ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)"

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа № 4 по курсу

по курсу «Операционные системы»

Группа: М8о-207Б-18

Студент:

Тояков Артем Олегович

Преподаватель:

Миронов Евгений Сергеевич

Оценка:

Дата:

Москва, 2020

Оглавление:

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.
Вариант № 15:

Родительский процесс считывает стандартной входной поток, отдает его дочернему процессу, который удаляет "задвоенные" пробелы и выводит его в файл (имя файла также передается от родительского процесса).

СТРУКТУРА ПРОГРАММЫ

В данной работе в моя программа состоит из трёх файлов:

1. main.c
2. child.c
3. Makefile

ОПИСАНИЕ ПРОГРАММЫ

Моя программа работает следующим образом. Вначале запускается файл main.c, в котором из командной строки считывается имя файла, в который в дальнейшем будет записан результат. Затем из стандартного потока ввода мы получаем наш текст и записываем его в массив. Далее с помощью системного вызова execl мы заменяем образ родительского процесса образом дочернего child. В параметрах этой функции мы передаём полное имя файла, который необходимо исполнить и 2 указателя на строки (входные данные и имя файла). Признак окончания списка параметров – NULL.

В дочернем процессае мы принимаем данные из родительского и с помощью простого алгоритма затираем двойные пробелы. Затем исполняется несколько системных вызовов. Первый из них это open. Она открывает file с указанными параметрами. После выполняется вызов lseek, который устанавливает указатель положения в файле, указанном дексриптором, в положение, указанное аргументами offset и origin (в нашем случае это SEEK_SET, то есть начало файла). Затем выполняется вызов mmap, который отражает в память count символов из файла, описанного дескриптором, с определёнными параметрами (например MAP_SHARED разделяет использование этого отражения с другими процессами, отражающими тот же объект. Запись в эту область памяти эквивалентна записи в файл). Mmap возвращает настоящее местоположение отражённых данных. Последней же выполняется функция memcpy(), которая копирует count символов из массива, на который указывает out, в массив, на который указывает dst.

ЛИСТИНГ ПРОГРАММЫ

//main.c

```
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
```

```
#include <sys/mman.h>
#include <stdbool.h>
#include <fcntl.h>
#include <string.h>


int main(int argc, char* argv[]) {
        char sym;
    int cap = 4;
        char* in = (char*) malloc(sizeof(char) * cap);
    char* out = argv[1];
    if (argc != 2) {
        perror("Use like: ./a.out <tofile>");
        exit(1);
    }
    int i = 0;
        while ((sym = getchar()) != EOF) {
        if (i > cap) {
            cap = cap * 4 / 3;
            in = (char*)realloc(in, sizeof(char) * cap);
        }
                in[i] = sym;
        i++;
        }
    in = (char*) realloc(in, sizeof(char) * i);
        execl("child", in, out, NULL);
        return 0;
}
```

//child.c

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

int main(int argc, char* argv[]) {
    int fdout;
    char c;
    void *dst;
    int cap = 4;
        char* out = (char*) malloc(sizeof(char) * cap);
    char* in = argv[0];
    char* output = argv[1];
    int count = 0;
        bool lspace = false;
    for (int i = 0; (c = in[i]) != '\0'; ++i) {
        //printf("%d\n", i);
        if (c == ' ' && lspace) continue;
        if (i > cap) {
            cap = cap * 4 / 3;
            out = (char*)realloc(out, sizeof(char) * cap);
        }
        out[count] = c;
        count++;
        if (c == ' ') {
            lspace = true;
        } else {
                lspace = false;
```

```
        }
    }
    out[count] = '\0';
    if ((fdout = open(output, O_CREAT | O_RDWR | O_TRUNC, 0666)) < 0 ){
        perror("Невозможно создать файл для записи");
        exit(1);
    }
    /*установить размер выходного файла*/
    if (lseek(fdout, count, SEEK_SET) == -1 ){
        perror("Ошибка вызова функции lseek");
        exit(1);
    }
    if (write(fdout, "", 1) != 1 ){
        perror("Ошибка вызова функции write");
        exit(1);
    }
    if ((dst = mmap(0, count, PROT_READ | PROT_WRITE, MAP_SHARED, fdout, 0)) ==
MAP_FAILED ){
        perror("Ошибка вызова функции mmap для выходного файла");
        exit(1);
    }
    memcpy(dst, out, count);
}
```

//Makefile

```
FLAGS=-pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm
COMPILLER=gcc

all: start child

start: main.o
        $(COMPILLER) $(FLAGS) -o lab4 main.o

main.o: main.c
        $(COMPILLER) -c $(FLAGS) main.c

child: child.c
        $(COMPILLER) $(FLAGS) -o child child.c

clear:
        -rm -f *.o *.gch child lab
```

РЕЗУЛЬТАТ РАБОТЫ

```
artoy@artoy:~/Desktop/Labs/3 sem/OS/Lab4$ make
gcc -c -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm main.c
gcc -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -o lab4 main.o
gcc -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -lm -o child child.c
artoy@artoy:~/Desktop/Labs/3 sem/OS/Lab4$ ./lab4 file_name
I want to delete
all spaces
Its interesting) .

file_name:

I want to delete
all spaces
Its interesting) .
```

ВЫВОД

Отображение файла в память (на память) - это способ работы с файлами в некоторых операционных системах, при котором всему файлу или некоторой непрерывной его части ставится в соответствие определённый участок памяти (диапазон адресов оперативной памяти). При этом чтение данных из этих адресов фактически приводит к чтению данных из отображенного файла, а запись данных по этим адресам приводит к записи этих данных в файл. Также данный файл может использоваться несколькими процессами, для записи или чтения. Ключевым является системный вызов mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset) с помощью которого можно отобразить файл с необходимыми параметрами.