МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 "Компьютерные науки и прикладная математика"

Кафедра 806 "Вычислительная математика и программирование"

Лабораторная работа №3

По курсу «Операционные системы»

Студент: Кириллова Е.К.

Группа: М8О-203Б-23

Вариант: 12

Преподаватель: Миронов Е. С.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Содержание**

1. Репозиторий

2. Постановка задачи

3. Общие сведения о программе

4. Общий метод и алгоритм решения

5. Исходный код

6. Выводы

**Репозиторий**

https://github.com/ElenaKirillova05/osLabs

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

Управление процессами в ОС

Обеспечение обмена данных между процессами посредством каналов

**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 каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами.Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

12 вариант) Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы

**Общие сведения о программе**

Разработанная система реализует межпроцессное взаимодействие с использованием общей памяти и сигналов для координации процессов программа состоит из трех компонентов: родительского процесса и двух дочерних процессов первый дочерний процесс преобразует все буквы строки в заглавные второй дочерний процесс удаляет лишние пробелы между словами данные передаются через общую память синхронизация осуществляется с использованием сигналов SIGUSR1 и SIGUSR2 родительский процесс управляет вводом и выводом обменивается данными с дочерними процессами и контролирует завершение программы

**Общий метод и алгоритм решения**

Общий метод и алгоритм решения родительский процесс создает объект общей памяти и отображает его в адресное пространство затем порождаются два дочерних процесса с помощью fork первый дочерний процесс ожидает сигнала SIGUSR1 затем преобразует строку в верхний регистр и отправляет сигнал родителю второй дочерний процесс ожидает сигнала SIGUSR2 затем удаляет лишние пробелы и уведомляет родителя о завершении родительский процесс передает введенную строку в общую память активирует первый дочерний процесс ожидает его завершения затем активирует второй дочерний процесс и выводит результат завершение программы осуществляется с помощью передачи сигнала SIGINT обоим дочерним процессам

**Исходный код**

child.c:

```

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#include <signal.h>

#define SHARED\_MEMORY\_FILE "/shared\_memory"

#define MEMORY\_SIZE 1024

volatile sig\_atomic\_t ready = 0;

volatile sig\_atomic\_t exit\_flag = 0;

void signal\_handler(int sig) {

ready = 1;

}

void exit\_handler(int sig) {

exit\_flag = 1;

}

int main() {

signal(SIGUSR1, signal\_handler);

signal(SIGINT, exit\_handler);

int shared\_fd = shm\_open(SHARED\_MEMORY\_FILE, O\_RDWR, 0666);

if (shared\_fd == -1) {

perror("shm\_opeфn failed");

exit(EXIT\_FAILURE);

}

char \*shared\_memory = mmap(NULL, MEMORY\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shared\_fd, 0);

if (shared\_memory == MAP\_FAILED) {

perror("mmap failed");

exit(EXIT\_FAILURE);

}

while (1) {

while (!ready && !exit\_flag) pause();

if (exit\_flag) break;

for (int i = 0; shared\_memory[i]; i++) {

shared\_memory[i] = toupper(shared\_memory[i]);

}

kill(getppid(), SIGUSR1);

ready = 0;

}

munmap(shared\_memory, MEMORY\_SIZE);

close(shared\_fd);

return 0;

}

```

child2.c:

```

#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#include <signal.h>

#define SHARED\_MEMORY\_FILE "/shared\_memory"

#define MEMORY\_SIZE 1024

volatile sig\_atomic\_t ready = 0;

volatile sig\_atomic\_t exit\_flag = 0;

void signal\_handler(int sig) {

ready = 1;

}

void exit\_handler(int sig) {

exit\_flag = 1;

}

int main() {

// Устанавливаем обработчик сигнала SIGUSR2

signal(SIGUSR2, signal\_handler);

// Устанавливаем обработчик сигнала SIGINT для завершения программы

signal(SIGINT, exit\_handler);

// Открываем файл общей памяти

int shared\_fd = shm\_open(SHARED\_MEMORY\_FILE, O\_RDWR, 0666);

if (shared\_fd == -1) {

perror("shm\_open failed");

exit(EXIT\_FAILURE);

}

// Отображаем файл общей памяти в адресное пространство процесса

char \*shared\_memory = mmap(NULL, MEMORY\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shared\_fd, 0);

if (shared\_memory == MAP\_FAILED) {

perror("mmap failed");

exit(EXIT\_FAILURE);

}

while (1) {

// Ждем сигнала готовности

while (!ready && !exit\_flag) pause();

if (exit\_flag) break;

// Указатели для копирования строки

char \*src = shared\_memory;

char \*dest = shared\_memory;

int last\_was\_space = 0;

while (\*src) {

if (\*src == ' ') {

if (!last\_was\_space) {

\*dest++ = ' ';

last\_was\_space = 1;

}

} else {

\*dest++ = \*src;

last\_was\_space = 0;

}

src++;

}

\*dest = '\0';

// Отправляем сигнал родительскому процессу

kill(getppid(), SIGUSR2);

// Сбрасываем флаг готовности

ready = 0;

}

// Освобождаем память

munmap(shared\_memory, MEMORY\_SIZE);

close(shared\_fd);

return 0;

}

```

parent.c:

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/wait.h>

#include <signal.h>

#define SHARED\_MEMORY\_FILE "/shared\_memory"

#define MEMORY\_SIZE 1024

volatile sig\_atomic\_t child1\_finished = 0;

volatile sig\_atomic\_t child2\_finished = 0;

void signal\_handler\_child1(int sig) {

child1\_finished = 1;

}

void signal\_handler\_child2(int sig) {

child2\_finished = 1;

}

int main() {

signal(SIGUSR1, signal\_handler\_child1);

signal(SIGUSR2, signal\_handler\_child2);

int shared\_fd = shm\_open(SHARED\_MEMORY\_FILE, O\_CREAT | O\_RDWR, 0666);

if (shared\_fd == -1) {

perror("shm\_open failed");

exit(EXIT\_FAILURE);

}

ftruncate(shared\_fd, MEMORY\_SIZE);

char \*shared\_memory = mmap(NULL, MEMORY\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shared\_fd, 0);

if (shared\_memory == MAP\_FAILED) {

perror("mmap failed");

exit(EXIT\_FAILURE);

}

pid\_t child1\_pid = fork();

if (child1\_pid == -1) {

perror("fork for child1 failed");

exit(EXIT\_FAILURE);

}

if (child1\_pid == 0) {

execl("./child1", "./child1", NULL);

perror("execl for child1 failed");

exit(EXIT\_FAILURE);

}

pid\_t child2\_pid = fork();

if (child2\_pid == -1) {

perror("fork for child2 failed");

exit(EXIT\_FAILURE);

}

if (child2\_pid == 0) {

execl("./child2", "./child2", NULL);

perror("execl for child2 failed");

exit(EXIT\_FAILURE);

}

while (1) {

printf("Enter the line (or 0 to end the program): ");

if (fgets(shared\_memory, MEMORY\_SIZE, stdin) == NULL) break;

if (shared\_memory[0] == '0' && (shared\_memory[1] == '\n' || shared\_memory[1] == '\0')) break;

child1\_finished = 0;

child2\_finished = 0;

kill(child1\_pid, SIGUSR1);

while (!child1\_finished) pause();

kill(child2\_pid, SIGUSR2);

while (!child2\_finished) pause();

printf("Result: %s\n", shared\_memory);

}

kill(child1\_pid, SIGINT);

kill(child2\_pid, SIGINT);

waitpid(child1\_pid, NULL, 0);

waitpid(child2\_pid, NULL, 0);

munmap(shared\_memory, MEMORY\_SIZE);

shm\_unlink(SHARED\_MEMORY\_FILE);

return 0;

}

```

ТЕСТЫ

tests.cpp:

```

#include <gtest/gtest.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#include <signal.h>

#include <cstring>

#define SHARED\_MEMORY\_FILE "/shared\_memory"

#define MEMORY\_SIZE 1024

class SharedMemoryTest : public ::testing::Test {

protected:

char \*shared\_memory;

int shared\_fd;

void SetUp() override {

shared\_fd = shm\_open(SHARED\_MEMORY\_FILE, O\_CREAT | O\_RDWR, 0666);

if (shared\_fd == -1) {

perror("shm\_open failed");

exit(EXIT\_FAILURE);

}

ftruncate(shared\_fd, MEMORY\_SIZE);

shared\_memory = (char \*)mmap(NULL, MEMORY\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shared\_fd, 0);

if (shared\_memory == MAP\_FAILED) {

perror("mmap failed");

exit(EXIT\_FAILURE);

}

}

void TearDown() override {

munmap(shared\_memory, MEMORY\_SIZE);

close(shared\_fd);

shm\_unlink(SHARED\_MEMORY\_FILE);

}

};

TEST\_F(SharedMemoryTest, MemoryOperationsTest) {

const char\* test\_input = "hello world";

strncpy(shared\_memory, test\_input, MEMORY\_SIZE);

EXPECT\_STREQ(shared\_memory, test\_input);

}

TEST\_F(SharedMemoryTest, SignalHandlingTest) {

const char\* input\_data = " hello world ";

strncpy(shared\_memory, input\_data, MEMORY\_SIZE);

for (int i = 0; shared\_memory[i]; i++) {

shared\_memory[i] = toupper(shared\_memory[i]);

}

EXPECT\_STREQ(shared\_memory, " HELLO WORLD ");

char \*src = shared\_memory;

char \*dest = shared\_memory;

int last\_was\_space = 0;

while (\*src) {

if (\*src == ' ') {

if (!last\_was\_space) {

\*dest++ = ' ';

last\_was\_space = 1;

}

} else {

\*dest++ = \*src;

last\_was\_space = 0;

}

src++;

}

\*dest = '\0';

EXPECT\_STREQ(shared\_memory, " HELLO WORLD ");

}

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

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

```

**Выводы**

Программа демонстрирует использование разделяемой памяти и сигналов для межпроцессного взаимодействия. Реализована проверка корректности ввода данных (строка должна начинаться с заглавной буквы). Программа корректно завершает работу, освобождая все ресурсы (память, файловые дескрипторы). Приложение может быть расширено для выполнения более сложных задач, таких как обработка больших объемов данных или параллельные вычисления.