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

Лабораторная работа №2 по курсу
«Операционные системы»

Студент: Павлов Иван Дмитриевич
Группа: М8О-207Б-21
Вариант: 11
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/Pavloffff/MAI_OS

Постановка задачи

Цель работы

Приобретение практических навыков в:

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

Задание

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.

Группа вариантов 3

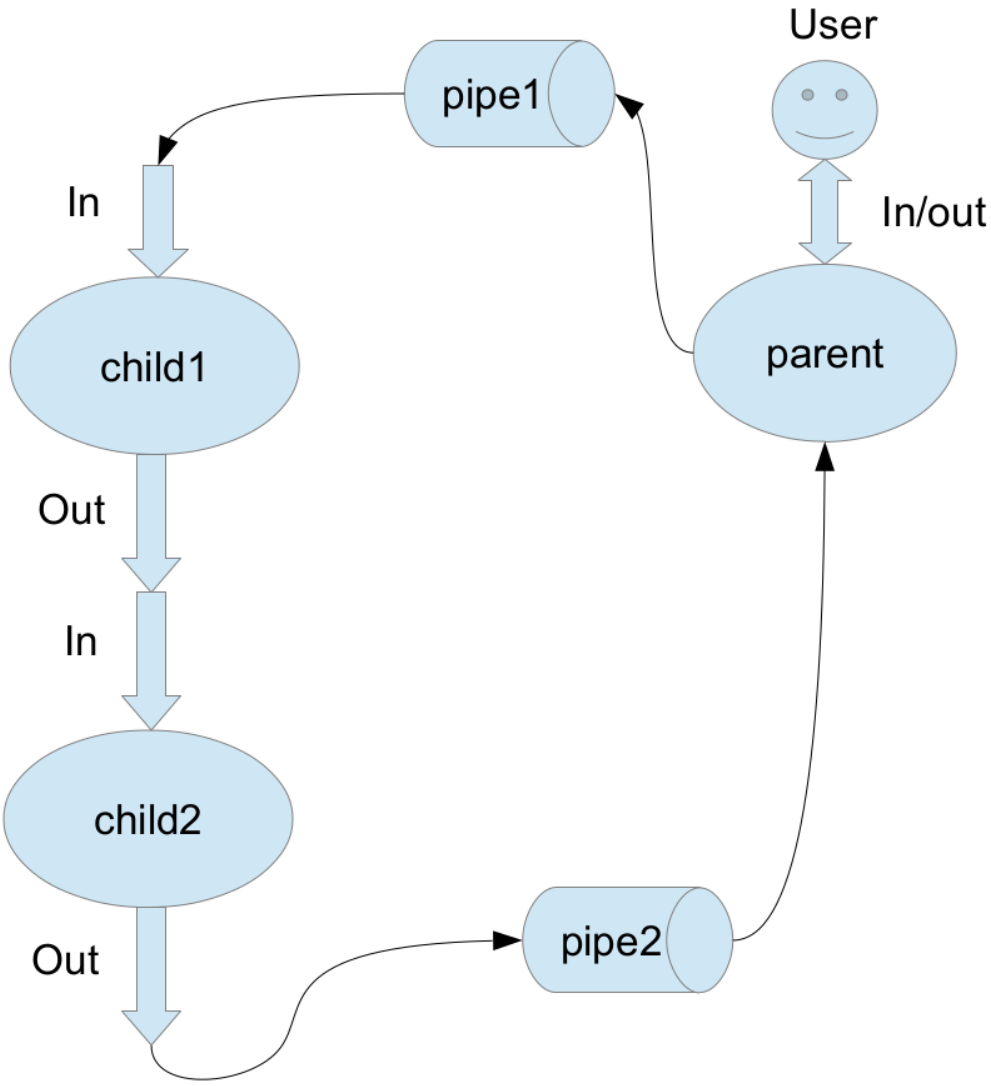
Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами.

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.

Вариант 11

Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «_».

Группа вариантов 3



Общие сведения о программе

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: unistd.h, iostream, string, algorithm. В программе используются следующие системные вызовы:

1. pipe() - существует для передачи информации между различными процессами.
2. fork() - создает новый процесс.
3. execpl() - передает процесс на исполнение другой программе.
4. read() - читает данные из файла.
5. write() - записывает данные в файл.
6. close() - закрывает файл.

Общий метод и алгоритм решения

ЛР выполнена на ОС UNIX, на языке C++.

В условии допущено соединение двух дочерних процессов с помощью дополнительного канала, поэтому создам 3 pipe; дескрипторы будут лежать в двумерном массиве fd[3][2]. Далее создаем первый дочерний процесс с помощью fork(). Проверяем id, которое вернул fork(): если id != 0, то это родительский процесс.

Согласно условию, родительский процесс считывает строки вместе с пробелами, до "\n"; поэтому использую while (getline(std::cin, str)) (до этого я объявил строку из std::string). Далее необходимо передать эти строки первому дочернему процессу (child1), используя write(). Так как write() - сишная функция, приведем строку к массиву char* методом std::string::c_str. Кроме того передаем размер строки, увеличенный на 1 (из-за "\0").

Далее, переходим к child1 (там где id == 0). По условию, родительский и дочерний процессы должны быть представлены разными программами. Поэтому использую execlp, и передаю туда в качестве параметров название исполняемого файла дочернего процесса, и все fd, приведенные к строке в формате Си.

Код дочерних процессов представлен в файле child.cpp. Из argv[] получим все fd. Далее с помощью fork() создадим child2, и если fork вернул id != 0, то приступаю к выполнению задачи child1.

Для того, чтобы считать строку для обработки, сперва с помощью read() считываем размер строки, затем создаем буфер дополнительной памяти этого размера, записываем туда строку и присваиваем строке из C++ этот буфер, после чего чистим память.

По условию child1 преобразует строки в верхний регистр, сделаем это с помощью std::transform(), итераторов и лямбда-выражений. Передадим через другой pipe это процессу 2.

Аналогичные действия совершает и процесс 2, который по завершении итерации передает через 3-й pipe обработанные строки родительскому процессу, который выводит их на экран.

При этом не забываем обрабатывать ошибки, связанные с системными вызовами.

Исходный код

main.cpp

```
#include <iostream>
#include <string>
#include <string.h>
#include <unistd.h>
#include <algorithm>
```

```
int main(int argc, char *argv[])
{
                                        int pipe1fd[2];
   if (pipe(pipe1fd) == -1) {
      std::cerr << "pipe1 error" << std::endl;
      return 1;
   }
                                        int pipe2fd[2];
                                        if (pipe(pipe2fd) == -1) {
      std::cerr << "pipe2 error" << std::endl;
      return 1;
   }
   int pipe3fd[2];
                                        if (pipe(pipe3fd) == -1) {
      std::cerr << "pipe3 error" << std::endl;
      return 1;
   }

                                        int id1 = fork();
                                        if (id1 == -1) {
                                                std::cerr << "fork error" << std::endl;
                                                return 1;
                                        } else if (id1 == 0) {
      char *argv1 = new char[sizeof(int)];
      sprintf(argv1, "%d", pipe1fd[0]);
      char *argv2 = new char[sizeof(int)];
      sprintf(argv2, "%d", pipe1fd[1]);
      char *argv3 = new char[sizeof(int)];
      sprintf(argv3, "%d", pipe3fd[0]);
      char *argv4 = new char[sizeof(int)];
      sprintf(argv4, "%d", pipe3fd[1]);
      execlp("./child1", argv1, argv2, argv3, argv4, NULL);
      delete[] argv1;
      delete[] argv2;
      delete[] argv3;
      delete[] argv4;
                                        } else {
                                                int id2 = fork();
```

```
                if (id2 == -1) {
                        std::cerr << "fork error" << std::endl;
                        return 1;
                } else  if(id2 == 0){
                        char *argv1 = new char[sizeof(int)];
                sprintf(argv1, "%d", pipe2fd[0]);
                char *argv2 = new char[sizeof(int)];
                sprintf(argv2, "%d", pipe2fd[1]);
                char *argv3 = new char[sizeof(int)];
                sprintf(argv3, "%d", pipe3fd[0]);
                char *argv4 = new char[sizeof(int)];
                sprintf(argv4, "%d", pipe3fd[1]);
                execlp("./child2", argv1, argv2, argv3, argv4, NULL);
                delete[] argv1;
                delete[] argv2;
                delete[] argv3;
                delete[] argv4;
                } else {
        std::string str;
        int sz;
        while (getline(std::cin, str)) {
           sz = str.size() + 1;
                        write(pipe1fd[1], &sz, sizeof(sz));
                        write(pipe1fd[1], str.c_str(), sz);
           read(pipe2fd[0], &sz, sizeof(sz));
           char *buf = new char[sz];
           read(pipe2fd[0], buf, sz);
           std::cout << buf << std::endl;
           delete[] buf;
                }
                }
        }
   close(pipe1fd[0]);
   close(pipe1fd[1]);
   close(pipe2fd[0]);
   close(pipe2fd[1]);
   close(pipe3fd[0]);
   close(pipe3fd[1]);
```

```
    return 0;
}
child1.cpp
#include <iostream>
#include <string>
#include <string.h>
#include <unistd.h>
#include <algorithm>

int main(int argc, char const *argv[])
{
    int pipe1fd[2];
                                            int pipe3fd[2];
                                            pipe1fd[0] = atoi(argv[0]);
                                            pipe1fd[1] = atoi(argv[1]);
                                            pipe3fd[0] = atoi(argv[2]);
                                            pipe3fd[1] = atoi(argv[3]);
                                            while (1) {
                                                    std::string str;
        int sz;
                                                    read(pipe1fd[0], &sz, sizeof(sz));
                                                    char *buf = new char[sz];
                                                    read(pipe1fd[0], buf, sz);
        str = buf;
        delete[] buf;
        std::transform(str.begin(),
                    str.end(),
                    str.begin(),
                    [](unsigned char c) { return std::toupper(c); });
                                                    write(pipe3fd[1], &sz, sizeof(sz));
                                                    write(pipe3fd[1], str.c_str(), sz);
                                            }
    close(pipe1fd[0]);
    close(pipe1fd[1]);
    close(pipe3fd[0]);
    close(pipe3fd[1]);
    return 0;
}
```

child2.cpp

```
#include <iostream>
#include <string>
#include <string.h>
#include <unistd.h>
#include <algorithm>

int main(int argc, char const *argv[])
{
                                        int pipe2fd[2];
                                        int pipe3fd[2];
                                        pipe2fd[0] = atoi(argv[0]);
                                        pipe2fd[1] = atoi(argv[1]);
                                        pipe3fd[0] = atoi(argv[2]);
                                        pipe3fd[1] = atoi(argv[3]);
                                        while (1) {
    std::string str;
                                                int sz = 0;
                                                read(pipe3fd[0], &sz, sizeof(sz));
                                                char *buf = new char[sz];
                                                read(pipe3fd[0], buf, sz);
    str = buf;
    delete[] buf;
    std::transform(str.begin(),
            str.end(),
            str.begin(),
            [](unsigned char c) { return (c == ' ') ? '_' : c; });
                                                write(pipe2fd[1], &sz, sizeof(sz));
                                                write(pipe2fd[1], str.c_str(), sz);
                                        }
  close(pipe2fd[0]);
  close(pipe2fd[1]);
  close(pipe3fd[0]);
  close(pipe3fd[1]);
  return 0;
}
```

Демонстрация работы программы

```
ggame@ggame:~/OS/ready/lab2$ g++ -o child child.cpp
ggame@ggame:~/OS/ready/lab2$ g++ -o main main.cpp
ggame@ggame:~/OS/ready/lab2$ ./main
aaaaaaa bbbbbb xxxxx qwertty
AAAAAAA_BBBBBB_XXXXX_QWERTTY
3523hj 3523     jihhi 999
3523HJ_3523_____JIHHI_999
        a
_________A
ggame@ggame:~/OS/ready/lab2$
```

Выводы

В ходе выполнения ЛР №2 я познакомился с системными вызовами, процессами.