Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу
«Операционные системы»

МЕЖПРОЦЕССНОЕ ВЗАИМОДЕЙСТВИЕ

Студент: Лукманова Аэлита
Группа: М8О–201Б–19
Вариант: 21
Преподаватель: Миронов Е. С.
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021.

Постановка задачи

Цель работы

Целью работы 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.

Вариант 21:

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами.
Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от фильтрации. Дочерние процессы производят работу над строками и пишут результат в стандартный вывод.
Правило фильтрации: нечетные строки в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, stdbool.h, sys/types.h, sys/stat.h, fcntl.h. В программе используются следующие системные вызовы:

1. **pipe** — принимает массив из двух целых чисел, в случае успеха массив будет содержать два файловых дескриптора, которые будут использоваться для конвейера, первое число в массиве предназначено для чтения, второе для записи, а так же вернется 0. В случае неуспеха вернется -1.
2. **fork** — создает новый процесс, который является копией родительского процесса, за исключением разных process ID и parent process ID. В

случае успеха fork() возвращает 0 для ребенка, число больше 0 для родителя – child ID, в случае ошибки возвращает -1.

3. **close —** принимает файловый дескриптор в качестве аргумента, удаляет файловый дескриптор из таблицы дескрипторов, в случае успеха вернет 0, в случае неуспеха вернет -1.
4. **open —** создает или открывает файл, если он был создан. В качестве аргументов принимает путь до файла, режим доступа (запись, чтение и т.п.), модификатор доступа (при создании можно указать права для файла). Возвращает в случае успеха файловый дескриптор – положительное число, иначе возвращает -1.
5. **read —** предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернет число прочитанных байт, иначе -1.
6. **write —** предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернет число записанных байт, иначе -1.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pipe, fork.
2. Организовать считывание названия файла и строки произвольной длины.
3. Реализовать функции для процесса-родителя и процесса-ребенка.
4. Реализовать сообщение между процессами при помощи каналов.
5. Реализовать обработку системных ошибок согласно заданию.

Основные файлы программы

main.c:

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

#define NUMBER_OF_STRING 25
#define MAX_STRING_SIZE 200


//Вариант 21

int main(int argc, char* argv[]) {

    //MARK: -названия файлов на запись
    char firstChildFile[200];
    char secChildFile[200];
    printf("Введите имя файла на запись для первого дочернего процесса: ");
    fgets(firstChildFile, 200, stdin);
    firstChildFile[strlen(firstChildFile) - 1] = '\0';

    printf("Спасибо, введите имя файла на запись для второго дочернего процесса: ");
    fgets(secChildFile, 200, stdin);
    secChildFile[strlen(secChildFile) - 1] = '\0';


    //MARK: -инициализация
    int fd[2];
    int fd2[2];
    if (pipe(fd) == -1) {
        return 1;
    }
    if (pipe(fd2) == -1) {
        return 1;
    }

    pid_t pid = fork();
    if (pid == -1) {
        return 2;
    }


    if (pid == 0) { //child1
        printf("child1");
```

```
        close(fd[1]);
        char arrayOfString1[NUMBER_OF_STRING][MAX_STRING_SIZE];
        int arrayCount1;


        if (read(fd[0], &arrayCount1, sizeof(int)) < 0) { return 6; }
        if (read(fd[0], arrayOfString1, sizeof(char) * NUMBER_OF_STRING *
MAX_STRING_SIZE) < 0) { return 6; }

        char *argv[200];

        argv[0] = "./child1";
        argv[1] = firstChildFile;
        for (int i = 2; i < arrayCount1+1; i++) {
            argv[i] = &arrayOfString1[i-2][0];
        }

        argv[arrayCount1+1] = NULL;
        wait(NULL);
        execv("./child", argv);
    }

    if (pid > 0) { //parent

        pid_t ch2 = fork();

        if (ch2 == 0) { //child2

            printf("child2");
            close(fd2[1]);
           int arrayCount2;
            char arrayOfString2[NUMBER_OF_STRING][MAX_STRING_SIZE];

            if (read(fd2[0], &arrayCount2, sizeof(int)) < 0) { return 6; }
            if (read(fd2[0], arrayOfString2, sizeof(char) * NUMBER_OF_STRING *
MAX_STRING_SIZE) < 0) { return 6; }

            char *argv[200];

            argv[0] = "./child2";
            argv[1] = secChildFile;
            for (int i = 2; i < arrayCount2+1; i++) {
                argv[i] = &arrayOfString2[i-2][0];
            }

            argv[arrayCount2+1] = NULL;
            //sleep(3);
            execv("./child", argv);

        } else {

            //MARK: -работа с pid
            printf("Hello, I'm a parent\n");

            close(fd[0]);
            close(fd2[0]);


            char arrayOfString[50][200];
            printf("Input strings: \n");
            int arrayCount = 0;
            char line[200];
            while(fgets(line, 200, stdin)){
                //чтобы не было в конце перевода строки
                line[strlen(line) - 1] = '\0';
                strcpy(arrayOfString[arrayCount], line);
                arrayCount++;
            }

            char arrayOfString1[NUMBER_OF_STRING][MAX_STRING_SIZE];
            char arrayOfString2[NUMBER_OF_STRING][MAX_STRING_SIZE];
```

```
            char currentStr[200];

            int arrayCount1 = 1;
            int arrayCount2 = 1;
            for (int i = 0; i < arrayCount; i++) {
                strcpy(currentStr, arrayOfString[i]);
                if (i%2 == 0) {
                    strcpy(arrayOfString1[arrayCount1-1], currentStr);
                    arrayCount1++;
                } else {
                    strcpy(arrayOfString2[arrayCount2-1], currentStr);
                    arrayCount2++;
                }
            }

            write(fd[1], &arrayCount1, sizeof(int));
            write(fd2[1], &arrayCount2, sizeof(int));
            write(fd[1], &arrayOfString1, sizeof(arrayOfString1));
            write(fd2[1],&arrayOfString2, sizeof(arrayOfString2));

            close(fd[1]);
            close(fd2[1]);
            printf("\n parent say goodbye\n");
            waitpid(ch2, NULL, 0);
            waitpid(pid, NULL, 0);
        }
    }
}
```

child.c:

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

int main(int argc, char* argv[]) {
    //MARK: -открытие файлов на запись
    FILE *fp;
    if ((fp = fopen(argv[1], "w")) == NULL)
    {
        printf("Не удалось открыть файл на запись");
        getchar();
        return 0;
    }


    printf("\nHello, I'm %s\nThis is result of my work:\n", argv[0]);
    for (int i = 2; i < argc; i++) {
        for (int j = 0, k = strlen(argv[i]) - 1; j < k; j++, k--)
                {
                    char temp = argv[i][j];
                    argv[i][j] = argv[i][k];
                    argv[i][k] = temp;
                }
        printf("%s\n", argv[i]);

        fprintf(fp, "%s\n", argv[i]);
    }

    fclose(fp);
    printf("%s say goodbye\n", argv[0]);
    return 0;
}
```

Пример работы

```
aelitalukmanova@MacBook-Pro-Aelita os-2 % ./main
Введите имя файла на запись для первого дочернего процесса: first.txt
Спасибо, введите имя файла на запись для второго дочернего процесса: second.txt
Hello, I'm a parent
Input strings:
This is 1st string
And this is 2nd one
3d
4th
fifth
This string will be 6th
^D
 parent say goodbye

Hello, I'm ./child2
This is result of my work:
eno dn2 si siht dnA
ht4
ht6 eb lliw gnirts sihT
./child2 say goodbye

Hello, I'm ./child1
This is result of my work:
gnirts ts1 si sihT
d3
htfif
./child1 say goodbye
aelitalukmanova@MacBook-Pro-Aelita os-2 % cat first.txt
gnirts ts1 si sihT
d3
htfif
aelitalukmanova@MacBook-Pro-Aelita os-2 % cat second.txt
eno dn2 si siht dnA
ht4
ht6 eb lliw gnirts sihT
aelitalukmanova@MacBook-Pro-Aelita os-2 %
```

Вывод

Здесь я узнала про процессы, о том, что можно один процесс поделить на несколько, при этом область памяти у них будет у каждого своя, то есть никакие глобальные переменные не получится поменять в одном процессе и изменить тем самым их в другом. Поэтому необходимы трубы, которые связывают процессы между собой, по ним передача становится возможной. Довольно интересно.