Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу

«Операционные системы»

Группа:М8О-215Б-23

Студент: Венгер Ирина Витальевна

Преподаватель: Миронов Е.С.

Оценка: _____

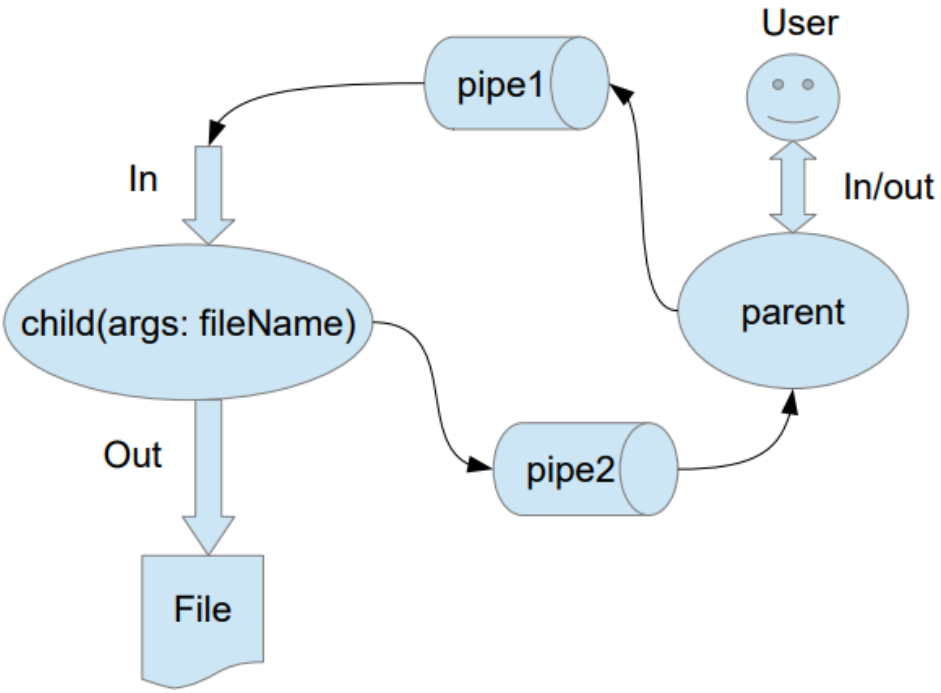
Дата: _____

Москва, 2024.

Содержание

Постановка задачи

Составить и отладить программу на языке С, родительский процесс которой считывает стандартный входной поток, отдает его дочернему процессу, который выполняет проверку строки на правило: «Строка начинается с большой буквы» и, если проверка пройдена успешно, то записывает строку в файл(имя файла также передается от родительского процесса), а в противном случае строки возвращаются в родительский процесс и выводятся в терминал.



3 вариант) Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

Общие сведения о программе

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 представлены разными программами main.c и child.c. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Программа состоит из 2 файлов: main.c и child.c, в которых разделена логика родительского и дочернего процесса соответственно.

Общий метод и алгоритм решения

Внутри main.c реализована функция main, в которой происходит создание двух pipe и форка от родительского процесса (создание дочернего процесса) при помощи. Pipe представляют собой массив из 3 элементов – файловых дескрипторов, файловые дескрипторы – неотрицательные числа, которые являются потоком ввода(0), вывода(1), ошибок(2). В данной лабораторной работе использовались только файловые дескрипторы ввода\вывода для упрощения логики программы. При помощи функции fork() мы создаём новый – дочерний процесс, а возвратным значением fork является process id(PID) – целое неотрицательное число. Если процесс создался, то возвращается 0, это значит, что мы находимся внутри ребёнка и мы запускаем child.c, перед этим закрыв конец записи у 1 пайпа и конец чтения у 2 пайпа. При помощи функции dup2() мы можем перенаправить pipe1 на стандартный ввод дочернего процесса. Аналогично у pipe2, но у стандартного вывода. Если же процесс успешно создался, то мы закрываем конец чтения у pipe1 и конец для записи у второго. Далее считываем имя файла, в который будет выведен результат. Если команда не “выход”, то отправляем её в дочерний процесс через pipe. Затем через pipe2 считываем результат и выводим его в род процессе. В конце работы закрываем оба pipe.

Код программы

Код программы смотрите в приложении 1.

Использование утилиты strace

Результаты strace смотрите в приложении 2.

По результатам strace видно, какие системные вызовы использует OS для реализации той концепции, которую мы реализовали. Nmap показывает, какие ячейки памяти выделяются в ходе выполнения программы. Функция clone является аналогом fork, только внутри ОС и создаёт новый процесс, который на стадии создания является таким же, как и его предок. Его PID=6458, при успешном создании процесса ему присвоился собственный ID. В ходе работы также можно в параллельном терминале убить этот процесс с помощью kill -9 (PID). Функции close используются при закрытии pipe в конце работы программы. Внутри ОС разные каналы pipe имеют разную нумерацию, чтобы не было путаницы у ОС, с каким pipe нужно работать.

Демонстрация работы программы

(base) boopie@MacBook-Air-Irina src % ./main

Введите имя файла: file.txt

Введите числа через пробел или 'выход' для завершения : 100 10 5

Результат деления: 2

Введите числа через пробел или 'выход' для завершения : 90 8 7

Результат деления: 1

Введите числа через пробел или 'выход' для завершения : 70 90 3 0

Результат деления: Деление на 0 запрещено. Выход из child.

Введите числа через пробел или 'выход' для завершения : выход

Вывод

В данной лабораторной работе я познакомилась с процессами и операциями с ними. Научилась налаживать взаимодействие между родительским и дочерним процессами, а также передавать необходимые данные посредством pipe. Также в ходе лабораторной работы использовалась утилита strace, которая наглядно и вполне понятно показывает системные вызовы. Благодаря ней можно отслеживать действия процессов и улучшать понимание того, какой процесс с каким ресурсом взаимодействует. Полученные навыки могут сильно помочь в будущем при работе с крупными проектами, где процессов может быть крайне много.

Приложения

Приложение 1 – код программы:

main.c

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

#define MAX_COMMAND_LEN 256

int main() {
    int pipe1[2], pipe2[2];
    pid_t pid;

    // Создаём  пайп 1 и пайп2
    if (pipe(pipe1) == -1 || pipe(pipe2) == -1) {
        perror("Ошибка");
        exit(1);
    }

    // форкаем и создаём дочерний процесс
    pid = fork();
    if (pid < 0) {
        perror("Ошибка создания процесса");
        exit(1);
    }

    if (pid == 0) {
        // Закрываем конец для записи у 1 пайпа, и закрываем конец чтения 2 пайпа
        close(pipe1[1]);
        close(pipe2[0]);

        // Перенаправляем pipe1[0] на стандартный ввод дочернего процесса
        dup2(pipe1[0], STDIN_FILENO);
        close(pipe1[0]);

        // делаем то же самое, но для стандартного вывода дочернего процесса
        dup2(pipe2[1], STDOUT_FILENO);
        close(pipe2[1]);

        // запускаем дочерний процесс
        execl("./child", "child", NULL);
        perror("Ошибка");
        exit(1);
    } else {
        //закрываем конец для чтения у 1 пайпа, и конец для записи у второго
```

```
        close(pipe1[0]);
        close(pipe2[1]);

        char filename[MAX_COMMAND_LEN];
        printf("Введите имя файла: ");
        fgets(filename, sizeof(filename), stdin);
        filename[strcspn(filename, "\n")] = 0;

        //отправляем имя файла, которое мы ввели через 1 пайп
        write(pipe1[1], filename, strlen(filename) + 1);

        char command[MAX_COMMAND_LEN];
        while (1) {
            printf("Введите числа через пробел или ‘выход’ для завершения: ");
            fgets(command, sizeof(command), stdin);

            // проверка на выход
            if (strstr(command, "выход") != NULL) {
                write(pipe1[1], "выход", strlen("выход") + 1);
                break;
            }

            //отправляем команду в дочерний проесс через первый пайп
            write(pipe1[1], command, strlen(command) + 1);

            // считываем результат из child через pipe2
            char result[MAX_COMMAND_LEN];
            read(pipe2[0], result, sizeof(result));
            printf("Результат деления: %s\n", result);
        }

        // ожидание завершения процесса child
        wait(NULL);
        close(pipe1[1]);
        close(pipe2[0]);
    }

    return 0;
}
```

child.c

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

#define MAX_COMMAND_LEN 256

int main() {
```

```
    char filename[MAX_COMMAND_LEN];
    char command[MAX_COMMAND_LEN];

    //считываем имя файла из 1 пайпа
    read(STDIN_FILENO, filename, sizeof(filename));

    FILE *file = fopen(filename, "a");
    if (file == NULL) {
        perror("Ошибка открытия файла");
        exit(1);
    }

    while (1) {
        // считали команду через первый пайп
        read(STDIN_FILENO, command, sizeof(command));

        // Проверка на exit
        if (strcmp(command, "выход") == 0) {
            printf("Выходим из child...\n");
            break;
        }

        // делим команду на числа, с которыми будем работать
        int num1, num2, result;
        char *token = strtok(command, " ");
        num1 = atoi(token);

        result = num1;
        while ((token = strtok(NULL, " ")) != NULL) {
            num2 = atoi(token);
            if (num2 == 0) {
                printf("Деление на 0 запрещено. Выход из child.\n");
                fprintf(file, "Деление на 0 запрещено. Выход из child.\n");
                fclose(file);
                exit(1);
            }

            result /= num2;
        }
        fprintf(file, "Результат деления: %d\n", result);
        fflush(file);

        // отправка результата в родительский процесс
        char result_str[MAX_COMMAND_LEN];
        sprintf(result_str, "%d", result);
        write(STDOUT_FILENO, result_str, strlen(result_str) + 1);
    }

    fclose(file);
    return 0;
}
```

Приложение 2 – результаты strace

```
root@d0402fc3bf27:/src# strace ./main
execve("./main", ["./main"], 0xfffff53807c0 /* 9 vars */) = 0
brk(NULL)                               = 0xaaaace171000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffffbaf80000
faccessat(AT_FDCWD, "/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=8467, ...}) = 0
mmap(NULL, 8467, PROT_READ, MAP_PRIVATE, 3, 0) = 0xffffbaf7d000
close(3)                                = 0
openat(AT_FDCWD, "/lib/aarch64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\360\206\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1722920, ...}) = 0
mmap(NULL, 1892240, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_DENYWRITE, -1, 0) = 0xffffbad7a000
mmap(0xffffbad80000, 1826704, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0) = 0xffffbad80000
munmap(0xffffbad7a000, 24576)           = 0
munmap(0xffffbaf3e000, 40848)           = 0
mprotect(0xffffbaf1a000, 77824, PROT_NONE) = 0
mmap(0xffffbaf2d000, 20480, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19d000) = 0xffffbaf2d000
mmap(0xffffbaf32000, 49040, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffffbaf32000
close(3)                                = 0
set_tid_address(0xffffbaf80fb0)         = 1193
set_robust_list(0xffffbaf80fc0, 24)     = 0
rseq(0xffffbaf81600, 0x20, 0, 0xd428bc00) = 0
mprotect(0xffffbaf2d000, 12288, PROT_READ) = 0
mprotect(0xaaaabd90f000, 4096, PROT_READ) = 0
mprotect(0xffffbaf86000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0xffffbaf7d000, 8467)            = 0
pipe2([3, 4], 0)                        = 0
pipe2([5, 6], 0)                        = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0xffffbaf80fb0) = 1194
close(3)                                = 0
close(6)                                = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
getrandom("\x4f\xea\x9f\xd0\xe4\x9e\xb6\xce", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0xaaaace171000
brk(0xaaaace192000)                     = 0xaaaace192000
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\32
0\260"..., 34Введите имя файла: ) = 34
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
```

```
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, file1.txt
"file1.txt\n", 1024)            = 10
write(4, "file1.txt\0", 10)             = 10
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260 \321\207\320\265\32
1\200"..., 100Введите числа через пробел или 'выход' для завершения : ) = 100
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 0xaaaace1716b0, 1024)           = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
read(0, 100 10 5
"100 10 5\n", 1024)             = 9
write(4, "100 10 5\n\0", 10)            = 10
read(5, "2\0", 256)                     = 2
write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 \320\264\320\265\320\273\320\265\320\275\320
```

```
\270\321"..., 37Результат деления: 2
) = 37
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260 \321\207\320\265\32
1\200"..., 100Введите числа через пробел или 'выход' для завершения : ) = 100
read(0, exit
"exit\n", 1024)                 = 5
write(4, "exit\n\0", 6)                 = 6
read(5, "0\0", 256)                     = 2
write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 \320\264\320\265\320\273\320\265\320\275\320
\270\321"..., 37Результат деления: 0
) = 37
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260 \321\207\320\265\32
1\200"..., 100Введите числа через пробел или 'выход' для завершения : ) = 100
read(0, выход
"\320\262\321\213\321\205\320\276\320\264\n", 1024) = 11
write(4, "\320\262\321\213\321\205\320\276\320\264\0", 11) = 11
wait4(-1, NULL, 0, NULL)                = 1194
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1194, si_uid=0, si_status=0, si_utime=0, si_stime=0} ---
close(4)                                = 0
close(5)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```