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

Лабораторная работа №4 по курсу

«Операционные системы»

Тема работы

Студент: Павлов Иван Дмитриевич
Группа: М8О-207Б-21
Вариант: 11
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/Pavloffff/MAI_OS/tree/main/lab4

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 3:

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Вариант 11:

Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «_».

Общие сведения о программе

Программа представлена четырьмя файлами: main.cpp, child1.cpp, child2.cpp, labtools.h

Общий метод и алгоритм решения

Опишу новые для себя системные вызовы:

shm_open

<sys/stat.h> + <fcntl.h>

Создает и открывает объект общей памяти POSIX, который эффективен для работы с несвязанными процессами, которые хотят использовать единый объект памяти. С флагом O_RDWR - открывает объект на чтение и запись. O_CREAT - создает объект, если он не существует. Аргумент mode означает права доступа, я их установил в переменной accessPerm, установив 644. В случае ошибки возвращает -1.

sem_open

<semaphore.h> + <fcntl.h>

Создает новый семафор POSIX, или открывает уже существующий. Семафор - число, не меньше 0. Семафоры можно уменьшать (sem_wait) и увеличивать (sem_post). При этом если применить операцию sem_wait к семафору, когда его значение 0, то sem_wait блокирует работу, пока значение не увеличится (для чего они и создавались). Именованные семафоры, также, как и объекты общей памяти, лежат на диске в директории /dev/shm. Если устанавлен аттрибут O_CREAT и семафор при этом существует, то аттрибуты значения и прав доступа игнорируются.

ftruncate

<unistd.h>

Устанавливает необходимую длину файла в байтах.

fstat

<sys/stat.h> + <sys/types.h>

Содержит информацию о файле, например, размер st_size, и заполняет буфер.

mmap

<sys/mman.h>

Создает отображение файла на память в пространстве процесса.

Алгоритм решения:

Считываем полностью текст из строк до EOF из потока ввода, затем открываем объект общей памяти, устанавливаем ему размер текста и отображаем на него текст.

Далее создаем семафор, увеличиваем / уменьшаем его значение до 2.

Вызываем fork(). Родительский процесс в цикле блокирует семафор, и ждет выполнения дочерних процессов.

Так как родительский и дочерние процессы представлены разными файлами, то придется заново закрывать и открывать объект общей памяти и семафор.

После вызова execl, дочерний процесс 1 открывает объект общей памяти, отображает его на буфер, устанавливает значение семафора на 1, выполняет свое преобразование, вызывает fork и execl.

Дочерний процесс 2 делает те же действия, уменьшая значение семафора до 0 (при этом родительский процесс заблокировался), затем ждет (так как одна итерация цикла while(1) может выполниться быстрее, чем дочерний процесс 2), и разблокирует родительский процесс, который выводит результат в консоль.

Исходный код

labtools.h

```
#ifndef __LABTOOLS_H_
#define __LABTOOLS_H_

#include <string>

std::string backFile = "main.back";
std::string semFile = "main.semaphore";
int accessPerm = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;

#define CHECK_ERROR(expr, err, message) \
    do { \
        auto __result = (expr); \
```

```
        if (__result == err) { \
            fprintf(stderr, "Error: %s\n", message); \
            fprintf(stderr, "errno = %s, file %s, line %d\n", strerror(errno), \
                    __FILE__, __LINE__); \
            exit(-1); \
        } \
    } while (0)

std::string myInput()
{
    size_t sz = 0;
    char *_str = (char *) calloc(1, sizeof(char));
    char c;
    while ((c = getchar()) != EOF) {
        _str[sz] = c;
        _str = (char *) realloc(_str, (++sz + 1) * sizeof(char));
    }
    _str[sz++] = '\0';
    std::string str = _str;
    free(_str);
    return str;
}

#endif
```

main.cpp

```
#include <iostream>

#include <string>

#include <algorithm>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <semaphore.h>

#include <fcntl.h>

#include <errno.h>
```

```
#include <string.h>
#include "labtools.h"


int main(int argc, char const *argv[])
{
    std::string in = myInput();
    int fd = shm_open(backFile.c_str(), O_RDWR | O_CREAT, accessPerm);
    CHECK_ERROR(fd, -1, "shm_open");
    sem_t *semaphore = sem_open(semFile.c_str(), O_CREAT, accessPerm);
    CHECK_ERROR(semaphore, SEM_FAILED, "sem_open");
    int state = 0, mapSize = in.size() + 1;
    ftruncate(fd, (int) mapSize);
    char *mapped = (char *) mmap(NULL,
                                 mapSize,
                                 PROT_READ | PROT_WRITE,
                                 MAP_SHARED,
                                 fd,
                                 0);
    CHECK_ERROR(mapped, MAP_FAILED, "mmap");
    memset(mapped, '\0', mapSize);
    sprintf(mapped, "%s", in.c_str());
    int semErrCheck = sem_getvalue(semaphore, &state);
    CHECK_ERROR(semErrCheck, -1, "sem_getvalue");
    while (state++ < 2) {
        sem_post(semaphore);
    }
    while (state-- > 3) {
        sem_wait(semaphore);
    }
```

```
    pid_t child1Id = fork();
    CHECK_ERROR(child1Id, -1, "fork");
    if (child1Id == 0) {
        munmap(mapped, mapSize);
        close(fd);
        sem_close(semaphore);
        int execlErrCheck = execl("child1", "child1", NULL);
        CHECK_ERROR(execlErrCheck, 0, "execl");
    }
    while (1) {
        int semErrCheck = sem_getvalue(semaphore, &state);
        CHECK_ERROR(semErrCheck, -1, "sem_getvalue");
        if (state == 0) {
            int semWaitErrCheck = sem_wait(semaphore);
            CHECK_ERROR(semWaitErrCheck, -1, "sem_wait");
            std::cout << mapped;
            return 0;
        }
    }
    return 0;
}
```

child1.cpp

```
#include <iostream>
#include <string>
#include <algorithm>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <semaphore.h>
```

```
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include "labtools.h"

int main(int argc, char const *argv[])
{
    int fd = shm_open(backFile.c_str(), O_RDWR, accessPerm);
    CHECK_ERROR(fd, -1, "shm_open");
    struct stat statBuf;
    fstat(fd, &statBuf);
    const size_t mapSize = statBuf.st_size;
    char *mapped = (char *)mmap(NULL,
                                mapSize,
                                PROT_READ | PROT_WRITE,
                                MAP_SHARED,
                                fd,
                                0);
    CHECK_ERROR(mapped, MAP_FAILED, "mmap");
    int state = 0;
    sem_t *semaphore = sem_open(semFile.c_str(), O_CREAT, accessPerm, 2);
    CHECK_ERROR(semaphore, SEM_FAILED, "sem_open");
    int semWaitErrCheck = sem_wait(semaphore);
    CHECK_ERROR(semWaitErrCheck, -1, "sem_wait");
    std::string str;
    str = mapped;
    std::transform(str.begin(),
                   str.end(),
                   str.begin(),
```

```
            [](unsigned char c) { return std::toupper(c); });
    memset(mapped, '\0', mapSize);
    sprintf(mapped, "%s", str.c_str());
    pid_t child2Id = fork();
    CHECK_ERROR(child2Id, -1, "fork");
    if (child2Id == 0) {
        munmap(mapped, mapSize);
        close(fd);
        sem_close(semaphore);
        int execlErrCheck = execl("child2", "child2", NULL);
        CHECK_ERROR(execlErrCheck, 0, "execl");
    }
    return 0;
}
```

child2.cpp

```
#include <iostream>
#include <string>
#include <algorithm>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include "labtools.h"

int main(int argc, char const *argv[])
{
```

```
int fd = shm_open(backFile.c_str(), O_RDWR, accessPerm);
CHECK_ERROR(fd, -1, "shm_open");
struct stat statBuf;
fstat(fd, &statBuf);
const size_t mapSize = statBuf.st_size;
char *mapped = (char *) mmap(NULL,
                             mapSize,
                             PROT_READ | PROT_WRITE,
                             MAP_SHARED,
                             fd,
                             0);
CHECK_ERROR(mapped, MAP_FAILED, "mmap");
sem_t *semaphore = sem_open(semFile.c_str(), O_CREAT, accessPerm, 2);
CHECK_ERROR(semaphore, SEM_FAILED, "sem_open");
int semWaitErrCheck = sem_wait(semaphore);
CHECK_ERROR(semWaitErrCheck, -1, "sem_wait");
std::string str;
str = mapped;
std::transform(str.begin(),
               str.end(),
               str.begin(),
               [](unsigned char c) { return (c == ' ') ? '_' : c; });
int mSize = str.size() + 1;
ftruncate(fd, (int) mSize);
memset(mapped, '\0', mSize);
sprintf(mapped, "%s", str.c_str());
close(fd);
usleep(00150000);
sem_post(semaphore);
```

```
    sem_close(semaphore);
    return 0;
}
```

Демонстрация работы программы

Ввод в консоль:

ggame@ggame:~/OS/lab4/t1/build$ cat test

#ifndef mashina turinga h

#define mashina turinga h

#endif

ggame@ggame:~/OS/lab4/t1/build$ ./main < test

#IFNDEF__MASHINA_TURINGA_H__

#DEFINE__MASHINA_TURINGA_H__

#ENDIF

Выводы

Проделав лабораторную работу, я приобрёл практические навыки, необходимые для работы с отображаемой памятью и семафорами.