Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу
«Операционные системы»

Тема работы
“Взаимодействие между процессами”

Студент: Тутаев Владимир Владимирович
Группа: М8О-201Б-23
Вариант: 18

Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

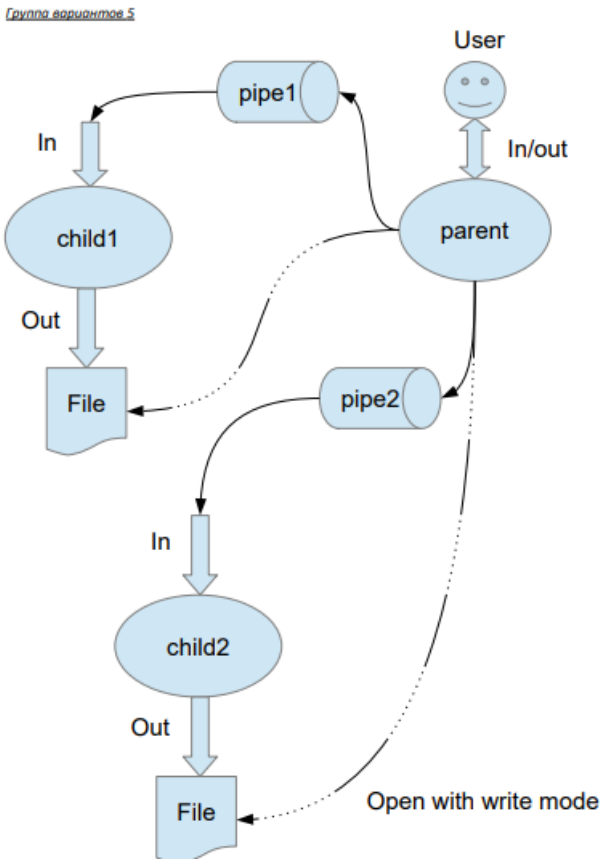
Москва, 2024

Репозиторий

https://github.com/Volan4ik/MAI_OS.git

Постановка задачи

Задача: Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работ



Вариант 18) Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы удаляют все гласные из строк.

Общие сведения

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Исходный код в Приложение 1

Strace в Приложение 2

Вывод:

При разработке программы, я ознакомился с различными функциями работы с процессами. Функция fork используется для создания дочерних процессов, pipe - для создания каналов для передачи данных между процессами, dup2 - для перенаправления потоков ввода-вывода, а execv - для запуска новой программы в контексте дочернего процесса.

Приложение 1

child.c

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

#define MAX_LINE_LENGTH 1024

void remove_vowels(char *str) {
    char *src = str, *dst = str;
    while (*src) {
        if (*src == 'a' || *src == 'e' || *src == 'i' || *src == 'o' || *src == 'u' ||
            *src == 'A' || *src == 'E' || *src == 'I' || *src == 'O' || *src == 'U') {
            src++;
        } else {
            *dst++ = *src++;
        }
    }
    *dst = '\0';
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    char *filename = argv[1];

    int file = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (file == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    char line[MAX_LINE_LENGTH];
    while (fgets(line, MAX_LINE_LENGTH, stdin)) {
        remove_vowels(line);
        write(file, line, strlen(line));
    }

    close(file);
    printf("Child process finished.\n");
    return 0;
}
```

parent.c

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

#define MAX_LINE_LENGTH 1024

int main() {
    int pipe1[2], pipe2[2];
    pid_t pid1, pid2;
    char filename1[MAX_LINE_LENGTH], filename2[MAX_LINE_LENGTH];
    char line[MAX_LINE_LENGTH];
    int line_number = 0;

    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    printf("Enter filename for child1: ");
    fgets(filename1, sizeof(filename1), stdin);
    filename1[strcspn(filename1, "\n")] = '\0';

    printf("Enter filename for child2: ");
    fgets(filename2, sizeof(filename2), stdin);
    filename2[strcspn(filename2, "\n")] = '\0';

    // Создаем первый дочерний процесс
    pid1 = fork();
    if (pid1 == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid1 == 0) {
        close(pipe1[1]);
        close(pipe2[0]);
        close(pipe2[1]);

        dup2(pipe1[0], STDIN_FILENO);
        close(pipe1[0]);

        execl("./child", "./child", filename1, NULL);
        perror("execl error");
        exit(EXIT_FAILURE);
    }
```

```
    // Создаем второй дочерний процесс
    pid2 = fork();
    if (pid2 == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid2 == 0) {
        close(pipe2[1]);
        close(pipe1[0]);
        close(pipe1[1]);

        dup2(pipe2[0], STDIN_FILENO);
        close(pipe2[0]);

        execl("./child", "./child", filename2, NULL);
        perror("execl error");
        exit(EXIT_FAILURE);
    }

    // Родительский процесс
    close(pipe1[0]);
    close(pipe2[0]);

    while (fgets(line, sizeof(line), stdin)) {
        line_number++;
        if (line_number % 2 == 1) {
            write(pipe1[1], line, strlen(line) + 1);
        } else {
            write(pipe2[1], line, strlen(line) + 1);
        }
    }

    close(pipe1[1]);
    close(pipe2[1]);

    wait(NULL);
    wait(NULL);

    printf("Parent process finished.\n");
    return 0;
}
```

Приложение 2

execve("./parent", ["./parent"], 0x7ffc4c19d440 /* 75 vars */) = 0

brk(NULL) = 0x5a587e06a000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7e5f1a6d6000

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

fstat(3, {st_mode=S_IFREG|0644, st_size=87915, ...}) = 0

mmap(NULL, 87915, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7e5f1a6c0000

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\0\220\243\2\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

fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7e5f1a400000

mmap(0x7e5f1a428000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7e5f1a428000

mmap(0x7e5f1a5b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7e5f1a5b0000

mmap(0x7e5f1a5ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7e5f1a5ff000

mmap(0x7e5f1a605000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7e5f1a605000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7e5f1a6bd000

arch_prctl(ARCH_SET_FS, 0x7e5f1a6bd740) = 0

set_tid_address(0x7e5f1a6bda10)   = 918927

set_robust_list(0x7e5f1a6bda20, 24)   = 0

rseq(0x7e5f1a6be060, 0x20, 0, 0x53053053) = 0

mprotect(0x7e5f1a5ff000, 16384, PROT_READ) = 0

mprotect(0x5a587c55f000, 4096, PROT_READ) = 0

mprotect(0x7e5f1a70e000, 8192, PROT_READ) = 0

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

munmap(0x7e5f1a6c0000, 87915)   = 0

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

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

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...}) = 0

getrandom("\x34\x6b\x64\x5e\xd0\x3c\xc8\xe8", 8, GRND_NONBLOCK) = 8

brk(NULL)   = 0x5a587e06a000

brk(0x5a587e08b000)   = 0x5a587e08b000

fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...}) = 0

write(1, "Enter filename for child1: ", 27) = 27

read(0, "test1.txt\n", 1024)   = 10

write(1, "Enter filename for child2: ", 27) = 27

read(0, "test2.txt\n", 1024)   = 10

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7e5f1a6bda10) = 918968

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7e5f1a6bda10) = 918969

close(3)   = 0

close(5)   = 0

read(0, "\321\207\321\221\321\202\320\275\320\276\320\265\n", 1024) = 13

write(4, "\321\207\321\221\321\202\320\275\320\276\320\265\n\0", 14) = 14

```
read(0, "\320\275\320\265\321\207\321\221\321\202\320\275\320\276\320\265\320\265\n", 1024) = 19
write(6, "\320\275\320\265\321\207\321\221\321\202\320\275\320\276\320\265\320\265\n\0", 20) = 20
read(0, 0x5a587e06a6b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGINT {si_signo=SIGINT, si_code=SI_KERNEL} ---
+++ killed by SIGINT +++
```