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-214Б-23

Студент: Маркелов Я.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.11.24

Москва, 2024

Постановка задачи

Вариант 15.

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

Вариант 15) Правило проверки: строка должна начинаться с заглавной буквы

Общий метод и алгоритм решения

Использованные системные вызовы:

- **int pipe(int *fd):**
  Создает пару файловых дескрипторов для межпроцессного взаимодействия (pipe). Один дескриптор используется для записи, другой - для чтения.
- **int open(const char *pathname, int flags, mode_t mode):**
  Открывает файл с заданным именем. Параметры определяют режим открытия (например, только для записи, создание нового файла, обрезка существующего файла) и права доступа для нового файла.
- **pid_t fork(void):**
  Создает новый процесс (дочерний) путем копирования текущего (родительского) процесса. Возвращает PID дочернего процесса в родительском процессе и 0 в дочернем.
- **int dup2(int oldfd, int newfd):**
  Дублирует файловый дескриптор oldfd в newfd, перенаправляя стандартный ввод или вывод (например, связывает ввод-вывод с pipe).
- **int execvp(const char *file, char *const argv[]):**
  Замещает текущий процесс новым процессом, запуская исполняемый файл с аргументами. Если выполнение не удается, возвращает -1.
- **ssize_t read(int fd, void *buf, size_t count):**
  Читает данные из файла (или pipe) в буфер. Возвращает количество фактически прочитанных байтов или 0, если достигнут конец файла.
- **ssize_t write(int fd, const void *buf, size_t count):**
  Записывает данные из буфера в файл (или pipe). Возвращает количество фактически записанных байтов.
- **int close(int fd):**
  Закрывает файловый дескриптор, освобождая ресурсы, связанные с ним.
- **pid_t wait(int *status):**
  Ожидает завершения дочернего процесса. Возвращает PID завершившегося процесса или -1 в случае ошибки.
- **void perror(const char *s):**

Выводит сообщение об ошибке, основанное на коде ошибки, связанной с последним системным вызовом, с предварительным сообщением, указанным в параметре s.

- **void exit(int status):**
  Завершает выполнение программы с указанным статусом, освобождая все ресурсы.
- **size_t strcspn(const char *s, const char *reject):**
  Вычисляет длину начального сегмента строки s, который не содержит ни одного из символов в строке reject.
- **int snprintf(char *str, size_t size, const char *format, ...):**
  Форматирует строку и записывает ее в буфер, гарантируя, что не произойдет переполнение буфера.
- **int isupper(int c):**
  Проверяет, является ли символ заглавной буквой.

Описание программы

Данный код состоит из двух частей: родительского процесса и дочернего процесса. Родительский процесс собирает ввод пользователя, передает данные дочернему процессу, который проверяет валидность строк и возвращает результаты обратно родительскому процессу. Все валидные и невалидные строки записываются в файл.

Основные функции программы:

1. **Создание pipe:**

   Программа создает два трубопровода (pipe) для межпроцессного взаимодействия. Один трубопровод используется для передачи строк от родительского процесса к дочернему, а другой — для передачи результатов обратно.

2. **Запрос имени файла:**

   Родительский процесс запрашивает у пользователя имя файла, в который будут записываться результаты.

3. **Открытие файла:**

   Файл открывается для записи, создается, если не существует, и обрезается, если существует.

4. **Создание дочернего процесса:**

   Используя fork(), программа создает дочерний процесс. Если создание дочернего процесса прошло успешно, он перенаправляет стандартный ввод и стандартный вывод через dup2() на соответствующие трубопроводы.

5. **Ввод строк:**

   Родительский процесс считывает строки из стандартного ввода (консоли) и передает их через трубопровод в дочерний процесс.

6. **Обработка строк в дочернем процессе:**

Дочерний процесс считывает строки из стандартного ввода (который теперь перенаправлен на первый трубопровод). Он проверяет каждую строку: если первая буква строки заглавная, считается, что строка валидная, и она помечается как "Валидная строка". Если строка невалидная, она помечается как "Не валидная строка".

7. **Запись результатов в файл:**

   Родительский процесс считывает результаты из второго трубопровода и записывает их в указанный файл.

8. **Завершение работы:**

   После завершения передачи данных родительский процесс ожидает завершения дочернего процесса с помощью wait() и закрывает открытые файловые дескрипторы.

Как работает программа:

- Программа сначала создает два трубопровода для межпроцессного взаимодействия.
- Она запрашивает у пользователя имя файла, в который будут записываться результаты.
- Далее, программа открывает файл и создает дочерний процесс с помощью fork().
- В дочернем процессе стандартный ввод и вывод перенаправляются на трубопроводы.
- Родительский процесс запрашивает ввод строк от пользователя и записывает их в первый трубопровод.
- Дочерний процесс считывает строки из первого трубопровода, проверяет их на валидность и отправляет результаты через второй трубопровод.
- Родительский процесс считывает результаты из второго трубопровода и записывает их в файл.
- После завершения всех операций происходит очистка: закрываются файловые дескрипторы и дочерний процесс завершается.

Примечания:

- Программа использует fgets() для считывания строк, что позволяет обрабатывать ввод с учетом пробелов.
- Для определения валидности строки используется функция isupper(), проверяющая, является ли первый символ строки заглавной буквой.
- В случае ошибок при работе с файлами или создании процессов, программа выводит сообщение об ошибке с помощью perror(), после чего завершается.

Код программы

parent.c

```c
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>
```

```c
#include <fcntl.h>
#include <sys/wait.h>

#define BUFFER_SIZE 1024
#define ERROR_BUFFER_SIZE 256

int main() {
    int pipe1[2];
    int pipe2[2];
    char filename[BUFFER_SIZE];

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

    printf("Введите имя файла: ");
    scanf("%s", filename);

    int file_descriptor = open(filename, O_WRONLY | O_CREAT |
O_TRUNC, S_IRUSR | S_IWUSR);
    if (file_descriptor == -1) {
        perror("Не удалось открыть файл");
        exit(EXIT_FAILURE);
    }

    pid_t child_pid = fork();
    if (child_pid == -1) {
        perror("Не удалось создать процесс");
        exit(EXIT_FAILURE);
```

```
    }

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

        dup2(pipe1[0], STDIN_FILENO);
        dup2(pipe2[1], STDOUT_FILENO);

        char *args[] = {"./child", NULL};
        execvp(args[0], args);

        perror("Не удалось запустить дочерний процесс");
        exit(EXIT_FAILURE);
    } else {
        char input[BUFFER_SIZE];
        char valid_msg[BUFFER_SIZE];

        close(pipe1[0]);
        close(pipe2[1]);

        printf("Введите строки (CTRL+D для завершения):\n");
        while (fgets(input, sizeof(input), stdin) != NULL) {
            input[strcspn(input, "\n")] = '\0';
            write(pipe1[1], input, strlen(input) + 1);
        }
        close(pipe1[1]);

        while (1) {
```

```
            ssize_t bytes_read = read(pipe2[0], valid_msg,
sizeof(valid_msg));

            if (bytes_read > 0) {

                valid_msg[bytes_read] = '\0'; // Завершение строки

                if (!strcmp(strstr(valid_msg, "Валидная строка:"),
valid_msg)) {

                    write(file_descriptor, valid_msg,
strlen(valid_msg)); // Запись валидной строки в файл

                } else {

                    write(file_descriptor, valid_msg,
strlen(valid_msg)); // Запись невалидной строки в файл

                }

            } else if (bytes_read == 0) {

                break; // Конец чтения

            }

        }


        wait(NULL);

        close(pipe2[0]);

        close(file_descriptor);

    }


    return 0;

}
```

child.c

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <ctype.h>
```

```c
#define BUFFER_SIZE 1024
#define ERROR_BUFFER_SIZE 256


int main() {
    char input[BUFFER_SIZE];
    char error_msg[ERROR_BUFFER_SIZE];


    // Чтение из стандартного ввода (должно быть по pipe)
    while (read(STDIN_FILENO, input, sizeof(input)) > 0) {
        // Удаляем символ новой строки
        input[strcspn(input, "\n")] = '\0';


        if (isupper(input[0])) {
            char tmp[256];
            int ret = snprintf(tmp, sizeof(tmp), "Валидная строка: %s\n", input);
            if (ret < 0){
                abort();
            }
            write(STDERR_FILENO, tmp, strlen(tmp));
            write(STDOUT_FILENO, tmp, strlen(tmp));


        } else {
            if (strlen(input)){
                int ret = snprintf(error_msg, sizeof(error_msg), "Не валидная строка: %s\n", input);
                if (ret < 0){
                    abort();
                }
                write(STDOUT_FILENO, error_msg, strlen(error_msg));
            }
```

```
        }
    }

    return 0;
}
```

Протокол работы программы

Тестирование:

```
$ gcc parent.c -o parent
$ gcc child.c -o child
$ ./parent
Введите имя файла: 1234
Введите строки (CTRL+D для завершения):
QQQQ
Валидная строка: QQQQ
WWWWWW qqqq
Валидная строка: WWWWWW qqqq
1234
52
Файл 1234:
Валидная строка: QQQQ
Валидная строка: WWWWWW qqqq
Не валидная строка: 1234
Не валидная строка: 52
```

Strace:

```
$ strace -f ./main

execve("./parent", ["./parent"], 0x7ffc33a8d6e0 /* 64 vars */) = 0
brk(NULL)                               = 0x55e87e83a000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe24bfaab0) = -1 EINVAL (Недопустимый аргумент)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f816be67000
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=81643, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 81643, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f816be53000
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\0i8\235HZ\227\223\333\350s\360\352,\223\340."..
., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0644, 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) = 0x7f816bc2b000
mmap(0x7f816bc53000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
 3, 0x28000) = 0x7f816bc53000
mmap(0x7f816bde8000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
 0x1bd000) = 0x7f816bde8000
mmap(0x7f816be40000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
 3, 0x214000) = 0x7f816be40000
mmap(0x7f816be46000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
 -1, 0) = 0x7f816be46000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f816bc28000
arch_prctl(ARCH_SET_FS, 0x7f816bc28740) = 0
set_tid_address(0x7f816bc28a10)         = 6058
set_robust_list(0x7f816bc28a20, 24)     = 0
rseq(0x7f816bc290e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f816be40000, 16384, PROT_READ) = 0
mprotect(0x55e87d315000, 4096, PROT_READ) = 0
mprotect(0x7f816bea1000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f816be53000, 81643)           = 0
pipe2([3, 4], 0)                        = 0
pipe2([5, 6], 0)                        = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH)
= 0
getrandom("\x94\x73\x3f\x1f\x11\xad\xa6\x97", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x55e87e83a000
brk(0x55e87e85b000)                     = 0x55e87e85b000
newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH)
= 0
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"..., 34Введите имя
 файла: ) = 34
read(0, QQQQ
"QQQQ\n", 1024)                 = 5
openat(AT_FDCWD, "QQQQ", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 7
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
 child_tidptr=0x7f816bc28a10) = 6061
close(3)                                = 0
close(6)                                = 0
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 (CTR"..., 66Введите строки (CTRL+D
для
 завершения):
) = 66
```

```
write(4, "\0", 1)                       = 1
read(0, 12345
"12345\n", 1024)              = 6
write(4, "12345\0", 6)                 = 6
read(0, qqqqqq
"qqqqqq\n", 1024)             = 7
write(4, "qqqqqq\0", 7)                = 7
read(0, wwwww
"wwwww\n", 1024)              = 6
write(4, "wwwww\0", 6)                 = 6
read(0, QQQQQ
"QQQQQ\n", 1024)              = 6
write(4, "QQQQQ\0", 6Валидная строка: QQQQQ
)                  = 6
read(0, "", 1024)                       = 0
close(4)                                = 0
read(5, "\320\235\320\265
 \320\262\320\260\320\273\320\270\320\264\320\275\320\260\321\217
 \321\201\321\202\321\200\320\276\320\272"..., 1024) = 164
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=6061, si_uid=1000,
 si_status=0, si_utime=0, si_stime=0} ---
write(7, "\320\235\320\265
 \320\262\320\260\320\273\320\270\320\264\320\275\320\260\321\217
 \321\201\321\202\321\200\320\276\320\272"..., 164) = 164
read(5, "", 1024)                       = 0
wait4(-1, NULL, 0, NULL)                = 6061
close(5)                                = 0
close(7)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Сегодня мы научились разрабатывать программу на C, которая использует межпроцессное взаимодействие через трубопроводы (pipes). Мы реализовали родительский процесс, который принимает пользовательский ввод и передает его дочернему процессу для проверки на валидность. В зависимости от результата проверки, валидные и невалидные строки записываются в указанный файл, что позволяет эффективно обрабатывать и сохранять данные. Возникли проблемы с чтением из pipe в child.c