Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работ №2 по курсу

«Операционные системы»

Группа: М80 – 207Б-18

Студент: Цапков Александр Максимович

Преподаватель: Миронов Евгений Сергеевич

Оценка: _____

Дата: _____

Москва, 2019.

Содержание

Постановка задачи.

Дочерний процесс представляет собой сервер по работе со стеками и принимает команды со стороны родительского процесса.

Общие сведения о программе

Программа компилируется в 2 исполняемых файла: главный (управляющий) и сервер. Сервер также использует заголовочные и .c файлы по работе со стеком. В лабораторной работе были использованы следующие системные вызовы.

1. **read** – для чтения данных из файла
2. **write** – для записи данных в файл
3. **pipe** – для создания однонаправленного канала, через который могут общаться два процесса.
4. **dup2** – создает копию файлдискриптора. В моей программе я использую этот системный вызов для перенаправления пайпа в стандартный ввод дочернего процесса.
5. **fork** – для создания дочернего процесса.
6. **close** – для закрытия файла.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Используя системный вызов pipe создать 2 каналов, по которым будут обмениваться данными процессы.
2. Используя системный вызов fork создать дочерний процесс.
3. В дочернем процессе перенаправить поток на сервер с помощью dup2 и запустить server.out с помощью execl и передаем в нее в качестве параметров.
4. В родительском процессе считывать данные со стандартного потока в цикле.
5. Как в родительском процессе данные считались, необходимо записать их в канал с помощью системного вызова write. Затем родительский процесс считывает результат из второго канала. Но пока дочерний процесс не запишет данные во второй канал, родительский процесс будет заблокирован.
6. Пока родительский процесс не записал данные в канал. Дочерний процесс блокируется. И как только родительский процесс записал данные в первый канал дочерний процесс считывает их с его стандартного ввода, производит вычисления и записывает результат во второй канал.
7. Как только дочерний процесс запишет результат вычислений во второй канал родительский процесс получит результат из второго канала и выведет их в стандартный поток. И затем снова будет ждать ввода со стандартного потока.

Основные файлы программы.

Файл main.cpp

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

int main() {
    int pipeFdToServ[2];
    int pipeFdFromServ[2];
    std::string outStr;
    if (pipe(pipeFdToServ) == -1) {
        std::cout << "Cannot create pipe\n";
        exit(-1);
    }
    if (pipe(pipeFdFromServ) == -1) {
        std::cout << "Cannot create pipe\n";
        exit(-1);
    }
    int serverPid = fork();
    if (serverPid < 0) {
        std::cout << "Cannot create server procces\n";
        exit(-1);
    }else if (serverPid == 0) {
        std::cout << "Create server procces\n";
        if (dup2(pipeFdToServ[0], STDIN_FILENO) == -1) {
            std::cout << "Cannot dep2\n";
            exit(-1);
        }
        close(pipeFdToServ[1]);
        execl("./server.out", std::to_string(pipeFdFromServ[0]).c_str(),
                        std::to_string(pipeFdFromServ[1]).c_str(), (char*)NULL);
        std::cout << "Cannot execute program\n" << errno << "\n";
    }
    close(pipeFdToServ[0]);
    close(pipeFdFromServ[1]);
    int msgSize, inMsg;
    while (true) {
        std::getline(std::cin, outStr);
        int a = outStr.length();
        outStr[a] = '\n';
        outStr[a + 1] = '\0';
        if (write(pipeFdToServ[1], outStr.c_str(), a + 1) == -1) {
            std::cout << "Cannot write to pipe\n";
```

```
            exit(-1);
        }
        read(pipeFdFromServ[0], &msgSize, sizeof(int));
        std::cout << "Mesage no: " << msgSize << '\n';
        if (msgSize == -2) {
            break;
        }
        for (int i = 0; i < msgSize; i++) {
            read(pipeFdFromServ[0], &inMsg, sizeof(int));
            std::cout << inMsg << '\n';
        }
    }
    return 0;
}
```

server.cpp

```
#include <iostream>
#include "stack.h"
#include "sort.h"
#include <string>
#include <unistd.h>

int main(int argc, char* argv[]) {;
    int pipeFd[2];
    int buf;
    //char strIn[100];
    pipeFd[0] = std::stoi(argv[0]);
    pipeFd[1] = std::stoi(argv[1]);
    close(pipeFd[0]);
    char c;
    int st;
    int val;
    Stack *A[10];
    for (int i = 0; i < 10; i++) {
        A[i] = NULL;
    }
    while (1){
        scanf("%c", &c);
        switch (c) {
            case 'c':
                scanf("%d", &st);
                if (st > 9 || st < 0) {
                    buf = -1;
                    write(pipeFd[1], &buf, sizeof(int));
```

```
            break;
        }
        A[st] = stack_create();
        buf = 0;
        write(pipeFd[1], &buf, sizeof(int));
        break;
    case 'd':
        scanf("%d", &st);
        if (st > 9 || st < 0) {
            buf = -1;
            write(pipeFd[1], &buf, sizeof(int));
            break;
        }
        if (A[st] != NULL) {
            stack_delete(&A[st]);
            buf = 0;
            write(pipeFd[1], &buf, sizeof(int));
        } else {
            buf = -1;
            write(pipeFd[1], &buf, sizeof(int));
        }
        break;
    case 'i':
        scanf("%d", &st);
        if (st > 9 || st < 0) {
            buf = -1;
            write(pipeFd[1], &buf, sizeof(int));
            break;
        }
        if (A[st] == NULL) {
            buf = -1;
            write(pipeFd[1], &buf, sizeof(int));
            break;
        }
        while (scanf("%d", &val)) {
            stack_push(A[st], val);
            c = getchar();
            if (c == '\n') {
                break;
            }
        }
        buf = 0;
        write(pipeFd[1], &buf, sizeof(int));
        break;
    case 'o':
```

```
            scanf("%d", &st);
            if (st > 9 || st < 0) {
                buf = -1;
                write(pipeFd[1], &buf, sizeof(int));
                break;
            }
            if (A[st] == NULL) {
                buf = -1;
                write(pipeFd[1], &buf, sizeof(int));
                break;
            }
            if (!stack_is_empty(A[st])) {
                buf = 1;
                write(pipeFd[1], &buf, sizeof(int));
                buf = stack_pop(A[st]);
                write(pipeFd[1], &buf, sizeof(int));
            } else {
                buf = -1;
                write(pipeFd[1], &buf, sizeof(int));
            }
            break;
        case 's':
            scanf("%d", &st);
            if (st > 9 || st < 0) {
                buf = -1;
                write(pipeFd[1], &buf, sizeof(int));
                break;
            }
            if (A[st] == NULL) {
                buf = -1;
                write(pipeFd[1], &buf, sizeof(int));
                break;
            }
            if (stack_size(A[st]) < 0) {
                buf = -1;
                write(pipeFd[1], &buf, sizeof(int));
                break;
            }
            sort(A[st]);
            buf = 0;
            write(pipeFd[1], &buf, sizeof(int));
            break;
        case 'p':
            scanf("%d", &st);
            if (st > 9 || st < 0) {
```

```
                buf = -1;
                write(pipeFd[1], &buf, sizeof(int));
                break;
            }
            if (A[st] == NULL) {
                buf = -1;
                write(pipeFd[1], &buf, sizeof(int));
                break;
            }
            buf = stack_size(A[st]);
            write(pipeFd[1], &buf, sizeof(int));
            stack_print(A[st], pipeFd[1]);
            break;
        case 'q':
            for (int i = 0; i < 10; i++) {
                if (A[i] != NULL) {
                    stack_delete(&A[i]);
                }
            }
            return 0;
            break;
        default:
            buf = -1;
            write(pipeFd[1], &buf, sizeof(int));
            break;
    }
    while(c != '\n') {
        c = getchar();
    }
  }
}
```

Демонстрация работы программы.

```
iPad-Alex:~/Documents/Study/Labs/OS/sem3os/2st# make
g++ -Wall -pedantic -c server.cpp
g++  -o server.out server.o stack.o sort.o
g++ -Wall -pedantic -o main.out main.cpp
iPad-Alex:~/Documents/Study/Labs/OS/sem3os/2st# ./main.out
Create server procces
c 1
Mesage no: 0
c 3
Mesage no: 0
i 1 0 1 2 3 4 5 322
Mesage no: 0
i 3 8 5 4 0 -3 545
Mesage no: 0
i 2 3 4 33
Mesage no: -1
i 1 6 7 8
Mesage no: 0
p 1
Mesage no: 10
0
1
2
3
4
5
322
6
7
8
p 3
Mesage no: 6
8
5
4
```

0
-3
545
s 3
Mesage no: 0
p 3
Mesage no: 6
-3
0
4
5
8
545
o 3
Mesage no: 1
545
o 3
Mesage no: 1
8
p 3
Mesage no: 4
-3
0
4
5
q

Вывод

Для начала из данной лабораторной работы я вынес то, что нужно лучше вычислять свой вариант и быть внимательнее. Процессы — это основа почти любой операционной системы и освоение работы с ними предоставляет огромную гибкость в написании программы, особенно из за распараллеливания задач и разделения ее как логически, так разделение ее вычислений.