

Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №1
по курсу «Операционные системы»
III Семестр

Студент:	Короткевич Л. В.
Группа:	М80-208Б-19
Преподаватель:	Миронов Е.С
Оценка:	
Дата:	

1. Постановка задачи

Приобретение практических навыков работы программного обеспечения.

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.

Используемое средство диагностики — утилита **strace**.

2. Метод решения

Для тестирования утилиты **strace** мной была выбрана программа из ЛР №2.

Программа компилируется из файла **main.c**. В ней происходит запуск двух дочерних процессов **child1**, **child2**, которые компилируются из **child1.c** и **child2.c**. Заголовочные файлы: **<stdlib.h>**, **<stdio.h>**, **<unistd.h>**, **<fcntl.h>**, **<string.h>**, **<sys/wait.h>**.

Используемые системные вызовы в **main.c**:

int pipe(int filedес[3]);

pipe создает пару файловых дескрипторов, указывающих на запись **inode** именованного канала, и помещает их в массив, на который указывает **filedes**. **filedes[0]** предназначен для чтения, а **filedes[1]** предназначен для записи, **filedes[2]** – информации об ошибках.

void exit(int status);

Функция **exit()** приводит к обычному завершению программы, и величина **status & 0377** (least significant byte of status) возвращается процессу-родителю.

pid_t fork(void);

fork создает процесс-потомок, который отличается от родительского только значениями **PID** (идентификатор процесса) и **PPID** (идентификатор родительского процесса), а также тем фактом, что счетчики использования ресурсов установлены в 0. Блокировки файлов и сигналы, ожидающие обработки, не наследуются.

int close(int fd);

close закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно. Все блокировки, находящиеся на соответствующем файле, снимаются (независимо от того, был ли использован для установки блокировки именно этот файловый дескриптор).

int open(const char *pathname, int flags, mode_t mode);

Вызов open() используется, чтобы преобразовать путь к файлу в описатель файла. Если системный вызов завершается успешно, возвращенный файловый описатель является наименьшим описателем, который еще не открыт процессом. Новый описатель файла будет оставаться открытым при выполнении функции exes(2). Указатель устанавливается в начале файла.

int dup2(int oldfd, int newfd);
int dup(int oldfd);

dup и dup2 создают копию файлового дескриптора oldfd.

pid_t waitpid(pid_t pid, int *status, int options);

Функция 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.

3. Использование утилиты strace

strace — это утилита, отслеживающая системные вызовы, которые представляют собой механизм трансляции, обеспечивающий интерфейс между процессом и операционной системой (ядром). Эти вызовы могут быть перехвачены и прочитаны, что позволяет лучше понять, что процесс пытается сделать в заданное время. Перехватывая эти вызовы, мы можем добиться лучшего понимания поведения процессов, особенно если что-то идет не так. Функциональность операционной системы, позволяющая отслеживать системные вызовы, называется ptrace. Strace вызывает ptrace и читает данные о поведении процесса, возвращая отчет.

Отображение всех вызовов:

```
[leo@pc src]$ strace ./main out1 out2 <test01.txt
execve("./main", ["/main", "out1", "out2"], 0x7fff8d5fa130 /* 58 vars */) = 0
brk(NULL)                               = 0x560c870ba000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd7edef110) = -1 EINVAL (Invalid argument)
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=208114, ...}) = 0
mmap(NULL, 208114, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f77afd07000
close(3)                                = 0
```

```

openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\202\2\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"..., 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", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\364[g\253(\257\25\201\313\250\344q>\17\323\262"...,
68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2159552, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f77afd05000
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"..., 784, 64) = 784
mmap(NULL, 1868448, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f77afb3c000
mmap(0x7f77afb62000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x26000) = 0x7f77afb62000
mmap(0x7f77afcaf000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x173000) = 0x7f77afcaf000
mmap(0x7f77afcfc000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1be000) = 0x7f77afcfc000
mmap(0x7f77afd01000, 12960, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7f77afd01000
close(3) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f77afb3a000
arch_prctl(ARCH_SET_FS, 0x7f77afd06580) = 0
mprotect(0x7f77afcfc000, 12288, PROT_READ) = 0
mprotect(0x560c85307000, 4096, PROT_READ) = 0
mprotect(0x7f77afd66000, 4096, PROT_READ) = 0
munmap(0x7f77afd07000, 208114) = 0
fstat(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0x2), ...}) = 0
brk(NULL) = 0x560c870ba000
brk(0x560c870db000) = 0x560c870db000
write(1, "Name of the 1st child-output fil"..., 40Name of the 1st child-output file: out1
) = 40
write(1, "Name of the 2nd child-output fil"..., 40Name of the 2nd child-output file: out2
) = 40
pipe([3, 4]) = 0
pipe([5, 6]) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLD, child_tidptr=0x7f77afd06850) = 13037
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLD, child_tidptr=0x7f77afd06850) = 13038
close(3) = 0
close(5) = 0
write(1, "Enter strings to process: \n", 27Enter strings to process:
) = 27
fstat(0, {st_mode=S_IFREG|0644, st_size=286, ...}) = 0
read(0, "\342\200\234The unexamined life is not wo"..., 4096) = 286
write(6, "\342\200\234The unexamined life is not wo"..., 59) = 59
write(6, "https://www.google.com/search?q="..., 130) = 130
write(4, "123\n", 4) = 4
write(4, "...biba\n", 8) = 8
write(4, "boba...\n", 8) = 8

```

```

write(6, "\342\200\234If God did not exist, it woul"..., 77) = 77
read(0, "", 4096) = 0
close(4) = 0
close(6) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=13037, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=13038, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
wait4(13037, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 13037
write(1, "Child 1 exited, returned 0\n", 28Child 1 exited, returned 0
) = 28
wait4(13038, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 13038
write(1, "Child 2 exited, returned 0\n", 28Child 2 exited, returned 0
) = 28
exit_group(0) = ?
+++ exited with 0 +++

```

Очень «многословно» и не очень познавательно. Слишком много не интересующих нас вызовов.

Отображение определенных вызовов:

Опция -e служит для отображения лишь определенных вызовов.

Например — отобразить только вызовы read():

```

[leo@pc src]$ strace -e close ./main out1 out2 <test01.txt
close(3) = 0
close(3) = 0
Name of the 1st child-output file: out1
Name of the 2nd child-output file: out2
close(3) = 0
close(5) = 0
Enter strings to process:
close(4) = 0
close(6) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=13590, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=13589, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
Child 1 exited, returned 0
Child 2 exited, returned 0
+++ exited with 0 +++

```

Можно и несколько вызовов сразу: -e trace= и через запятую — список вызовов.

```

[leo@pc src]$ strace -e trace=mmap,close ./main out1 out2 <test01.txt
mmap(NULL, 208114, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f5573d47000
close(3) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f5573d45000

```

```

mmap(NULL, 1868448, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f5573b7c000
mmap(0x7f5573ba2000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x26000) = 0x7f5573ba2000
mmap(0x7f5573cef000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x173000) = 0x7f5573cef000
mmap(0x7f5573d3b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1be000) = 0x7f5573d3b000
mmap(0x7f5573d41000, 12960, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7f5573d41000
close(3) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f5573b7a000
Name of the 1st child-output file: out1
Name of the 2nd child-output file: out2
close(3) = 0
close(5) = 0
Enter strings to process:
close(4) = 0
close(6) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=16380, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=16379, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
Child 1 exited, returned 0
Child 2 exited, returned 0
+++ exited with 0 +++

```

Но вот на fork или open, например, взглянуть без определенного ключа не удастся, ибо сей вызов находится в части программы, обрабатываемой дочерним процессом.

Отслеживание дочерних процессов:

Отслеживать дерево процессов целиком помогает флаг -f, с которым strace отслеживает системные вызовы в процессах-потомках. К каждой строке вывода при этом добавляется pid процесса, делающего системный вызов:

```

[leo@pc src]$ strace -f -e trace=fork ./main out1 out2 <test01.txt
Name of the 1st child-output file: out1
Name of the 2nd child-output file: out2
strace: Process 17178 attached
strace: Process 17179 attached
Enter strings to process:
[pid 17179] +++ exited with 0 +++
[pid 17177] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17179,
si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
[pid 17178] +++ exited with 0 +++
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17178, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
Child 1 exited, returned 0

```

Child 2 exited, returned 0
+++ exited with 0 +++

В данном случае будет полезной **фильтрация по группам вызовов:**

```
[leo@pc src]$ strace -f -e trace=%process ./main out1 out2 <test01.txt
execve("./main", ["/main", "out1", "out2"], 0x7ffea199a518 /* 60 vars */) = 0
Name of the 1st child-output file: out1
Name of the 2nd child-output file: out2
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLD, child_tidptr=0x7f871c760850) = 17210
strace: Process 17210 attached
[pid 17209] clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|
CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f871c760850) = 17211
strace: Process 17211 attached
Enter strings to process:
[pid 17211] execve("./child2", ["child2"], 0x7ffda96841a8 /* 60 vars */) = 0
[pid 17210] execve("./child1", ["child1"], 0x7ffda96841a8 /* 60 vars */) = 0
[pid 17209] wait4(17210, <unfinished ...>
[pid 17211] exit_group(0) = ?
[pid 17211] +++ exited with 0 +++
[pid 17209] <... wait4 resumed>0x7ffda9683e34, 0, NULL) = ? ERESTARTSYS (To be restarted if
SA_RESTART is set)
[pid 17209] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17211,
si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
[pid 17209] wait4(17210, <unfinished ...>
[pid 17210] exit_group(0) = ?
[pid 17210] +++ exited with 0 +++
<... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 17210
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17210, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
Child 1 exited, returned 0
wait4(17211, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 17211
Child 2 exited, returned 0
exit_group(0) = ?
+++ exited with 0 +++
```

Пути к файлам вместо дескрипторов:

Ключ -y позволяет взглянуть на название файлов, с которыми работает процесс, вместо файловых дескрипторов:

```
[leo@pc src]$ strace -y -e read ./main out1 out2 <test01.txt
read(3</usr/lib/libc-2.32.so>, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\
0\1\0\0\0\220\202\2\0\0\0\0"..., 832) = 832
Name of the 1st child-output file: out1
Name of the 2nd child-output file: out2
Enter strings to process:
```

```

read(0</home/leo/programming/OS/Lab1/src/test01.txt>, "\342\200\234The unexamined life is not
wo"..., 4096) = 286
read(0</home/leo/programming/OS/Lab1/src/test01.txt>, "", 4096) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17353, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17352, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
Child 1 exited, returned 0
Child 2 exited, returned 0
+++ exited with 0 +++

```

Отображение времени выполнения вызова:

Посмотреть время выполнения того или иного вызова позволяет ключ -t:

```

[leo@pc src]$ strace -y -t -e read ./main out1 out2 <test01.txt
18:05:03 read(3</usr/lib/libc-2.32.so>, "\177ELF\2\1\1\3\0\0\0\0\0\0\3\0>\
0\1\0\0\0\220\202\2\0\0\0\0"..., 832) = 832
Name of the 1st child-output file: out1
Name of the 2nd child-output file: out2
Enter strings to process:
18:05:03 read(0</home/leo/programming/OS/Lab1/src/test01.txt>, "\342\200\234The unexamined
life is not wo"..., 4096) = 286
18:05:03 read(0</home/leo/programming/OS/Lab1/src/test01.txt>, "", 4096) = 0
18:05:03 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17476,
si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
Child 1 exited, returned 0
18:05:03 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=17477,
si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
Child 2 exited, returned 0
18:05:03 +++ exited with 0 +++

```

Статистика системных вызовов:

С помощью опции -c — можно получить наглядную статистику выполнения программы:

```

[leo@pc src]$ strace -c ./main out1 out2 <test01.txt
Name of the 1st child-output file: out1
Name of the 2nd child-output file: out2
Enter strings to process:
Child 1 exited, returned 0
Child 2 exited, returned 0
% time    seconds  usecs/call   calls   errors syscall
-----
72.69    0.002167    1083        2      wait4
9.26     0.000276     138        2      clone
6.84     0.000204      18       11      write
2.65     0.000079      26         3      brk
1.85     0.000055      18         3      mprotect
1.61     0.000048       8         6      close

```


1.51	0.000045	45	1	munmap
1.51	0.000045	22	2	pipe
1.07	0.000032	8	4	fstat
1.01	0.000030	10	3	read
0.00	0.000000	0	8	mmap
0.00	0.000000	0	4	pread64
0.00	0.000000	0	1	1 access
0.00	0.000000	0	1	execve
0.00	0.000000	0	2	1 arch_prctl
0.00	0.000000	0	2	openat

100.00	0.002981	54	55	2 total

4. Листинг программы

main.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <sys/wait.h>

#define MAX_FNAME_LENGTH 100
#define MAX_STR_LENGTH 300

#define RD 0 // Read end of pipe
#define WR 1 // Write end of pipe

int main(int argc, char *argv[])
{
    if (argc < 3)
    {
        printf("Usage: ./main filename1 filename2\n");
        exit(-1);
    }

    char filename1[MAX_FNAME_LENGTH], filename2[MAX_FNAME_LENGTH];
    printf("Name of the 1st child-output file: %s\n", argv[1]);
    printf("Name of the 2nd child-output file: %s\n", argv[2]);
    strcpy(filename1, argv[1]);
    strcpy(filename2, argv[2]);

    int fd[2][2];
    for (int i = 0; i < 2; ++i)
    {
        if (pipe(fd[i]) < 0)
        {
            perror("Pipe error");
            exit(1);
        }
    }
}
```

```

}

pid_t pid1, pid2;

pid1 = fork();
if (pid1 < 0)
{
    perror("Fork err");
    exit(1);
}
else if (pid1 == 0)
{ // child1
    // closing useless pipes
    close(fd[1][0]);
    close(fd[1][1]);
    close(fd[0][WR]);

    int file_out = open(filename1, O_WRONLY | O_CREAT | O_TRUNC, 0777);
    if (file_out < 0)
    {
        perror("File err");
        exit(1);
    }

    int new_out = dup2(file_out, STDOUT_FILENO);
    close(file_out);
    if (new_out < 0)
    {
        perror("Duping child1 stdout err");
        exit(1);
    }

    int new_in = dup2(fd[0][RD], STDIN_FILENO);
    close(fd[0][RD]);
    if (new_in < 0)
    {
        perror("Duping child1 stdin err");
        exit(1);
    }

    if (execlp("./child1", "child1", NULL) < 0)
    {
        perror("Execl err");
        exit(1);
    }
}

pid2 = fork();
if (pid2 < 0)
{
    perror("Fork err");
    exit(1);
}

```

```

}
else if (pid2 == 0)
{ // child2
    // closing useless pipes
    close(fd[0][0]);
    close(fd[0][1]);
    close(fd[1][WR]);

    int file_out = open(filename2, O_WRONLY | O_CREAT | O_TRUNC, 0777);
    if (file_out < 0)
    {
        perror("File err");
        exit(1);
    }

    int new_out = dup2(file_out, STDOUT_FILENO);
    close(file_out);
    if (new_out < 0)
    {
        perror("Duping child2 stdout err");
        exit(1);
    }

    int new_in = dup2(fd[1][RD], STDIN_FILENO);
    close(fd[1][RD]);
    if (new_in < 0)
    {
        perror("Duping child1 stdin err");
        exit(1);
    }

    if (execlp("./child2", "child2", NULL) < 0)
    {
        perror("Execl err");
        exit(1);
    }
}

close(fd[0][RD]);
close(fd[1][RD]);

printf("Enter strings to process: \n");
char msg[MAX_STR_LENGTH];
while (fgets(msg, MAX_STR_LENGTH, stdin))
{
    if (strlen(msg) <= 10)
    {
        if (write(fd[0][WR], msg, strlen(msg)) < 0)
        {
            perror("Write err");
            exit(1);
        }
    }
}

```

```

    }
    else
    {
        if (write(fd[1][WR], msg, strlen(msg)) < 0)
        {
            perror("Write err");
            exit(1);
        }
    }
}

close(fd[0][WR]);
close(fd[1][WR]);

int statusChild1, statusChild2;
waitpid(pid1, &statusChild1, 0);
if (WIFEXITED(statusChild1))
{
    printf("Child 1 exited, returned %d\n", WEXITSTATUS(statusChild1));
}
else
{
    fprintf(stderr, "Something is wrong with 1st child process\n");
}
waitpid(pid2, &statusChild2, 0);
if (WIFEXITED(statusChild1))
{
    printf("Child 2 exited, returned %d\n", WEXITSTATUS(statusChild2));
}
else
{
    fprintf(stderr, "Something is wrong with 2nd child process\n");
}

return 0;
}

```

child1.c:

```

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>

#define INBUFSIZE 300 // Buffer size

int isVowel(char t)
{
    t = tolower(t);
    if (t == 'a' || t == 'e' || t == 'i' || t == 'o' || t == 'u' || t == 'y')
        return 1;
    return 0;
}

```

```

int main(void)
{
    char buff[INBUFSIZE], res[INBUFSIZE];

    pid_t mypid = getpid();
    printf("1st Child %d: Started!\n", mypid);

    while (fgets(buff, INBUFSIZE, stdin))
    {
        printf("Received line: %s", buff);

        int d = 0;
        for (int i = 0; i < strlen(buff); ++i)
        {
            if (!isVowel(buff[i]))
            {
                res[d++] = buff[i];
            }
        }
        res[d] = '\0';

        printf("Processed line: %s", res);
    }
    printf("1st Child %d: I'm Done!\n", mypid);

    return 0;
}

```

child2.c:

```

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>

#define INBUFSIZE 300 // Buffer size

int isVowel(char t)
{
    t = tolower(t);
    if (t == 'a' || t == 'e' || t == 'i' || t == 'o' || t == 'u' || t == 'y')
        return 1;
    return 0;
}

int main(void)
{
    char buff[INBUFSIZE], res[INBUFSIZE];

    pid_t mypid = getpid();
    printf("2nd Child %d: Started!\n", mypid);
}

```

```

while (fgets(buff, INBUFSIZE, stdin))
{
    printf("Received line: %s", buff);

    int d = 0;
    for (int i = 0; i < strlen(buff); ++i)
    {
        if (!isVowel(buff[i]))
        {
            res[d++] = buff[i];
        }
    }
    res[d] = '\0';

    printf("Processed line: %s", res);
}
printf("2nd Child %d: I'm Done!\n", mypid);

return 0;
}

```

Вывод

По мере выполнения данной лабораторной работы я освоил диагностику работы ПО с помощью утилиты `strace` — простого и надёжного инструмента. `Strace` позволяет отслеживать выполнение системных вызовов и сигналов к ядру системы. Ее функционал прост, но широк; работать с ней комфортно и, главное, полезно.

Я научился просматривать как все вызовы сразу, так и несколько отдельных; получилось отследить работу и дочерних процессов. Научился отображать названия, пути к файлам, с к-рыми работает программа, вместо дескрипторов. Получилось взглянуть на удобную статистику системных вызовов и наблюдать время выполнения тех или иных вызовов.

Все это — лишь малая часть функционала утилиты `strace`. С ее помощью можно делать еще много интересных вещей: намеренно ломать программу (инъекция ошибок) для выявления уязвимостей, наблюдать стек процесса в момент системного вызова и другое.

Но больше всего мне понравилась возможность отследить время тех или иных вызовов, т.к это в будущем позволит определить, в каком участке кода программа могла заикнуться; или понять, в каком месте стоит произвести оптимизацию. Также интересной и полезной я нахожу возможность отобразить статистику системных вызовов в удобной форме, в виде таблицы.