Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

# Лабораторная работа №1 по курсу

# «Операционные системы»

Группа: М8О-213Б-23

Студент: Колесник Д.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 04.10.24

Москва, 2024

# Постановка задачи

**Вариант 15.**

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

Правило проверки: строка должна начинаться с заглавной буквы

# Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int32_t execv(const char *__path, char *const *__argv); – заменяет текущий образ программы новым образом, загружая и выполняя программу, путь к которой передается в аргументах.
- ssize_t read(int __fd, void *__buf, size_t __nbytes); – читает данные из файлового дескриптора и записывает их в буфер.
- ssize_t write(int __fd, void *__buf, size_t __nbytes); – записывает данные из буфера в файловый дескриптор.
- int32_t open(const char*__file, int __oflag, …); – открывает файл и возвращает файловый дескриптор.
- int close(int __fd); – закрывает файл.
- pid_t wait(int *__stat_loc); – приостанавливает выполнение родительского процесса до тех пор, пока дочерний не будет выполнен.
- int pipe(int *__pipedes); – создает канал для однонаправленной передачи данных между двумя процессами.

Программа начинает с того, что родительский процесс создает два канала для передачи данных между собой и дочерним процессом. Родитель затем создает дочерний процесс с помощью fork(). После этого родительский процесс ожидает пользовательский ввод и пересылает его через первый канал дочернему процессу.

Дочерний процесс принимает данные, проверяет, начинается ли строка с заглавной буквы. Если проверка проходит успешно, он отправляет эту строку обратно родителю через второй канал. Если строка не начинается с заглавной буквы, дочерний процесс отправляет сообщение об ошибке.

Родительский процесс получает результат из второго канала — либо проверенную строку, либо сообщение об ошибке — и выводит его на экран.

Таким образом, программа организует взаимодействие между двумя процессами через каналы: один передает пользовательский ввод дочернему процессу для проверки, другой возвращает результат обратно родителю для вывода на экран.

## Код программы

**server.c**

```
#include <stdint.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>

static char CLIENT_PROGRAM_NAME[] = "client";

int main(int argc, char **argv) {
	if (argc == 1) {
		char msg[1024];
		uint32_t len = snprintf(msg, sizeof(msg) - 1, "usage: %s filename\n",
argv[0]);
		write(STDERR_FILENO, msg, len);
		exit(EXIT_SUCCESS);
	}

	char progpath[1024];
	{
		ssize_t len = readlink("/proc/self/exe", progpath, sizeof(progpath) - 1);
		if (len == -1) {
			const char msg[] = "error: failed to read full program path\n";
			write(STDERR_FILENO, msg, sizeof(msg));
			exit(EXIT_FAILURE);
		}

		while (progpath[len] != '/')
			--len;

		progpath[len] = '\0';
	}
```

```
	int channel[2];
	if (pipe(channel) == -1) {
		const char msg[] = "error: failed to create pipe\n";
		write(STDERR_FILENO, msg, sizeof(msg));
		exit(EXIT_FAILURE);
	}

	const pid_t child = fork();

	switch (child) {
	case -1: {
		const char msg[] = "error: failed to spawn new process\n";
		write(STDERR_FILENO, msg, sizeof(msg));
		exit(EXIT_FAILURE);
	} break;

	case 0: {
		pid_t pid = getpid();

		dup2(STDIN_FILENO, channel[STDIN_FILENO]);
		close(channel[STDOUT_FILENO]);

		{
			char msg[64];
			const int32_t length = snprintf(msg, sizeof(msg), "%d: I'm a
child\n", pid);
			write(STDOUT_FILENO, msg, length);
		}

		{
			char path[1024];
			snprintf(path, sizeof(path) - 1, "%s/%s", progpath,
CLIENT_PROGRAM_NAME);

			char *const args[] = {CLIENT_PROGRAM_NAME, argv[1], NULL};

			int32_t status = execv(path, args);

			if (status == -1) {
				const char msg[] = "error: failed to exec into new
executable image\n";
				write(STDERR_FILENO, msg, sizeof(msg));
				exit(EXIT_FAILURE);
			}
		}
	} break;

	default: {
		pid_t pid = getpid();
```

```
            {
                  char msg[64];
                  const int32_t length = snprintf(msg, sizeof(msg), "%d: I'm a
parent, my child has PID %d\n", pid, child);
                  write(STDOUT_FILENO, msg, length);
            }

            int child_status;
            wait(&child_status);

            if (child_status != EXIT_SUCCESS) {
                  const char msg[] = "error: child exited with error\n";
                  write(STDERR_FILENO, msg, sizeof(msg));
                  exit(child_status);
            }
      } break;
      }
}
```

**client.c**

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <ctype.h> // Для isupper()



// Проверить, что строка начинается с заглавной буквы

int main(int argc, char **argv) {
      char buf[4096];
      ssize_t bytes;

      pid_t pid = getpid();


      int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
      if (file == -1) {
            const char msg[] = "error: failed to open requested file\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
      }

      {
            char msg[128];
```

```
            int32_t len = snprintf(msg, sizeof(msg) - 1,
    "%d: Start typing lines of text.Press 'Ctrl-D' or 'Enter' with no input to exit\n",
    pid);
            write(STDOUT_FILENO, msg, len);
        }

        while (bytes = read(STDIN_FILENO, buf, sizeof(buf))) {
            if (bytes < 0) {
                const char msg[] = "error: failed to read from stdin\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            } else if (buf[0] == '\n') {
                break;
            }

            if (!isupper(buf[0])) {
                const char msg[] = "error: string does not start with uppercase letter\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                continue;
            }

            {
                char msg[32];
                int32_t len = snprintf(msg, sizeof(msg) - 1, "");

                int32_t written = write(file, msg, len);
                if (written != len) {
                    const char msg[] = "error: failed to write to file\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    exit(EXIT_FAILURE);
                }
            }

            {
                buf[bytes - 1] = '\0';

                int32_t written = write(file, buf, bytes);
                if (written != bytes) {
                    const char msg[] = "error: failed to write to file\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    exit(EXIT_FAILURE);
                }
            }
        }

    if (bytes == 0) {
```

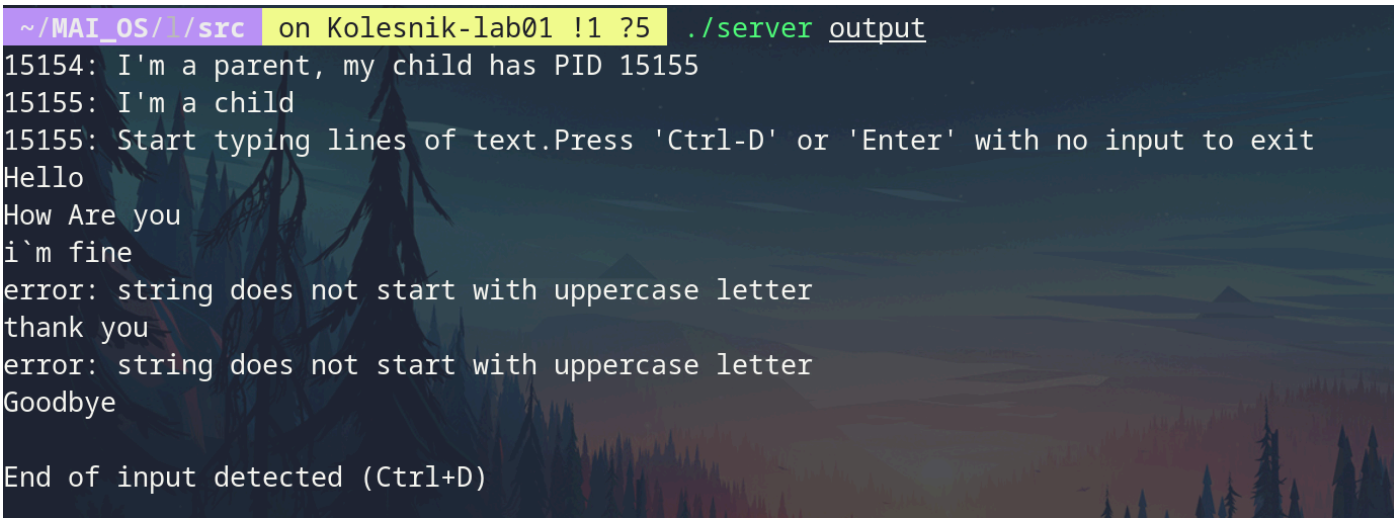
```
            const char msg[] = "\nEnd of input detected (Ctrl+D)\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
        }

        const char term = '\0';
        write(file, &term, sizeof(term));

        close(file);
}
```

## Протокол работы программы

**Тестирование:**




**Strace:**

```
$ strace -f ./server output
execve("./server", ["./server", "output"], 0x7ffc770c3d00 /* 64 vars */) = 0
brk(NULL)                               = 0x5ef6288a5000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fff5b1b4e30) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7a8aaeadd000
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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=116959, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 116959, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a8aaeac0000
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\0P\237\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 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48,
848) = 48
```

```
pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68,
896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
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, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a8aae800000
mprotect(0x7a8aae828000, 2023424, PROT_NONE) = 0
mmap(0x7a8aae828000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7a8aae828000
mmap(0x7a8aae9bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7a8aae9bd000
mmap(0x7a8aaea16000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7a8aaea16000
mmap(0x7a8aaea1c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7a8aaea1c000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7a8aaeabd000
arch_prctl(ARCH_SET_FS, 0x7a8aaeabd740) = 0
set_tid_address(0x7a8aaeabda10)         = 15411
set_robust_list(0x7a8aaeabda20, 24)     = 0
rseq(0x7a8aaeabe0e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7a8aaea16000, 16384, PROT_READ) = 0
mprotect(0x5ef627b23000, 4096, PROT_READ) = 0
mprotect(0x7a8aaeb17000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7a8aaeac0000, 116959)          = 0
readlink("/proc/self/exe", "/home/ares/MAI_OS/lab01/src/serv"..., 1023) = 34
pipe2([3, 4], 0)                        = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace:
Process 15412 attached
, child_tidptr=0x7a8aaeabda10) = 15412
[pid 15411] getpid( <unfinished ...>
[pid 15412] set_robust_list(0x7a8aaeabda20, 24 <unfinished ...>
[pid 15411] <... getpid resumed>)       = 15411
[pid 15412] <... set_robust_list resumed>) = 0
[pid 15411] write(1, "15411: I'm a parent, my child ha"..., 44 <unfinished ...>
[pid 15412] getpid(15411: I'm a parent, my child has PID 15412
 <unfinished ...>
[pid 15411] <... write resumed>)        = 44
[pid 15412] <... getpid resumed>)       = 15412
[pid 15411] wait4(-1,  <unfinished ...>
[pid 15412] dup2(0, 3)                  = 3
[pid 15412] close(4)                    = 0
[pid 15412] write(1, "15412: I'm a child\n", 1915412: I'm a child
) = 19
[pid 15412] execve("/home/ares/MAI_OS/lab01/src/client", ["client", "output"],
0x7fff5b1b5010 /* 64 vars */) = 0
```

```
[pid 15412] brk(NULL)                   = 0x583b6b03f000
[pid 15412] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd92d3cbd0) = -1 EINVAL (Invalid
argument)
[pid 15412] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7872de1a3000
[pid 15412] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
[pid 15412] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 4
[pid 15412] newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=116959, ...},
AT_EMPTY_PATH) = 0
[pid 15412] mmap(NULL, 116959, PROT_READ, MAP_PRIVATE, 4, 0) = 0x7872de186000
[pid 15412] close(4)                    = 0
[pid 15412] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 4
[pid 15412] read(4,
"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0"..., 832) = 832
[pid 15412] pread64(4,
"\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
[pid 15412] pread64(4, "\4\0\0\0
\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
[pid 15412] pread64(4,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68,
896) = 68
[pid 15412] newfstatat(4, "", {st_mode=S_IFREG|0755, st_size=2220400, ...},
AT_EMPTY_PATH) = 0
[pid 15412] pread64(4,
"\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
[pid 15412] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 4, 0) =
0x7872dde00000
[pid 15412] mprotect(0x7872dde28000, 2023424, PROT_NONE) = 0
[pid 15412] mmap(0x7872dde28000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x28000) = 0x7872dde28000
[pid 15412] mmap(0x7872ddfbd000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x1bd000) = 0x7872ddfbd000
[pid 15412] mmap(0x7872de016000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x215000) = 0x7872de016000
[pid 15412] mmap(0x7872de01c000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7872de01c000
[pid 15412] close(4)                    = 0
[pid 15412] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7872de183000
[pid 15412] arch_prctl(ARCH_SET_FS, 0x7872de183740) = 0
[pid 15412] set_tid_address(0x7872de183a10) = 15412
[pid 15412] set_robust_list(0x7872de183a20, 24) = 0
[pid 15412] rseq(0x7872de1840e0, 0x20, 0, 0x53053053) = 0
[pid 15412] mprotect(0x7872de016000, 16384, PROT_READ) = 0
[pid 15412] mprotect(0x583b6a988000, 4096, PROT_READ) = 0
[pid 15412] mprotect(0x7872de1dd000, 8192, PROT_READ) = 0
[pid 15412] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
[pid 15412] munmap(0x7872de186000, 116959) = 0
```

```
[pid 15412] getpid()                    = 15412
[pid 15412] openat(AT_FDCWD, "output", O_WRONLY|O_CREAT|O_TRUNC|O_APPEND, 0600) = 4
[pid 15412] write(1, "15412: Start typing lines of tex"..., 8215412: Start typing lines
of text.Press 'Ctrl-D' or 'Enter' with no input to exit
) = 82
[pid 15412] read(0,
"\n", 4096)         = 1
[pid 15412] write(4, "\0", 1)           = 1
[pid 15412] close(4)                    = 0
[pid 15412] exit_group(0)               = ?
[pid 15412] +++ exited with 0 +++
<... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 15412
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=15412, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
exit_group(0)                           = ?
+++ exited with 0 +++
```

# Вывод

С помощью системных вызовов и возможностей языка Си можно удобно организовать взаимодействие между разными процессами, связать их, передавать и обрабатывать данные между ними. Основными системными вызовами для этого являются: fork(), pipe(), а также read(), write() и open().