

Московский Авиационный Институт
(Национальный Исследовательский Университет)



Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу

«Операционные системы»

Студентка:

Варламова Анна Борисовна

Группа: М80-207Б-20

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 08.11.2021

Москва, 2021

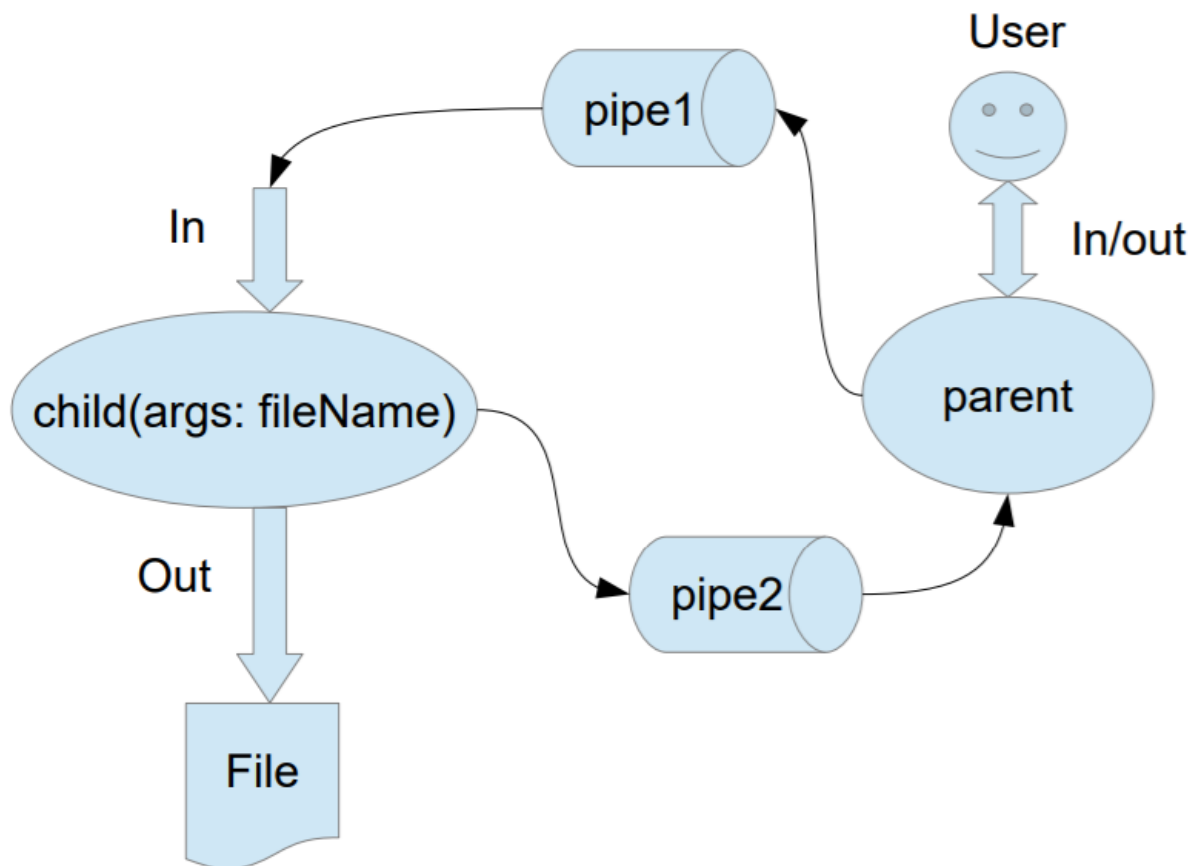
Описание

Данная лабораторная работа будет выполняться в ОС Unix.

Процесс – абстракция, описывающая выполняющуюся программу. В моем случае, имеется два процесса: процесс-родитель и процесс-ребенок. Взаимодействие между процессами будет осуществляться с помощью неименованных каналов (fd1, fd2).

Ниже представлена схема сообщения между родительским процессом и дочерним, они представлены разными программами.

Группа вариантов 1



Родительский процесс создаёт дочерний. Первой строчкой пользователь вводит имя файла, которое будет передано в дочерний, туда запишутся результаты работы дочернего процесса. Далее родительский процесс считывается команды вида: <число число число endl>, числа типа float. Родительский процесс через pipe1 передаёт команды в дочерний, который в свою очередь делит первое число команды на последующие и записывает результат в открытый вначале файл. Если встречается деление на 0, через pipe2 дочерний процесс передаёт

информацию об этом родительскому процессу, оба процесса завершаются.

Правильная последовательность работы процессов осуществляется при помощи `wait`. Этот системный вызов приостанавливает выполнение текущего процесса до тех пор, пока не завершится дочерний процесс.

Системный вызов — обращение прикладной программы к ядру операционной системы для выполнения какой-либо операции.

Использованные системные вызовы:

`pid_t fork(void);` — создает дочерний процесс. Если возвращает 0, то созданный текущий процесс — ребенок, если >0 , то — родитель, если <0 , то — ошибка(и текущий процесс — родитель).

`exit(int status);` — выходит из процесса с заданным статусом.

`pid_t wait(int *status);` — он же **`pid_t waitpid(pid_t pid, int *status, int options);`** где `options=0`. Приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится.

`int pipe(int fd);` — предоставляет средства передачи данных между двумя процессами(неименованный канал).

`int close(int fd);` — закрывает файловый дескриптор.

`int read(int fd, void *buffer, int nbyte);` — читает `nbyte` байтов из файлового дескриптора `fd` в буффер `buffer`.

`int dup(int oldfd)` — создаёт копию файлового дескриптора `oldfd`, дальше их можно использовать как взаимозаменяемые.

`int dup2(int newfd, int oldfd)` — тоже создаёт копию, теперь `newfd` заменяет `oldfd`

Исходный код

```
//parent.cpp
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <vector>
#include <iostream>
#include <sys/wait.h>
```

```

int main(void) {
    char *filename = NULL;
    size_t len = 0;
    std::cout << "Enter the name of file for answers: ";
    fflush(stdout);
    if (getline(&filename, &len, stdin) == -1) {
        perror("getline");
        _exit(EXIT_FAILURE);
    }
    filename[strlen(filename) - 1] = '\0';

    std::cout << "Enter numerous:\n" << std::endl;

    int fd1[2], fd2[2];
    if ((pipe(fd1) == -1) || (pipe(fd2) == -1)){
        perror("pipe");
        _exit(EXIT_FAILURE);
    }

    int old_stdout = dup(fileno(stdout));

    pid_t pid = fork();
    if (pid == -1) {
        perror("fork");
        _exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        close(fd2[0]);
    }
}

```

```

close(fd1[1]);

if (dup2(fd1[0], fileno(stdin)) == -1) {
    perror("child, dup2, stdin");
    _exit(EXIT_FAILURE);
}
close(fd1[0]);

if (dup2(fd2[1], fileno(stdout)) == -1) {
    perror("child, dup2, stdout");
    _exit(EXIT_FAILURE);
}

if (execl("child.out", "child.out", filename, NULL) == -1) {
    perror("execl");
    _exit(EXIT_FAILURE);
}
}

int status;
if (pid > 0) {
    close(fd1[0]);
    close(fd2[1]);
    if (dup2(fd1[1], fileno(stdout)) == -1) {
        perror("parent, dup2, stdout");
        _exit(EXIT_FAILURE);
    }
    char err = '1', c;
    float x;

```

```

std::vector<float> vec;

while (scanf("%f%c", &x, &c) > 0) {
    vec.push_back(x);
    if (c == '\n') {
        int n = vec.size();
        std::cout << n << " ";
        for (int i = 0; i < n; ++i) {
            std::cout << vec[i] << " ";
        }
        vec.clear();
        fflush(stdout);
        read(fd2[0], &err, sizeof(1));
        if (err == '0') {
            wait(&status);
            close(fd2[0]);
            close(fd1[1]);
            dup2(old_stdout, fileno(stdout));
            close(old_stdout);
            std::cout << "Divizion by zero" << std::endl;
            fflush(stdout);
            _exit(EXIT_FAILURE);
        }
        if (err == '2') {
            wait(&status);
            close(fd2[0]);
            close(fd1[1]);
            dup2(old_stdout, fileno(stdout));
            close(old_stdout);
            std::cout << "Opening file" << std::endl;

```

```

        fflush(stdout);
        _exit(EXIT_FAILURE);
    }
}

close(fd1[1]);
close(fd2[0]);
close(fileno(stdout));
}

if (wait(&status) == -1) {
    perror("wait");
    _exit(EXIT_FAILURE);
}

return 0;
}

```

//child.cpp

```

#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <vector>
#include <iostream>

int main(int argc, char *argv[]) {
    char err = '1';

    int myfile = open(argv[1], O_CREAT | O_WRONLY, S_IRWXU);

    if (myfile == -1) {
        err = '2';
    }
}

```

```

std::cout << err;

fflush(stdout);

_exit(EXIT_FAILURE);
}

float x, res;

int n, i;

while (std::cin >> n) {
    for (i = 0; i < n; ++i) {
        std::cin >> x;

        if (i == 0) {
            res = x;
        } else {
            if (x == 0) {
                err = '0';

                std::cout << err;

                fflush(stdout);

                close(myfile);

                _exit(EXIT_FAILURE);
            } else {
                if (i > 0) {
                    res /= x;
                }
            }
        }
    }

    dprintf(myfile, "%f\n", res);

    fflush(stdout);

    std::cout << err;
}

```



```

    fflush(stdout);
}
close(myfile);
return 0;
}

```

Тесты

```
ann@ann:~/os/lab2$ ./parent.out
```

Enter the name of file for answers: test.txt

Enter numerous:

45 9 5

35 7 5

1000 10 2

36.4 56.3 2.1

256 2 2

23 3

```
ann@ann:~/os/lab2$ cat test.txt
```

1.000000

1.000000

50.000000

0.307875

64.000000

7.666667

```
ann@ann:~/os/lab2$ ./parent.out
```

Enter the name of file for answers: 1.txt

Enter numerous:

123455 3 4

4 0 5

Divizion by zero

```
ann@ann:~/os/lab2$ cat 1.txt
```

10287.916992

Вывод

Благодаря данной лабораторной работе я на практике изучила принципы работы с неименованными каналами для межпроцессного взаимодействия, я разобралась, как перенаправлять потоки ввода/вывода, а также научилась использовать системные вызовы и обращаться с файловыми дескрипторами.

Также я по достоинству оценила функцию `fflush`, которая решила большую часть моих проблем, важно не забывать её использовать, чтобы протолкнуть данные по каналу, иначе без неё в некоторых ситуациях программа может зависнуть. Также важно вовремя и уместно закрывать файловые дескрипторы.

Очевидно, что в сложных программах используется большее количество неименованных каналов, чем два. Эта лабораторная научила базовому обращению с ними.