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: А. Р. Боташев
Преподаватель: Е. С. Миронов
Группа: М8О-201Б-21
Вариант: 12
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

задвоенные пробелы и передает полученные строки родительскому процессу через pipe2

Результирующие строки родительский процесс считывает из pipe2

4 Листинг программы

main.cpp

```
#include "include/parent.h"
#include <vector>

int main() {
    std::vector <std::string> input;

    std::string s;
    while (getline(std::cin, s)) {
        input.push_back(s);
    }

    std::vector <std::string> output = ParentRoutine("child1", "child2", input);

    for (const auto &res : output){
        std::cout << res << std::endl;
    }
    return 0;
}
```

parent.cpp

```
#include "parent.h"


std::vector<std::string> ParentRoutine(char const *pathToChild1, char const *
    pathToChild2,
                                       const std::vector<std::string> &input){

    std::vector<std::string> output;

    int firstPipe[2];
    CreatePipe(firstPipe);
    int pipeBetweenChildren[2];
    CreatePipe(pipeBetweenChildren);

    int pid = fork();



    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;
```

```
}
```

utils.cpp

```
#include "utils.h"

void CreatePipe(int fd[]) {
    if (pipe(fd) != 0) {
        std::cout << "Couldn't create pipe" << std::endl;
        exit(EXIT_FAILURE);
    }
}

void GetForkError() {
    std::cout << "fork error" << std::endl;
    exit(EXIT_FAILURE);
}

void MakeDup2(int oldFd, int newFd) {
    if (dup2(oldFd, newFd) == -1) {
        std::cout << "dup2 error" << std::endl;
        exit(EXIT_FAILURE);
    }
}

void GetExecError(std::string const &executableFile) {
    std::cout << "Exec \"" << executableFile << "\" error." << std::endl;
}
```

5 Демонстрация работы программы

```
botashev@botashev-laptop:~/ClionProjects/os_labs/tests$ cat lab2_test.cpp
#include <gtest/gtest.h>

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


TEST(FirstLabTests,SimpleTest) {

constexpr int inputSize = 4;

std::array<std::vector<std::string>,inputSize>input;
```

```
input[0] = {
"abcabc",
"qwerty qwerty",
"A n O t H e R             TeSt",
"oNe1 Two2  thr3ee   5fiVe     Ei8ght        13thiRTEEN             ...",
"2 + 2 = 4",
"0123456789 abcdefghijklmnopqrstuvwxyz"
};
input[1] = {
"second test",
"1234567890/.,']['",
".            .             .                          ..............",
"!?+-*/_;",
};
input[2] = {
"",
"                   "
};
input[3] = {
"AAAAAAAAAAAAAAAAAAAAAAAAAAAA"
};

std::array<std::vector<std::string>,inputSize>expectedOutput;
expectedOutput[0] = {
"ABCABC",
"QWERTY QWERTY",
"A N O T H E R TEST",
"ONE1 TWO2 THR3EE 5FIVE EI8GHT 13THIRTEEN ...",
"2 + 2 = 4",
"0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ"
};
expectedOutput[1] = {
"SECOND TEST",
"1234567890/.,']['",
". . . ..............",
"!?+-*/_;",
};
expectedOutput[2] = {
"",
" ",
};
```

```
expectedOutput[3] = {
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
};


for (int i = 0; i <inputSize; i++) {
auto result = ParentRoutine(getenv("child1"),getenv("child2"),input[i]);
EXPECT_EQ(result,expectedOutput[i]);
}
}
botashev@botashev-laptop:~/ClionProjects/os_labs/tests$ ./../cmake-build-debug/tests/l
Running main() from /home/botashev/ClionProjects/os_labs/cmake-build-debug/_deps/googl
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from FirstLabTests
[ RUN      ] FirstLabTests.SimpleTest
[       OK ] FirstLabTests.SimpleTest (7 ms)
[----------] 1 test from FirstLabTests (7 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (7 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 библеотека unistd.h. Эта библеотека 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, испрользуя для этого функцию exec, а обеспечить связь с помощью pipe.

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