

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

Лабораторная работа №2 по курсу
«Операционные системы»

Тема работы
Процессы в ОС и обмен данными между ними

Студент: Полонский Кирилл Андреевич
Группа: М8О-208Б-20
Вариант: 15
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021
Содержание

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

Репозиторий

https://github.com/kirillpolonskii/OS/tree/master/os_lab2

Постановка задачи

Родительский процесс получает на вход имя файла, использующегося для записи, и строки произвольной длины, после чего создаёт дочерний процесс и пересылает их туда через `pipe1`. Дочерний процесс проверяет строки на соответствие правилу "Строка должна начинаться с маленькой буквы"; правильные строки выводятся в стандартный поток вывода дочернего процесса, неправильные через `pipe2` посылаются обратно в дочерний процесс и выводятся в стандартный поток вывода родительского процесса.

Общие сведения о программе

Файл `parent.cpp` содержит весь исходный код двух процессов, сборка осуществляется с помощью утилиты `make`. В `Makefile` описаны флаги `-fsanitize=address` и `-g` для отслеживания ошибок и строк, в которых они возникают.

Общий метод и алгоритм решения

Программа принимает на вход имя файла для вывода, создаёт два неименованных канала и системным вызовом `fork()` создаёт дочерний процесс.

Родительский процесс принимает количество строк и сами строки, записывая их в `pipe1`. Дочерний процесс читает строки из основного потока ввода, который системным вызовом `dup2` заменяется на конец для чтения в `pipe1`,

после чего проверяет их первый символ на принадлежность отрезку в кодировке ASCII и в соответствии с правилом либо записывает их в файл, либо пересылает в родительский процесс через pipe2. Родительский процесс читает из pipe2 количество ошибочных строк и сами строки, а затем выводит их в стандартный поток вывода и записывает в файл.

Программа запускается без ключей.

Исходный код

```
#include <iostream>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <string>
#include "string.h"
#include <vector>
#include <fcntl.h>
#include "unistd.h"

int main() {
    std::string childProgName("child.cpp");
    std::cout << "Enter file name:" << std::endl;
    std::string fileName;
    std::cin >> fileName;

    int ruleFileDs[2];
    int errorsFileDs[2];
    int pipe1 = pipe(ruleFileDs); // pipe1 for check the rule
    int pipe2 = pipe(errorsFileDs); // pipe2 for sending error messages to the parent

    int END_CODE = 100000;
    int file;
    if ((file = open(fileName.c_str(), O_WRONLY | O_TRUNC | O_CREAT,
S_IWUSR | S_IRUSR)) == -1){
        return 1;
    }
    if (pipe1 == -1 || pipe2 == -1){
        std::cout << "Pipe1 or pipe2 has errors.\n";
    }
    int pid = fork(); // When fork () is called, two completely identical processes arise.
```

// All code after the fork () is executed twice, both in
the child and parent processes

```
switch (pid) {
    case -1:{
        std::cout << "Fork has errors.\n";
        return -1;
    }
    case 0:{// It's child process. Now we need to decide on the direction of data
transfer -
        // if we need to transfer data from parent to child, then the parent closes the
descriptor
        // for reading, and the child closes the descriptor for writing

        dup2(ruleFileDs[0], STDIN_FILENO);
        dup2(file, STDOUT_FILENO);
        close(ruleFileDs[1]);
        close(errorsFileDs[0]);
        int strAmount;
        read(STDIN_FILENO, &strAmount, sizeof(int));
        int errorsAmount = 0;
        int curStrLength;
        for (int i = 0; i < strAmount; ++i) {
            read(STDIN_FILENO, &curStrLength, sizeof(int));
            char* c_curStr = (char *) malloc(sizeof(char) * curStrLength);
            read(STDIN_FILENO, c_curStr, sizeof(char) * curStrLength);

            if (c_curStr[0] >= 65 && c_curStr[0] <= 90) {
                char message[17] = "Correct string: ";
                char* newC_curStr = (char*) malloc(sizeof(char) * (curStrLength +
17));

                for(int j = 0; j < 16; ++j){
                    newC_curStr[j] = message[j];
                }
                for(int j = 16; j < curStrLength + 16; ++j){
                    newC_curStr[j] = c_curStr[j-16];
                }
                newC_curStr[curStrLength + 16] = '\n';

                write(STDOUT_FILENO, newC_curStr, sizeof(char) * (curStrLength +
17));

                free(c_curStr);
            }
        }
    }
}
```

```

        free(newC_curStr);
    } else {
        write(errorsFileDs[1], &curStrLength, sizeof(int));
        write(errorsFileDs[1], c_curStr, sizeof(char) * curStrLength);
        free(c_curStr);
    }
}

write(errorsFileDs[1], &END_CODE, sizeof(int));
close(ruleFileDs[0]);
close(errorsFileDs[1]);
break;
}
default: { // It's a parent process.
    close(ruleFileDs[0]);
    close(errorsFileDs[1]);

    std::string curStr;
    std::cout << "Enter amount of strings:" << std::endl;
    int strAmount;
    std::cin >> strAmount;
    write(ruleFileDs[1], &strAmount, sizeof(int)); // writing in pipe1
    std::cout << "Enter " << strAmount << " strings:" << std::endl;
    for (int i = 0; i < strAmount; ++i){
        std::cin >> curStr;
        int curStrLength = curStr.size();
        char* c_curStr = (char*) malloc(sizeof(char) * curStrLength);
        for (int j = 0; j < curStrLength; ++j){
            c_curStr[j] = curStr[j];
        }

        write(ruleFileDs[1], &curStrLength, sizeof(int));
        write(ruleFileDs[1], c_curStr, sizeof(char) * curStrLength);
        free(c_curStr);
    }
    // at this moment pipe1 is needed only for reading
    close(ruleFileDs[1]);
    int curStrLength;
    while (true){
        read(errorsFileDs[0], &curStrLength, sizeof(int));
        if (curStrLength == END_CODE){
            break;
        }
    }
}

```

```

char *c_errorStr = (char *) malloc(sizeof(char) * curStrLength);
read(errorsFileDs[0], c_errorStr, sizeof(char) * curStrLength);

write(STDOUT_FILENO, "In PARENT: <", sizeof(char) * 12);
write(STDOUT_FILENO, c_errorStr, sizeof(char) * curStrLength);
write(STDOUT_FILENO, "> doesn't start with capital\n", sizeof(char)*
29);

char message[19] = "Incorrect string: ";
char* newC_errorStr = (char*) malloc(sizeof(char) * (curStrLength +
19));

for(int j = 0; j < 18; ++j){
    newC_errorStr[j] = message[j];
}
for(int j = 18; j < curStrLength + 18; ++j){
    newC_errorStr[j] = c_errorStr[j-18];
}
newC_errorStr[curStrLength + 18] = '\n';
write(file, newC_errorStr, sizeof(char) * (curStrLength + 19));

free(c_errorStr);
free(newC_errorStr);
}
close(errorsFileDs[0]);

}
close(file);
}

return 0;
}

```

Демонстрация работы программы

```
kirill@kirill-acpire:~/labsMAI/sem3/OS/os_lab2/src$ ./parent
Enter file name:
o.txt
Enter amount of strings:
5
Enter 5 strings:
qwe
DGBT
H
p
SGfgb
In PARENT: <qwe> doesn't start with capital
In PARENT: <p> doesn't start with capital
kirill@kirill-acpire:~/labsMAI/sem3/OS/os_lab2/src$ cat o.txt
Correct string: DGBT
Correct string: H
Correct string: SGfgb
Incorrect string: qwe
Incorrect string: p
kirill@kirill-acpire:~/labsMAI/sem3/OS/os_lab2/src$
```

Выводы

В ходе выполнения лабораторной работы я на практике познакомился с процессами в Linux и связанными с ними системными вызовами, а также научился межпроцессорному взаимодействию через неименованные каналы.