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: Ильин И. О.

Группа: М8О–206Б–19

Вариант: 2

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**

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

1. **pipe** ­­–– принимает массив из двух целых чисел, в случае успеха массив будет содержать два файловых дескриптора, которые будут использоваться для конвейера, первое число в массиве предназначено для чтения, второе для записи, а так же вернется 0. В случае неуспеха вернется -1.
2. **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.
3. **close ––** принимает файловый дескриптор в качестве аргумента, удаляет файловый дескриптор из таблицы дескрипторов, в случае успеха вернет 0, в случае неуспеха вернет -1.
4. **open ––** создает или открывает файл, если он был создан. В качестве аргументов принимает путь до файла, режим доступа (запись, чтение и т.п.), модификатор доступа ( при создании можно указать права для файла ). Возвращает в случае успеха файловый дескриптор – положительное число, иначе возвращает -1.
5. **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.
6. **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.
7. **waitpid** — приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс, указанный в параметре pid, не завершит выполнение, или пока не появится сигнал, который либо завершает текущий процесс либо требует вызвать функцию-обработчик. Если указанный дочерний процесс к моменту вызова функции уже завершился (так называемый "зомби"), то функция немедленно возвращается. Системные ресурсы, связанные с дочерним процессом, освобождаются.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы утилиты strace в ОС Ubuntu.
2. Запустить утилиту strace с ключом -f на одном из тестов.
3. Проанализировать вывод утилиты.

**Основные файлы программы**

**main.c**

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <stdbool.h>

#define ERR\_MALLOC 3

#define ERR\_REALLOC 4

#define ERR\_READ 5

#define ERR\_WRITE 6

#define ERR\_OPEN 7

#define ERR\_CLOSE 8

#define ERR\_PIPE 9

#define ERR\_FORK 10

#define ERR\_WAIT 11

#define ERR\_DIV\_0 20

#define ERR\_INV\_INT 21

#define ERR\_NO\_STATUS 22

int get\_int\_length(int num) { // Считает длину числа

int length = 0;

if (num < 0) {

++length;

}

do {

num /= 10;

++length;

} while (num != 0);

return length;

}

void int\_to\_string(char\* s, int x) { // Преобразует число в строку

bool is\_negative = x < 0;

if (is\_negative) {

x \*= -1;

}

bool is\_first = true;

char\* p;

do {

if (is\_first) {

p = s;

is\_first = false;

} else {

++p;

}

\*p = '0' + (x % 10);

x /= 10;

} while (x != 0);

if (is\_negative) {

++p;

\*p = '-';

}

while (s < p) {

char tmp = \*s;

\*s = \*p;

\*p = tmp;

++s;

--p;

}

}

void read\_strings(int fd, char\*\*\* ptr\_buffer, size\_t\* res\_size) {

// Считывает строки через пробел до конца ввода

char c; // Текущий считываемый символ

int str\_idx = 0; // Индекс текущего слова

size\_t buffer\_size = 1; // Кол-во строк

char\*\* buffer = malloc(sizeof(char\*) \* buffer\_size);

if (!buffer) { perror("Malloc error"); exit(ERR\_MALLOC); }

int\* lengths = malloc(sizeof(int) \* buffer\_size);

if (!lengths) { perror("Malloc error"); exit(ERR\_MALLOC); }

lengths[0] = 0;

int str\_size = 1; // Длина текущей строки

buffer[0] = malloc(sizeof(char) \* str\_size);

if (!buffer[0]) { perror("Malloc error"); exit(ERR\_MALLOC); }

size\_t read\_bytes = 0;

while ((read\_bytes = read(fd, &c, sizeof(char))) > 0) {

if ((read\_bytes != sizeof(char)) && (read\_bytes != 0)) {

perror("Read error");

exit(ERR\_READ);

}

// Конец ввода слов

if (c == '\n') { break; }

// Следующее слово

if (c == ' ') {

++str\_idx;

str\_size = 1;

continue;

}

// Увеличиваем буфер строк и длин

if (str\_idx >= buffer\_size) {

++buffer\_size;

str\_size = 1;

buffer = realloc(buffer, sizeof(char\*) \* buffer\_size);

if (!buffer) { perror("Realloc error"); exit(ERR\_REALLOC); }

buffer[buffer\_size - 1] = malloc(sizeof(char) \* str\_size);

if (!buffer[buffer\_size - 1]) { perror("Malloc error"); exit(ERR\_MALLOC);}

for (size\_t i = 0; i < str\_size; ++i) {

buffer[buffer\_size - 1][i] = 0;

}

lengths = realloc(lengths, sizeof(int) \* buffer\_size);

if (!lengths) { perror("Realloc error"); exit(ERR\_REALLOC); }

lengths[buffer\_size - 1] = 0;

}

// Увеличиваем длину строки

if (lengths[str\_idx] == str\_size) {

//printf("Increasing str len\n");

str\_size += 16;

buffer[str\_idx] = realloc(buffer[str\_idx], sizeof(char) \* str\_size);

if (!buffer[str\_idx]) { perror("Realloc error"); exit(ERR\_REALLOC);}

}

buffer[str\_idx][lengths[str\_idx]] = c;

++lengths[str\_idx];

}

for (int i = 0; i < buffer\_size; ++i) {

buffer[i] = realloc(buffer[i], sizeof(char) \* lengths[i]);

if (!buffer[i]) { perror("Realloc error"); exit(ERR\_REALLOC); }

buffer[i][lengths[i]] = '\0';

}

\*res\_size = buffer\_size;

\*ptr\_buffer = buffer;

}

// Считывает одну строку

void read\_string(int fd, char\*\* str, size\_t\* size) {

char c; // Текущий считываемый символ

size\_t str\_len = 0;

size\_t str\_size = 1; // Длина текущей строки

char\* buffer = malloc(sizeof(char) \* str\_size);

if (!buffer) { perror("Malloc error"); exit(ERR\_MALLOC); }

size\_t read\_bytes = 0;

while ((read\_bytes = read(fd, &c, sizeof(char))) > 0) {

if ((read\_bytes != sizeof(char)) && (read\_bytes != 0)) {

perror("Read error");

exit(ERR\_READ);

}

if (c == '\n') { break; }

// Увеличиваем длину строки

if (str\_len == str\_size) {

str\_size += 16;

buffer = realloc(buffer, sizeof(char) \* str\_size);

if (!buffer) { perror("Realloc error"); exit(ERR\_REALLOC); }

}

buffer[str\_len] = c;

++str\_len;

}

buffer = realloc(buffer, sizeof(char) \* str\_len);

if (!buffer) { perror("Realloc error"); exit(ERR\_REALLOC); }

buffer[str\_len] = '\0';

\*size = str\_len;

\*str = buffer;

}

// Преобразует массив строк в массив int'ов

void str\_array\_to\_int(char\*\* strs, int nums[], size\_t n) {

for (size\_t i = 0; i < n; ++i) {

char\* p = strs[i];

bool is\_negative = false;

int result = 0;

while(\*p != '\0') {

if (\*p == '-' || \*p == '+') {

if (p != strs[i]) { perror("Invalid int"); exit(ERR\_INV\_INT); }

if (\*p == '-') {

is\_negative = true;

}

} else if ('0' <= \*p && \*p <= '9') {

result = result \* 10 + (\*p - '0');

} else {

perror("Invalid int");

exit(ERR\_INV\_INT);

}

++p;

}

if (is\_negative) { result \*= -1; }

nums[i] = result;

}

}

void write\_to\_fd(int fd, const void \*buf, size\_t nbytes) {

if (write(fd, buf, nbytes) != nbytes) { perror("Write error"); exit(ERR\_WRITE);}

}

void read\_from\_fd(int fd, void \*buf, size\_t nbytes) {

if (read(fd, buf, nbytes) != nbytes) { perror("Read error"); exit(ERR\_READ);}

}

int main() {

int fd[2]; // массив файловых дескрипторов

// создаем pipe

if (pipe(fd) != 0) { perror("Piping error"); exit(ERR\_PIPE); }

int status;

int id = fork();

if (id == -1) {

perror("fork() error\n");

exit(ERR\_FORK);

} else if (id == 0) {

// Ребенок

size\_t path\_size;

size\_t size;

int divident;

int divisor = 1;

// Читаем размер строки и строку

read\_from\_fd(fd[0], &path\_size, sizeof(size\_t));

char\* path = malloc(sizeof(char) \* path\_size);

if (!path) { exit(ERR\_REALLOC); }

read\_from\_fd(fd[0], path, sizeof(char) \* path\_size);

// Читаем числа

read\_from\_fd(fd[0], &size, sizeof(size\_t));

read\_from\_fd(fd[0], &divident, sizeof(int));

for (int i = 0; i < size - 1; ++i) {

int num;

read\_from\_fd(fd[0], &num, sizeof(int));

divisor \*= num;

}

close(fd[0]);

close(fd[1]);

if (divisor == 0) { exit(ERR\_DIV\_0); }

int res = divident / divisor;

int length = get\_int\_length(res);

char s[length];

int\_to\_string(s, res);

int fp = open(path, O\_CREAT | O\_WRONLY); // создаем, если не сущ., только для записи

if (fp == -1) { // Если не получилось открыть файл

free(path);

close(fp);

exit(ERR\_OPEN);

}

write\_to\_fd(fp, s, sizeof(char) \* length);

close(fp);

free(path);

exit(0);

} else {

// Родитель, id = child\_id

char\*\* strs = NULL;

size\_t size = 0;

char\* path;

size\_t path\_size = 0;

printf("[%d] Reading...\n", getpid()); fflush(stdout);

read\_string(0, &path, &path\_size); // Читает имя выходного файла

read\_strings(0, &strs, &size); // Читаем числа до \n в массив char\* и переводим в массив int'ов

int nums[size];

str\_array\_to\_int(strs, nums, size);

printf("[%d] Red all values.\n", getpid()); fflush(stdout);

for (size\_t i = 0; i < size; ++i) {

printf("%zu) %d\n", i, nums[i]);

}

printf("[%d] Writing...\n", getpid()); fflush(stdout);

write\_to\_fd(fd[1], &path\_size, sizeof(size\_t));

write\_to\_fd(fd[1], path, sizeof(char) \* path\_size);

write\_to\_fd(fd[1], &size, sizeof(size\_t));

for (size\_t i = 0; i < size; ++i) {

write\_to\_fd(fd[1], nums + i, sizeof(int));

}

close(fd[0]);

close(fd[1]);

printf("[%d] Wrote all values.\n", getpid());

free(strs);

if (waitpid(id, &status, 0) == -1) {

perror("wait() error\n");

exit(ERR\_WAIT);

}

if (WIFEXITED(status)) { // Если ребенок завершился со статусом завершения

int exit\_code = WEXITSTATUS(status);

switch (exit\_code) {

case ERR\_DIV\_0:

perror("error: you cant divide by zer0\n");

exit(exit\_code);

break;

case ERR\_OPEN:

perror("error: can't open file\n");

exit(exit\_code);

break;

case ERR\_CLOSE:

perror("error: can't close file\n");

exit(exit\_code);

break;

case ERR\_WRITE:

perror("error: can't write to file\n");

exit(exit\_code);

break;

case ERR\_READ:

perror("error: can't read from file\n");

exit(exit\_code);

break;

case ERR\_REALLOC:

perror("error: realloc error\n");

exit(exit\_code);

break;

default:

printf("[%d] Child ended with status: %d\n", getpid(), status);

fflush(stdout);

break;

}

} else {

perror("error: child exit with no status\n");

exit(ERR\_NO\_STATUS);

}

}

return 0;

}

**Пример работы**

parallels@parallels:~/Desktop/os\_lab\_02/src$ strace -f ./a.out <../test/in01.txt

execve("./a.out", ["./a.out"], 0x7fffab6aaec8 /\* 64 vars \*/) = 0

brk(NULL) = 0x56295f854000

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

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

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=95109, ...}) = 0

mmap(NULL, 95109, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f0d1b437000

close(3) = 0

access("/etc/ld.so.nohwcap", F\_OK) = -1 ENOENT (No such file or directory)

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\260\34\2\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2030544, ...}) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f0d1b435000

mmap(NULL, 4131552, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f0d1ae37000

mprotect(0x7f0d1b01e000, 2097152, PROT\_NONE) = 0

mmap(0x7f0d1b21e000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f0d1b21e000

mmap(0x7f0d1b224000, 15072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f0d1b224000

close(3) = 0

arch\_prctl(ARCH\_SET\_FS, 0x7f0d1b4364c0) = 0

mprotect(0x7f0d1b21e000, 16384, PROT\_READ) = 0

mprotect(0x56295d881000, 4096, PROT\_READ) = 0

mprotect(0x7f0d1b44f000, 4096, PROT\_READ) = 0

munmap(0x7f0d1b437000, 95109) = 0

pipe([3, 4]) = 0

clone(strace: Process 32580 attached

child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f0d1b436790) = 32580

[pid 32580] read(3, <unfinished ...>

[pid 32579] getpid() = 32579

[pid 32579] fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(136, 2), ...}) = 0

[pid 32579] brk(NULL) = 0x56295f854000

[pid 32579] brk(0x56295f875000) = 0x56295f875000

[pid 32579] write(1, "[32579] Reading...\n", 19[32579] Reading...

) = 19

[pid 32579] read(0, ".", 1) = 1

[pid 32579] read(0, ".", 1) = 1

[pid 32579] read(0, "/", 1) = 1

[pid 32579] read(0, "t", 1) = 1

[pid 32579] read(0, "e", 1) = 1

[pid 32579] read(0, "s", 1) = 1

[pid 32579] read(0, "t", 1) = 1

[pid 32579] read(0, "/", 1) = 1

[pid 32579] read(0, "o", 1) = 1

[pid 32579] read(0, "u", 1) = 1

[pid 32579] read(0, "t", 1) = 1

[pid 32579] read(0, "0", 1) = 1

[pid 32579] read(0, "1", 1) = 1

[pid 32579] read(0, ".", 1) = 1

[pid 32579] read(0, "t", 1) = 1

[pid 32579] read(0, "x", 1) = 1

[pid 32579] read(0, "t", 1) = 1

[pid 32579] read(0, "\n", 1) = 1

[pid 32579] read(0, "1", 1) = 1

[pid 32579] read(0, "0", 1) = 1

[pid 32579] read(0, " ", 1) = 1

[pid 32579] read(0, "2", 1) = 1

[pid 32579] read(0, "\n", 1) = 1

[pid 32579] getpid() = 32579

[pid 32579] write(1, "[32579] Red all values.\n", 24[32579] Red all values.

) = 24

[pid 32579] write(1, "0) 10\n", 60) 10

) = 6

[pid 32579] write(1, "1) 2\n", 51) 2

) = 5

[pid 32579] getpid() = 32579

[pid 32579] write(1, "[32579] Writing...\n", 19[32579] Writing...

) = 19

[pid 32579] write(4, "\21\0\0\0\0\0\0\0", 8) = 8

[pid 32580] <... read resumed> "\21\0\0\0\0\0\0\0", 8) = 8

[pid 32579] write(4, "../test/out01.txt", 17) = 17

[pid 32579] write(4, "\2\0\0\0\0\0\0\0", 8 <unfinished ...>

[pid 32580] brk(NULL <unfinished ...>

[pid 32579] <... write resumed> ) = 8

[pid 32580] <... brk resumed> ) = 0x56295f854000

[pid 32579] write(4, "\n\0\0\0", 4 <unfinished ...>

[pid 32580] brk(0x56295f875000 <unfinished ...>

[pid 32579] <... write resumed> ) = 4

[pid 32580] <... brk resumed> ) = 0x56295f875000

[pid 32579] write(4, "\2\0\0\0", 4 <unfinished ...>

[pid 32580] read(3, <unfinished ...>

[pid 32579] <... write resumed> ) = 4

[pid 32580] <... read resumed> "../test/out01.txt", 17) = 17

[pid 32579] close(3 <unfinished ...>

[pid 32580] read(3, <unfinished ...>

[pid 32579] <... close resumed> ) = 0

[pid 32580] <... read resumed> "\2\0\0\0\0\0\0\0", 8) = 8

[pid 32579] close(4 <unfinished ...>

[pid 32580] read(3, <unfinished ...>

[pid 32579] <... close resumed> ) = 0

[pid 32580] <... read resumed> "\n\0\0\0", 4) = 4

[pid 32579] getpid( <unfinished ...>

[pid 32580] read(3, <unfinished ...>

[pid 32579] <... getpid resumed> ) = 32579

[pid 32580] <... read resumed> "\2\0\0\0", 4) = 4

[pid 32579] write(1, "[32579] Wrote all values.\n", 26 <unfinished ...>

[32579] Wrote all values.

[pid 32580] close(3 <unfinished ...>

[pid 32579] <... write resumed> ) = 26

[pid 32580] <... close resumed> ) = 0

[pid 32579] wait4(32580, <unfinished ...>

[pid 32580] close(4) = 0

[pid 32580] openat(AT\_FDCWD, "../test/out01.txt", O\_WRONLY|O\_CREAT, 000) = 3

[pid 32580] write(3, "5", 1) = 1

[pid 32580] close(3) = 0

[pid 32580] exit\_group(0) = ?

[pid 32580] +++ exited with 0 +++

<... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 32580

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=32580, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

getpid() = 32579

write(1, "[32579] Child ended with status:"..., 35[32579] Child ended with status: 0

) = 35

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В ходе выполнения данной лабораторной работы я научился работать с утилитой strace. Она позволяет отслеживать все системные вызовы, которые использовались программой в ходе выполнения, что позволяет производит дебаг и анализ поведения программы на более низком уровне, находить ошибки.