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

Лабораторная работа №2 по курсу

«Операционные системы»

Студент: Шумилова Александра
Группа: М8О-207Б-21
Вариант: 20
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/tonsoleils/OS

Постановка задачи

Цель работы

Приобретение практических навыков в:

- Управлении процессами в ОС
- Обеспечении обмена данных между процессами посредством каналов

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 каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Общие сведения о программе

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

1. pipe() - создание ненаправленного канала для межпроцессной коммуникации.
2. fork() - создание процесса, который является практически копией родительского процесса.
3. dup2(oldfd, newfd) - делает newfd копией oldfd, закрывая newfd, если требуется.

Общий метод и алгоритм решения

Родительский процесс считывает названия двух файлов, открывает их для чтения и создаёт 2 дочерних потока. Затем родительский процесс в цикле (пока не прекратили ввод) считывает строки и, в зависимости от их длины, отправляет в соответствующие дочерние потоки сначала размер строки (это необходимо для того, чтобы знать, сколько считать с пайпа символов), а затем и саму строку с помощью пайпов. Дочерний процесс считывает размер, затем считывает строку заданного размера, подменяет дескриптор стандартного вывода на дескриптор открытого соответствующего файла, инвертирует строку и выводит

ее в стандартный вывод. Благодаря подмене дескрипторов строка выводится не в терминал, а в открытый файл. При окончании ввода строк все дескрипторы пайпов и все файлы закрываются.

Исходный код

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

#define READ 0
#define WRITE 1

void reverse(char *string) { // Функция для инвертирования строки
   int length = strlen(string); // получаем длинну строки
   int middle = (length - 1) / 2; // получаем индекс середины строки (игнорируя последний терминальный символ)

   char temp;

   for (int i = 0; i < middle; i++) {
      // переставляем местами символы
      temp = string[i];
      string[i] = string[length - i - 2];  // -2 потому что игнорируем последний терминальный символ
      string[length - i - 2] = temp;
   }
}

int main() {
```

```
    pid_t child_a, child_b; // идентификаторы потоков
    int fd_a[2], fd_b[2]; // файловые дескрипторы для pipe'ов
    int file_a; // дескриптор файла A
    int file_b; // дескриптор файла B

    if (pipe(fd_a) == -1) { // Создаем пайп A, если ошибка - завершаем работу
        fprintf(stderr, "Pipe Failed");
        exit(EXIT_FAILURE);
    }
    if (pipe(fd_b) == -1) { // Создаем пайп B, если ошибка - завершаем работу
        fprintf(stderr, "Pipe Failed");
        exit(EXIT_FAILURE);
    }

    char file_name_a[256]; // переменная для имени файла A
    char file_name_b[256]; // переменная для имени файла B
    fgets(file_name_a, sizeof file_name_a, stdin); // Считываем первое имя файла
    fgets(file_name_b, sizeof file_name_b, stdin); // Считываем второе имя файла
    file_name_a[strcspn(file_name_a, "\n")] = 0; // удаляем символ перевода каретки
    file_name_b[strcspn(file_name_b, "\n")] = 0;
    file_a = open(file_name_a, O_RDWR | O_CREAT | O_TRUNC, 0777); // открываем файл для
записи
    if (!file_a) {
        fprintf(stderr, "Could not open file\n");
        exit(1);
    }
    file_b = open(file_name_b, O_RDWR | O_CREAT | O_TRUNC, 0777); // открываем файл для
записи
    if (!file_b) {
        fprintf(stderr, "Could not open file\n");
        exit(2);
    }
```

```
    child_a = fork(); // Создаём новый процесс, который является практически копией родителя-процесса

    if (child_a < 0) { // Если при создании fork произошла ошибка
        fprintf(stderr, "child process A not created\n");
        exit(3);
    } else if (child_a == 0) {
        /* ------- CHILD A ------- */

        printf("Run child A\n");
        int size; // Размер строки, которую передал родительский процесс
        while(read(fd_a[READ], &size, sizeof(int)) > 0) { // считываем размер строки
            // т.е. когда в пайп попадает размер строки, мы продолжаем работу
            // значит в пайп попала и строка этого размера
            // и мы считываем эти параметры
            char buffer_a[255];
            read(fd_a[READ], &buffer_a, sizeof(char) * size); // считываем строку размера size
            dup2(file_a, STDOUT_FILENO); // подменяем дескриптор стандартного вывода на дескриптора файла
            // это нужно для того, чтобы при выводе строки в стандартный поток, строка записывалась в файл
            reverse(buffer_a); // инвертируем строку
            printf("%s", buffer_a); // выводим в стандартный поток, но так как файловые дескрипторы подменены,
                                    // строка выводится в файл
        }
    } else {
        // Мы в родительском процессе
        // Создаём второй дочерний поток

        child_b = fork();

        if (child_b < 0) {
```

```
        fprintf(stderr, "child process B not created\n");
        exit(3);
    } else if (child_b == 0) {
        /* ------- CHILD B ------- */
        // Всё то же самое и для второго потомка родительского процесса

        printf("Run child B\n");
        int size;
        while(read(fd_b[READ], &size, sizeof(int)) > 0) {
            char buffer_b[255];
            read(fd_b[READ], &buffer_b, sizeof(char) * size);
            dup2(file_b, STDOUT_FILENO);
            reverse(buffer_b);
            printf("%s", buffer_b);
        }
    } else {
        /* ------- PARENT ------- */

        char buffer[255];
        memset(buffer, 0, sizeof(char) * 255); // очищаем строку
        close(fd_a[READ]); // закрываем дескрипторы пайпа для чтения, так как родитель ничего не читает
        close(fd_b[READ]);
        while(fgets(buffer, sizeof(char) * 255, stdin) != NULL) { // считываем строку
            if (strlen(buffer) > 10) { // если ее длина больше 10 - отправляем в потомка B
                int size = strlen(buffer) + 1; // размер строки + терминальный символ (как мне объяснил индус)
                write(fd_b[WRITE], &size, sizeof(int)); // отправляем в пайп размер строки
                write(fd_b[WRITE], &buffer, sizeof(char) * size); // отправляем в пайп саму строку
            } else { // длинна строки меньше или равна 10 - всё то же самое, что и с потомком B, только A
                int size = strlen(buffer) + 1;
                write(fd_a[WRITE], &size, sizeof(int));
```

```
                write(fd_a[WRITE], &buffer, sizeof(char) * size);
            }
        }
        close(fd_a[WRITE]); // закрываем пайпы для записи
        close(fd_b[WRITE]);
        close(file_a); // закрываем файлы
        close(file_b);
    }
  }
}
```

Демонстрация работы программы

tonsoleils@LAPTOP-31GE9NQM:/mnt/d/OS/lab1$ ./main

a.txt

b.txt

Run child A

Run child B

abc

def

super long string

another super long string

short

string

tonsoleils@LAPTOP-31GE9NQM:/mnt/d/OS/lab1$ cat a.txt

cba

fed

trohs

gnirts

tonsoleils@LAPTOP-31GE9NQM:/mnt/d/OS/lab1$ cat b.txt

gnirts gnol repus

gnirts gnol repus rehtona

Выводы

Проделав данную работу я научилась управлять процессами в ОС, пользоваться системными вызовами, обеспечивать коммуникацию между процессами. Также я научилась подменять файловые дескрипторы, что в некоторых ситуациях бывает весьма полезно, как, например, в моей лабораторной работе, когда дочерним процессам нужно вывести данные в файл, который создал родительский процесс. Для этого достаточно подменить дескриптор стандартного вывода на дескриптор файла, и, при попытке вывести информацию в стандартный вывод, данные будут выведены в файл.