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: Рылов Александр Дмитриевич
Группа: М8О-207Б-21
Вариант: 14
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/Brokiloene/os/tree/main

Постановка задачи

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

Задание
Родительский процесс создает два дочерних процесса. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их pipe1. Процесс child1 и child2 производят пересылает результат работу над в строками. Child2 своей работы родительскому процессу. Родительский процесс полученный результат выводит стандартный поток вывода. Правило фильтрации: 14 вариант Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

Общие сведения о программе

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

1. shm_open - создаёт/открывает объекты общей памяти POSIX.
2. sem_open - инициализирует и открывает именованный семафор.
3. ftruncate - обрезает файл до заданного размера.
4. mmap, munmap - отображает файлы или устройства в памяти, или удаляет их отображение.
5. sem_close - закрывает именованный семафор.
6. execlp - запуск файла на исполнение.
7. sem_wait - блокирует семафор.
8. sem_post - разблокирует семафор.

Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить работу с отображением файла в память(mmap и munmap).
2. Изучить работу с процессами(fork).
3. Создать 2 дочерних и 1 родительский процесс.
4. В каждом процессе отобразить файл в память, преобразовать в соответствии с вариантом и снять отображение(mmap, munmap).

Исходный код

main.c

```c
// 14 вариант Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

#include <stdio.h>    // STDOUT_FILENO & size_t & printf & getline
#include <sys/mman.h> // mmap & shm_open
#include <sys/stat.h> // S_xxxx
#include <fcntl.h>    // O_xxxx
#include <unistd.h>   // ftruncate
#include <string.h>   // memcpy
#include <semaphore.h>// sem_t
#include <sys/wait.h> // wait

#include "lab4_utils.h"

void print(char *str, size_t size)
{
    for (size_t i = 0; i < size; ++i) {
        printf("%c", str[i]);
    }
    printf("\n");
}

int main(int argc, char const *argv[])
{
    char *shmpath = "/shmlab4";
    int fd = shm_open(shmpath, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    check_err(fd, "shm_open error");

    check_err(ftruncate(fd, sizeof(Shared_str)), "ftruncate error");
    Shared_str *shr_str = mmap(NULL, sizeof(*shr_str), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (shr_str == MAP_FAILED) {check_err(-1, "mmap error")};

    check_err(sem_init(&shr_str->p_to_c1, 1, 0), "sem_init error");
    check_err(sem_init(&shr_str->c1_to_c2, 1, 0), "sem_init error");
    check_err(sem_init(&shr_str->c2_to_p, 1, 0), "sem_init error");

    int pid = fork();
    check_err(pid, "fork error");
    if (pid == 0) { // child1
        check_err(execlp("./child", "child", (char *)NULL), "execlp child1 error");
    }

    int pid2 = fork();
    check_err(pid, "fork error");
    if (pid2 == 0) { // child2
        check_err(execlp("./child2", "child2", (char *)NULL), "execlp child2 error");
    }

```

```
    }

    int pid2 = fork();
    check_err(pid, "fork error");
    if (pid2 == 0) { // child2
        check_err(execlp("./child2", "child2", (char *)NULL), "execlp child2 error");
    }

    size_t len = 16;
    int size = 0;
    char* text = NULL;
    size = getline(&text, &len, stdin);
    check_err(size, "getline error");
    text[size - 1] = '\0';
    while(text[0] != '0') {
        if (size > MAX_SIZE) {
            printf("size of string shall be <= 1024\n");
            break;
        }
        shr_str->size = size;
        memcpy(&shr_str->str, text, size);
        check_err(sem_post(&shr_str->p_to_c1), "sem_post error");

        check_err(sem_wait(&shr_str->c2_to_p), "sem_wait error");
        print(shr_str->str, shr_str->size);

        size = getline(&text, &len, stdin);
        check_err(size, "getline error");
        text[size - 1] = '\0';
    }
    shr_str->size = STOP;
    check_err(sem_post(&shr_str->p_to_c1), "sem_post error");
    free(text);

    check_err(sem_close(&shr_str->p_to_c1), "sem_close error");
    check_err(sem_close(&shr_str->c1_to_c2), "sem_close error");
    check_err(sem_close(&shr_str->c2_to_p), "sem_close error");
    check_err(munmap(shr_str, sizeof(*shr_str)), "munmap error");
    shm_unlink(shmpath);
    wait(NULL);
    wait(NULL);

    return 0;
}

```

child.c

```
// child1

#include <stdio.h>    // STDOUT_FILENO & size_t
#include <sys/mman.h> // mmap & shm_open
#include <fcntl.h>    // O_xxxx
#include <ctype.h>    // tolower
#include <semaphore.h>// sem_t

#include "lab4_utils.h"

int main(int argc, char const *argv[])
{
    char *shmpath = "/shmlab4";
    int fd = shm_open(shmpath, O_RDWR, 0);
    check_err(fd, "shm_open error");

    Shared_str *shr_str = mmap(NULL, sizeof(*shr_str), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (shr_str == MAP_FAILED) {check_err(-1, "mmap error")};

    while(1) {
        check_err(sem_wait(&shr_str->p_to_c1), "sem_wait error");
        if (shr_str->size == STOP) {
            check_err(sem_post(&shr_str->c1_to_c2), "sem_post error");
            break;
        }
        for (int i = 0; i < shr_str->size; ++i) {
            shr_str->str[i] = tolower(shr_str->str[i]);
        }
        check_err(sem_post(&shr_str->c1_to_c2), "sem_post error");
    }
    shm_unlink(shmpath);

    return 0;
}

```

child2.c

```c
// child2

#include <sys/mman.h> // mmap & shm_open
#include <fcntl.h>    // O_xxxx
#include <semaphore.h>// sem_t

#include "lab4_utils.h"

int main(int argc, char const *argv[])
{
    char *shmpath = "/shmlab4";
    int fd = shm_open(shmpath, O_RDWR, 0);
    check_err(fd, "shm_open error");

    Shared_str *shr_str = mmap(NULL, sizeof(*shr_str), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (shr_str == MAP_FAILED) {check_err(-1, "mmap error")};

    while(1) {
        check_err(sem_wait(&shr_str->c1_to_c2), "sem_wait error");
        if (shr_str->size == STOP) {
            break;
        }
        for (int i = 0, end = shr_str->size - 1; i < end; i++) {
            if (shr_str->str[i] == ' ' && shr_str->str[i] == shr_str->str[i + 1]) {
                for (int j = i, end = shr_str->size - 1; j < end; j++) {
                    shr_str->str[j] = shr_str->str[j + 1];
                }
            shr_str->size--;
            i--;
            }
        }
        check_err(sem_post(&shr_str->c2_to_p), "sem_post error");
    }
    shm_unlink(shmpath);

    return 0;
}

```

lab4_utils.h

```c
#ifndef _LAB4_UTILS_
#define _LAB4_UTILS_

#include <stdio.h>    // perror
#include <stdlib.h>   // EXIT_FAILURE
#include <semaphore.h>// sem_t

#define STOP -1
#define MAX_SIZE 1024

#define check_err(foo, msg) do { int __res = foo; \
if (__res == -1) { perror(msg); exit(EXIT_FAILURE); } \
} while(0);

typedef struct {
    sem_t p_to_c1;
    sem_t c1_to_c2;
    sem_t c2_to_p;
    int size;
    char str[MAX_SIZE];
} Shared_str;

#endif

```

Демонстрация работы программы

```
user@brokiloene:~/Desktop/all/os/lab_4/src/build$ ./main
HELLO
hello
lkasdfjlk KKKK qq Q
lkasdfjlk kkkk qq q
0
```

Выводы

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.