Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

**ОТЧЁТ**

к лабораторной работе

на тему

Взаимодействие процессов: обмен данными

Выполнил: студент группы 153505

Кудласевич Артур Иванович

Проверил: Сиротко Сергей Иванович

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Результаты выполнения лабораторной работы 6](#_Toc146631500)

[Выводы 7](#_Toc146631501)

[Приложение А (обязательное) Листинг кода 8](#_Toc146631503)

**1 ПОСТАНОВКА ЗАДАЧИ**

Целью лабораторной работы является изучение подходов, системных объектов и функций для обеспечения передачи данных между взаимодействующими процессами и/или совместной их обработки. Изучение типичных проблем, возникающих при организации взаимодействия, и пути их решения.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Каждый процесс обладает своим адресным пространством. Процесс не может обратиться в чужое пространство. При обращении к участку памяти другого процесса, процессором генерируется исключительная ситуация segmentation violation. Это дает защиту от случайного или намеренного нарушения данных одного процесса другим. Таким образом, самостоятельно процессы не обладают средствами взаимодействия друг с другом. Такое средство предоставляет только ОС.

Ограждение влияния на процесс со стороны другого процесса – это часть защиты операционной системы. Случайный или преднамеренный воздействия. Высококонкурентная среда. Несколько пользователей на машине, соперничающие процессы одного пользователя, вредоносные процессы, отладка. Поэтому если процессы хотят обмениваться данными друг с другом, они должны заявить о своем желании ОС. Или по-другому, воспользоваться одним из множества механизмов межпроцессного взаимодействия.

Сложная программа разбивается на взаимодействующие процессы. Следующий уровень модульности. Процессы взаимодействуют друг с другом через данные, их форматы и протоколы. В противоположность разбиению на модули и взаимодействие через структуры данных. Методика проектирования программы, в котором она разделяется на взаимодействующие процессы, называется **мультипрограммированием** (**многопроцессорной обработкой**).

В-третьих, несколько процессов используются для увеличения производительности программы за счет разделения объема работы по нескольким физическим процессорам и распределенным компьютерам.

Упрощает разработку программы за счет использования уже существующих программ. Для загрузки из сети файла используют curl, для редактирования документа — существующий текстовый редактор.

Философия UNIX коротко гласит: “Делайте что-то одно, но делайте это хорошо”. Она поощряет разработку маленьких программ, соединенных методами межпроцессного взаимодействия. Это стало возможным благодаря:

* малозатратное создание процессов;
* простые методы обмена данными между процессами;
* простые текстовые форматы данных.

fork – процесс раздваивается + copy-on-write. Ядро Linux, процессы создаются также быстро, как и потоки.

Существуют разные способы взаимодействия процессов друг с другом. Но суть у всех одна — передать данные от одного процесса другому.

* каналы (pipelines);
* очереди сообщений;
* семафоры;
* разделяемая память;
* сокеты;
* сигналы;
* файлы.

Чем обусловлено разнообразие методов? Некоторые методы дублируют друг друга. Некоторые методы возникли раньше других. Они появились в различных системах, развивались одновременно. Стандарт POSIX взял лучшее у обеих методов.

Взаимодействующие процессы могут располагаться на одной машине или связанных по сети машинах.

Передача данных может быть односторонней или двухсторонней. Передача данных в одном направлении (полудуплексный) или в обоих направлениях (дуплексный).

Процессы могут быть равноправными или состоять в подчинении.

Способы отличаются по сложности реализации и вносимую в программу неопределенности. Требование дополнительных методов синхронизации.

Каждый из способов имеет различную эффективность взаимодействия. Эта эффективность определяется двумя параметрами — пропускной способностью и задержкой. Пропускная способность показывает количество данных, которое может быть обработано за единицу времени. Задержка показывает время между отправкой данных и их получением.

Передача данных: отдельными завершенными сообщениями или потоком байтов.

Объект, через который происходит обмен данными. Он задается явно операционной системой. Объект похож на телефон или почтовый ящик. Обладание им процессом означает, что может отправлять данные или принимать данные от других процессов. Объект является ресурсом и в конце должен быть освобожден или уничтожен. Обычно объект – это дескриптор файла (указатель на файл). Для отправки данных используется функция записи в файл, для приема данных – функция чтения из файла.

**3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ     РАБОТЫ**

В рамках выполнения лабораторной работы было создано приложение, которое организовывает обмен чрез разделяемую память. Сначала процессы – “пользователи” захватывают буфер, затем они заполняют этот буфер данными, потом процессы освобождают его, для чтения другими процессами.

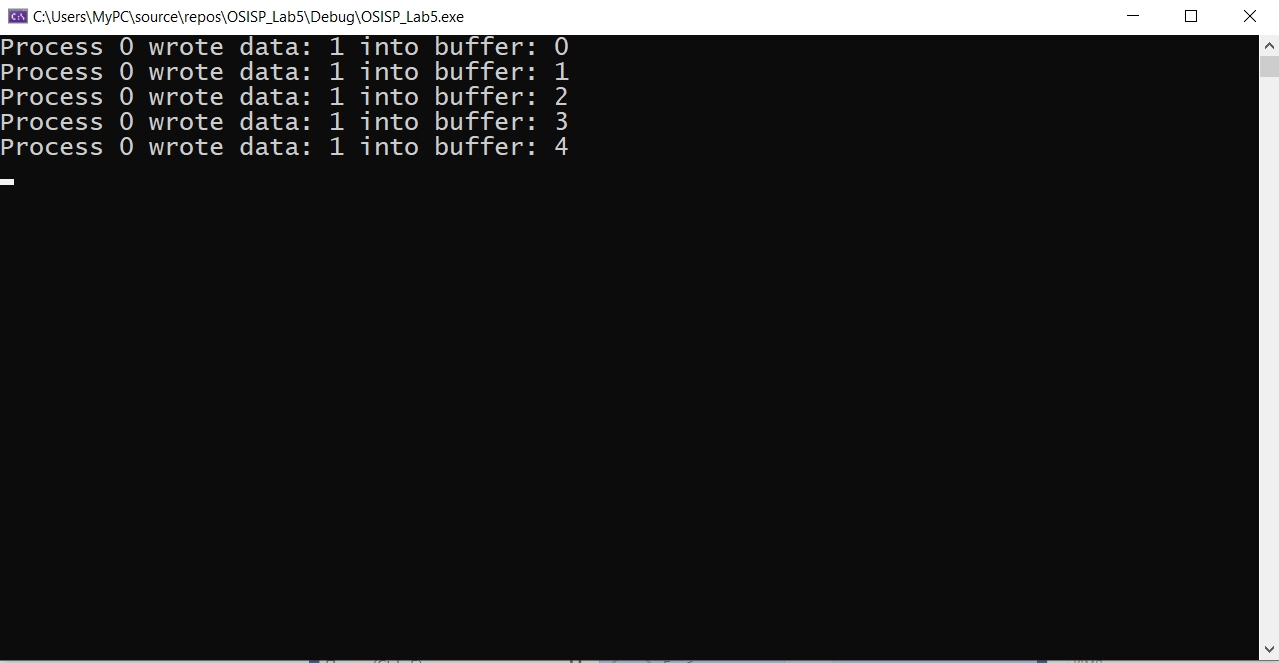


Рисунок 1 – Первый процесс записывает данные в буфер

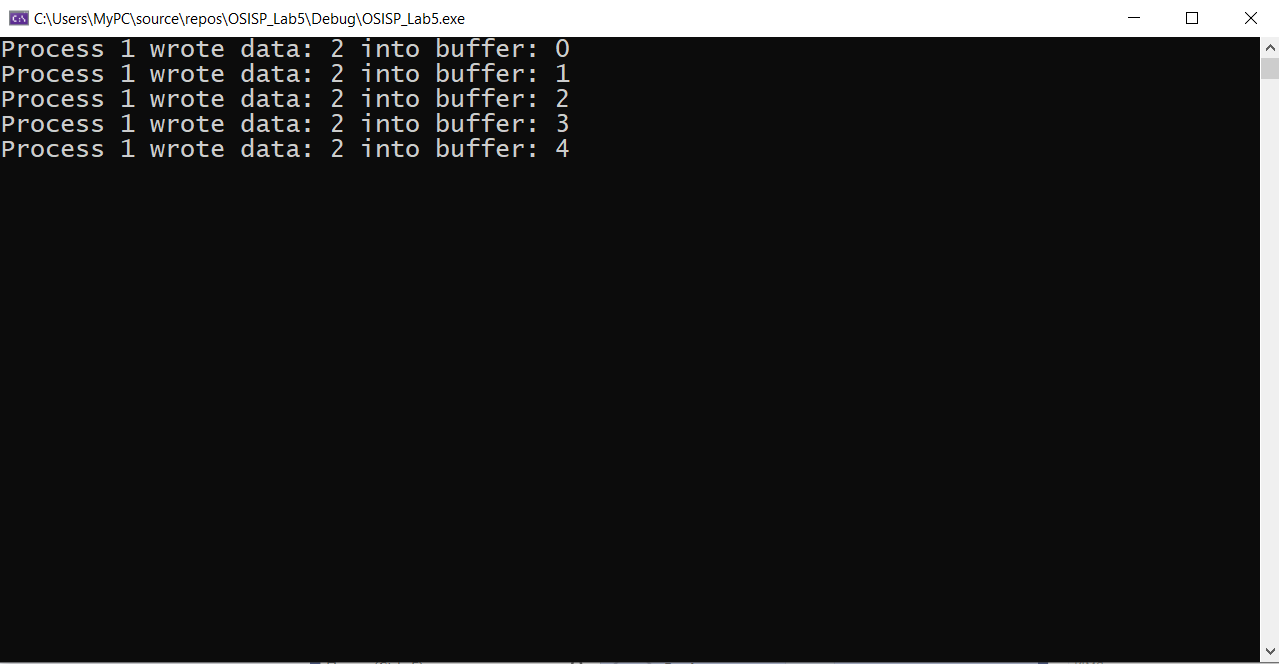


Рисунок 2 – Второй процесс записывает данные в буфер

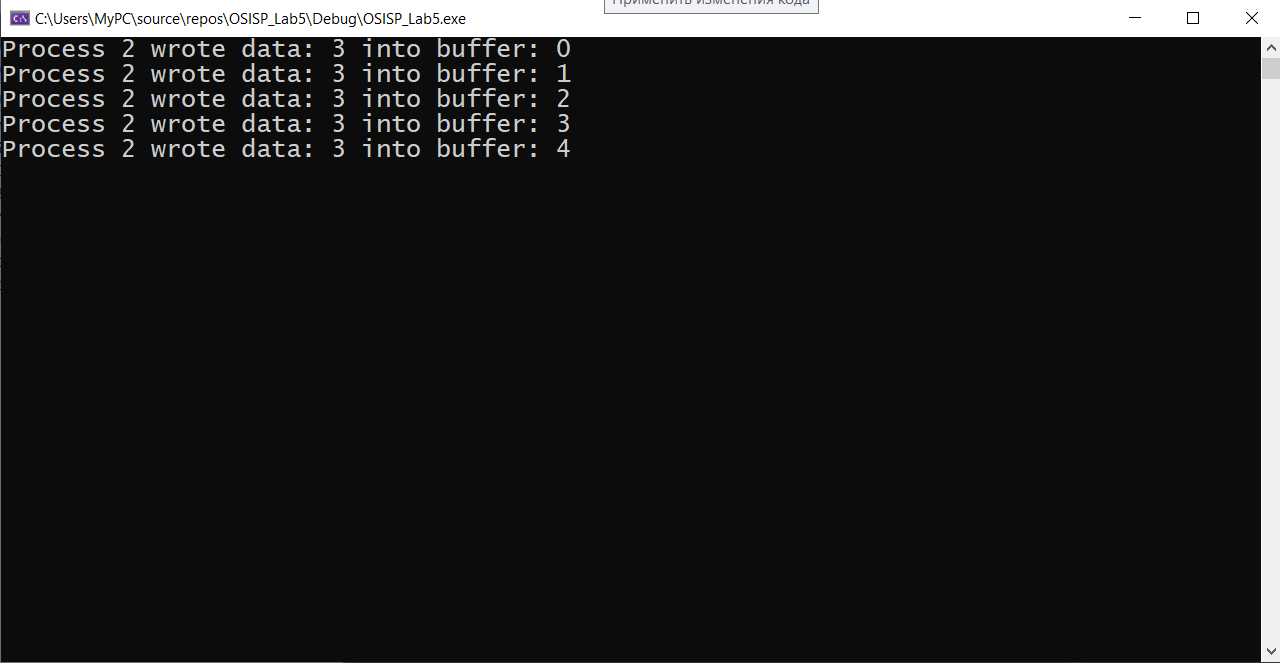


Рисунок 3 – Третий процесс записывает данные в буфер

**ВЫВОДЫ**

В результате выполнения данной лабораторной работы была разработана программа с использованием C++, которая организовывает обмен через разделяемую память.

**ПРИЛОЖЕНИЕ А**

## **(обязательное)**

**Листинг кода**

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <condition\_variable>

const int BUFFER\_SIZE = 5; // Размер пула буферов

const int NUM\_PROCESSES = 3; // Количество процессов-пользователей

std::vector<int> buffer(BUFFER\_SIZE); // Пул буферов

std::vector<std::mutex> mutexes(BUFFER\_SIZE); // Мьютексы для каждого буфера

std::vector<std::condition\_variable> cvars(BUFFER\_SIZE); // Условные переменные для каждого буфера

int currentBuffer = 0; // Текущий доступный буфер

// Функция для процесса-пользователя

void processData(int id) {

int data = id + 1;

while (true) {

{

std::unique\_lock<std::mutex> lock(mutexes[currentBuffer]); // Захватываем мьютекс буфера

// Ждем, пока буфер не станет доступен для записи

cvars[currentBuffer].wait(lock, [] { return buffer[currentBuffer] == 0; });

// Записываем данные в буфер

buffer[currentBuffer] = data;

std::cout << "Process " << id << " wrote data: " << data << " into buffer: " << currentBuffer << std::endl;

// Освобождаем буфер для чтения другим процессом

cvars[currentBuffer].notify\_one();

}

// Переходим к следующему буферу

currentBuffer = (currentBuffer + 1) % BUFFER\_SIZE;

}

}

int main() {

std::vector<std::thread> threads;

// Создаем и запускаем процессы-пользователей

for (int i = 0; i < NUM\_PROCESSES; i++) {

threads.emplace\_back(processData, i);

}

// Ожидаем завершения всех процессов

for (auto& thread : threads) {

thread.join();

}

return 0;

}