Отчет по лабораторной работе № 2 по курсу
Операционные системы

Студент группы М8О-206Б-21 Синюков Антон Сергеевич, № по списку 19

Контакты www, e-mail, icq, skype vk.com/antonckya

Работа выполнена: « 26 » ноября 2022 г.

Преподаватель: Миронов Евгений

Входной контроль знаний с оценкой _____

Отчет сдан « » _____ 202 __ г., итоговая оценка ___

Подпись преподавателя _____

1. **Тема:** Каналы и процессы

2. **Цель работы:** Цель работы - приобретение практических навыков в управление процессами в ОС и обеспечение обмена данных между процессами посредством каналов

3. **Задание** (*вариант № 3*): Пользователь вводит команды вида: «число число число<endline>». 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.

4. **Оборудование**:
ЭВМ _____, процессор _____, имя узла сети _____с ОП _____Мб,
НМД _____Мб. Терминал _____адрес _____. Принтер _____
Другие устройства _____

5. **Программное обеспечение**:
Операционная система семейства _____, наименование _____версия _____
интерпретатор команд _____версия _____
Система программирования _____версия _____
Редактор текстов _____версия _____
Утилиты операционной системы _____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных _____

6. **Идея, метод, алгоритм** решение задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)

main.c делает fork и создает дочерний процесс с программой child.c, создается два канала для передачи данных между процессами, один для передачи чисел в child.c, второй для передачи результата работы программы, и также передачи кода, сигнализирующего деление на 0. Если ввод окончен, main.c заносит в пайп для сигнализации child.c завершения работы и началу записи в файл. Путь к файлу передается child.c как аргумент при записи.

7. **Сценарий выполнения работы** (план работы, первоначальный текст программы в черновике [можно на отдельном листе] и тесты либо соображения по тестированию)

1. Изучить работу с pipes.
2. Написать программу main.c как родительский процесс.
3. Написать программу child.c как дочерний процесс.
3. Скомпилировать и протестировать программу.

8. **Выводы:** Лабораторная работа оказалась довольно интересной, до этого мне не приходилость писать программы для межпроцессорного взаимодействия на уровне системных вызовов.

Подпись студента _____

9. Код

main.c

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

int fd1[2], fd2[2];
int pipe1, pipe2;

int main (int argc, char *argv[]){

    int arg_len = strlen(argv[1]);

    if (argc != 2){
                printf("The number of arguments is different than required\n");
                return 2;
        }
    if (arg_len > 255){
        printf("The len of the file is higher than required\n");
                return 3;
    }

    pipe1 = pipe(fd1);
    if (pipe1 == -1){
        printf("Pipe1 opening error\n");
                return 4;
    }
    pipe2 = pipe(fd2);
    if (pipe2 == -1){
        printf("Pipe2 opening error\n");
                return 5;
    }

    int id = fork();

    if (id > 0){

        if (close(fd1[0]) == -1){
            printf("Pipe1 (read) closing error\n");
                    return 6;
        }
        if (close(fd2[1]) == -1){
            printf("Pipe2 (write) closing error\n");
                    return 7;
        }

        int message;
        while (scanf("%d", &message) == 1){
                        if (write(fd1[1], &message, sizeof(message)) == -1){
```

```
                printf("Error while writing to Pipe1\n");
                        return 8;
            }
        }
        write(fd1[1], "\n", sizeof(char));
        int res;
        if (read(fd2[0], &res, sizeof(res)) == -1){
            printf("Error while reading from Pipe2\n");
                    return 9;
        }

        if (res == 2){
            printf("Division by zero exception\n");
                    return 10;
        } else if (res != 0){
            printf("Something wrong in child process\n");
                    return 11;
        }
        printf("i'm fine\n");

        if (close(fd1[1]) == -1){
            printf("Pipe1 (write) closing error\n");
                    return 12;
        }
        if (close(fd2[0]) == -1){
            printf("Pipe2 (read) closing error\n");
                    return 13;
        }

    } else if (id == 0){

        if (dup2(fd1[0], 0) == -1){
            printf("Error in dub2 Pipe1 (read)");
            return 14;
        }

        if (dup2(fd2[1], 1) == -1){
            printf("Error in dub2 Pipe2 (write)");
            return 15;
        }

        if (close(fd1[1]) == -1){
            printf("Pipe1 (write) closing error\n");
                    return 16;
        }

        if (close(fd2[0]) == -1){
            printf("Pipe2 (read) closing error\n");
                    return 17;
        }
```

```
        // execl execlp execv execvp
        if (execl("b.out", "b.out", argv[1], (char *)NULL) == -1){
            printf("Error in executing child.c\n");
            return 18;
        }


    } else {
        printf("Fork raise error\n");
                return 19;
    }
    return 0;
}
```

child.c

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

int main(int argc, char *argv[]){

    int message, return_code = 0, count = 1;
    double result;
    while (read(0, &message, sizeof(message)) == sizeof(message)){

                if (count > 1 && message == 0){
            return_code = 2;
            write(1, &return_code, sizeof(return_code));
        }

        if (count == 1){
            result = message;
            count++;
            continue;
        }

        result = result / message;
        count++;
        }
    if (return_code == 2){
        write(1, &return_code, sizeof(return_code));
        return 0;
    }
    write(1, &return_code, sizeof(return_code));

    FILE *fout = fopen(argv[1], "w");
    fprintf(fout, "%lf\n", result);
    fclose(fout);
```

```
    return 0;
}
```

10. Протокол strace

```
root@Anton-Sinyukov:/mnt/c/Users/sinyu/Desktop/os_2# strace ./a.out res.txt
execve("./a.out", ["./a.out", "res.txt"], 0x7ffe11a60018 /* 20 vars */) = 0
brk(NULL)                               = 0x556e197f5000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fff40efc590) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f61213d0000
access("/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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=16355, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 16355, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f61213cc000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-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>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0i8\235HZ\227\223\333\350s\360\352,\223\340."..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2216304, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2260560, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f61211a4000
mmap(0x7f61211cc000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f61211cc000
mmap(0x7f6121361000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f6121361000
mmap(0x7f61213b9000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x214000) = 0x7f61213b9000
mmap(0x7f61213bf000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f61213bf000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f61211a1000
arch_prctl(ARCH_SET_FS, 0x7f61211a1740) = 0
set_tid_address(0x7f61211a1a10)         = 741
set_robust_list(0x7f61211a1a20, 24)     = 0
rseq(0x7f61211a20e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f61213b9000, 16384, PROT_READ) = 0
mprotect(0x556e18849000, 4096, PROT_READ) = 0
mprotect(0x7f612140a000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f61213cc000, 16355)           = 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=0x7f61211a1a10) = 742
close(3)                                = 0
close(6)                                = 0
newfstatat(0, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0x1), ...}, AT_EMPTY_PATH) = 0
getrandom("\xc0\x51\x1f\x5b\xdf\xc7\x96\xa0", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x556e197f5000
brk(0x556e19816000)                     = 0x556e19816000
read(0, 10 2 2 2
"10 2 2 2\n", 1024)             = 9
write(4, "\n\0\0\0", 4)                 = 4
write(4, "\2\0\0\0", 4)                 = 4
write(4, "\2\0\0\0", 4)                 = 4
write(4, "\2\0\0\0", 4)                 = 4
write(4, "\2\0\0\0", 4)                 = 4
write(4, "\2\0\0\0", 4)                 = 4
read(0, "", 1024)                       = 0
write(4, "\n", 1)                       = 1
read(5, "\0\0\0\0", 4)                  = 4
newfstatat(1, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0x1), ...}, AT_EMPTY_PATH) = 0
write(1, "i'm fine\n", 9i'm fine
)               = 9
close(4)                                = 0
close(5)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```