Московский авиационный институт
1 (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: Чирикова П. С.
Преподаватель: Е.С. Миронов
Группа: М8О-201Б-21
Вариант: Дата:
Оценка:
Подпись:

Москва, 2023

2 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.

3 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

4 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

5 Листинг программы

main.cpp

```
#include
"parent.hpp"
#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("chil
d1", "child2",
input);

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

parent.cpp

```
#include <sys/wait.h>
#include    <unistd.h>

#include  "parent.hpp"
#include     "utils.hpp"


std::vector<std::string
>   ParentRoutine(char
const    *pathToChild1,
char                const
*pathToChild2,

const
std::vector<std::string
>        &input)        {
    std::vector<std::stri
ng>                output;

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

```
    CreatePipe(pipeBet
weenChildren);

       int  pid  =  fork();

       if  (pid  ==  0)  {

        close(firstPipe[W
RITE_END]);
       close(pipeBetwee
nChildren[READ_END])
;

        MakeDup2(firstPi
pe[READ_END],
STDIN_FILENO);
        MakeDup2(pipeB
etweenChildren[WRITE
_END],
STDOUT_FILENO);

                        if
(execl(pathToChild1,
"",  nullptr)  ==  -1)  {
          GetExecError(p
athToChild1);
                        }
       close(firstPipe[RE
AD_END]);
        close(firstPipe[W
RITE_END]);
    } else if (pid == -1)
{
          GetForkError();
             }    else    {
       close(firstPipe[RE
AD_END]);
               for  (const
std::string  &s: input)  {
           auto str = s +
"\n";
          write(firstPipe[
WRITE_END],
str.c_str(),     str.size());
                        }
        close(firstPipe[W
RITE_END]);

       int secondPipe[2];
```

```
        CreatePipe(secon
dPipe);

            pid  =  fork();

          if (pid == 0) {
           close(secondPi
pe[READ_END]);
           close(pipeBetw
eenChildren[WRITE_EN
D]);

            MakeDup2(pip
eBetweenChildren[REA
D_END],
STDIN_FILENO);
           MakeDup2(sec
ondPipe[WRITE_END],
STDOUT_FILENO);

                          if
(execl(pathToChild2,
"",  nullptr)  ==  -1)  {
              GetExecError
(pathToChild2);
                          }
        } else if (pid ==
-1)                       {
           GetForkError();
                 }  else  {
           close(secondPi
pe[WRITE_END]);
           close(pipeBetw
eenChildren[WRITE_EN
D]);
           close(pipeBetw
eenChildren[READ_EN
D]);
              wait(nullptr);
                   char  ch;
                std::string s;
           for(size_t i = 0;
i < input.size(); ++i) {
                   s.clear();
               while(read(s
econdPipe[READ_END]
, &ch, 1) && ch != '\n')
{
                   s += ch;
```

```
                        }
            output.push_
back(std::move(s));
                        }
            std::cout <<
std::endl;
          close(secondPi
pe[READ_END]);
                        }
                        }
         return   output;
}
```

child1.cpp

```
#include
"utils.hpp"
#include
<fstream>

int main() {
   std::string s;
   while
(getline(std::cin,
s)) {
      os << s <<
std:: endl;
      std::cout <<
UpReg(s) << '\n';
   }
   return 0;
}
```

child2.cpp

```
#include "utils.hpp"
#include <fstream>

int main() {
    std::string s;
    while (getline(std::cin, s)) {
       os << WithoutDoubleSpace(s) << std:: endl;
    std::cout << WithoutDoubleSpace(s) << '\n';
    }
    return 0;
}
```

utils.cpp

```
1 #include <iostream>
  #include <string>

  #include "utils.hpp"
```

```
std::string UpReg(const std::string& s) {
    std::string upRegStr;
    upRegStr.reserve(s.size());
    for (char i : s) {
        upRegStr += toupper(i);
    }
    return upRegStr;
}

std::string WithoutDoubleSpace(const std::string& s) {
    std::string outDoubleSpace;
    outDoubleSpace += s[0];
    for (size_t i = 1; i < s.length(); i++) {
        if (s[i - 1] != ' ' || s[i] != ' ') {
            outDoubleSpace += s[i];
        }
    }
    return outDoubleSpace;
}

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

6 Демонстрация работы программы

polina@polina-Vostro-3400:~/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] = {
"AAAAAAAAAAAAAAAAAAAAAAAAAAAA"
};


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/
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.
```

7 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 состоят более, чем из одного процесса.