED 1

#define KILL 2

#define OK 0

#define FILE\_OPEN\_ERR 1

#define FORK\_ERR 2

#define MMAP\_ERR 3

#define CLOSE\_FILE\_ERR 4

#define NULL\_DIV 5

int mmap\_clear(float\* mf, int\* sig) {

int to\_return = OK;

if (munmap(mf, PAGESIZE) == -1) {

to\_return = MMAP\_ERR;

}

if (munmap(sig, sizeof(int)) == -1) {

to\_return = MMAP\_ERR;

}

return to\_return;

}

int child(char\* file\_name, float\* mf, int\* sig) {

FILE\* fp;

if ((fp = fopen(file\_name, "w")) == NULL) {

perror("unable to open file");

exit(FILE\_OPEN\_ERR);

}

while (\*sig == CHILD\_BLOCKED) {}

while (\*sig != KILL) {

int size = mf[0];

float result = mf[1];

for (int i = 2; i < size + 1; ++i) {

if (mf[i] == 0.0) {

\*sig = KILL;

if (fclose(fp) != 0) {

perror("unable to close file");

exit(CLOSE\_FILE\_ERR);

}

exit(NULL\_DIV);

}

result /= mf[i];

}

fprintf(fp, "%f\n", result);

\*sig = CHILD\_BLOCKED;

while (\*sig == CHILD\_BLOCKED) {}

}

if (fclose(fp) != 0) {

perror("unable to close file");

exit(CLOSE\_FILE\_ERR);

}

exit(OK);

}

int parent(float\* mf, int\* sig) {

int get\_return = RF\_VALID;

while (true) {

int it = 0;

while (true) {

float num;

get\_return = read\_float(&num);

if (get\_return == RF\_INVALID) {

perror("invalid num");

break;

}

mf[it + 1] = num;

it++;

if (get\_return != RF\_VALID) {

break;

}

}

if (get\_return == RF\_EOF) {

break;

} else if (get\_return == RF\_INVALID) {

continue;

}

mf[0] = it;

\*sig = PARENT\_BLOCKED;

while (\*sig == PARENT\_BLOCKED) {}

if (\*sig == KILL) {

if (mmap\_clear(mf, sig) == MMAP\_ERR) {

perror("munmap error");

exit(MMAP\_ERR);

}

exit(NULL\_DIV);

}

}

\*sig = KILL;

if (mmap\_clear(mf, sig) == MMAP\_ERR) {

perror("munmap error");

exit(MMAP\_ERR);

}

exit(OK);

}

int main() {

char file\_name[BUF\_SIZE];

read\_file\_name(file\_name);

float\* mf = (float\*)mmap(NULL, PAGESIZE, PROT\_READ | PROT\_WRITE , MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

if (mf == MAP\_FAILED) {

perror("mmap error");

exit(MMAP\_ERR);

}

int\* sig = (int\*)mmap(NULL, sizeof(int), PROT\_READ | PROT\_WRITE , MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

if (sig == MAP\_FAILED) {

perror("mmap error");

exit(MMAP\_ERR);

}

\*sig = CHILD\_BLOCKED;

int id = fork();

if (id < 0) {

perror("fork error");

exit(FORK\_ERR);

} else if (id == 0) {

child(file\_name, mf, sig);

} else {

parent(mf, sig);

}

return 0;

}

**Пример работы**

kak@MacBookAir-K:~/Рабочий стол/OS/os\_lab\_04/src$ strace -f ./lab4 < test.txt

execve("./lab4", ["./lab4"], 0x7ffcb89522d8 /\* 44 vars \*/) = 0

brk(NULL) = 0x564315f2f000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe1673b210) = -1 EINVAL (Недопустимый аргумент)

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

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

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=67778, ...}) = 0

mmap(NULL, 67778, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fc9a9d85000

close(3) = 0

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

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360\215\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0~\303\347M\250B\312<j\233\242\v!0<\341"..., 68, 880) = 68

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=1995896, ...}) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fc9a9d83000

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2004064, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fc9a9b99000

mprotect(0x7fc9a9bbf000, 1810432, PROT\_NONE) = 0

mmap(0x7fc9a9bbf000, 1495040, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x26000) = 0x7fc9a9bbf000

mmap(0x7fc9a9d2c000, 311296, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x193000) = 0x7fc9a9d2c000

mmap(0x7fc9a9d79000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1df000) = 0x7fc9a9d79000

mmap(0x7fc9a9d7f000, 13408, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fc9a9d7f000

close(3) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fc9a9b97000

arch\_prctl(ARCH\_SET\_FS, 0x7fc9a9d84580) = 0

mprotect(0x7fc9a9d79000, 12288, PROT\_READ) = 0

mprotect(0x564314b22000, 4096, PROT\_READ) = 0

mprotect(0x7fc9a9dc4000, 4096, PROT\_READ) = 0

munmap(0x7fc9a9d85000, 67778) = 0

read(0, "t", 1) = 1

read(0, "e", 1) = 1

read(0, "s", 1) = 1

read(0, "t", 1) = 1

read(0, "\_", 1) = 1

read(0, "f", 1) = 1

read(0, "i", 1) = 1

read(0, "l", 1) = 1

read(0, "e", 1) = 1

read(0, "\_", 1) = 1

read(0, "n", 1) = 1

read(0, "a", 1) = 1

read(0, "m", 1) = 1

read(0, "e", 1) = 1

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

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED|MAP\_ANONYMOUS, -1, 0) = 0x7fc9a9d95000

mmap(NULL, 4, PROT\_READ|PROT\_WRITE, MAP\_SHARED|MAP\_ANONYMOUS, -1, 0) = 0x7fc9a9d94000

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLDstrace: Process 5404 attached

, child\_tidptr=0x7fc9a9d84850) = 5404

[pid 5403] read(0, "4", 1) = 1

[pid 5403] read(0, "2", 1) = 1

[pid 5403] read(0, "\n", 1) = 1

[pid 5404] brk(NULL) = 0x564315f2f000

[pid 5404] brk(0x564315f50000) = 0x564315f50000

[pid 5404] openat(AT\_FDCWD, "test\_file\_name", O\_WRONLY|O\_CREAT|O\_TRUNC, 0666) = 3

[pid 5404] fstat(3, {st\_mode=S\_IFREG|0664, st\_size=0, ...}) = 0

[pid 5403] read(0, "4", 1) = 1

[pid 5403] read(0, " ", 1) = 1

[pid 5403] read(0, "2", 1) = 1

[pid 5403] read(0, " ", 1) = 1

[pid 5403] read(0, "2", 1) = 1

[pid 5403] read(0, "\n", 1) = 1

[pid 5403] read(0, "3", 1) = 1

[pid 5403] read(0, " ", 1) = 1

[pid 5403] read(0, "1", 1) = 1

[pid 5403] read(0, "\n", 1) = 1

[pid 5403] read(0, "6", 1) = 1

[pid 5403] read(0, "4", 1) = 1

[pid 5403] read(0, " ", 1) = 1

[pid 5403] read(0, "2", 1) = 1

[pid 5403] read(0, " ", 1) = 1

[pid 5403] read(0, "2", 1) = 1

[pid 5403] read(0, " ", 1) = 1

[pid 5403] read(0, "2", 1) = 1

[pid 5403] read(0, " ", 1) = 1

[pid 5403] read(0, "2", 1) = 1

[pid 5403] read(0, "\n", 1) = 1

[pid 5403] read(0, "", 1) = 0

[pid 5403] munmap(0x7fc9a9d95000, 4096 <unfinished ...>

[pid 5404] write(3, "42.000000\n1.000000\n3.000000\n4.00"..., 37 <unfinished ...>

[pid 5403] <... munmap resumed>) = 0

[pid 5403] munmap(0x7fc9a9d94000, 4 <unfinished ...>

[pid 5404] <... write resumed>) = 37

[pid 5403] <... munmap resumed>) = 0

[pid 5404] close(3 <unfinished ...>

[pid 5403] exit\_group(0) = ?

[pid 5404] <... close resumed>) = 0

[pid 5404] exit\_group(0) = ?

[pid 5403] +++ exited with 0 +++

+++ exited with 0 +++

**Вывод**

Выполняя данную лабораторную работу, я познакомился с утилитой strace. С ее помощью можно отслеживать системные вызовы, которые использовались в программе во время ее работы. Это позволяет лучше понять, что процесс пытается сделать в заданное время, понять, где в программе что-то идет не так. Особенно это полезно в низкоуровневом программировании, когда часто происходит обращение к ОС для выделения памяти, создания процессов, потоков, каналов, работы с файлами и т.п. В заключении можно сказать, что strace очень удобный инструмент для диагностики и анализа программ.