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: Ворошилов Кирилл Сергеевич
Группа: М8О-201Б-21
Вариант:
Преподаватель Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/kiviyi/operation-system/tree/lab-2

Постановка задачи

Цель работы

Научиться создавать процессы и взаимодействовать с ними через pipe

Задание

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.
14 вариант) Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

Общие сведения о программе

Программа компилируется из файла 2lab.cpp. Также подключаются файлы child1.cpp, child2.cpp через execlp в качестве отдельной программы.

1. pipe() – создает связь между памятью процессов
2. fork() – создает второй процесс
3. dup2() – копирует old_file_descriptor в new_file_descriptor.

Общий метод и алгоритм решения

Сначала создаем два pipe.

Затем создаем два процесса: 1 занимается обработкой pipe1, второй – pipe2; Родительский процесс занимается заполнением pipe1 и pipe2 перед их обработкой дочерними процессами.

В файлы pipe1.txt и pipe2.txt заносятся выходные данные.

Исходный код

2lab.cpp:

```
#include "../include/parent.h"
#include <unistd.h>
#include "../include/utils.h"
#include <vector>
#include <stdio.h>
using namespace std;

string ParentRoutine(char const *pathToChild1, char const *pathToChild2,
                     const string &input){

   string output;

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

   int pid = fork();



   if (pid == 0) {

      close(firstPipe[WRITE_END]);
      close(pipeBetweenChildren[READ_END]);

      MakeDup2(firstPipe[READ_END], STDIN_FILENO);
      MakeDup2(pipeBetweenChildren[WRITE_END], STDOUT_FILENO);

      if (execl(pathToChild1, pathToChild1, NULL) == -1) {
         GetExecError(pathToChild1);
      }
      close(firstPipe[READ_END]);
      close(firstPipe[WRITE_END]);
```

```
    } else if (pid == -1) {
        GetForkError();
    } else {
        close(firstPipe[READ_END]);
        write(firstPipe[WRITE_END], input.c_str(), input.size());
        close(firstPipe[WRITE_END]);

        int secondPipe[2];
        CreatePipe(secondPipe);

        pid = fork();

        if (pid == 0) {
            close(secondPipe[READ_END]);
            close(pipeBetweenChildren[WRITE_END]);

            MakeDup2(pipeBetweenChildren[READ_END], STDIN_FILENO);
            MakeDup2(secondPipe[WRITE_END], STDOUT_FILENO);

            if (execl(pathToChild2, pathToChild2, NULL) == -1) {
                GetExecError(pathToChild2);
            }
            close(pipeBetweenChildren[READ_END]);
            close(secondPipe[WRITE_END]);
        } else if (pid == -1) {
            GetForkError();
        } else {
            close(secondPipe[WRITE_END]);
            close(pipeBetweenChildren[WRITE_END]);
            close(pipeBetweenChildren[READ_END]);

            wait(nullptr);
            char ch;

            while (read(secondPipe[READ_END], &ch, 1) && ch != '\n'){
                output += ch;
```

```
        }
        close(secondPipe[READ_END]);
      }
   }
   return output;
}
```

child1:

```
#include<unistd.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
#include <string>

using namespace std;

int main() {
   char c[100];
   int n;
   while(n=read(0, &c, 100)) {
      for (int i = 0; i < n; ++i) {
         if (c[i] != '\n' && c[i] != '\0') {
            c[i] = tolower(c[i]);
         } else {
            break;
         }
      }

      write(1, &c, n);
      exit(0);
   }
   return 0;
}
```

child2:

```
#include<unistd.h>
#include <string>
#include <valarray>
```

```
#include <iostream>
#include <cstring>

using namespace std;

int main() {
    char c[100];
    int n;
    while(n=read(0, &c, 100)){
        int j = 0;
        char lch = '\0';
        for (int i = 0; i < n; i++){
            if (lch != ' ' || c[i] != ' '){
                c[j] = c[i];
                j++;
            }
            lch = c[i];
        }
        for (int i = 0; i < j; i++) {
            cout << c[i];
        }
        cout << '\n';
    }
    return 0;
}
```

utils:

```
#include "../include/utils.h"
#include <unistd.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;
}
```

Демонстрация работы программы

INPUT:

```
QRWERWE        WERWQERWE
```

OUTPUT:

```
kivi@LAPTOP-DGOM1IRE:~/os/lab_2/lab2/l2$ ./lab2

QRWERWE        WERWQERWE

qrwerwe werwqerwe
```

Выводы

С помощью с и с++ можно создавать процессы, которые значительно ускоряют работу программы. Связь между ними можно осуществить с помощью pipe(так называемой трубки), что очень круто!