Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

Студент: Куценко Б.Д.

Группа: М80-207Б-20

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата:

Содержание

1. Постановка задачи.
2. Общие сведения о программе.
3. Общий метод и алгоритм решения.
4. Код программы.
5. Демонстрация работы программы.
6. Вывод.

## **Постановка задачи**

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.

**Группа вариантов № 2:**

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 переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода 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.

**Вариант 9:**

В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. 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 состоит из одного файла lab4.c.

Программа использует следующие системные вызовы:

1. **sem\_open –** для создания нового именнованного семафора.
2. **sem\_unlink** – для удаления именованного семафора.
3. **open -** для создания файла и его открытия.
4. **close** – для закрытия файлового дескриптора.
5. **mmap –** для отображения файла в память.
6. **fork** – для создания дочернего процесса.
7. **sem\_wait –** для блокировки семафора.
8. **sem\_post** – для разблокировки семафора.
9. **dup2** – для перенаправления потока вывода.
10. **getpid** - для получения id процесса.
11. **ftruncate** — обрезает/расширяет файл до заданного размера.
12. **remove** — для удаления файла.

**Общий метод и алгоритм решения**

* создать именованный семофор, проверить, создать файл pipe1.txt**,** отобразить его в память c помощью **mmap** как pipe\_1, создать дочерний процесс(с помощью **fork**) и обработать возможные ошибки.

То же самое проделать с pipe2.txt.

* Из родительского процесса: блокируем семафор записываем в разделенную память данные (список чисел), разблокируем семафор.
* В дочернем процесс блокируем семафор, читаем из разделенной памяти список чисел и выводим результат деления первого числа на последующие.
* В конце родительского процесса удаляем созданные файлы, удаляем семафор

## **Код программы**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <string.h>

#include <fcntl.h>

#include <semaphore.h>

#include <pthread.h>

#include <math.h>

char\* get\_string() {

    int len = 0;

    int capacity = 1;

    char \*s = (char\*) malloc(sizeof(char));

    char c = getchar();

    while (c != '\n') {

        s[(len)++] = c;

        if (len >= capacity) {

            capacity \*= 2;

            s = (char\*) realloc(s, capacity \* sizeof(char));

        }

        c = getchar();

    }

    s[len] = '\0';

    return s;

}

void reverse(char\* str, int len) {

    int i = 0, j = len - 1, temp;

    while (i < j) {

        temp = str[i];

        str[i] = str[j];

        str[j] = temp;

        i++;

        j--;

    }

}

int intToStr(int x, char str[], int d)

{

    int i = 0;

    int flag = 0;

    if (x < 0) {

        x \*= -1;

        flag = 1;

    }

    while (x) {

        str[i++] = (x % 10) + '0';

        x = x / 10;

    }

    while ((i < d) || flag == 1) {

        if (flag == 1) {

            str[i++] = '-';

            flag = 0;

        } else str[i++] = '0';

    }

    reverse(str, i);

    str[i] = '\0';

    return i;

}

void ftoa(float n, char\* res, int afterpoint) {

    int ipart = (int)n;

    float fpart = n - (float)ipart;

    if (fpart < 0) fpart \*= -1;

    int i = intToStr(ipart, res, 0);

    if (afterpoint != 0) {

        res[i] = '.';

        fpart = fpart \* pow(10, afterpoint);

        intToStr((int)fpart, res + i + 1, afterpoint);

    }

}

int numberOfIntpart(int i) {

    if (i < 0) i \*= -1;

    int l = 1;

    while (i / 10) {

        i /= 10;

        l++;

    }

    return l;

}

int main() {

    const char \*in\_sem\_name = "input\_semaphor";

    const char \*out\_sem\_name = "output\_semaphor";

    sem\_unlink(in\_sem\_name);

    sem\_unlink(out\_sem\_name);

    sem\_t \*input\_semaphore = sem\_open(in\_sem\_name, O\_CREAT, 777, 1);

    sem\_t \*output\_semaphore = sem\_open(out\_sem\_name, O\_CREAT, 777, 1);

    int pipe1 = open("pipe1.txt", O\_RDWR | O\_CREAT, 0777);

    int pipe2 = open("pipe2.txt", O\_RDWR | O\_CREAT, 0777);

    if (pipe1 == -1) {

        printf("ERROR: creating map of file mapping was failed\n");

        exit(EXIT\_FAILURE);

    }

    if (pipe2 == -1) {

        printf("ERROR: creating map of file mapping was failed\n");

        exit(EXIT\_FAILURE);

    }

    char \*pipe\_1 = (char\*)mmap(NULL, getpagesize(), PROT\_READ | PROT\_WRITE,  MAP\_SHARED, pipe1, 0);

    char \*pipe\_2 = (char\*)mmap(NULL, getpagesize(), PROT\_READ | PROT\_WRITE,  MAP\_SHARED, pipe2, 0);

    if (pipe\_1 == MAP\_FAILED) {

        printf("ERROR: file1 mappinf was failed\n");

        exit(EXIT\_FAILURE);

}

    if (pipe\_2 == MAP\_FAILED) {

        printf("ERROR: file1 mappinf was failed\n");

        exit(EXIT\_FAILURE);

    }

    pid\_t pid = fork();

    if (pid < 0) {

        printf("ERROR: fork was failed\n");

        exit(EXIT\_FAILURE);

    } else if (pid == 0) { // child

        sem\_wait(input\_semaphore);

        struct stat st;

        if(fstat(pipe1, &st)) {

            printf("Error: fstat was failed\n");

            exit(EXIT\_FAILURE);

        }

        char\* receivedStr = malloc((st.st\_size + 1) \* sizeof(char));

        \*(receivedStr + st.st\_size) = '\0';

        for(int i = 0; i < st.st\_size; i++) {

            receivedStr[i] = pipe\_1[i];

        }

        int file = open(receivedStr, O\_RDONLY);

        if (file == -1) {

            printf("ERROR: file '%s' wasn't open. Check the name of file.\n", receivedStr);

            exit(EXIT\_FAILURE);

        } else {

            printf("File '%s' was successfully opened.\n", receivedStr);

        }

        free(receivedStr);

        dup2(file, STDIN\_FILENO);

        close(file);

        float curr;

        char end;

        float \*arr = NULL;

        int counter = 0;

        int numerator = 0;

        int length = 0;

        int writedlength = 0;

        while (scanf("%f", &curr) != EOF) {

            counter++;

            if (counter == 1) {

                numerator = curr;

            }

            if ((arr = realloc(arr, sizeof(curr) \* counter - 1)) == NULL) {

                printf("ERROR: realloc was failed\n");

            }

            if (counter > 1) {

                if (curr == 0) {

                    printf("ERROR: division by zero. Stopping program...\n");

                    exit(EXIT\_SUCCESS);

                }

                arr[counter - 2] = numerator / curr;

            }

            if (((end = getchar()) == EOF) || (end == '\n')) {

                for(int i = 0; i < counter -1; i++) {

                    char res[20];

                    ftoa(arr[i], res, 4);

                    length += (strlen(res) + 1 )\* sizeof(char);

                    // printf("%s ", res);

                }

                // putchar('\n');

                if(ftruncate(pipe2, length)) {

                    printf("ERROR: ftruncate was failed\n");

                    remove("pipe1.txt");

                    remove("pipe2.txt");

                    free(arr);

                    exit(EXIT\_FAILURE);

                }

                for (int i = 0; i < counter - 1; i++){

                    char res[20];

                    ftoa(arr[i], res, 4);

                    for (int j = 0; j < strlen(res); j++) {

                        pipe\_2[writedlength + j] = res[j];

                    }

                    writedlength += strlen(res) - 1;

                    pipe\_2[++writedlength] = ((i == counter - 2) ? '\n' : ' ');

                    writedlength++;

                }

                counter = 0;

                free(arr);

                // printf("length = %d, writedlength = %d\n", length, writedlength);

            }

        }

        sem\_post(input\_semaphore);

        close(file);

        exit(0);

    } else { // parent

        sem\_wait(input\_semaphore);

        printf("Parent's id: %d, Child's id: %d\n", getpid(), pid);

        printf("Enter the file's name: ");

        sem\_post(output\_semaphore);

        int c = 0;

        char\* filename = get\_string();

        int number = strlen(filename);

        if(ftruncate(pipe1, number)) {

            printf("ERROR: ftruncate was failed\n");

            exit(EXIT\_FAILURE);

        }

        for (int i = 0; i < number; i++) {

            pipe\_1[i] = filename[i];

        }

        free(filename);

        sem\_post(input\_semaphore);

        wait(NULL);

        struct stat st;

        if(fstat(pipe2, &st)) {

            printf("ERROR: fstat of pipe2 was failed\n");

            exit(EXIT\_FAILURE);

        }

        // printf("Size of pipe2 = %ld\n", st.st\_size);

        printf("\n\*\*\*Result of division\*\*\*\n");

        for(int i = 0; i < st.st\_size; i++) {

            printf("%c",pipe\_2[i]);

        }

        close(pipe1);

        close(pipe2);

        remove("pipe1.txt");

        remove("pipe2.txt");

        sem\_destroy(input\_semaphore);

        sem\_destroy(output\_semaphore);

    }

}

## 

## **Распечатка протокола работы программы**

## **Вывод**

При выполнении данной работы я вспомнил как работать с процессами в си. Научился синхронизировать работу процессов с помощью семафора. Так как в ЛР2 обмен данными осуществлялся через пайп, там не обязательно было использовать мьютексы, семафоры и другие приспособления для синхронизации. Однако в ЛР4 обмен данными происходит через файл, который отображен в память, работа с ним идет быстрее нежели с обычным файлом, но, чтобы процессы не мешали друг другу, мы вынуждены использовать, например, семафор, который бы блокировал один процесс, пока другой делал что-то с общей памятью.