Отчет по лабораторной работе № 4 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: « 7 » января 2023 г.

Преподаватель: Миронов Евгений

Входной контроль знаний с оценкой _____

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

Подпись преподавателя _____

1. **Тема:** Файлы, отображаемые в память

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

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)

Логика работы аналогична логике из ЛР2: main.c делает fork и создает дочерний процесс с программой child.c, создается mmap для передачи данных между процессорами вида ключ - значение. Ключ отвечает за то, идет ли в мапе запись или чтение, это было сделано чтобы предотвратить гонку в данном участке памяти. При делении на 0 child.c заносит в ключ значение 3. Если ввод окончен main.c заносит в ключ код 2 для сигнализации child.c завершения работы и началу записи в файл.

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 memory map.
2. Написать программу main.c как родительский процесс.
3. Написать программу child.c как дочерний процесс.
3. Скомпилировать и протестировать программу.

8. **Выводы:** Мапы оказались, как по мне, более удобными чем пайпы. При работе с ними нужно создавать в 2 раза меньше файловых дескрипторов, они используются как массивы, а не как потоки данных. Единственная сложность при работе с ними - это предотвращение гонки, но это можно исправить сигналами, shared mutex, или, как в моем случае, флагом в начале mmap.

Подпись студента _____

9. Код

main.c

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

#define True 1

int fd;

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;
    }

    fd = open("temp", O_RDWR | O_CREAT | O_TRUNC, 0777);
    /*
    O_RDWR - режим доступа на чтение и запись
    O_CREAT - файл будет создан если не существует
    O_TRUNC - при открытии файл отчистится
    0777 (a.k.a. S_IRWXU | S_IRWXG | S_IRWXO)
    */
    if (fd < 0){
                printf("Error while opening a file\n");
                return 4;
        }

    ftruncate(fd, 2 * sizeof(int)); // изменение размера файла с файловым дескриптором fd

    int *ptr;
    ptr = mmap(NULL, 2 * sizeof(int), PROT_WRITE | PROT_READ, MAP_SHARED, fd, 0); // отражение на па
    /*
    mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset);
    PROT_WRITE - данные можно читать
    PROT_READ - в данную область можно записывать
    MAP_SHARED - эту область могут использовать другие процессы
```

```
    */
    if (ptr == MAP_FAILED){
                printf("Error while memory map created\n");
                return 5;
        }
    ptr[0] = 0;
    /*
    ptr[0] = 0 - готово к записи
    ptr[0] = 1 - запись произошла, можно читать
    ptr[0] = 2 - запись окончена, а.к.а. данных больше не будет
    ptr[0] = 3 - в дочернем процессе произошло деление на 0
    ptr[0] = 4 - другие исключения в работе дочернего процесса
    */

    int id = fork();

    if (id > 0){
        int message;
        int reading;

        while (True){
            while (ptr[0] == 1){
                // активное ожидание
            }
            reading = scanf("%d", &message);

            printf("Readed in main.c: %d\n", message);

            if (reading > 0){
                ptr[1] = message;
                if (ptr[0] == 3){
                    break;
                }
                ptr[0] = 1;
            } else {
                if (ptr[0] == 3){
                    break;
                }
                ptr[0] = 2;
                break;
            }
            if (ptr[0] == 3){
                break;
            }
        }

        printf("Reading ended in main.c\n");

        if (ptr[0] == 3){
            printf("Division by zero\n");
```

```
                return 0;
        }

        if (close(fd) == -1){
                printf("Error while closing a file\n");
                return 17;
        }

        munmap(ptr, 2*sizeof(int));

    } else if (id == 0){
        printf("Forking success in main.c\n");

        if (execl("c.out", "c.out", NULL, (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 <string.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>

#define True 1

int fd;

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

    printf("Forking success in child.c\n");

    fd = open("temp", O_RDWR);
    if (fd < 0){
                printf("Error while opening a file in child process\n");
                return 4;
        }

    int *ptr;
```

```
ptr = mmap(NULL, 2 * sizeof(int), PROT_WRITE | PROT_READ, MAP_SHARED, fd, 0);
if (ptr == MAP_FAILED){
            printf("Error while memory map created in child process\n");
            return 5;
    }

int message, count = 1;
double result;

while(True){
    while(ptr[0] == 0){
        // активное ожидание
    }
    if (ptr[0] == 1){
        ptr[0] = 0;
        message = ptr[1];
        printf("Readed in child.c: %d, result = %f\n", message, result);

        if (count > 1 && message == 0){
            ptr[0] = 3;
            printf("Division by zero in child.c\n");
            break;
        }

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

        result = result / message;
        count++;
    } else if (ptr[0] == 2){
        FILE *fout = fopen(argv[1], "w");
        printf("Writing to file in child.c, file name = %s\n", argv[1]);
        fprintf(fout, "%f\n", result);
        fclose(fout);
        break;
    }

}

printf("Reading ended in child.c, result = %f\n", result);

if (close(fd) == -1){
            printf("Error while closing a file in child process\n");
            return 17;
}
munmap(ptr, 2*sizeof(int));
```

```
    return 0;
}
```

10. Протокол strace

```
root@Anton-Sinyukov:/mnt/c/Users/sinyu/Desktop/os_4# strace ./a.out res.txt
execve("./a.out", ["./a.out", "res.txt"], 0x7fffcce5e8e8 /* 20 vars */) = 0
brk(NULL)                               = 0x56317c9bb000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffffe638360) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa1ef3ae000
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) = 0x7fa1ef3aa000
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) = 0x7fa1ef182000
mmap(0x7fa1ef1aa000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fa1ef1aa000
mmap(0x7fa1ef33f000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7fa1ef33f000
mmap(0x7fa1ef397000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x214000) = 0x7fa1ef397000
mmap(0x7fa1ef39d000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fa1ef39d000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa1ef17f000
arch_prctl(ARCH_SET_FS, 0x7fa1ef17f740) = 0
set_tid_address(0x7fa1ef17fa10)         = 706
set_robust_list(0x7fa1ef17fa20, 24)     = 0
rseq(0x7fa1ef1800e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7fa1ef397000, 16384, PROT_READ) = 0
mprotect(0x56317af00000, 4096, PROT_READ) = 0
mprotect(0x7fa1ef3e8000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fa1ef3aa000, 16355)           = 0
openat(AT_FDCWD, "temp", O_RDWR|O_CREAT|O_TRUNC, 0777) = 3
ftruncate(3, 8)                         = 0
mmap(NULL, 8, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7fa1ef3e7000
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fa1ef17fa10) = 707
newfstatat(0, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0x1), ...}, AT_EMPTY_PATH) = 0
getrandom("\x7c\x13\x2f\x87\xf4\xbd\x38\xdb", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x56317c9bb000
brk(0x56317c9dc000)                     = 0x56317c9dc000
read(0, Forking success in main.c
Forking success in child.c
10 2 2 2
"10 2 2 2\n", 1024)             = 9
newfstatat(1, "", {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0x1), ...}, AT_EMPTY_PATH) = 0
write(1, "Readed in main.c: 10\n", 21Readed in main.c: 10
)  = 21
write(1, "Readed in main.c: 2\n", 20Readed in main.c: 2
write(1, "Readed in main.c: 10\n", 21Readed in main.c: 10
)  = 21
write(1, "Readed in main.c: 2\n", 20Readed in main.c: 2
)   = 20
Readed in child.c: 10, result = 0.000000
Readed in child.c: 2, result = 10.000000
write(1, "Readed in main.c: 2\n", 20Readed in main.c: 2
)   = 20
Readed in child.c: 2, result = 5.000000
write(1, "Readed in main.c: 2\n", 20Readed in main.c: 2
)   = 20
Readed in child.c: 2, result = 2.500000
read(0, "", 1024)                       = 0
write(1, "Readed in main.c: 2\n", 20Readed in main.c: 2
)   = 20
write(1, "Reading ended in main.c\n", 24Reading ended in main.c
) = 24
close(3)                                = 0
munmap(0x7fa1ef3e7000, 8)               = 0
exit_group(0)                           = ?
Writing to file in child.c, file name = res.txt
+++ exited with 0 +++
root@Anton-Sinyukov:/mnt/c/Users/sinyu/Desktop/os_4# Reading ended in child.c, result = 1.250000
```