Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

**ПРОЦЕССЫ И ПОТОКИ**

Студент: Модин-Глазков Богдан Арсеньевич

Группа: М8О–212Б–22 Вариант: 13

Преподаватель: Оценка:

Дата:

Подпись:

Москва, 2023.

**Постановка задачи**

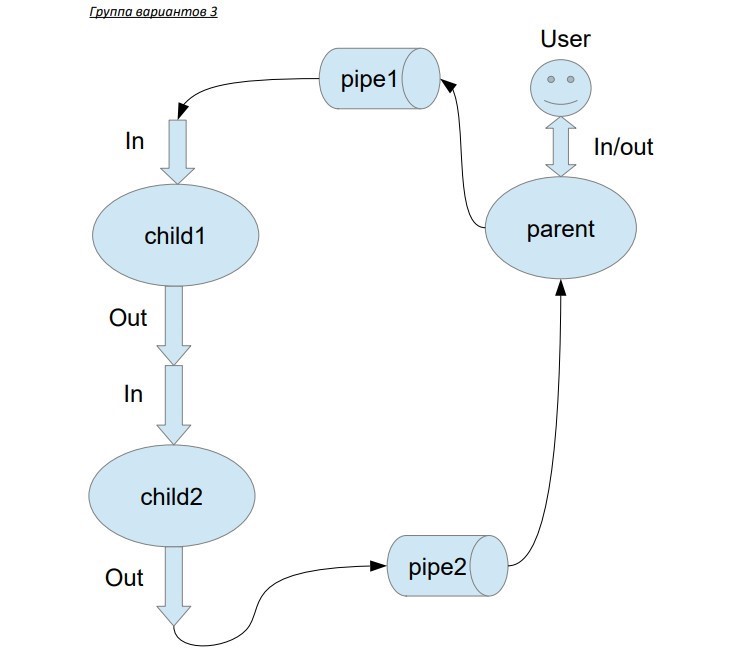
**Цель работы**

Целью является приобретение практических навыков в:

* Управление процессами в ОС
* Обеспечение обмена данных между процессами посредством каналов

**Задание**

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.



Вариант 13) Child1 переводит строки в нижний регистр. Child2 превращает все пробельные символы в символ «\_».

**Общие сведения о программе**

Программа компилируется при помощи утилиты CMake и запускается путем запуска ./main. Также используется заголовочные файлы: iostream, string, stdio.h, unistd.h, cstdlib, sys/wait.h, fstream, fcntl.h, sys/stat.h. В программе используются следующие системные вызовы:

1. **read** – функция read() считывает count байт из файла, описываемого аргументом fd, в буфер, на который указывает аргумент buf Указателю положения в файле дается приращение на количество считанных байт. Если файл открыт в текстовом режиме, то может иметь место транслирование символов.
2. **write** – функция переписывает count байт из буфера, на который указывает bufy в файл, соответствующий дескриптору файла handle. Указателю положения в файле дается приращение на количество записанных байт. Если файл открыт в текстовом режиме, то символы перевода строки автоматически дополняются символами возврата каретки.
3. **pipe** – создаёт механизм ввода вывода, который называется конвейером.

Возвращаемый файловый дескриптор можно использовать для операций

чтения и записи. Когда в конвейер что-то записывается, то буферизуется до 504 байтов данных, после чего процесс записи приостанавливается.

1. **fork** - вызов создаёт новый процесс посредством копирования вызывающего процесса. Новый процесс считается дочерним процессом. Вызывающий процесс считается родительским процессом.
2. **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).
3. **dup2** - системная функция используется для создания копии существующего файлового дескриптора.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, pipe, read, write, close, exec\*, dup2.
2. Написать две программмы для родительского и дочернего процесса, а так же написать библиотеку common.h, для работы со стандартными потоками ввода и вывода через read и write.
3. Использовать в parent.c fork, чтобы запустить дочерний процесс.
4. При помощи конструкции if/else организовать работу с дочерним и родительским процессом.
5. В дочернем процессе скопировать файловые дескрипторы пайпов в stdin и stdout и запустить child.c при помощи execl.
6. Скомпилировать обе программы при помощи CMake и запустить

./parent.out.

**Основные файлы программы**

**main.cpp:**

|  |
| --- |
| **#include <sys/types.h>**  **#include <sys/wait.h>**  **#include <unistd.h>** |
|  |
|  |
| **#include <iostream>**  **#include <cerrno>**  **#include <cstring>** |
|  |
|  |
| **using namespace std;**  **int main() {**  **string line; getline(cin, line);**  **int pipe1[2];** |
| **int pipe2[2]; int pipe3[2];**  **if (pipe(pipe1) == -1 || pipe(pipe2) == -1 || pipe(pipe3) == -1) {** |

|  |
| --- |
| **cerr << "Pipe error" << endl; return 1;**  **}** |
| **size\_t line\_len = line.length();** |
| **write(pipe1[1], &line\_len, sizeof(size\_t));**  **write(pipe1[1], line.c\_str(), sizeof(char) \* line.length());** |
| **pid\_t pid1 = fork();** **if (pid1 == -1) { cerr << "Fork error" << endl;** **return 1;**  **}**  **if (pid1 == 0) {** **close(pipe1[1]); size\_t received\_line\_len;**  **read(pipe1[0], &received\_line\_len, sizeof(size\_t)); char received\_line[received\_line\_len + 1];**  **read(pipe1[0], received\_line, received\_line\_len \* sizeof(char)); received\_line[received\_line\_len] = '\0';** |
| **close(pipe3[0]);**  **if (dup2(pipe3[1], STDOUT\_FILENO) == -1) {**  **cerr << "Dup2 error" << endl; return 1;**  **}** |
| **if (execl("./child1", "./child1", received\_line, nullptr) == -1) { cerr << "Execl error" << strerror(errno) << endl;** **return 1;**  **}**  **}**  **pid\_t pid2 = fork();** **if (pid2 == -1) {** |

|  |
| --- |
| **cerr << "Fork error" << endl; return 1;**  **}**  **if (pid2 == 0) {** **close(pipe3[1]);**  **if (dup2(pipe3[0], STDIN\_FILENO) == -1) {**  **cerr << "Dup2 error" << endl; return 1;**  **}**  **string received\_line;** **getline(cin, received\_line);**  **close(pipe2[0]);**  **if (dup2(pipe2[1], STDOUT\_FILENO) == -1) {** **cerr << "Dup2 error" << endl; return 1;**  **}**  **if (execl("./child2", "./child2", received\_line.c\_str(), nullptr) ==**  **-1) {**  **cerr << "Execl error" << strerror(errno) << endl;** **return 1;**  **}**  **}** |
| **wait(nullptr);**  **if (pid1 > 0 and pid2 > 0) { close(pipe2[1]);**  **if (dup2(pipe2[0], STDIN\_FILENO) == -1) { cerr << "Dup2 error" << endl;**  **return 1;**  **}**  **getline(cin, line); cout << line << endl;**  **}**    **return 0;**  **}** |

**child1.cpp**

|  |
| --- |
| **#include <iostream>**  **#include <cctype>** **#include <string>**  **using namespace std;**  **int main(int argc, const char \*argv[]) {** |
|  |
|  |
| **if (argc < 2) { cerr << "Too few arguments — please provide input line" << endl;** **return 1;**  **}**    **string word = argv[1]; for (char &c: word) {** |
|  |
|  |
| **c = tolower(c);**  **}** |
| **word += '\n'; cout << word;**  **return 0;** |
|  |
|  |
| **}** |

**child2.cpp**

|  |
| --- |
| **#include <iostream>**  **#include <sstream>** **#include <string>**  **using namespace std;**  **int main(int argc, char \*argv[]) {**  **if (argc < 2) { cerr << "Too few arguments — please provide input line" << endl;**  **return 1;**  **}**  **string word = argv[1];**  **for (char &c: word) {** **if (c == ' ') { c = '\_';**  **}**  **}**  **word += '\n'; cout << word;** **return 0;**  **}** |

# Пример работы

aleksandr@dots:~/labsOC/lab1var13$ ./main HFCH kyvg etgeg VYVYG hfch\_kyvg\_\_\_\_etgeg\_vyvyg **STRACE**

aleksandr@dots:~/labsOC/lab1var13$ strace ./main

execve("./main", ["./main"], 0x7fffb2d5cb40 /\* 45 vars \*/) = 0

brk(NULL) = 0x55ab40943000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffcfc3e1410) = -1 EINVAL (Недопустимый

аргумент)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff346244000 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

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=68703, ...}, AT\_EMPTY\_PATH)

= 0

mmap(NULL, 68703, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7ff346233000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libstdc++.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\0\0\0\0\0\0\0\0"..., 832)

= 832 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=2260296, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 2275520, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0)

= 0x7ff346000000 mprotect(0x7ff34609a000, 1576960, PROT\_NONE) = 0 mmap(0x7ff34609a000, 1118208, PROT\_READ|PROT\_EXEC,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x9a000) = 0x7ff34609a000 mmap(0x7ff3461ab000, 454656, PROT\_READ,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ab000) = 0x7ff3461ab000 mmap(0x7ff34621b000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x21a000) =

0x7ff34621b000

mmap(0x7ff346229000, 10432, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff346229000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libgcc\_s.so.1", O\_RDONLY|O\_CLOEXEC) =

3 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=125488, ...}, AT\_EMPTY\_PATH)

= 0

mmap(NULL, 127720, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7ff345fe0000

mmap(0x7ff345fe3000, 94208, PROT\_READ|PROT\_EXEC,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x3000) = 0x7ff345fe3000 mmap(0x7ff345ffa000, 16384, PROT\_READ,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1a000) = 0x7ff345ffa000 mmap(0x7ff345ffe000, 8192, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1d000) = 0x7ff345ffe000

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\0\244;\374\204(\337f#\315I\214\234\f\256\271\3

2"...,

68, 896) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2216304, ...}, 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, 2260560, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0)

= 0x7ff345c00000

mmap(0x7ff345c28000, 1658880, PROT\_READ|PROT\_EXEC,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7ff345c28000 mmap(0x7ff345dbd000, 360448, PROT\_READ,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7ff345dbd000 mmap(0x7ff345e15000, 24576, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x214000) =

0x7ff345e15000

mmap(0x7ff345e1b000, 52816, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7ff345e1b000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.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\0\0\0\0\0\0\0\0"..., 832) = 832 newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=940560, ...}, AT\_EMPTY\_PATH)

= 0

mmap(NULL, 942344, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) =

0x7ff345ef9000

mmap(0x7ff345f07000, 507904, PROT\_READ|PROT\_EXEC,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xe000) = 0x7ff345f07000 mmap(0x7ff345f83000, 372736, PROT\_READ,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x8a000) = 0x7ff345f83000 mmap(0x7ff345fde000, 8192, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xe4000) = 0x7ff345fde000

close(3) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff346231000

arch\_prctl(ARCH\_SET\_FS, 0x7ff3462323c0) = 0

set\_tid\_address(0x7ff346232690) = 5577 set\_robust\_list(0x7ff3462326a0, 24) = 0 rseq(0x7ff346232d60, 0x20, 0, 0x53053053) = 0 mprotect(0x7ff345e15000, 16384,

PROT\_READ) = 0 mprotect(0x7ff345fde000, 4096, PROT\_READ)

= 0 mprotect(0x7ff345ffe000, 4096, PROT\_READ) = 0 mmap(NULL, 8192, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7ff34622f000 mprotect(0x7ff34621b000, 45056, PROT\_READ) = 0

mprotect(0x55ab3f765000, 4096, PROT\_READ) = 0

mprotect(0x7ff34627e000, 8192, PROT\_READ) = 0 prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024,

rlim\_max=RLIM64\_INFINITY}) = 0 munmap(0x7ff346233000,

68703) = 0

getrandom("\x60\xe7\x4e\xf0\x98\xec\x48\x0c", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55ab40943000

brk(0x55ab40964000) = 0x55ab40964000

futex(0x7ff34622977c, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

newfstatat(0, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...},

AT\_EMPTY\_PATH) = 0 read(0, VVIHBJ bkjbn

"VVIHBJ bkjbn\n", 1024) = 13 pipe2([3, 4], 0)

= 0 pipe2([5, 6], 0) = 0 pipe2([7, 8], 0) = 0

write(4, "\f\0\0\0\0\0\0\0", 8) = 8 write(4,

"VVIHBJ bkjbn", 12) = 12 clone(child\_stack=NULL,

flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7ff346232690) = 5580 clone(child\_stack=NULL,

flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD,

child\_tidptr=0x7ff346232690) = 5581

wait4(-1, NULL, 0, NULL) = 5580

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=5580,

si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

close(6) = 0 dup2(5, 0) = 0

read(0, "vvihbj\_bkjbn\n", 1024) = 13

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

write(1, "vvihbj\_bkjbn\n", 13vvihbj\_bkjbn

) = 13

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=5581, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} --exit\_group(0)

= ?

+++ exited with 0 +++

**Вывод**

В первой лабораторной работе я научился работать с процессами программ. Изучив работу каждого системного вызова и разобрав работу стандартных потоков, я разобрался с данной темой. Разобрался с read и write. Также разобрал системные вызовы для управления процессами( dup2, close и wait )