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-211Б-23

Студент: Королев И.А

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 31.10.24

Москва, 2024

Постановка задачи

Вариант 13.

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

Общий метод и алгоритм решения

Использованные системные вызовы:

- **pid_t fork(void);** – создает дочерний процесс, возвращая PID дочернего процесса родительскому процессу, и 0 – в дочернем. В родительском процессе используется для создания двух дочерних процессов, которые будут выполнять обработку данных.
- **int pipe(int *fd);** – создает однонаправленный канал для передачи данных между процессами. При успешном вызове, в массиве fd устанавливаются два файловых дескриптора: fd[0] для чтения и fd[1] для записи. В программе используются три канала для обмена данными между процессами: первый для передачи данных из родительского процесса в первый дочерний, второй для передачи данных из первого дочернего во второй дочерний, и третий для передачи результата из второго дочернего обратно родителю.
- **ssize_t read(int fd, void *buf, size_t count);** – читает данные из файлового дескриптора fd в буфер buf. В программе используется для чтения пользовательского ввода в родительском процессе и передачи данных через пайпы в дочерние процессы, а также для получения данных, обработанных дочерними процессами.
- **ssize_t write(int fd, const void *buf, size_t count);** – записывает данные из буфера buf в файловый дескриптор fd. Применяется для отправки данных между процессами через пайпы, а также для вывода на экран обработанных данных.
- **int execl(const char *path, const char *arg, ...);** – заменяет текущий образ процесса другим, выполняя указанную программу. В программе каждый дочерний процесс использует execl для вызова child1 или child2, чтобы выполнить их основную функцию.
- **pid_t waitpid(pid_t pid, int *status, int options);** – приостанавливает выполнение родительского процесса до завершения указанного дочернего процесса. Используется для ожидания завершения обоих дочерних процессов перед завершением работы родительского процесса.

Код программы

parent.c

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

#define BUF_SIZE 4096

int main() {
    int pipe1[2], pipe2[2], pipe3[2];
    char buffer[BUF_SIZE];

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1 || pipe(pipe3) == -1) {
```

```
        perror("Ошибка при создании pipe");
        exit(EXIT_FAILURE);
    }

    pid_t child1 = fork();
    if (child1 == -1) {
        perror("Ошибка при создании дочернего процесса");
        exit(EXIT_FAILURE);
    }

    if (child1 == 0) {
        close(pipe1[STDOUT_FILENO]);
        close(pipe2[STDIN_FILENO]);
        close(pipe3[STDIN_FILENO]);
        close(pipe3[STDOUT_FILENO]);

        dup2(pipe1[STDIN_FILENO], STDIN_FILENO);
        close(pipe1[STDIN_FILENO]);

        dup2(pipe2[STDOUT_FILENO], STDOUT_FILENO);
        close(pipe2[STDOUT_FILENO]);

        execl("./child1", "child1", NULL);
        perror("Ошибка при выполнении child1");
        exit(EXIT_FAILURE);
    }

    pid_t child2 = fork();
    if (child2 == -1) {
        perror("Ошибка при создании дочернего процесса");
        exit(EXIT_FAILURE);
    }

    if (child2 == 0) {
        close(pipe1[STDIN_FILENO]);
        close(pipe1[STDOUT_FILENO]);
        close(pipe2[STDOUT_FILENO]);
        close(pipe3[STDIN_FILENO]);

        dup2(pipe2[STDIN_FILENO], STDIN_FILENO);
        close(pipe2[STDIN_FILENO]);

        dup2(pipe3[STDOUT_FILENO], STDOUT_FILENO);
        close(pipe3[STDOUT_FILENO]);

        execl("./child2", "child2", NULL);
        perror("Ошибка при выполнении child2");
        exit(EXIT_FAILURE);
    }

    close(pipe1[STDIN_FILENO]);
    close(pipe2[STDIN_FILENO]);
    close(pipe2[STDOUT_FILENO]);
    close(pipe3[STDOUT_FILENO]);

    printf("Введите строки (нажмите Enter для завершения):\n");
    ssize_t bytes;
    while ((bytes = read(STDIN_FILENO, buffer, sizeof(buffer))) > 0) {
        if (buffer[0] == '\n' && bytes == 1) {
            break;
        }
        write(pipe1[STDOUT_FILENO], buffer, bytes);

        ssize_t nread = read(pipe3[STDIN_FILENO], buffer, sizeof(buffer));
        if (nread > 0) {
            write(STDOUT_FILENO, buffer, nread);
        }
```

```
    }

    close(pipe1[STDOUT_FILENO]);
    close(pipe3[STDIN_FILENO]);

    waitpid(child1, NULL, 0);
    waitpid(child2, NULL, 0);

    return 0;
}
```

child1.c

```
#include <stdint.h>
#include <stdbool.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define BUF_SIZE 4096

int main() {
    char buffer[BUF_SIZE];

    while (true) {
        ssize_t bytes = read(STDIN_FILENO, buffer, sizeof(buffer));

        if (bytes < 0) {
            const char msg[] = "error: failed to read from stdin in child1\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }

        if (bytes == 0) {
            break;
        }

        for (int i = 0; i < bytes; i++) {
            buffer[i] = tolower((unsigned char)buffer[i]);
        }

        if (write(STDOUT_FILENO, buffer, bytes) != bytes) {
            const char msg[] = "error: failed to write to stdout in child1\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
    }

    exit(EXIT_SUCCESS);
}
```

child2.c

```c
#include <stdint.h>
#include <stdbool.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define BUF_SIZE 4096

int main() {
    char buffer[BUF_SIZE];

    while (true) {
        ssize_t bytes = read(STDIN_FILENO, buffer, sizeof(buffer));

        if (bytes < 0) {
            const char msg[] = "error: failed to read from stdin in child2\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }

        if (bytes == 0) {
            break;
        }

        for (int i = 0; i < bytes; i++) {
            if (isspace((unsigned char)buffer[i])) {
                buffer[i] = '_';
            }
        }

        if (write(STDOUT_FILENO, buffer, bytes) != bytes) {
            const char msg[] = "error: failed to write to stdout in child2\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }

        if (write(STDOUT_FILENO, "\n", 1) != 1) {
            const char msg[] = "error: failed to write newline to stdout in child2\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
    }

    exit(EXIT_SUCCESS);
}
```

Протокол работы программы

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs# cd lab1/src

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs/lab1/src# gcc parent.c -o parent -lm

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs/lab1/src# gcc child1.c -o child1 -lm

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs/lab1/src# gcc child2.c -o child2 -lm

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs/lab1/src# ./parent

Введите строки (нажмите Enter для завершения):

Hi! My name Is John

hi!_my_name_is_john_

I AM 19 YEARS OLD

i_am_19_years_old___

I REALLY WANT to ComplEtE thIS LAB

i__really_want_to_complete_this_lab_

SOME MORE TeStS

some_more_tests_

And one mo reE

and__one_mo_ree_

LEEEts go

leeets__go_

Strace:

root@DESKTOP-VOD4IPT:/mnt/d/ClionProjects/OSlabs/lab1/src# strace ./parent

execve("./parent", ["./parent"], 0x7fffc60efb40 /* 27 vars */) = 0

brk(NULL)   = 0x55f10acf2000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdb581d610) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7faa1beab000

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=18647, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 18647, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7faa1bea6000

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) = 0x7faa1bc7d000

mprotect(0x7faa1bca5000, 2023424, PROT_NONE) = 0

mmap(0x7faa1bca5000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7faa1bca5000

mmap(0x7faa1be3a000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7faa1be3a000

mmap(0x7faa1be93000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7faa1be93000

mmap(0x7faa1be99000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7faa1be99000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7faa1bc7a000

arch_prctl(ARCH_SET_FS, 0x7faa1bc7a740) = 0

set_tid_address(0x7faa1bc7aa10) = 10441

set_robust_list(0x7faa1bc7aa20, 24) = 0

rseq(0x7faa1bc7b0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7faa1be93000, 16384, PROT_READ) = 0

mprotect(0x55f108ef3000, 4096, PROT_READ) = 0

mprotect(0x7faa1bee5000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7faa1bea6000, 18647) = 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=0x7faa1bc7aa10) = 10442

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7faa1bc7aa10) = 10443

close(3) = 0

close(5) = 0

close(6) = 0

close(8) = 0

```
newfstatat(1, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH)
= 0

getrandom("\x26\xf9\x9a\x3a\x39\x61\x31\xc7", 8, GRND_NONBLOCK) = 8

brk(NULL)                               = 0x55f10acf2000

brk(0x55f10ad13000)                     = 0x55f10ad13000

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \
321\201\321\202\321\200\320\276\320\272\320\270 (\320\275\320"..., 80Введите строки (нажмите
Enter для завершения):

) = 80

read(0, 0x7ffdb581c6b0, 4096)           = ? ERESTARTSYS (To be restarted if SA_RESTART is
set)

--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---

read(0, 4

"4\n", 4096)                    = 2

write(4, "4\n", 2)                      = 2

read(7, "4_\n", 4096)                   = 3

write(1, "4_\n", 34_

)                       = 3

read(0, PriVET

"PriVET\n", 4096)               = 7

write(4, "PriVET\n", 7)                 = 7

read(7, "privet_\n", 4096)              = 8

write(1, "privet_\n", 8privet_

)              = 8

read(0, NOW THE  STRACE WORKS

"NOW THE  STRACE WORKS\n", 4096) = 22

write(4, "NOW THE  STRACE WORKS\n", 22) = 22

read(7, "now_the__strace_works_\n", 4096) = 23

write(1, "now_the__strace_works_\n", 23now_the__strace_works_

) = 23

read(0,

"\n", 4096)                     = 1

close(4)                                = 0

close(7)                                = 0
```

wait4(10442, NULL, 0, NULL) = ? ERESTARTSYS (To be restarted if SA_RESTART is set)

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=10443, si_uid=0, si_status=0, si_utime=0, si_stime=0} ---

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=10442, si_uid=0, si_status=0, si_utime=0, si_stime=0} ---

wait4(10442, NULL, 0, NULL) = 10442

wait4(10443, NULL, 0, NULL) = 10443

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы была разработана программа, в которой родительский процесс принимает входные данные и отправляет их в дочерний процесс, который обрабатывает эти данные и передает их второму дочернему процессу, который, в свою очередь, тоже обрабатывает данные и возвращает их в родительский процесс, который выводит эти данные на экран. В ходе выполнения я столкнулся с трудностями использования двух пайпов для общения между тремя процессами, все удалось решить добавив дополнительный пайп между двумя дочерними процессами.