

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: Тулин Иван Денисович
Группа: М8О-201Б-21
Вариант:
Преподаватель Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

Репозиторий

<https://github.com/YusayuSharingan/Opsys-labs/tree/main/lab2>

Постановка задачи

Цель работы

Научиться создавать процессы и взаимодействовать с ними через pipe

Задание

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами

10 вариант) В файле записаны команды вида: «число<newline>». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются. Количество чисел может быть произвольным.

Общие сведения о программе

Программа компилируется из файла parent.cpp. Также подключаются файлы child.cpp через exel 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

Получаем в родительском процессе имя входного файла и создаем pipe.

Создаем дочерний процесс и заменяем его стандартный поток ввода дескриптором входного файла, заменяем образ дочернего процесса через `execl`, обрабатываем данные и через pipe возвращаем ответ в родительский процесс.

Исходный код

parent.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <vector>
#include "checkers.h"
#include "parent.h"

std::vector <int> ParentRoutine(char child[], char filename[]){
    std::vector <int> output = {};

    int fd[2];
    CreatePipe(fd);

    int pid = fork();
    if (pid == ERROR){           // fork error check
        ForkError();
    } else if (pid == CHILD_ID) { // for child process
        close(fd[0]);
        FILE *fi;
        if (!(fi = freopen(filename, "r", stdin))){
            FileError();
        }
        MakeDup2(fd[1], STDOUT_FILENO);
        if (execl(child, child, NULL) == ERROR){
            ExecError();
        }
    } else {                    // for parent process
```

```

        close(fd[1]);
        int result;
        while(read(fd[0], &result, sizeof(int)) != STOP_SIG){
            if (result == BREAKER){
                return output;
            }
            output.push_back(result);
        }
    }
    return output;
}

```

checkers.cpp

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "checkers.h"

```

```

void CreatePipe(int fd[]) {
    if (pipe(fd) != 0) {
        printf("Error: couldn't create pipe\n");
        exit(EXIT_FAILURE);
    }
}

```

```

void ForkError() {
    printf("Error: couldn't create new process\n");
    exit(EXIT_FAILURE);
}

```

```

void FileError(){
    printf("Error: couldn't open file\n");
    exit(EXIT_FAILURE);
}

```

```

void MakeDup2(int oldFd, int newFd) {
    if (dup2(oldFd, newFd) == ERROR) {
        printf("Error: couldn't change child stdin\n");
    }
}

```

```

        exit(EXIT_FAILURE);
    }
}

void ExecError(){
    printf("Error: couldn't execute child program");
    exit(EXIT_FAILURE);

}

```

child.cpp

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "functions.h"

int main(){
    char number[32];
    while (scanf("%s", number) != STOP_SIG){
        int n = GetNumber(number);
        if (n < 0){
            write(STDOUT_FILENO, &NEGATIVE, sizeof(int));
            exit(EXIT_FAILURE);
        } else if (IsPrime(n)){
            write(STDOUT_FILENO, &NEGATIVE, sizeof(int));
            exit(EXIT_FAILURE);
        } else {
            write(STDOUT_FILENO, &n, sizeof(int));
        }
    }
    return 0;
}

```

functions.cpp

```

#include "functions.h"

```

```

int GetNumber(char* s){

```

```

int code = ZERO;
int cntr = 0;
if (s[cntr] - MINUS == 0){
    return -1;
}
int sum = 0;
while (s[cntr] != EOS){
    code = s[cntr] - ZERO;
    if(code < 0 || code > 9){
        break;
    }
    sum = sum*10 + code;
    cntr++;
}
return sum;
}

```

```

int IsPrime(int num){
    if (num % 2 == 0){
        return num == 2;
    }
    int div = 3;
    while (div * div <= num && num % div != 0){
        div += 2;
    }
    return div * div > num;
}

```

Демонстрация работы программы

```
yorokobeshounen@YS:~/Рабочий стол/OpSys/lab2/build$ cat input1.txt
```

4

-3

22

12

46

89

91

42

12

68

55

-22

1

```
yorokobeshounen@YS:~/Рабочий стол/OpSys/lab2/build$ ./lab2
```

Input name of file

input1.txt

4

Выводы

С помощью `c` и `c++` можно создавать процессы, которые значительно ускоряют работу программы.

Связь между ними можно осуществить с помощью `pipe` (так называемой трубы).