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: Недосекин Александр Александрович
Группа: М8О–209Б–22
Вариант: 13
Преподаватель: Гапонов Н.А.
Оценка:_____
Дата:_____
Подпись: _____

Москва, 2023.

Постановка задачи

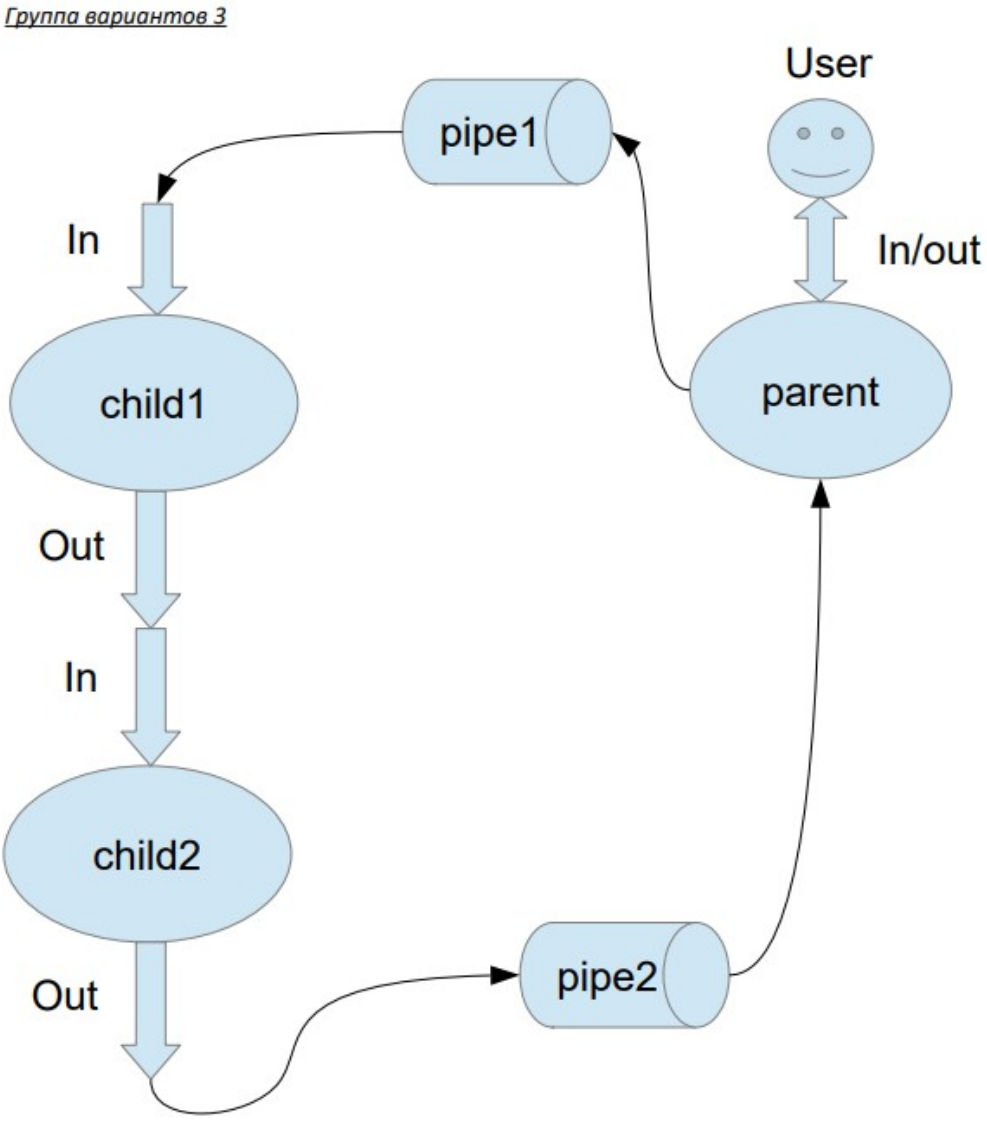
Цель работы

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.

4. **fork** - вызов создаёт новый процесс посредством копирования вызывающего процесса. Новый процесс считается дочерним процессом. Вызывающий процесс считается родительским процессом.
5. **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).
6. **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:

```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\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"..., 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\32"...,
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"..., 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", 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, путем изучения их мануалов и информации из интернета и разобрав работу стандартных потоков, я понял, что умение и понимание этого позволит в будущем понимать более глубоко устройство программ и их процессов в работе. Любая современная функция работы с вводом/выводом в наше время, работает на основе read и write. А такие низкоуровневые функции, как exec* используются по сей день в улучшенных оболочках. Управление процессами путем dup2, close и wait помогут в будущем более умело пользоваться многопроцессорными программами.