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-210Б-23

Студент: Иванченко Макар Дмитриевич

Преподаватель: Бахарев В.Д. (ФИИТ)

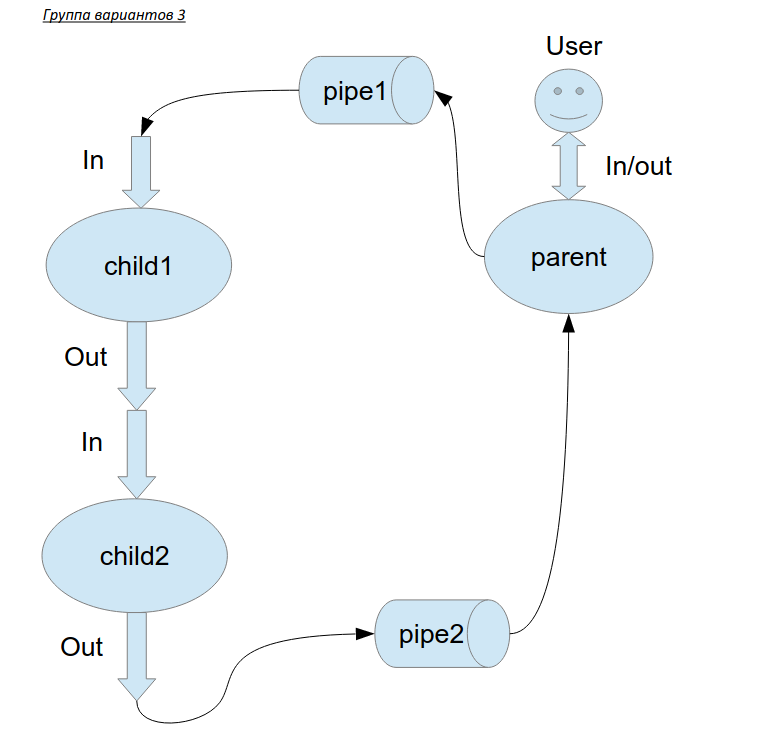
Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 17.10.24

Москва, 2024

**Постановка задачи**

**Вариант 12.**

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.

Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t fork(void); – создает дочерний процесс.
* int pipe(int \*fd); – создаёт неименованный канал и помещает дескрипторы для чтения и записи в fd[0], fd[1] соответственно.
* ssize\_t write(int fd, const void\* buff, int count); – записывает по дескриптору fd count байт из buff.
* void exit(int number); – завершает программу с кодом number.
* int dup2(int fd1, int fd2); – копирует дескриптор fd1 в дескриптор fd2.
* int exec(char\* path, const char\* argм); – запускает программу по пути path с аргументами argv и текущим окружением.
* int close(int fd); – закрывает дескриптор fd, возвращает 0 если операция успешна и -1 иначе.
* pid\_t wait(int\* status) — функция, которая приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится. Записывает код завершения процесса в status
* pid\_t waitpid(pid\_t \_\_pid, int \*\_\_stat\_loc, int \_\_options) - ожидает завершения дочернего процесса pid с дополнительными опциями \_\_options, код завершения помещает в \_\_stat\_loc.
* int kill(pid\_t pid, int sig) – завершает процесс pid с сигналом sig. Возвращает 0 в случае успеха, иначе -1

Сначала программа создает три неименованных канала для обмена данными. После этого создается два дочерних процесса – child1 и child2. Их ввод и вывод перенаправляются в соответствии со схемой, указанной в задании. После этого процесс-родитель считывает строку из стандартного потока ввода, после чего помещает ее в канал pipe1. Child1 читает эту строку из pipe1, после чего переводит все символы в ней в верхний регистр и помещает ее в канал p\_children. Из этого канала ее считывает child2, который проходится по строке в цикле for, в котором убирает все сдвоенные пробелы, после чего помещает результат работы в канал pipe2. Процесс-родитель читает данные из канала pipe2 и выводит их на стандартный вывод.

Если на вход поступает символ ‘\n’, то родитель посылает этот символ в детей и ожидает их завершения. Каждый ребенок, получив этот символ, завершает свою работу.

**Код программы**

**parent.c**

#include <stdint.h>

#include <stdbool.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

#include "string.h"

#include <signal.h>

int main()

{

char path1[4096] = "/media/sf\_univer/OSlabs/lab1 var12/child1";

char path2[4096] = "/media/sf\_univer/OSlabs/lab1 var12/child2";

int p1[2];

int p2[2];

int p\_childrens[2];

pipe(p1); // [1] - запись [0] - чтение

pipe(p2);

pipe(p\_childrens);

pid\_t child1 = fork();

int status;

int res\_status;

if (child1 == -1)

{

exit(EXIT\_FAILURE);

}

if (child1 == 0)

{

// child

dup2(p1[0], STDIN\_FILENO);

dup2(p\_childrens[1], STDOUT\_FILENO);

char \*args[] = {NULL};

status = execv(path1, args);

if (status == -1)

{

const char msg[] = "error: failed to exec into new exectuable image\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

}

else

{

//

// parent

pid\_t child2 = fork();

if (child2 == -1)

{

exit(EXIT\_FAILURE);

}

if (child2 == 0)

{

dup2(p\_childrens[0], STDIN\_FILENO);

dup2(p2[1], STDOUT\_FILENO);

char \*args[] = {NULL};

status = execv(path2, args);

if (status == -1)

{

const char msg[] = "error: failed to exec into new exectuable image\n";

write(STDERR\_FILENO, msg, sizeof(msg));

kill(child1, SIGKILL);

exit(EXIT\_FAILURE);

}

}

else

{

char input\_string[4096];

char output\_string[4096];

int n\_inp = read(STDIN\_FILENO, input\_string, sizeof(input\_string));

int n\_out;

while (input\_string[0] != '\n')

{

int status1, status2;

waitpid(child1, &status1, WNOHANG);

waitpid(child2, &status2, WNOHANG);

if (status1 != 0 || status2 != 0)

{

if (status1 == 0){

kill(child1, SIGKILL);

} else {

kill(child2, SIGKILL);

}

}

// fprintf(stderr, "p %s", input\_string);

input\_string[n\_inp - 1] = '\0';

write(p1[1], input\_string, n\_inp);

// fprintf(stderr, "wait");

n\_out = read(p2[0], output\_string, sizeof(output\_string));

// fprintf(stderr, "parent read %s end", output\_string);

int really\_written = write(STDOUT\_FILENO, output\_string, n\_out);

char temp = '\n';

if (really\_written != n\_out || write(STDOUT\_FILENO, &temp, 1) != 1)

{

const char msg[] = "error: failed to write to stdout\n";

write(STDERR\_FILENO, msg, sizeof(msg));

kill(child1, SIGKILL);

kill(child2, SIGKILL);

exit(EXIT\_FAILURE);

}

n\_inp = read(STDIN\_FILENO, input\_string, sizeof(input\_string));

}

// fprintf(stderr, "exit");

char temp = '\n';

if(write(p1[1], &temp, 1) != 1){

const char msg[] = "error: failed to write to pipe\n";

write(STDERR\_FILENO, msg, sizeof(msg));

kill(child1, SIGKILL);

kill(child2, SIGKILL);

exit(EXIT\_FAILURE);

}

wait(&res\_status);

}

}

return res\_status;

}

**child1\_source.c**

#include <stdint.h>

#include <stdbool.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int main(int argsc, char\*\* args){

char buf[4096];

size\_t bytes;

while(bytes = read(STDIN\_FILENO, buf, sizeof(buf))){

if (buf[0] == '\n')

{

int written = write(STDOUT\_FILENO, &buf[0], 1);

if (written != 1)

{

const char msg[] = "error: failed to write to pipe\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

\_exit(0);

}

if (bytes < 0) {

const char msg[] = "error: failed to read from stdin\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

// fprintf(stderr, "%s\n", buf);

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

int n = strlen(buf);

for (int i = 0; i < n; i++){

buf[i] = toupper(buf[i]);

}

int written = write(STDOUT\_FILENO, buf, strlen(buf));

// fprintf(stderr, "c1write");

// int written = bytes;

if (written != strlen(buf)){

const char msg[] = "error: failed to write to pipe\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

}

return 0;

}

**child2\_source.c**

#include <stdint.h>

#include <stdbool.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int main(int argsc, char \*\*args)

{

char buf[4096];

size\_t bytes;

while (bytes = read(STDIN\_FILENO, buf, sizeof(buf)))

{

if (buf[0] == '\n'){

int written = write(STDOUT\_FILENO, &buf[0], 1);

if (written != 1)

{

const char msg[] = "error: failed to write to pipe\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

\_exit(0);

}

if (bytes < 0)

{

const char msg[] = "error: failed to read from stdin\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

// fprintf(stderr, "c2read %s", buf);

buf[bytes] = '\0';

int n = strlen(buf);

char res[4096];

strcpy(res, "");

// fprintf(stderr,"%s", buf);

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

{

if (buf[i] == ' ' && buf[i + 1] == ' ')

{

i++;

}

else

{

strncat(res, &buf[i], 1);

}

}

strncat(res, &buf[n - 1], 1);

// fprintf(stderr, "%ld", bytes);

// fprintf(stderr,"2 %s", res);

int written = write(STDOUT\_FILENO, res, strlen(res));

// fprintf(stderr, "c2write %s end", res);

if (written != strlen(res))

{

const char msg[] = "error: failed to write to pipe\n";

write(STDERR\_FILENO, msg, sizeof(msg));

exit(EXIT\_FAILURE);

}

}

return 0;

}

**Протокол работы программы**

**Тестирование:**



**Strace:**

execve("./a.out", ["./a.out"], 0x7ffe940e3570 /\* 57 vars \*/) = 0

brk(NULL) = 0x55b0d0a45000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffebfdfc250) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x765fb901c000

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=59615, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 59615, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x765fb900d000

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) = 0x765fb8c00000

mprotect(0x765fb8c28000, 2023424, PROT\_NONE) = 0

mmap(0x765fb8c28000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x765fb8c28000

mmap(0x765fb8dbd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x765fb8dbd000

mmap(0x765fb8e16000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x765fb8e16000

mmap(0x765fb8e1c000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x765fb8e1c000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x765fb900a000

arch\_prctl(ARCH\_SET\_FS, 0x765fb900a740) = 0

set\_tid\_address(0x765fb900aa10) = 5819

set\_robust\_list(0x765fb900aa20, 24) = 0

rseq(0x765fb900b0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x765fb8e16000, 16384, PROT\_READ) = 0

mprotect(0x55b0d00da000, 4096, PROT\_READ) = 0

mprotect(0x765fb9056000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x765fb900d000, 59615) = 0

pipe2([3, 4], 0) = 0

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

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

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x765fb900aa10) = 5820

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x765fb900aa10) = 5821

read(0, Sfd De D gg Das

"Sfd De D gg Das\n", 4096) = 18

wait4(5820, 0x7ffebfdf8280, WNOHANG, NULL) = 0

wait4(5821, 0x7ffebfdf8284, WNOHANG, NULL) = 0

write(4, "Sfd De D gg Das\0", 18) = 18

read(5, "SFDDE D GGDAS", 4096) = 13

write(1, "SFDDE D GGDAS", 13SFDDE D GGDAS) = 13

write(1, "\n", 1) = 1

read(0, "\n", 4096) = 1

write(4, "\n", 1) = 1

wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 5820

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=5820, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=5821, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В ходе лабораторной работе я приобрел базовые навыки по работе с системными вызовами в Си, а также по работе с процессами. Я узнал, как происходит обмен данными между процессами, что такое pipe и как он работает. При выполнении возникли некоторые трудности с перенаправлением ввода и вывода, а также с обработкой ошибок.