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: Чекменев В.А.
Группа: М8О–207Б–20
Вариант: 2
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Постановка задачи

Цель работы
Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

Задание
Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль
родительского процесса пишет имя файла, которое будет передано при создании дочернего
процесса. Родительский процесс передает команды пользователя через pipe1, который связан с
стандартным входным потоком дочернего процесса. Дочерний процесс принеобходимости
передает данные в родительский процесс через pipe2. Результаты своей работы дочерний
процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не
перенаправляя стандартный поток вывода.
Вариант задания
Пользователь вводит команды вида: «число число число<endline>». Далее эти числа
передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и
выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

Общие сведения о программе

Программа компилируется из файла lab2.c. . В программе используются следующие системные вызовы:

fork - системный вызов в Unix-подобных операционных системах, создающий новый процесс (потомок), который является практически полной копией процесса-родителя, выполняющего этот вызов.Между процессом-потомком, порождаемым вызовом fork(), и процессом-родителем существуют различия:

1) PID процесса-потомка отличен от PID процесса-родителя;

2) значению PPID процесса-потомка присваивается значение PID процесса-родителя;

3) процесс-потомок получает собственную таблицу файловых дескрипторов, являющуюся копией таблицы процесса-родителя на момент вызова fork(); это означает, что открытые файлы наследуются, но если процесс-потомок, например, закроет какой-либо файл, то это не повлияет на таблицу дескрипторов процесса-родителя;

3) для процесса-потомка очищаются все ожидающие доставки сигналы;

4) временная статистика выполнения процесса-потомка в таблицах ОС обнуляется;

5) блокировки памяти и записи, установленные в процессе-родителе, не наследуются.

pipe - создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.

read - Считывает данные из файла

write - записывает в файл

Код программы lab2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include "functions.h"

int main()
{
    pid_t child_pid;
    int pipe1[2];
    int returnstatus_pipe;
    char file_name[20];
    char read_file_name[20];
    char sequence_of_numbers[100];
    char read_sequence_of_numbers[100];

    returnstatus_pipe = pipe(pipe1);

    // Error: can not create pipe
    if (returnstatus_pipe == -1) {
        printf("Unable to create pipe\n");
        return 1;
    }

    child_pid = fork();

    // Error: can not create child process
    if (child_pid < 0) {
        perror("fork error");
```

```
        return -1;
    }
    // Child Process
    else if (child_pid == 0) {
        int read_cnt;
        read(pipe1[0], read_file_name, sizeof(read_file_name)); // read 1
                                                read(pipe1[0],
read_sequence_of_numbers,    sizeof(read_sequence_of_numbers));    //
read 2
        read(pipe1[0], &read_cnt, sizeof(int)); // read 3

        printf("[Child Process,  id=%d]: Reading from the pipe. Message is
file name: %s\n", getpid(), read_file_name);
        fflush(stdout);

        printf("[Child Process,  id=%d]: Reading from the pipe. Message is
sequence of numbers: %s\n", getpid(), read_sequence_of_numbers);
        fflush(stdout);

        remove(read_file_name);
        FILE *write_seq = fopen(read_file_name, "w");
        fputs(read_sequence_of_numbers, write_seq);
        fclose(write_seq);

        FILE *read_seq = fopen(read_file_name, "r");
        float res = sum(read_seq, read_cnt);
        fclose(read_seq);

        FILE *write_res = fopen(read_file_name, "w");
        write_res = fopen(read_file_name, "w");
        fprintf(write_res, "%f", res);
        fclose(write_res);
```

```
        printf("[Child Process,  id=%d]: result of sum is %f\n", getpid(), res);
        fflush(stdout);

            printf("[Child Process,  id=%d]: writing result to file %s\n", getpid(), read_file_name);
        fflush(stdout);

        close(pipe1[0]);
        close(pipe1[1]);
    }
    // Parent Process
    else {
            printf("[Parent Process, id=%d]: Enter name of the file where result will store: ", getpid());
        fflush(stdout);
        fgets(file_name, 20, stdin);
        if (file_name[strlen(file_name) - 1] == '\n')
            file_name[strlen(file_name) - 1] = '\0';
        fflush(stdin);

         printf("[Parent Process, id=%d]: Enter the sequence of real type numbers: ", getpid());
        fflush(stdout);
        fgets(sequence_of_numbers, 100, stdin);
         if (sequence_of_numbers[strlen(sequence_of_numbers) - 1] == '\n')
            sequence_of_numbers[strlen(sequence_of_numbers) - 1] = '\0';
        fflush(stdin);

         printf("[Parent Process, id=%d]: Writing to the pipe. Message is file name: %s\n\n", getpid(), file_name);
```

```
        fflush(stdout);

        int cnt = count(sequence_of_numbers);

        write(pipe1[1], file_name, sizeof(file_name)); // write 1
                                        write(pipe1[1],      sequence_of_numbers,
sizeof(sequence_of_numbers)); // write 2
        write(pipe1[1], &cnt, sizeof(int)); // write 3

        /* printf("[Parent Process, id=%d]: Writing to the pipe - Message is
sequence of numbers: \"%f, %f, %f\"\n", getpid(), x, y, z);
        fflush(stdout); */

        close(pipe1[0]);
        close(pipe1[1]);
    }
    return 0;
}
```

Код программы functions.h

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

int count(char *str)
{
   int counter = 1;
   for (int i = 0; i < strlen(str); ++i)
      if (str[i] == ' ')
         counter++;
   return counter;
}

float sum(FILE *f, int count)
{
   float res = 0.0, a;
   rewind(f);
   for (int i = 0; i < count; ++i) {
      fscanf(f, "%f", &a);
      res += a;
   }
   return res;
}
```

Листинг терминала

[suraba04@asusx512fl code_data]$ pwd

/home/suraba04/labs/sem3_OS/lab2/code_data

[suraba04@asusx512fl code_data]$ ll

total 44K

-rw-r--r-- 1 suraba04 suraba04 378 ноя 8 20:10 functions.h

-rw-r--r-- 1 suraba04 suraba04 3,3K ноя 8 20:10 lab2.c

-rw-r--r-- 1 suraba04 suraba04   82 ноя 8 20:14 .~lock.report.docx#

-rw-r--r-- 1 suraba04 suraba04 32K ноя 8 19:36 report.docx

[suraba04@asusx512fl code_data]$ wc functions.h lab2.c

 21 62 378 functions.h

 102 319 3342 lab2.c

 123 381 3720 total

[suraba04@asusx512fl code_data]$ du -h functions.h lab2.c

4,0K functions.h

4,0K lab2.c

[suraba04@asusx512fl code_data]$ gcc lab2.c -o lab2.o

[suraba04@asusx512fl code_data]$ ll

total 64K

-rw-r--r-- 1 suraba04 suraba04 378 ноя 8 20:10 functions.h

-rw-r--r-- 1 suraba04 suraba04 3,3K ноя 8 20:10 lab2.c

-rwxr-xr-x 1 suraba04 suraba04 17K ноя 8 20:28 lab2.o*

-rw-r--r-- 1 suraba04 suraba04   82 ноя 8 20:14 .~lock.report.docx#

-rw-r--r-- 1 suraba04 suraba04 32K ноя 8 19:36 report.docx

[suraba04@asusx512fl code_data]$ ./lab2.o

[Parent Process, id=32477]: Enter name of the file where result will store: output_file.txt

[Parent Process, id=32477]: Enter the sequence of real type numbers: 1.2 2 3 4

[Parent Process, id=32477]: Writing to the pipe. Message is file name: output_file.txt

[Child Process, id=32478]: Reading from the pipe. Message is file name: output_file.txt

[Child Process, id=32478]: Reading from the pipe. Message is sequence of numbers: 1.2 2 3 4

[Child Process, id=32478]: result of sum is 10.200000

[Child Process, id=32478]: writing result to file output_file.txt

[suraba04@asusx512fl code_data]$ ll

total 68K

-rw-r--r-- 1 suraba04 suraba04 378 ноя 8 20:10 functions.h

-rw-r--r-- 1 suraba04 suraba04 3,3K ноя 8 20:10 lab2.c

-rwxr-xr-x 1 suraba04 suraba04 17K ноя 8 20:28 lab2.o*

-rw-r--r-- 1 suraba04 suraba04   82 ноя 8 20:14 .~lock.report.docx#

-rw-r--r-- 1 suraba04 suraba04   9 ноя 8 20:29 output_file.txt

-rw-r--r-- 1 suraba04 suraba04 32K ноя 8 19:36 report.docx

[suraba04@asusx512fl code_data]$ cat output_file.txt

10.200000[suraba04@asusx512fl code_data]$

Вывод

Управление процессами – одна из ключевых задач операционной системы. Обычно ОС сама создаёт необходимые для себя и для других программ процессы, но возникают ситуации, когда пользователю требуется вмешаться в работу системы.

Язык Си при подключении библиотеки unistd.h (для Unix-подобных ОС) обладает возможностью совершать системные вызовы, связанные с вводом/выводом данных, управлением файлами и каталогами и, что самое важное, управлением процессами.

В данной лабораторной работе я познакомился с такими понятиями как процесс, дочерний процесс, родительский процесс, пайп, потоки.

Также, я пользовался некторыми функциями языка Си, такими как: pipe, fork, open, close, read, write etc.

После выполнения данной работы я приобрел новые знания, которые позволят по-другому взглянуть на обычные программы.

Однако не только язык Си способен совершать системные вызовы, связанные с управлением процессами. Похожие библиотеки есть на многих других языках программирования, ведь современное программное обеспечение крайне редко состоит из одного процесса.