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**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**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 (причем у него может быть любое значение, но в основном 0, который программа вернула при прошлом использовании), 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.