Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу

«Операционные системы»

ПРОЦЕССЫ В ОС

Студент: [Лошманов Юрий Андреевич]
Группа: М8О–206Б–20
Вариант: 7
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2020.

Постановка задачи

Цель работы

Целью является 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.

Общие сведения о программе

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.

В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pipe, fork, exec, wait.
2. Подключить библиотеки unistd.h и sys/wait.h, необходимые для работы с процессами.
3. Написать файл потомка child.c.
4. Написать основной файл main.c, который создаёт процесс, запускающий файл потомка.
5. Создать текстовый файл, который потомок будет принимать на чтение.

Основные файлы программы

main.c:

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

int main(int argc, const char *argv[]) {
    int fd[2];
    if (pipe(fd) == -1) {
        fprintf(stderr, "pipe error\n");
        exit(1);
    }

    pid_t pid = fork();
    if (pid < 0) {
        fprintf(stderr, "fork error\n");
        exit(2);
    } else if (pid == 0) {
        FILE *f = NULL;
        if (argc > 1) {
            f = fopen(argv[1], "r");
        } else {
            char file_name[50];
            printf("Enter file name: ");
            scanf("%[^\n]%*c", file_name);
            f = fopen(file_name, "r");
        }
        if (!f) {
            fprintf(stderr, "can't open file\n");
            exit(3);
        }

        dup2(fileno(f), STDIN_FILENO);
        dup2(fd[1], STDOUT_FILENO);
```

```
            dup2(fd[1], STDERR_FILENO);

            close(fd[0]);
            close(fd[1]);
            fclose(f);

            if (execl("child", "child", (char *) NULL) == -1) {
                fprintf(stderr, "exec error\n");
                exit(4);
            }
        } else {
            close(fd[1]);
            waitpid(pid, (int *)NULL, 0);

            float result;
            while (read(fd[0], &result, sizeof(result))) {
                printf("%f\n", result);
            }
            close(fd[0]);
        }
        return 0;
    }
```

child.c:

```
    #include <stdio.h>
    #include <unistd.h>

    int main() {
        float a, b, c;
        while (scanf("%f %f %f", &a, &b, &c) != -1) {
            float result = a + b + c;
            write(STDOUT_FILENO, &result, sizeof(result));
        }
        return 0;
    }
```

Пример работы

```
yuryloshmanov@air-urij Desktop % git clone https://github.com/yuryloshmanov/
os_lab_2
Клонирование в «os_lab_2»…
remote: Enumerating objects: 107, done.
remote: Counting objects: 100% (107/107), done.
remote: Compressing objects: 100% (66/66), done.
remote: Total 107 (delta 29), reused 94 (delta 22), pack-reused 0
Получение объектов: 100% (107/107), 124.64 KiB | 397.00 KiB/s, готово.
Определение изменений: 100% (29/29), готово.
yuryloshmanov@air-urij Desktop % cd os_lab_2
yuryloshmanov@air-urij os_lab_2 % cd test
yuryloshmanov@air-urij test % ./run.sh
-- The C compiler identification is AppleClang 12.0.0.12000032
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc
- skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
```

```
-- Build files have been written to: /Users/yuryloshmanov/Desktop/os_lab_2/src
Scanning dependencies of target child
[ 25%] Building C object CMakeFiles/child.dir/child.c.o
[ 50%] Linking C executable child
[ 50%] Built target child
Scanning dependencies of target main
[ 75%] Building C object CMakeFiles/main.dir/main.c.o
[100%] Linking C executable main
[100%] Built target main

test1.txt
1.3 1.2 33.5
1.3 1.2 33.5
1.3 1.00 33.5
1.3 1.2 33.5
1.3 1.2 4

Answer:
36.000000
36.000000
35.799999
36.000000
6.500000


test2.txt
0.112 44 22
-1 -2 -3

Answer:
66.112000
-6.000000


test3.txt

Answer:


test4.txt
55.212 344.1234 0
-2 -1 3

Answer:
399.335419
0.000000

yuryloshmanov@air-urij test %
```

Вывод

Это была моя первая лабораторная работа в курсе «операционные системы». В ней я столкнулся с такими системными вызовами, как fork, pipe и exec. Для меня было совсем не тривиальной задачей разобраться в принципе создания процесса. Раньше я понятия не имел, что один и тот же код может раздвоиться на определённой строчке и начать выполняться по-разному.

Данная лабораторная работа помогла мне узнать как создавать процессы и поддерживать между ними связь в операционных системах типа *nix на языке Си.