Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Операционные системы»

Лабораторная работа № 2

Студент: Марочкин И.А.
Группа: М8О-206Б-19
Преподаватель: Соколов А.А.
Дата: 23.04.2021
Оценка:

Москва, 2020

Цель работы

Приобретение практических навыков в:

- Управлении 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:

В файле записаны команды вида: “число число число<endline>”. Дочерний процесс производит деление первого числа в команде на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. числа имеют тип float.

Общие сведения о программе

Программа компилируется из файла laba.c. Для реализации поставленной задачи в программе используются следующие системные вызовы:

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

fork - создает копию текущего процесса, который является дочерним процессом для текущего процесса.

open - открывает файл. Аргументом можно задать, открыть файл на запись или чтение.

close - закрывает файл.

write - записывает в файл из области памяти указанное количество байт.

read - читает из файла указанное количество байт в области памяти.

dup2 - создает дубликат файлового дескриптора.

perror - вывод сообщения об ошибке.

Листинг программы

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <math.h>
#include <stdint.h>

void Swap(char* a, char* b){
   char tmp = *a;
   *a = *b;
   *b = tmp;
}

char* reverse(char* str){
   char* begin = str;
   char* end = str;

   for (; *end; ++end) { ; }

   for (--end; begin < end; ++begin, --end)
   {
      Swap(begin, end);
   }

   return str;
}

char* IntToAnsi(int dig, char str[], size_t len){
   int i = 0;

   while (dig && (i < len - 1))
   {
      str[i++] = (dig % 10) + '0';
      dig /= 10;
   }

   if ((i == 0) && (i < len))
   {
      str[i++] = '0';
   }

   str[i] = '\0';

   reverse(str);

   return str;
}

char* DoubleToAnsi(double dig, size_t acc, char str[], size_t len){
   int hi = (int) dig;
   int lo = (dig - hi) * pow(10, acc);
   char* pstr = str;

   IntToAnsi(hi, pstr, len);

   if (lo)
   {
      for (; *pstr; ++pstr) { ; }

      *pstr++ = '.';
```

```
      IntToAnsi(lo, pstr, len - (pstr - str));
   }
   return str;
}

#define F_MESSAGE_SIZE 18
#define S_MESSAGE_SIZE 20
char* message;

int main(){
    int pipe1[2];
    int pipe2[2];

    pipe(pipe1);
    pipe(pipe2);

    // посимвольно считываем имя файла:
    message = "Enter file name: ";
    write(1, message, F_MESSAGE_SIZE);

    char file_name[256];
    char simb;
    int i = 0;
    do {
        read(0, &simb, 1);
        if (simb != '\n'){
            file_name[i] = simb;
            ++i;
        }
    } while (simb != '\n');

    // открываем файл с введенным именем:
    int commands = open(file_name, O_RDONLY);
    if (commands == -1){
        perror("Can't open file");
        exit(-2);
    }

    // создаем дочерний процесс:
    int id = fork();

    if (id == -1){
        perror("Fork error");
        exit(-3);
    }
    else if (id == 0){

        close(pipe1[0]);
        close(pipe2[0]);

        double num[50];
        uint8_t check = 1;

        for (int i = 0; i < 3; ++i){
            num[i] = 0;
        }

        // перенаправляем стандартный поток ввода дочернего процесса на открытый файл:
        dup2(commands, 0);

        // посимвольно считываем 3 числа типа double из файла:
        int i = 0;
        int flag = 0;
        double fract = 0.1;
        while (read(0, &simb, 1) > 0) {
            if (simb == ' '){
                ++i;
```

```
                flag = 0;
                fract = 0.1;
                continue;
            }
            if (simb == '.'){
                flag = 1;
                continue;
            }
            if (flag == 0){
                num[i] = num[i] * 10 + (simb - 48);
            }
            else {
                num[i] += fract * (simb - 48);
                fract *= 0.1;
            }
        }

        // делаем проверку деления на 0:
        for (int j = 1; j < i + 1; ++j){
            if (num[j] == 0){
                perror("Can't divide");
                check = 0;
                write(pipe2[1], &check, sizeof(uint8_t)); // если возможно деление на 0,
отправляем по pipe2 сообщение родительскому процессу и завершаем работу.
                close(pipe2[1]);
                exit(-5);
            }
        }

        // если деления на 0 нет, отправляем по pipe2 сообщение родительскому процессу:
        write(pipe2[1], &check, sizeof(uint8_t));

        // считаем и отправляем по pipe1 результат родительскому процессу:
        double res = num[0];
        for (int j = 1; j < i + 1; ++j){
            res /= num[j];
        }

        write(pipe1[1], &res, sizeof(double));
        close(pipe1[1]);
        close(pipe2[1]);

    }
    else {
        close(pipe1[1]);
        close(pipe2[1]);
        double res = 0;
        uint8_t check = 0;
        // читаем сообщение от дочернего процесса:
        read(pipe2[0], &check, sizeof(uint8_t));
        if (check == 0) exit(-5); // завершаем работу, если в дочернем процессе было
деление на 0
        // считываем результат от дочернего процесса и выводим его на экран:
        read(pipe1[0], &res, sizeof(double));
        char str_res[15];
        DoubleToAnsi(res, 4, str_res, sizeof(str_res));
        message = "Result from child: ";
        write(1, message, S_MESSAGE_SIZE);
        write(1, &str_res, 15);
        write(1, "\n", 1);

        close(pipe1[0]);
        close(pipe2[0]);
    }
    close(commands);
    return 0;
}
```

Пример работы

test1.text: 1000 2 2 2 5 5

```
Vanya:Src ivan$ gcc laba.c
Vanya:Src ivan$ ./a.out
Enter file name: ./Tests/test1.txt
Result from child: 5
```

Вывод

Существуют специальные системные вызовы(fork) для создания процессов, а специальные каналы pipe позволяют связать процессы для обмена данными. При использовании fork важно помнить, что фактически создается копию текущего процесса и неправильная работа может привести к неожиданным результатам и последствиям, однако создание процессов очень удобно, когда нужно выполнять несколько действий параллельно. Также у каждого процесса есть свой id, по которому его можно определить. При работе с чтением и записью из файла важно помнить, что read и write возвращают количество успешно считанных/записанных байт и оно не обязательно равно тому значению, которое вы указали. И, конечно, важно не забывать, что открыв файл при помощи open, его обязательно надо будет впоследствии закрыть с помощью close.