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: Кудаева В.А.

Преподаватель: Бахарев В.Д.

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

Дата: 31.10.24

Москва, 2024

**Постановка задачи**

**Вариант 16:**

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. 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. Правило проверки: строка должна оканчиваться на «.» или «;».

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t fork(void); – создает дочерний процесс.
* int pipe(int \*fd); – создаёт пайп и помещает дескрипторы в fd[0], fd[1], для чтения и записи.
* int write(int fd, const void\* buff, int count); – записывает по дескриптору fd count байт из buff.
* void exit(int number); – вызывает нормальное завершение программы с кодом number.
* int dup2(int fd1, int fd2); – делает эквиваентными дескрипторы fd1 и fd2.
* int exec(char\* path, const char\* argc); – заменяет текущий процесс на процесс path, с аргументами argc;
* int close(int fd); – закрывает дескриптор fd.
* pid\_t wait(int status) — функция, которая приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится.

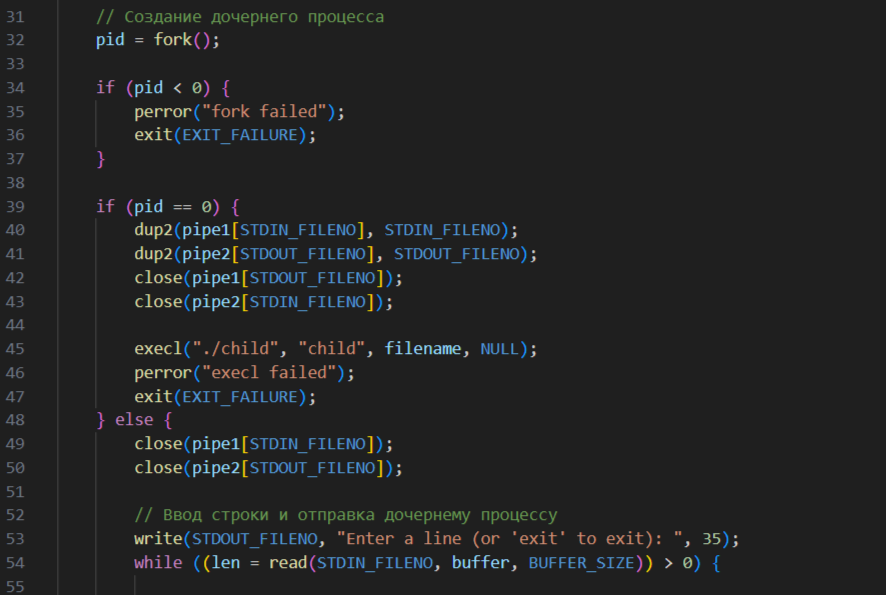
В файле parent.c создается родительский и дочерний процессы, которые обмениваются данными через два канала (pipe1 и pipe2). Родительский процесс запрашивает имя файла и отправляет строки на запись в этот файл дочернему процессу, который принимает строки через канал, проверяет их на валидность (оканчиваются ли они на . или ;) и записывает в файл, если строка валидна. Дочерний процесс отвечает родителю сообщением "OK" или об ошибке, а родитель выводит это сообщение или завершает передачу, если пользователь вводит "exit".

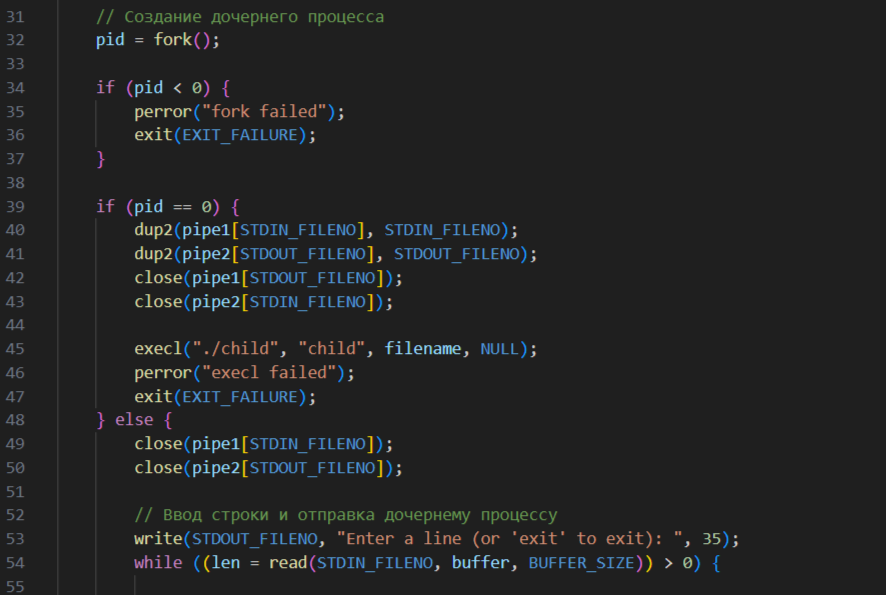
В файле child.c дочерний процесс принимает имя файла, которое передается как аргумент командной строки, и открывает файл для записи. После получения строки через стандартный ввод, процесс проверяет её на корректное окончание и записывает в файл, если строка валидна, отправляя родительскому процессу подтверждение "OK" или сообщение об ошибке.

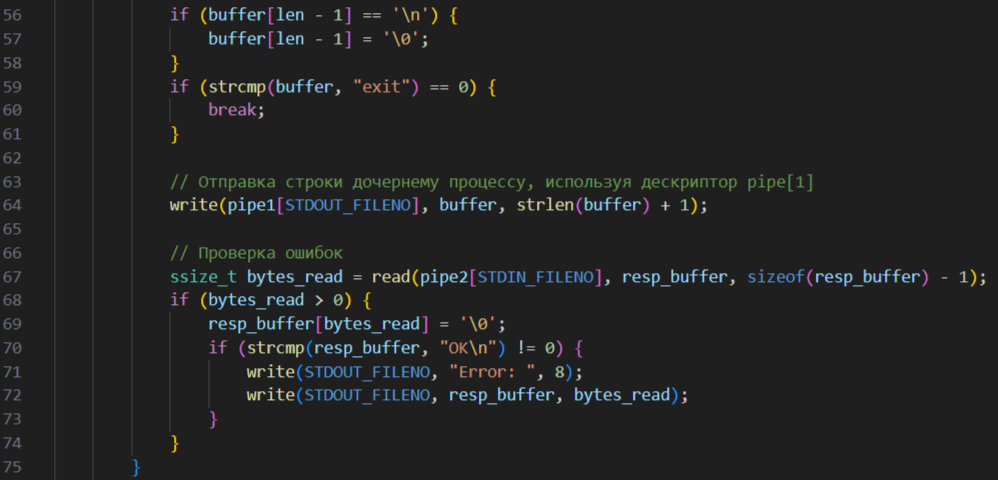
**Код программы**

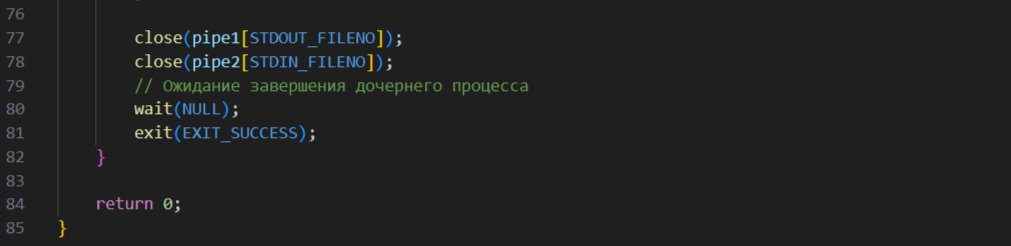
**parent.c**

****

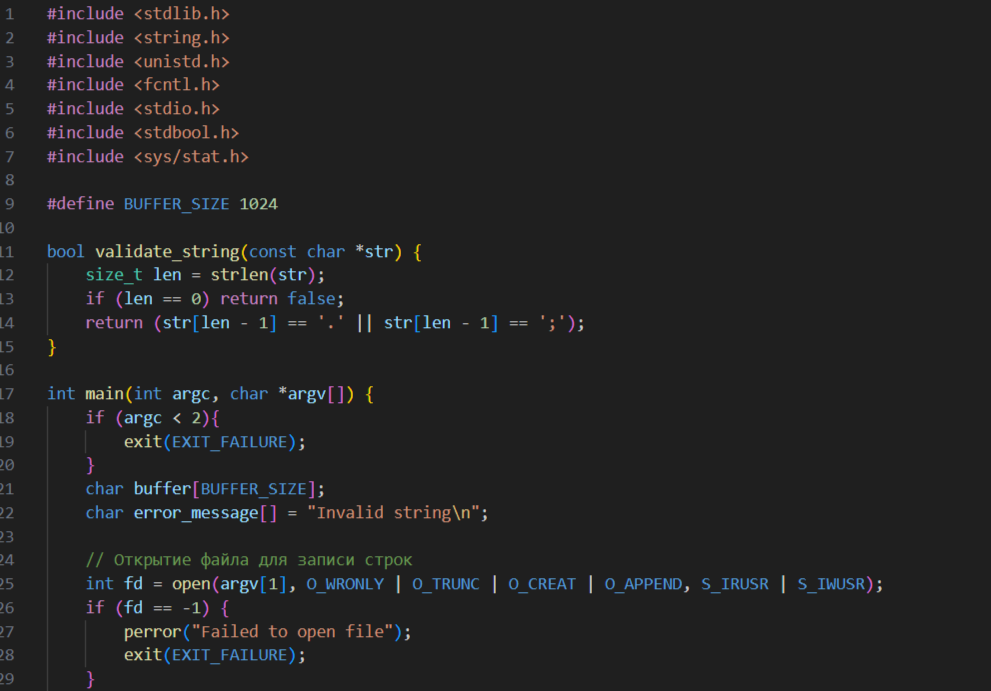
****

****

****

****

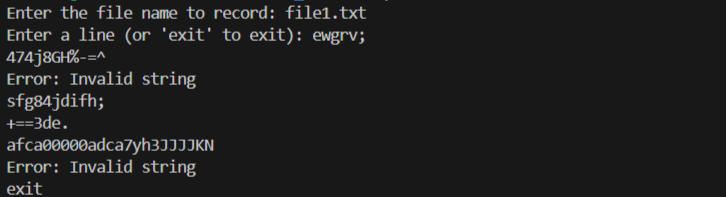
**child.c**

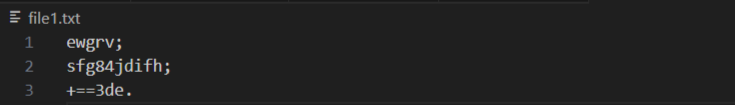
****

****

**Протокол работы программы**

**Тестирование**

****

****

strace ./parent

execve("./parent", ["./parent"], 0x7ffcf4df51b0 /\* 72 vars \*/) = 0

brk(NULL) = 0x558ec6415000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fff7293bdd0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f1a0177d000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/home/ksenoox/OpenFOAM/ksenoox-v2312/platforms/linux64GccDPInt32Opt/lib/glibc-hwcaps/x86-64-v4/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/usr/lib/openfoam/openfoam2312/platforms/linux64GccDPInt32Opt/lib/dummy/libc.so.6", O\_RDONLY|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/lib/openfoam/openfoam2312/platforms/linux64GccDPInt32Opt/lib/dummy", {st\_mode=S\_IFDIR|0755, st\_size=4096, ...}, 0) = 0

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=27211, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 27211, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f1a01776000

**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) = 0x7f1a0154d000

mprotect(0x7f1a01575000, 2023424, PROT\_NONE) = 0

mmap(0x7f1a01575000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f1a01575000

mmap(0x7f1a0170a000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f1a0170a000

mmap(0x7f1a01763000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f1a01763000

mmap(0x7f1a01769000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f1a01769000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f1a0154a000

arch\_prctl(ARCH\_SET\_FS, 0x7f1a0154a740) = 0

set\_tid\_address(0x7f1a0154aa10) = 334966

set\_robust\_list(0x7f1a0154aa20, 24) = 0

rseq(0x7f1a0154b0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f1a01763000, 16384, PROT\_READ) = 0

mprotect(0x558ec5867000, 4096, PROT\_READ) = 0

mprotect(0x7f1a017b7000, 8192, PROT\_READ) = 0

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

munmap(0x7f1a01776000, 27211) = 0

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

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

**write(1, "Enter the file name to record: \0", 32Enter the file name to record: ) = 32**

**read(0, svsf;**

**"svsf;\n", 256) = 6**

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f1a0154aa10) = 335084**

**close(3) = 0**

**close(6) = 0**

**write(1, "Enter a line (or 'exit' to exit)"..., 35Enter a line (or 'exit' to exit): ) = 35**

**read(0, dfb**

**"dfb\n", 1024) = 4**

**write(4, "dfb\0", 4) = 4**

**read(5, "Invalid string\n", 1023) = 15**

**write(1, "Error: \0", 8Error: ) = 8**

**write(1, "Invalid string\n", 15Invalid string**

**) = 15**

**read(0, exit**

**"exit\n", 1024) = 5**

**close(4) = 0**

**close(5) = 0**

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=335084, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

**wait4(-1, NULL, 0, NULL) = 335084**

**exit\_group(0) = ?**

+++ exited with 0 +++

**Вывод**

В ходе лабораторной работы мне удалось реализовать взаимодействие между родительским и дочерним процессами с использованием каналов pipe и системных вызовов. Я узнала, что такое процесс и канал, а также научилась пользоваться некоторыми системными вызовами.