Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**РАБОТА С ОБЩЕЙ ПАМЯТЬЮ**

Студент: Баталин Дмитрий Андреевич

Группа: М8О–212Б–20

Вариант: 7

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Приобретение практических навыков в:

1. Освоение принципов работы с файловыми системами
2. Обеспечение обмена данных между процессами посредством технологии «File mapping»

## Задание

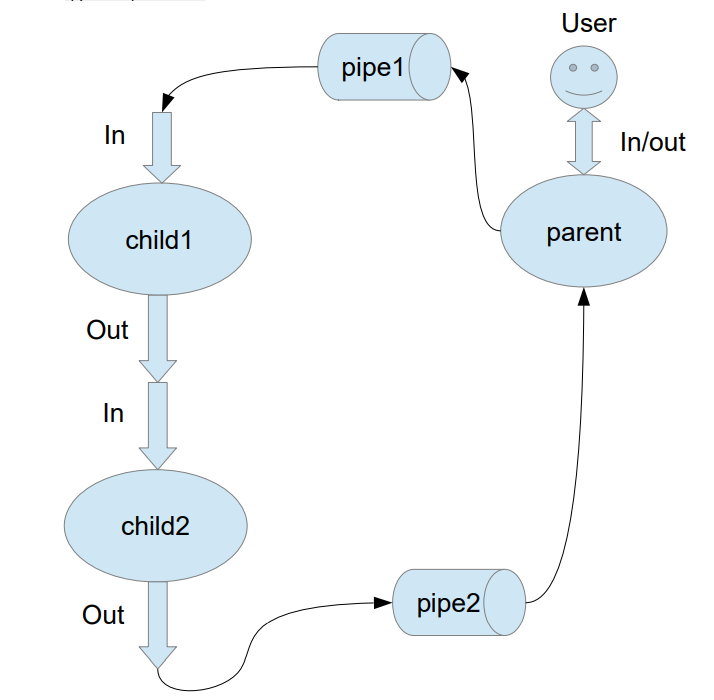
Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Формулировка задания по варианту:

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. 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 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.

14 вариант) Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.



**Общие сведения о программе**

Программа компилируется из файла main.c, child1.c и child2.c с помощью утилиты CMake. Внутри main.c 3 функции:

1. int main(), создающая дочерние процессы, семафоры и т. д.
2. int create\_process() - создаёт процесс с помощью форк и возвращает айди процесса.
3. void read\_input(char \*buffer, size\_t bufferSize) – считывает входные данные с помощью системных вызовов.

Внутри child1 2 функции:

1. int main(int argc, char\* argv[]) – принимает аргументы, передающиеся ей через execl и обрабатывает строку.
2. void to\_lower(char \*str) – переводит буквы в нижний регистр.

Внутри child2 2 функции:

1. int main(int argc, char\* argv[]) – выполняет примерно те же действия, что и в child1
2. void remove\_spaces(char\* str) – удаляет задвоенные пробелы

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Более подробно изучить принципы работы семафоров. Разобраться в средствах работы с общей памятью (mmap, shm и т.д.)
2. Взять функции to\_lower и remove\_spaces из лабораторной работы №1, так как они никак не поменяются.
3. Создать объект разделяемой памяти с помощью shm\_open, дабы именовать область виртуальной памяти, установить нужный нам размер этой памяти с помощью ftruncate и получить доступ к разделяемой памяти с помощью mmap.
4. Создать три семафора, которые будут ограничивать работу процессов main, child1 и child2
5. Создать дочерние процессы при помощи execl и передать в них нужные имена семафоров и имя объекта разделяемой памяти
6. Обработать внутри дочерних процессов строки и на ряду с этим увеличивать и ожидать сброса значения семафоров.
7. Вывести итоговую строку

**Основные файлы программы**

**main.c:**

#include <sys/types.h>

#include <sys/mman.h>

#include <stdio.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdlib.h> // pid\_t declaration is here

#include <unistd.h> // fork() and getpid() declarations are here

#include <pthread.h>

#include <string.h>

#include "semaphore.h"

#include "sys/wait.h"

#define STRING\_SIZE 100

**int** **create\_process**();

**void** **read\_input**(**char** \*buffer, **size\_t** bufferSize);

**int** **main**() {

**const** **char**\* mmapped\_file\_name;

mmapped\_file\_name = "mmaped\_file";

**const** **char**\* semaphores\_names[**3**];

semaphores\_names[**0**] = "/semaphoreOne"; // семафор для child1

semaphores\_names[**1**] = "/semaphoreTwo"; // семафор для child2

semaphores\_names[**2**] = "/semaphoreThree"; // семафор для parent

shm\_unlink(mmapped\_file\_name);

sem\_unlink(semaphores\_names[**0**]);

sem\_unlink(semaphores\_names[**1**]);

sem\_unlink(semaphores\_names[**2**]);

**int** mmapped\_file\_descriptor;

**char**\* mmapped\_file\_pointer;

mmapped\_file\_descriptor = shm\_open(mmapped\_file\_name, O\_RDWR | O\_CREAT, **0777**);

ftruncate(mmapped\_file\_descriptor, STRING\_SIZE);

mmapped\_file\_pointer = mmap(NULL, STRING\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, mmapped\_file\_descriptor, **0**);

**sem\_t**\* semaphores[**3**];

semaphores[**0**] = sem\_open(semaphores\_names[**0**], O\_CREAT, **0777**, **0**);

semaphores[**1**] = sem\_open(semaphores\_names[**1**], O\_CREAT, **0777**, **0**);

semaphores[**2**] = sem\_open(semaphores\_names[**2**], O\_CREAT, **0777**, **0**);

**pid\_t** pid = create\_process();

**if** (pid == **0**) // child1

{

execl("child1", "", mmapped\_file\_name, semaphores\_names[**0**], semaphores\_names[**1**], NULL);

perror("exec");

exit(-**3**);

}

**else**

{

**pid\_t** pid = create\_process();

**if** (pid == **0**) // child2

{

execl("child2", "", mmapped\_file\_name, semaphores\_names[**1**], semaphores\_names[**2**], NULL);

perror("exec");

exit(-**3**);

}

**else** // parent

{

**char** string[STRING\_SIZE];

read\_input(string, STRING\_SIZE); // читаем входные данные в string

**for** (**int** i = **0**; i < strlen(string); ++i) {

mmapped\_file\_pointer[i] = string[i];

}

write(STDOUT\_FILENO, "Строка считана и выглядит так: ", **57** \* **sizeof**(**char**));

write(STDOUT\_FILENO, mmapped\_file\_pointer, **100** \* **sizeof**(**char**));

sem\_post(semaphores[**0**]);

sem\_wait(semaphores[**2**]);

wait(NULL);

munmap(mmapped\_file\_pointer, **0**);

shm\_unlink(mmapped\_file\_name);

**for** (**int** i = **0**; i < **3**; i++) {

sem\_unlink(semaphores\_names[i]);

}

write(STDOUT\_FILENO, mmapped\_file\_pointer, STRING\_SIZE \* **sizeof**(**char**));

}

}

**return** **0**;

}

**int** **create\_process**() {

**pid\_t** pid = fork();

**if** (-**1** == pid)

{

perror("Error while fork");

exit(-**2**);

}

**return** pid;

}

**void** **read\_input**(**char** \*buffer, **size\_t** bufferSize) {

**ssize\_t** bytesRead;

bytesRead = read(STDIN\_FILENO, buffer, bufferSize - **1**);

**if** (bytesRead == -**1**) {

perror("Read error");

exit(-**1**);

}

**for** (**int** i = bytesRead; i < bufferSize; i++) {

buffer[i] = **0**;

}

}

**child1.c:**

#include <sys/types.h>

#include <sys/mman.h>

#include <stdio.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdlib.h> // pid\_t declaration is here

#include <unistd.h> // fork() and getpid() declarations are here

#include <pthread.h>

#include <string.h>

#include "semaphore.h"

#define STRING\_SIZE 100

**void** **to\_lower**(**char**\* str);

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

// write(STDOUT\_FILENO, "1 дочерний процесс запущен!\n", 51 \* sizeof(char));

**char** mmapped\_file\_name[STRING\_SIZE];

strcpy(mmapped\_file\_name, argv[**1**]);

// write(STDOUT\_FILENO, "Название mmap файла: ", 35 \* sizeof(char));

// write(STDOUT\_FILENO, mmapped\_file\_name, 100 \* sizeof(char));

**char** semaphore\_child1\_name[STRING\_SIZE];

**char** semaphore\_child2\_name[STRING\_SIZE];

strcpy(semaphore\_child1\_name, argv[**2**]); // семафор для child1

strcpy(semaphore\_child2\_name, argv[**3**]); // семафор для child2

**int** mmapped\_file\_descriptor = shm\_open(mmapped\_file\_name, O\_RDWR, **0777**);

ftruncate(mmapped\_file\_descriptor, STRING\_SIZE);

**char**\* mmapped\_file\_pointer = mmap(NULL, STRING\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, mmapped\_file\_descriptor, **0**);

**sem\_t**\* semaphore\_child1 = sem\_open(semaphore\_child1\_name, **0**);

**sem\_t**\* semaphore\_child2 = sem\_open(semaphore\_child2\_name, **0**);

**char** string[STRING\_SIZE];

// write(STDOUT\_FILENO, "Я около семафора в 1 процессе!\n", 55 \* sizeof(char));

sem\_wait(semaphore\_child1);

to\_lower(mmapped\_file\_pointer);

munmap(mmapped\_file\_pointer, **0**);

sem\_post(semaphore\_child2);

sem\_close(semaphore\_child1);

sem\_close(semaphore\_child2);

}

**void** **to\_lower**(**char** \*str) {

**while** (\*str) {

**if** ((\*str >= **65**) && (\*str <= **90**))

\*str = \*str + **32**;

str += **1**;

}

}

**child2.c:**

#include <sys/types.h>

#include <sys/mman.h>

#include <stdio.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdlib.h> // pid\_t declaration is here

#include <unistd.h> // fork() and getpid() declarations are here

#include <pthread.h>

#include <string.h>

#include "semaphore.h"

#define STRING\_SIZE 100

**void** **remove\_spaces**(**char**\* str);

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

// write(STDOUT\_FILENO, "2 дочерний процесс запущен!\n", 51 \* sizeof(char));

**char** mmapped\_file\_name[STRING\_SIZE];

strcpy(mmapped\_file\_name, argv[**1**]);

**char** semaphore\_child2\_name[STRING\_SIZE];

**char** semaphore\_parent\_name[STRING\_SIZE];

strcpy(semaphore\_child2\_name, argv[**2**]); // семафор для child1

strcpy(semaphore\_parent\_name, argv[**3**]); // семафор для child2

**int** mmapped\_file\_descriptor = shm\_open(mmapped\_file\_name, O\_RDWR, **0777**);

ftruncate(mmapped\_file\_descriptor, STRING\_SIZE);

**char**\* mmapped\_file\_pointer = mmap(NULL, STRING\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, mmapped\_file\_descriptor, **0**);

**sem\_t**\* semaphore\_child2 = sem\_open(semaphore\_child2\_name, **0**);

**sem\_t**\* semaphore\_parent = sem\_open(semaphore\_parent\_name, **0**);

**char** string[STRING\_SIZE];

// write(STDOUT\_FILENO, "Я около семафора в 2 процессе!\n", 55 \* sizeof(char));

sem\_wait(semaphore\_child2);

remove\_spaces(mmapped\_file\_pointer);

munmap(mmapped\_file\_pointer, **0**);

sem\_post(semaphore\_parent);

sem\_close(semaphore\_child2);

sem\_close(semaphore\_parent);

}

**void** **remove\_spaces**(**char**\* str) {

**int** len = STRING\_SIZE;

**if** (len <= **1**) {

**return**;

}

**char**\* dest = str;

\*dest++ = str[**0**];

**int** i = **1**;

**for** (; str[i] != **0** && i < len ; i++) {

**if** (!(str[i-**1**] == ' ' && str[i] == ' '))

{

\*dest = str[i];

dest++;

}

}

**for** (; i < len; i++) {

\*dest = **0**;

dest++;

}

}

**Пример работы**

moses@moses:~/cs/os/build/lab3/src$ ./lab3 < ../../../lab3/test/test01.txt

Строка считана и выглядит так: BIG STRING WITH SPACES

big string with spacesmoses@moses:~/cs/os/build/lab3/src$ ./lab3 < ../../../lab3/test/test02.txt

Строка считана и выглядит так:

moses@moses:~/cs/os/build/lab3/src$ ./lab3 < ../../../lab3/test/test03.txt

Строка считана и выглядит так:

moses@moses:~/cs/os/build/lab3/src$ ./lab3 < ../../../lab3/test/test04.txt

Строка считана и выглядит так: SoMe TeXt SoMe TeXt

SoMe TeXt in TWO STRINGS

some text some text

some text in two stringsmoses@moses:~/cs/os/build/lab3/src$

**Вывод**

В течение выполнения данной лабораторной работы я научился работать с семафорами на практике, я очень хорошо понял, как работают механизмы разделяемой памяти. Такой системный вызов как mmap позволяет выделять часть виртуальной памяти под свои нужды. Используя этот вызов в связке с функцией shm\_open можно выделить область памяти и дать ей имя, благодаря чему можно будет обращаться к этой области из разных процессов. Также, я узнал, что все семафоры и такие объекты разделяемой памяти лежат в unix в папке /dev/shm. Используя эту папку, можно посмотреть какие файлы в данный момент существуют на оперативной памяти, что очень круто, так как эти файлы лежат не на диске, а в виртуальной памяти. Благодаря этой папке, например, можно узнать, что не все объекты были удалены после выполнения программы с помощью unlink.