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: Н. П. Ежов
Преподаватель: Е. С. Миронов
Группа: М8О-204Б
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 6: 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 int.
Алгоритм решения задачи. В основном процессе до создания дочернего считаем имя и попробуем переопределить поток ввода. В случае успеха создадим канал для связи родительского и дочернего процесса. Т.к. в задании требуется представить дочерний процесс в отдельном файле, то используем execv для его запуска. В самом процессе будем считывать числа из файла, а писать результат вычислений в стандартный поток вывода (который на самом деле канал). Родительский процесс будет читать из канала и выводить на экран значение суммы. Листинг программы

Файл с основным процессом - main.c, файл с дочерним - child.c

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

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

signed main() {
    char s [256];
    for (int i = 0; i < 256; i++) {
        s[i] = 0;
    }
    printf("Enter filename: ");
    scanf(" %s", s);
    FILE* input = NULL;
    input = freopen(s, "r", stdin);
    int fd[2];
    if (pipe(fd) == -1) {
        printf("Error creating pipe!");
        return 1;
    }
    int id = fork();
    if (id == -1) {
        printf("Error creating process!");
        return 2;
    }
    else if (id == 0) {
        close(fd[0]);
        if (input == NULL) {
            printf("Error opening file!\n");
            return 3;
        }
        if (dup2(fd[1], STDOUT_FILENO) == -1) {
            printf("Error changing stdout!\n");
            return 4;
        }
        char *argv[] = {"child", NULL};;
        if (execv("child", argv) == -1) {
            printf("Error executing child process!\n");
            return 5;
        }
    }
    else {
        close(fd[1]);
        int res;
        while (read(fd[0], &res, sizeof(int)) > 0) {
            printf("%d\n", res);
        }
```

```
        close(fd[0]);
    }
    return 0;
}
```

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

int main(){
    char c;
    int num = 0, res = 0, minus = 0;
    while (scanf("%c", &c) > 0) {
        if (c == ' ' || c == '\t') {
            if (minus) {
                res = res - num;
            }
            else {
                res = res + num;
            }
            num = 0;
            minus = 0;
        }
        else if (c == '-') {
            minus = 1;
        }
        else if (c == '\n') {
            if (minus) {
                res = res - num;
            }
            else {
                res = res + num;
            }
            num = 0;
            minus = 0;
            write(STDOUT_FILENO, &res, sizeof(int));
            res = 0;
        }
        else if ('0' <= c && c <= '9') {
            num = num * 10 + c - '0';
        }
    }
    return 0;
}
```

2 Тесты и протокол исполнения

Заранее подготовим файл test.txt, из которого будет читать дочерний процессом test.txt

test.txt

```
10 5 25 10
0 6 -7
2 3 4
```

```
(base) nikita@nikita-desktop:~/CLionProjects/OS_labs/lab2$ make
cc -std=c99 main.c -o main
cc -std=c99 child.c -o child
(base) nikita@nikita-desktop:~/CLionProjects/OS_labs/lab2$ ./main
Enter filename: test.txt
50
-1
9
```

strace

```
write(1, "Enter filename: ", 16Enter filename: ) = 16
read(0, test.txt
"test.txt\n", 1024) = 9
lseek(0, -1, SEEK_CUR) = -1 ESPIPE (Invalid offset operation)
openat(AT_FDCWD, "test.txt", O_RDONLY) = 3
dup3(3, 0, 0) = 0
close(3) = 0
pipe([3, 4]) = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
    child_tidptr=0x7fcdb8531810) = 6367
close(4) = 0
read(3, "2\0\0\0", 4) = 4
write(1, "50\n", 350
) = 3
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=6367, si_uid=1000, si_status
    =0, si_utime=0, si_stime=0} ---
read(3, "\377\377\377\377", 4) = 4
write(1, "-1\n", 3-1
) = 3
read(3, "\t\0\0\0", 4) = 4
write(1, "9\n", 29
) = 2
read(3, "", 4) = 0
close(3) = 0
exit_group(0) = ?
+++ exited with 0 +++
```

3 Выводы

Я изучил работу процессов и каналов в ОС Linux (fork и pipe), составил и отладил программу на языке C. Во время выполнения работы я столкнулся с проблемой взаимодействия процессов, с чем мне помогли freopen и dup2, которые перенаправляют потоки ввода и вывода для программ. Грамотное разделение программы на процессы может уменьшить время выполнения в целом.

Список литературы

[1] *Изучаем процессы в Linux — Habr.*
URL: `https://habr.com/ru/post/423049/` (дата обращения: 06.10.2020).

[2] *Таненбаум Э., Бос Х. Современные операционные системы - 4-е изд.*(дата обращения: 06.10.2020).