

Московский авиационный институт  
(национальный исследовательский университет)

Факультет информационных технологий и прикладной  
математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Операционные системы»

Управление процессами в ОС. Обеспечение обмена данных между  
процессами посредством каналов.

Студент: В. А. Амурский  
Преподаватель: Е. С. Миронов  
Группа: М8О-201Б-21  
Вариант: 11  
Дата:  
Оценка:  
Подпись:

Москва, 2023

# 1 Постановка задачи

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы.

# 2 Сведения о программе

Программа написанна на C++ в Unix подобной операционной системе на базе ядра Linux.

При запуске программы пользователь вводит строки в стандартный поток ввода. Программа создает два дочерних процесса для преобразования введенных строк.

По завершении работы программа выводит в стандартный поток вывода введенные строки в верхнем регистре, удалив все задвоенные пробелы

# 3 Общий метод и алгоритм решения

Родительский процесс создает первый дочерний процесс, передав через pipe1 строки, полученные от пользователя. Затем родительский процесс создает второй дочерний процесс.

Первый дочерний прочесс принимает строки и приводит все символы в верхний регистр, после чего передавая полученные строки во второй дочерний процесс через pipe3

Второй дочерний процесс принимает строки через pipe3, после чего удаляет все

завоенные пробелы и передает полученные строки родительскому процессу через pipe2

Результирующие строки родительский процесс считывает из pipe2

## 4 Листинг программы

### main.cpp

```
1 | #include "include/parent.h"
2 | #include <vector>
3 |
4 | int main() {
5 |     std::vector<std::string> input;
6 |
7 |     std::string s;
8 |     while (getline(std::cin, s)) {
9 |         input.push_back(s);
10 |    }
11 |
12 |    std::vector<std::string> output = ParentRoutine("child1", "child2", input);
13 |
14 |    for (const auto &res : output){
15 |        std::cout << res << std::endl;
16 |    }
17 |    return 0;
18 | }
```

### parent.cpp

```
1 | #include "parent.h"
2 |
3 |
4 | std::vector<std::string> ParentRoutine(char const *pathToChild1, char const *
    pathToChild2,
5 |                                     const std::vector<std::string> &input){
6 |
7 |     std::vector<std::string> output;
8 |
9 |     int firstPipe[2];
10 |    CreatePipe(firstPipe);
11 |    int pipeBetweenChildren[2];
12 |    CreatePipe(pipeBetweenChildren);
13 |
14 |    int pid = fork();
15 |
16 |
17 |
18 |    if (pid == 0) {
```

```

19
20     close(firstPipe[WRITE_END]);
21     close(pipeBetweenChildren[READ_END]);
22
23     MakeDup2(firstPipe[READ_END], STDIN_FILENO);
24     MakeDup2(pipeBetweenChildren[WRITE_END], STDOUT_FILENO);
25
26     if (execl(pathToChild1, "", nullptr) == -1) {
27         GetExecError(pathToChild1);
28     }
29     close(firstPipe[READ_END]);
30     close(firstPipe[WRITE_END]);
31 } else if (pid == -1) {
32     GetForkError();
33 } else {
34     close(firstPipe[READ_END]);
35     for (const std::string & s : input) {
36         std::string s_tmp = s + "\n";
37         write(firstPipe[WRITE_END], s_tmp.c_str(), s_tmp.size());
38     }
39     close(firstPipe[WRITE_END]);
40
41     int secondPipe[2];
42     CreatePipe(secondPipe);
43
44     pid = fork();
45
46     if (pid == 0) {
47         close(secondPipe[READ_END]);
48         close(pipeBetweenChildren[WRITE_END]);
49
50         MakeDup2(pipeBetweenChildren[READ_END], STDIN_FILENO);
51         MakeDup2(secondPipe[WRITE_END], STDOUT_FILENO);
52
53         if (execl(pathToChild2, "", nullptr) == -1) {
54             GetExecError(pathToChild2);
55         }
56         close(pipeBetweenChildren[READ_END]);
57         close(secondPipe[WRITE_END]);
58     } else if (pid == -1) {
59         GetForkError();
60     } else {
61         close(secondPipe[WRITE_END]);
62         close(pipeBetweenChildren[WRITE_END]);
63         close(pipeBetweenChildren[READ_END]);
64
65         wait(nullptr);
66
67         for (size_t i = 0; i < input.size(); i++) {

```

```

68         std::string res;
69         char ch;
70         while (read(secondPipe[READ_END], &ch, 1) && ch != '\n'){
71             res += ch;
72         }
73         output.push_back(res);
74     }
75     close(secondPipe[READ_END]);
76 }
77 }
78 return output;
79 }

```

### child1.cpp

```

1  #include "utils.h"
2
3  int main() {
4      std::string s;
5      while(getline(std::cin, s)) {
6          for (char & ch : s) {
7              ch = toupper(ch);
8          }
9          std::cout << s << '\n';
10     }
11     return 0;
12 }

```

### child2.cpp

```

1  #include "utils.h"
2
3
4  int main() {
5      std::string s;
6      while (getline(std::cin, s)) {
7          int j = 0;
8          char lastCh = '\0';
9          for (int i = 0; i < s.size(); i++){
10             if (lastCh != ' ' || s[i] != ' '){
11                 s[j] = s[i];
12                 j++;
13             }
14             lastCh = s[i];
15         }
16         for (int i = 0; i < j; i++) {
17             std::cout << s[i];
18         }
19         std::cout << '\n';
20     }
21     return 0;

```

22 || }

## utils.cpp

```
1 | #include "utils.h"
2 |
3 | void CreatePipe(int fd[]) {
4 |     if (pipe(fd) != 0) {
5 |         std::cout << "Couldn't create pipe" << std::endl;
6 |         exit(EXIT_FAILURE);
7 |     }
8 | }
9 |
10 | void GetForkError() {
11 |     std::cout << "fork error" << std::endl;
12 |     exit(EXIT_FAILURE);
13 | }
14 |
15 | void MakeDup2(int oldFd, int newFd) {
16 |     if (dup2(oldFd, newFd) == -1) {
17 |         std::cout << "dup2 error" << std::endl;
18 |         exit(EXIT_FAILURE);
19 |     }
20 | }
21 |
22 | void GetExecError(std::string const &executableFile) {
23 |     std::cout << "Exec \"" << executableFile << "\" error." << std::endl;
24 | }
```

## 5 Демонстрация работы программы

```
vazy1@vazy1-legion:~/mai/OS-labs/tests$ cat lab2_test.cpp
#include <gtest/gtest.h>
```

```
#include <array>
#include <memory>
#include <parent.h>
#include <stdlib.h>
#include <vector>
```

```
TEST(FirstLabTests, SimpleTest) {
```

```
constexpr int inputSize = 4;
```

```

std::array<std::vector<std::string>,inputSize>input;
input[0] = {
"abcabc",
"ad SD da",
"USE_LESS",
"KEK sdf_doupf"
};
input[1] = {
" ",
""
};
input[2] = {
"__A ",
" _ _ _"
};
input[3] = {
"\b "
};
std::array<std::vector<std::string>,inputSize>expectedOutput;
expectedOutput[0] = {
"ABCABC",
"AD_SD_DA",
"USE_LESS",
"KEK_SDF_DOUPF"
};
expectedOutput[1] = {
"__",
""
};
expectedOutput[2] = {
"__A__",
" _ _ _ _"
};
expectedOutput[3] = {
"\b_"
};
for (int i = 0; i <inputSize; i++) {
auto result = ParentProcces("/home/vazy1/mai/OS-labs/build/lab2/child1","/home/vazy1/r
EXPECT_EQ(result,expectedOutput[i]);
}
}

```

```
vazy1@vazy1-legion:~/mai/OS-labs/build/tests$ /home/vazy1/mai/OS-labs/build/tests/lab2
Running main() from /home/vazy1/mai/OS-labs/build/_deps/googletest-src/googletest/src
[=====] Running 1 test from 1 test suite.
[-----] Global test environment set-up.
[-----] 1 test from FirstLabTests
[ RUN      ] FirstLabTests.SimpleTest
[          OK ] FirstLabTests.SimpleTest (3 ms)
[-----] 1 test from FirstLabTests (3 ms total)

[-----] Global test environment tear-down
[=====] 1 test from 1 test suite ran. (4 ms total)
[ PASSED   ] 1 test.
```



## 6 Вывод

Одна из основных задач операционной системы - это управление процессами. В большинстве случаев она сама создает процессы для себя и при запуске других программ. Тем не менее бывают случаи, когда необходимо создавать процессы вручную.

В языке Си есть функционал, который позволит нам внутри нашей программы создать дополнительный, дочерний процесс. Этот процесс будет работать параллельно с родительским.

Для этого в языке Си на Unix-подобных ОС используется библиотека `unistd.h`. Эта библиотека позволяет совершать системные вызовы, которые связаны с вводом/выводом, управлением файлами, каталогами и работой с процессами и запуском программ. Для создания дочерних процессов используется функция `fork`. При этом с помощью ветвлений в коде можно отделить код родителя от ребенка. У ребенка при этом можно заменить программу, используя для этого функцию `exec`, а обеспечить связь с помощью `pipe`.

Подобный функционал есть во многих языках программирования, так как большинство современных программ состоят более, чем из одного процесса.