Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу
«Операционные системы»

Студент: Юнусов Руслан Асифович
Группа: М8О-209Б-23
Вариант: 5
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий..... 2
Постановка задачи..... 3
Общие сведения о программе.....4
Общий метод и алгоритм решения..... 4
Исходный код..... 5
Демонстрация работы программы..... 13
Выводы..... 13

Репозиторий

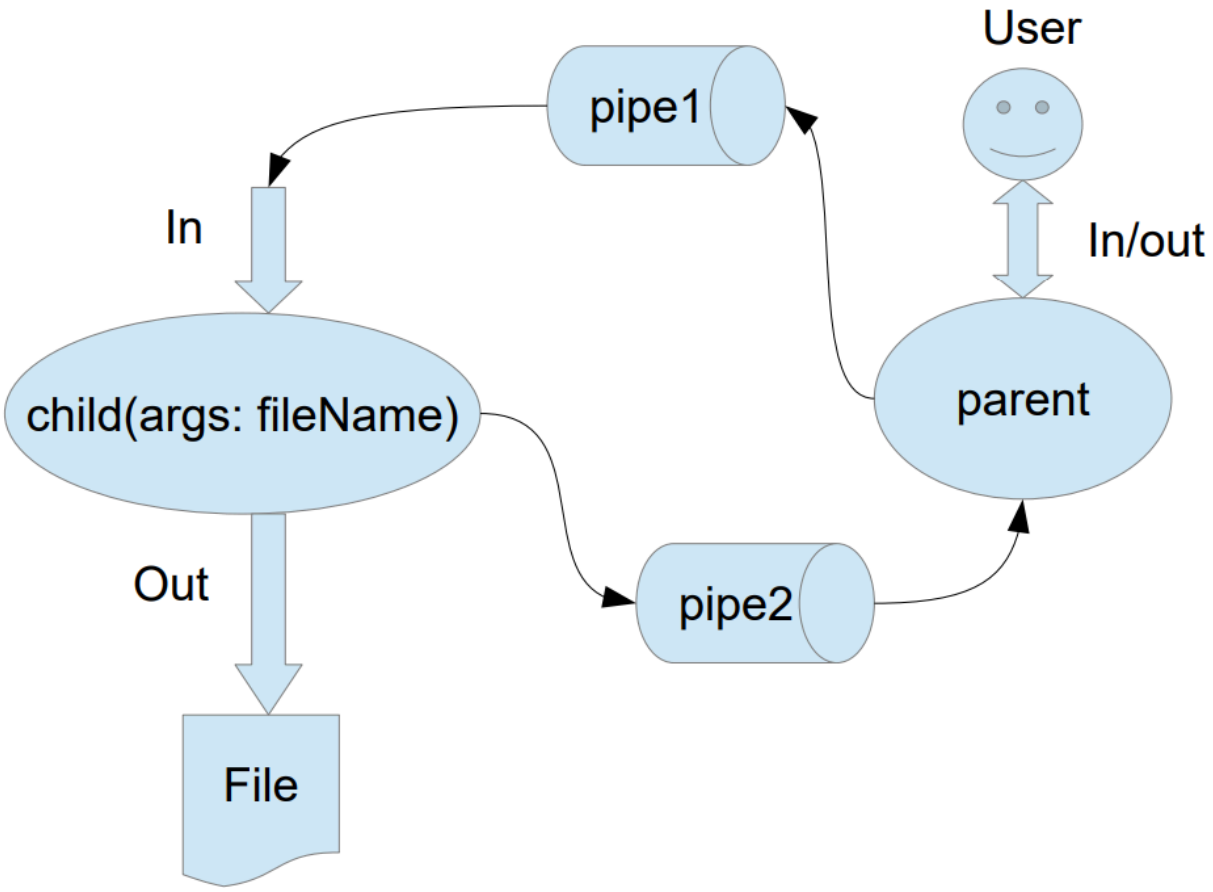
https://github.com/Rissochek/OSLabs/tree/main/lab4

Постановка задачи

Цель работы
Приобретение практических навыков в:
· Освоение принципов работы с файловыми системами
· Обеспечение обмена данных между процессами посредством технологии «File mapping»

Задание
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 создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.



4 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 float. Количество чисел может быть произвольным.

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: unistd.h, stdio.h , stdlib.h, fcntl.h, sys/mman.h, sys/stat.h, string.h, stdbool.h, sys/wait.h. В программе используются следующие системные вызовы:

1. shm_open - создаёт/открывает объекты общей памяти POSIX.
2. ftruncate - обрезает файл до заданного размера.
3. mmap, munmap - отображает файлы или устройства в памяти, или удаляет их отображение.
4. memset - заполнение памяти значением определённого байта.
5. close - закрывает файловый дескриптор.
6. execl - запуск файла на исполнение.
7. perror – вывод сообщения об ошибке.
8. exit - завершает выполнение программы.
9. wait - получает статус завершения дочернего процесса.
10. close - закрывает файл, а также файловые дескрипторы.

Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, mmap, munmap, execl, close, read, write, shm_open, ftruncate, memset.

2. Написать программу, которая будет работать с 2-мя процессами: один из них родительский и один дочерний, процессы обмениваются данными при помощи выделенной памяти.
3. Организовать работу с выделением памяти под строку неопределенной длины. Грамотно передать данные между процессами. Реализовать функцию проверки строки на наличие нулей, что противоречит правилам деления. Провести калькуляцию. Провести работу связанную с файлами и записать в файл результат вычислений.
4. Освободить всю выделенную память, а также проверить на наличие утечек при помощи специализированных программ.

Исходный код

main.c

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

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

{
   const char* back_name = "Lab3.back";
   unsigned perms = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;
   char  *input_data = malloc(sizeof(char) * 50);
```

```
size_t   size = 50;
size_t map_size = 0;
pid_t  cpid;
int    counter = 0;
char ch = ' ';

if (argc != 2) {
    fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
    exit(EXIT_FAILURE);
    }
while ((ch = getchar()) != EOF && ch != '\n'){
    if (counter < size){
        input_data[counter++] = ch;
    } else{
        size *= 2;
        char *buffer = realloc(input_data, size);
        if (buffer == NULL) {
            perror("realloc failed");
            exit(EXIT_FAILURE);
        }else{
            input_data = buffer;
            input_data[counter++] = ch;
        }
    }
}
input_data[counter] = '\0';

int fd = shm_open(back_name, O_RDWR | O_CREAT, perms);
  if (fd == -1) {
```

```
    perror("OPEN");
    exit(EXIT_FAILURE);
  }

  map_size = size + (size_t)strlen(argv[1]) + (size_t)1;
  ftruncate(fd, (off_t)size);

  caddr_t memptr = mmap(
    NULL,
    map_size,
    PROT_READ | PROT_WRITE,
    MAP_SHARED,
    fd,
    0);

  if (memptr == MAP_FAILED) {
    perror("MMAP");
    exit(EXIT_FAILURE);
  }
  memset(memptr, '\0', map_size);
  sprintf(memptr, "%s", argv[1]);
  sprintf(memptr + strlen(memptr), "%s", "|");
  sprintf(memptr + strlen(memptr), "%s", input_data);

  cpid = fork();
  if (cpid == -1) {
    perror("fork");
    exit(EXIT_FAILURE);
  }
```

```
    if (cpid == 0){
        munmap(memptr, size);
            close(fd);
        execl("child", "child", NULL);
        perror("EXECL");
            exit(EXIT_FAILURE);
    }else{
        int status = 0;
        wait(&status);
        free(input_data);
        if (WIFEXITED(status) && WEXITSTATUS(status) == 0) {
            exit(EXIT_SUCCESS);
        } else {
            exit(EXIT_FAILURE);
        }
    }
}
```

child_program.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

```
#include <sys/wait.h>

bool check_on_zeros(char* data, int size);
float calc_func(char* data, int size);

int main(){
    const char* back_name = "Lab3.back";
    unsigned perms = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;
    size_t counter = 0;
    char filename[20];
    size_t readed_data_id = 0;
    int map_fd = shm_open(back_name, O_RDWR, perms);
    if (map_fd < 0) {
        perror("SHM_OPEN");
        exit(EXIT_FAILURE);
    }

    struct stat statbuf;
    fstat(map_fd, &statbuf);
    const size_t map_size = statbuf.st_size;

    caddr_t memptr = mmap(
        NULL,
        map_size,
        PROT_READ | PROT_WRITE,
        MAP_SHARED,
        map_fd,
        0);
    if (memptr == MAP_FAILED) {
```

```
    perror("MMAP");
    exit(EXIT_FAILURE);
}

char  *input_data = malloc(sizeof(char) * (map_size + 1));

for (size_t i = 0; i < map_size; i++){
    if (memptr[i] != '|'){
        filename[i] = memptr[i];
        } else{
            filename[i] = '\0';
            readed_data_id = i+1;
            break;
        }
}
for (size_t i = readed_data_id; i < map_size; i++) {
    input_data[i - readed_data_id] = memptr[i];
}
float tmp = 0;
if (check_on_zeros(input_data, map_size) == 0){
    tmp = calc_func(input_data, map_size);
} else{
    perror("Cannot devide by zero\n");
    free(input_data);
    _exit(EXIT_FAILURE);
}

FILE *fptr;
fptr = fopen(filename, "w");
```

```
    fprintf(fptr, "%f", tmp);
    fclose(fptr);
    free(input_data);
    munmap(memptr, map_size);
    close(map_fd);
    _exit(EXIT_SUCCESS);
}

bool check_on_zeros(char* data, int size){
    for (int i = 1; i < size; i++){
        if (data[i] == '0'){
            if (data[i+1] == ' ' || data[i+1] == '\0' || data[i+1] == '\n'){
                if (data[i-1] == ' '){
                    return true;
                }
            }
            if (data[i+1] == '.' || data[i+1] == ','){
                int non_zeros_counter = 0;
                i += 2;
                while(data[i] != ' ' && data[i] != '\0'){
                    char sth = data[i++];
                    if (sth != '0'){
                        if (sth != '\0'){
                            if (sth != '\n'){
                                non_zeros_counter += 1;
                            }
                        }
                    }
                }
```

```
            if (non_zeros_counter == 0){
                return true;
            }
            non_zeros_counter = 0;
        }
    }
}
return false;
}

float calc_func(char* data, int size){
    int to_alloc = 100;
    char *buffer = malloc(sizeof(char) * to_alloc);
    int j = 0;
    char ch;
    float first_number;
    bool flag = false;
    float number = 1;
    float tmp = 0;
    fflush(stdout);
    for (int i = 0; i < size; i++){
    fflush(stdout);
        if (data[i] != '\0'){
            while ((ch = data[i]) != ' ' && ch != '\n' && ch != '\0') {
                buffer[j] = ch;
                j++;
                i++;
                if (i >= size){
                    break;
```

```
            }
        }
        if (flag == false){
            sscanf(buffer, "%f", &first_number);
            number = first_number;
            flag = true;
        }else{
            sscanf(buffer, "%f", &tmp);
            number = number / tmp;
        }
        j = 0;
        while (buffer[j] != '\0'){
            buffer[j++] = ' ';
            if (j >= to_alloc){
                break;
            }
        }
        j = 0;
    }else{
        break;
    }
  }
  free(buffer);
  return number;
}
```

Демонстрация работы программы

parol1@riss:~/Labs/OSLabs/lab3$ make build

```
gcc  -o parent main.c
gcc  -o child child_program.c
parol1@riss:~/Labs/OSLabs/lab3$ ./parent test.txt
2 2
parol1@riss:~/Labs/OSLabs/lab3$ cat test.txt
1.000000
```

Выводы

В ходе проделанной работы на Unix-подобной ОС я узнал много нового о работе процессов, а также смог реализовать программу, которая реализует работу, приведенную в задании моего варианта. В Си помимо механизма общения между процессами через pipe, также существуют и другие способы взаимодействия, например отображение файла в память, такой подход работает быстрее, за счет отсутствия постоянных вызовов read, write и тратит меньше памяти под кэш. После отображения возвращается void*, который можно привести к своему указателю на тип и обрабатывать данные как массив, где возвращенный указатель – указатель на первый элемент.