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: Д. А. Тарпанов
Преподаватель: Е. С. Миронов
Группа: М8О-204Б-19
Дата:
Оценка:
Подпись:

Москва, 2020

1 Постановка задачи

Задача: 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 создавать для решения 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 7: 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 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.
Алгоритм решения задачи. В начале работы программы считывается имя файла, из которого будут прочитаны числа. Имя файла ограничено 128 сиволами. Этим файлом заменяется поток стандартного ввода. После этого создается канал, необходимый для связи родительского и дочернего процесса. На следующем этапе вызывается функция fork(), возвращаемое значение которой запоминается в переменную. Это нужно для идентификации процесса: если процесс имеет идентификатор, равный единице, то он является родительским. Если идентификатор равен нулю, то это процесс потомок. Случай, когда идентификатор процесса равен минус единице, означает, что произошла ошибка. Далее в конструкции switch-case процесс идентифицируется и выполняются следующие действия: для дочернего процесса закрывается канал на чтение, переназначается файловый дескриптор и с помощью функци execl() вызывается программа, считывающая и суммирующая числа. Считывание происходит посимвольно до конца файла. Так как в условии сказано, что числа должны иметь тип float, целые числа должны вводится со знаками ".0"в конце для упрощения ввода. В результате работы программа, запускаемая дочерним процессом выводит в канал результаты вычислений. В случае, если процесс идентифицирован как родительский, происходит закрытие канала на запись, считывание результатов из канала и вывод суммы в консоль. По ходу программы предусмотрены различные ошибки и их вывод в консоль в случае срабатывания исключения.

Листинг программы

В файле main.c находится код основной программы, в child.c - дочерней.

```
//
// main.c
//

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

int main() {
    char *inputFileName = (char*)malloc(sizeof(char) * 128);
    for(int i = 0; i < 128; ++i) {
        inputFileName[i] = 0;
    }
    float result;
    printf("Enter input file name: ");
    scanf(" %s", inputFileName);
    FILE * inputFile = freopen(inputFileName, "r", stdin);
    int fd[2];
    if(pipe(fd) == -1) {
        printf("Error creating pipe");
        exit(-1);
    }
    int pid = fork();
    switch(pid) {
        case -1: //error case
            printf("Error creating process");
            exit(1);
        case 0: //child case
            close(fd[0]);
            if(inputFile == NULL) {
                printf("Error opening file");
                exit(2);
            }
            if(dup2(fd[1], STDOUT_FILENO) == -1) {
                printf("Error changing file descriptor");
                exit(3);
            }
            if(execl("child", NULL) == -1) {
                printf("Error executing child process");
                exit(4);
            }
            exit(0);
        default: //father case
            close(fd[1]);
            while(read(fd[0], &result, sizeof(float)) > 0) {
                printf("%f\n", result);
            }
```

```
            close(fd[0]);
            exit(0);
    }
    free(inputFileName);
    return 0;
}
```

```
//
// child.c
//
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main() {
    char c;
    float result = 0, currNum = 0;
    while((c = getchar()) != EOF) {
        if (c == '-') {
            scanf("%f", &currNum);
            result -= currNum;
        }
        if (c <= '9' && c >= '0') {
            currNum = c - '0';
            while((c = getchar()) != '.') {
                currNum = currNum * 10 + (c - '0');
            }
            float temp = 0;
            float i = 10;
            while((c = getchar()) != ' ' && c != '\n') {
                temp += ((float)(c - '0'))/i;
                i *= 10;
            }
            result += (currNum + temp);
        }
        if (c == '\n') {
            write(STDOUT_FILENO, &result, sizeof(float));
            result = 0;
            currNum = 0;
        }
    }

}
```

2 Тесты и протокол исполнения

Файл с тестовыми данными:
test.txt

```
123.0 1.75 1.25 14.0 10.0
11.5 19.5 -123.5
0.5 0.5 -0.25 -0.25 0.5 -1.0
```

```
(base) kng@Legion:/mnt/c/vsc/os/lab2$ make
cc -std=c99 main.c -o main
cc -std=c99 child.c -o child
(base) kng@Legion:/mnt/c/vsc/os/lab2$ ./main
Enter input file name: test.txt
150.00
-92.50
0.00
```

3 Выводы

Попробовал поработать с процессами и каналами в ОС Linux. Написал программу на языке C, в которой использовались новые для меня функции, такие как: fork, pipi, dup2, freopen и execl. Удивился тому, что больше всего времени ушло на написание и отладку кода, считывающего посимвольно числа типа float и считающего их сумму, даже учитывая то, что при этом были использованы "костыли"в виде ограничений на ввод. В целом, познакомился с процессами и теперь хочу узнать насколько сильно они могут ускорить выполнение программы.

Список литературы

[1] *Изучаем процессы в Linux — Habr.*
URL: https://habr.com/ru/post/423049/ (дата обращения: 5.11.2020).

[2] *Создание процессов с помощью вызова fork().*
URL: https://www.opennet.ru/docs/RUS/linux$_p$*arallel*/*node*7.*html*(:
5.11.2020).*dup*2
URL : http://www.codenet.ru/progr/cpp/spr/078.php(: 5.11.2020).

[3] *Запуск процессов с помощью вызова exec().*
URL: https://www.opennet.ru/docs/RUS/linux$_p$*arallel*/*node*8.*html*(: 5.11.2020).