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-213Б-23

Студент: Солодова С. М.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 04.10.24

Москва, 2024

Постановка задачи

Вариант 22.

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

Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы инвертируют строки.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int32_t execv(const char *__path, char *const *__argv); – заменяет текущий образ программы новым образом, загружая и выполняя программу, путь к которой передается в аргументах.
- ssize_t read(int __fd, void *__buf, size_t __nbytes); – читает данные из файлового дескриптора и записывает их в буфер.
- ssize_t write(int __fd, void *__buf, size_t __nbytes); – записывает данные из буфера в файловый дескриптор.
- int32_t open(const char*__file, int __oflag, …); – открывает файл и возвращает файловый дескриптор.
- int close(int __fd); – закрывает файл.
- pid_t wait(int *__stat_loc); – приостанавливает выполнение родительского процесса до тех пор, пока дочерний не будет выполнен.
- int pipe(int *__pipedes); – создает канал для однонаправленной передачи данных между двумя процессами.
- int dup2(int oldfd, int newfd) - переназначение файлового дескриптора

1. Родительский процесс (`parent.c`) запрашивает у пользователя имена файлов для двух дочерних процессов.

2. Затем создает два канала связи (pipes) с помощью системных вызовов `pipe()`.

3. После этого родительский процесс создает два дочерних процесса с помощью системного вызова `fork()`.

4. Каждый дочерний процесс выполняет следующие действия:

    - Закрывает ненужные дескрипторы каналов.

    - Перенаправляет стандартный ввод на чтение из соответствующего канала.

    - Открывает файл для записи с помощью системного вызова `open()`.

    - Перенаправляет стандартный вывод на запись в открытый файл.

    - Запускает программу `child` с помощью системного вызова `execl()`.

5. Родительский процесс закрывает ненужные дескрипторы каналов.

6. Родительский процесс читает строки, введенные пользователем, с помощью функции `read_line()`.

7. Для каждой строки, введенной пользователем, родительский процесс генерирует случайное число с помощью `rand()`.

8. В зависимости от значения случайного числа, родительский процесс записывает строку в соответствующий канал связи с помощью `write()`.

9. Родительский процесс закрывает записи в оба канала.

10. Далее он ожидает завершения дочерних процессов с помощью `waitpid()`.

11. В конце концов процесс освобождает выделенную память..

Код программы

parent.c

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

```
#include <errno.h>
#include <time.h>

#define BUFFER_SIZE 1024

void error_exit(const char *msg) {
    ssize_t len = 0;
    while (msg[len] != '\0') len++;
    write(STDERR_FILENO, msg, len);
    exit(EXIT_FAILURE);
}

void write_str(int fd, const char *str) {
    size_t len = 0;
    while (str[len] != '\0') len++;
    ssize_t total_written = 0;
    while (total_written < (ssize_t)len) {
        ssize_t written = write(fd, str + total_written, len - total_written);
        if (written == -1) {
            error_exit("write failed\n");
        }
        total_written += written;
    }
}

char* read_line() {
    size_t size = BUFFER_SIZE;
    size_t len = 0;
    char *buffer = (char*)malloc(size);
    if (!buffer) {
        error_exit("malloc failed\n");
    }
    while (1) {
        char c;
        ssize_t bytes = read(STDIN_FILENO, &c, 1);
        if (bytes == -1) {
            free(buffer);
            error_exit("read failed\n");
        } else if (bytes == 0) {
```

```
            if (len == 0) {
                buffer[0] = '\0';
            }
            break;
        }
        if (c == '\n') {
            break;
        }
        buffer[len++] = c;
        if (len >= size) {
            size += BUFFER_SIZE;
            char *new_buffer = realloc(buffer, size);
            if (!new_buffer) {
                free(buffer);
                error_exit("realloc failed\n");
            }
            buffer = new_buffer;
        }
    }
    buffer[len] = '\0';
    return buffer;
}

int main() {
    if (time(NULL) == ((time_t) -1)) {
        error_exit("time failed\n");
    }
    srand(time(NULL));

    write_str(STDOUT_FILENO, "Введите имя файла для child1: ");
    char *filename1 = read_line();
    if (filename1 == NULL) {
        error_exit("Failed to read filename1\n");
    }
    if (strlen(filename1) == 0) {
        free(filename1);
        error_exit("Filename1 is empty\n");
    }
```

```
write_str(STDOUT_FILENO, "Введите имя файла для child2: ");
char *filename2 = read_line();
if (filename2 == NULL) {
    free(filename1);
    error_exit("Failed to read filename2\n");
}
if (strlen(filename2) == 0) {
    free(filename1);
    free(filename2);
    error_exit("Filename2 is empty\n");
}

int pipe1[2];
if (pipe(pipe1) == -1) {
    free(filename1);
    free(filename2);
    error_exit("pipe1 creation failed\n");
}

int pipe2[2];
if (pipe(pipe2) == -1) {
    close(pipe1[0]);
    close(pipe1[1]);
    free(filename1);
    free(filename2);
    error_exit("pipe2 creation failed\n");
}

pid_t pid1 = fork();
if (pid1 == -1) {
    close(pipe1[0]);
    close(pipe1[1]);
    close(pipe2[0]);
    close(pipe2[1]);
    free(filename1);
    free(filename2);
    error_exit("fork1 failed\n");
}
if (pid1 == 0) {
```

```
    if (close(pipe1[1]) == -1) {
        error_exit("child1 close pipe1 write end failed\n");
    }
    if (close(pipe2[0]) == -1) {
        error_exit("child1 close pipe2 read end failed\n");
    }
    if (close(pipe2[1]) == -1) {
        error_exit("child1 close pipe2 write end failed\n");
    }

    if (dup2(pipe1[0], STDIN_FILENO) == -1) {
        error_exit("child1 dup2 pipe1 failed\n");
    }
    if (close(pipe1[0]) == -1) {
        error_exit("child1 close pipe1 read end after dup2 failed\n");
    }

    int file_fd = open(filename1, O_WRONLY | O_CREAT | O_TRUNC, 0666);
    if (file_fd == -1) {
        error_exit("child1 open file1 failed\n");
    }

    if (dup2(file_fd, STDOUT_FILENO) == -1) {
        close(file_fd);
        error_exit("child1 dup2 file1 failed\n");
    }
    if (close(file_fd) == -1) {
        error_exit("child1 close file1 failed\n");
    }

    execl("./child", "child", NULL);
    error_exit("child1 execl child failed\n");
}

pid_t pid2 = fork();
if (pid2 == -1) {
    close(pipe1[0]);
    close(pipe1[1]);
    close(pipe2[0]);
```

```
    close(pipe2[1]);
    free(filename1);
    free(filename2);
    waitpid(pid1, NULL, 0);
    error_exit("fork2 failed\n");
}
if (pid2 == 0) {
    if (close(pipe2[1]) == -1) {
        error_exit("child2 close pipe2 write end failed\n");
    }
    if (close(pipe1[0]) == -1) {
        error_exit("child2 close pipe1 read end failed\n");
    }
    if (close(pipe1[1]) == -1) {
        error_exit("child2 close pipe1 write end failed\n");
    }

    if (dup2(pipe2[0], STDIN_FILENO) == -1) {
        error_exit("child2 dup2 pipe2 failed\n");
    }
    if (close(pipe2[0]) == -1) {
        error_exit("child2 close pipe2 read end after dup2 failed\n");
    }

    int file_fd = open(filename2, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (file_fd == -1) {
        error_exit("child2 open file2 failed\n");
    }

    if (dup2(file_fd, STDOUT_FILENO) == -1) {
        close(file_fd);
        error_exit("child2 dup2 file2 failed\n");
    }
    if (close(file_fd) == -1) {
        error_exit("child2 close file2 failed\n");
    }

    execl("./child", "child", NULL);
    error_exit("child2 execl child failed\n");
```

```
}

if (close(pipe1[0]) == -1) {
    write_str(STDERR_FILENO, "parent close pipe1 read end failed\n");
}
if (close(pipe2[0]) == -1) {
    write_str(STDERR_FILENO, "parent close pipe2 read end failed\n");
}

write_str(STDOUT_FILENO, "Введите строку (Ctrl+D для завершения):\n");

while (1) {
    char *line = read_line();
    if (line == NULL) {
        break;
    }

    if (line[0] == '\0') {
        free(line);
        break;
    }

    int r = rand() % 10;
    if (r < 8) {
        write_str(pipe1[1], line);
        write_str(pipe1[1], "\n");
    } else {
        write_str(pipe2[1], line);
        write_str(pipe2[1], "\n");
    }
    free(line);
}

if (close(pipe1[1]) == -1) {
    write_str(STDERR_FILENO, "parent close pipe1 write end failed\n");
}
if (close(pipe2[1]) == -1) {
    write_str(STDERR_FILENO, "parent close pipe2 write end failed\n");
}
```

```
    int status;
    if (waitpid(pid1, &status, 0) == -1) {
        write_str(STDERR_FILENO, "waitpid for child1 failed\n");
    }
    if (waitpid(pid2, &status, 0) == -1) {
        write_str(STDERR_FILENO, "waitpid for child2 failed\n");
    }

    free(filename1);
    free(filename2);

    return 0;
}
```

child.c

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

void error_exit(const char *msg) {
    ssize_t len = 0;
    while (msg[len] != '\0') len++;
    write(STDERR_FILENO, msg, len);
    exit(EXIT_FAILURE);
}

void write_str(int fd, const char *str) {
    size_t len = 0;
    while (str[len] != '\0') len++;
    ssize_t total_written = 0;
    while (total_written < (ssize_t)len) {
        ssize_t written = write(fd, str + total_written, len - total_written);
        if (written == -1) {
            error_exit("write failed\n");
        }
        total_written += written;
    }
}

char* read_line() {
    size_t size = 1024;
    size_t len = 0;
    char *buffer = (char*)malloc(size);
```

```
    if (!buffer) {
        error_exit("malloc failed\n");
    }

 while (1) {
        char c;
        ssize_t bytes = read(STDIN_FILENO, &c, 1);
        if (bytes == -1) {
            free(buffer);
            error_exit("read failed\n");
        } else if (bytes == 0) {
            if (len == 0) {
                buffer[0] = '\0';
            }
            break;
        }
        if (c == '\n') {
            break;
        }
        buffer[len++] = c;
        if (len >= size) {
            size += 1024;
            char *new_buffer = realloc(buffer, size);
            if (!new_buffer) {
                free(buffer);
                error_exit("realloc failed\n");
            }
            buffer = new_buffer;
        }
    }
    buffer[len] = '\0';
    return buffer;
}

void reverse_string(char *str) {
    size_t len = 0;
    while (str[len] != '\0') len++;
    for (size_t i = 0; i < len / 2; i++) {
        char tmp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = tmp;
    }
}

int main() {
    while (1) {
        char *line = read_line();
        if (line == NULL) {
            break;
```

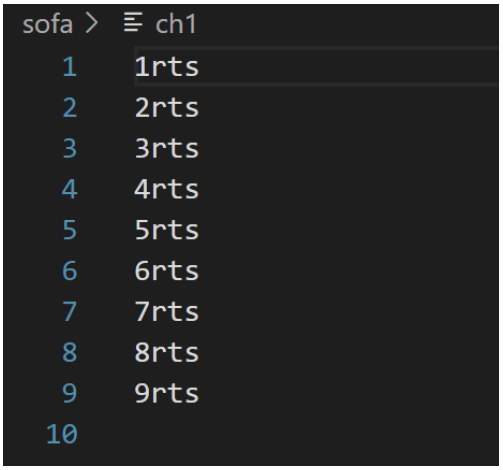
```
        }

        if (line[0] == '\0') {
            free(line);
            break;
        }

        reverse_string(line);

        write_str(STDOUT_FILENO, line);
        write_str(STDOUT_FILENO, "\n");

        free(line);
    }
    return 0;
}
```

Протокол работы программы

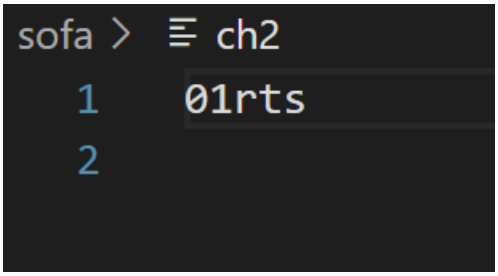
Тестирование:

```
wabisabi@wabisabi:/mnt/c/Users/zlata/OneDrive/Рабочий стол/labs/sofa$ gcc -o parent parent.c
wabisabi@wabisabi:/mnt/c/Users/zlata/OneDrive/Рабочий стол/labs/sofa$ gcc -o child child.c
wabisabi@wabisabi:/mnt/c/Users/zlata/OneDrive/Рабочий стол/labs/sofa$ ./parent
Введите имя файла для child1: ch1
Введите имя файла для child2: ch2
Введите строку (Ctrl+D для завершения):
str1
str2
str3
str4
str5
str6
str7
str8
str9
str10
wabisabi@wabisabi:/mnt/c/Users/zlata/OneDrive/Рабочий стол/labs/sofa$
```

Ch1:



Ch2:



Strace: wabisabi@wabisabi:/mnt/c/Users/zlata/OneDrive/Рабочий стол/labs/sofa$ strace ./parent

execve("./parent", ["./parent"], 0x7ffc3524a8a0 /* 35 vars */) = 0
brk(NULL) = 0x5588e7858000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fdf6ce20000
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=19243, ...}) = 0
mmap(NULL, 19243, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fdf6ce1b000
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\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) = 0x7fdf6cc09000
mmap(0x7fdf6cc31000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fdf6cc31000
mmap(0x7fdf6cdb9000, 323584, PROT_READ,

MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7fdf6cdb9000
mmap(0x7fdf6ce08000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7fdf6ce08000
mmap(0x7fdf6ce0e000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fdf6ce0e000
close(3) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fdf6cc06000
arch_prctl(ARCH_SET_FS, 0x7fdf6cc06740) = 0
set_tid_address(0x7fdf6cc06a10) = 24574
set_robust_list(0x7fdf6cc06a20, 24) = 0
rseq(0x7fdf6cc07060, 0x20, 0, 0x53053053) = 0
mprotect(0x7fdf6ce08000, 16384, PROT_READ) = 0
mprotect(0x5588bafd9000, 4096, PROT_READ) = 0
mprotect(0x7fdf6ce58000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fdf6ce1b000, 19243) = 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"..., 48Введите имя файла
для child1:) = 48
getrandom("\x3b\x98\xe5\x51\x9a\x64\xe0\x08", 8, GRND_NONBLOCK) = 8
brk(NULL) = 0x5588e7858000
brk(0x5588e7879000) = 0x5588e7879000
read(0, ch1
"c", 1) = 1
read(0, "h", 1) = 1
read(0, "1", 1) = 1
read(0, "\n", 1) = 1
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"..., 48Введите имя файла
для child2:) = 48
read(0, ch2
"c", 1) = 1
read(0, "h", 1) = 1
read(0, "2", 1) = 1
read(0, "\n", 1) = 1
pipe2([3, 4], 0) = 0
pipe2([5, 6], 0) = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fdf6cc06a10) = 24926
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fdf6cc06a10) = 24928
close(3) = 0
close(5) = 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\321\203 (Ctr"..., 66Введите строку (Ctrl+D для
завершения):

```
) = 66
read(0, str1
"s", 1)                 = 1
read(0, "t", 1)                 = 1
read(0, "r", 1)                 = 1
read(0, "1", 1)                 = 1
read(0, "\n", 1)                = 1
write(6, "str1", 4)               = 4
write(6, "\n", 1)                = 1
read(0, str2
"s", 1)                 = 1
read(0, "t", 1)                 = 1
read(0, "r", 1)                 = 1
read(0, "2", 1)                 = 1
read(0, "\n", 1)                = 1
write(6, "str2", 4)               = 4
write(6, "\n", 1)                = 1
read(0, str3
"s", 1)                 = 1
read(0, "t", 1)                 = 1
read(0, "r", 1)                 = 1
read(0, "3", 1)                 = 1
read(0, "\n", 1)                = 1
write(4, "str3", 4)               = 4
write(4, "\n", 1)                = 1
read(0, "", 1)                  = 0
close(4)                        = 0
close(6)                        = 0
wait4(24926, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 24926
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=24926, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
wait4(24928, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 24928
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=24928, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
exit_group(0)                   = ?
+++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы была успешно разработана программа на языке C, реализующая модель взаимодействия между родительским процессом и двумя дочерними процессами. Эта программа была нацелена на демонстрацию работы с потоками ввода-вывода и принципами управления процессами в операционной системе Linux. Основными системными вызовами для этого являются: fork(), pipe(), а также read(), write() и open(). Лабораторная работа послужила отличным практическим примером использования теоретических знаний о системах, работе с процессами и управлении памятью, что особенно актуально для дальнейшего изучения программирования на языке C и разработки приложений в Linux.