

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 №3  
на тему

**ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ: ОБМЕН ДАННЫМИ**

Выполнил: студент гр. 253503  
Тимошевич К.С.

Проверил: ассистент кафедры  
информатики Гриценко Н.Ю.

Минск 2024

## СОДЕРЖАНИЕ

1 Постановка задачи.....	3
2 Описание работы программы.....	4
2.1 Запись данных в разделяемую память.....	4
2.2 Чтение данных из разделяемой памяти.....	4
2.3 Синхронизация через мьютексы.....	4
3 Ход выполнения программы.....	5
3.1 Примеры выполнения задания.....	5
Вывод.....	6
Список использованных источников.....	7
Приложение А (справочное) Исходный код.....	8

## 1 ПОСТАНОВКА ЗАДАЧИ

Организация обмена через разделяемую память: совместное использование содержимого разделяемой памяти несколькими процессами (потоками) с предотвращением коллизий. В данной лабораторной работе будет реализован обмен данными между двумя процессами через разделяемую память с защитой от коллизий с использованием мьютексов. Один процесс (*Writer*) будет записывать данные в разделяемую память, другой процесс (*Reader*) читать эти данные.

## 2 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

В этом разделе будут рассмотрены основные функции программы, которые были задействованы для выполнения работы.

### 2.1 Запись данных в разделяемую память

Модуль *writer* реализует процесс записи данных в разделяемую память с синхронизацией доступа с помощью мьютекса. Программа создает разделяемую память с помощью функции *CreateFileMapping* [1] и отображает её в адресное пространство процесса с помощью функции *MapViewOfFile*. После этого создается мьютекс через *CreateMutex* [2] для защиты доступа к памяти.

В бесконечном цикле программа запрашивает ввод от пользователя, блокируя мьютекс функцией *WaitForSingleObject*. Введенная строка сохраняется в разделяемую память с помощью *CopyMemory*. При вводе строки «*exit*» программа завершает работу, записав в память сигнал завершения и освободив мьютекс через *ReleaseMutex*. По окончании работы программа освобождает память и закрывает дескрипторы объектов.

### 2.2 Чтение данных из разделяемой памяти

Модуль *reader* отвечает за чтение данных, записанных в разделяемую память модулем *writer*. Программа открывает уже существующую разделяемую память через *OpenFileMapping* и отображает её в адресное пространство с помощью *MapViewOfFile* [3]. Аналогичным образом открывается мьютекс через *OpenMutex* для синхронизации доступа к памяти.

В бесконечном цикле программа ожидает получения доступа к разделяемой памяти, блокируя мьютекс через *WaitForSingleObject* [4]. Она выводит содержимое памяти на экран, проверяя, была ли записана строка «*exit*», сигнализирующая о завершении работы *writer*. В случае обнаружения сигнала завершения, программа выводит сообщение и завершает свою работу, освобождая мьютекс и закрывая дескрипторы объектов.

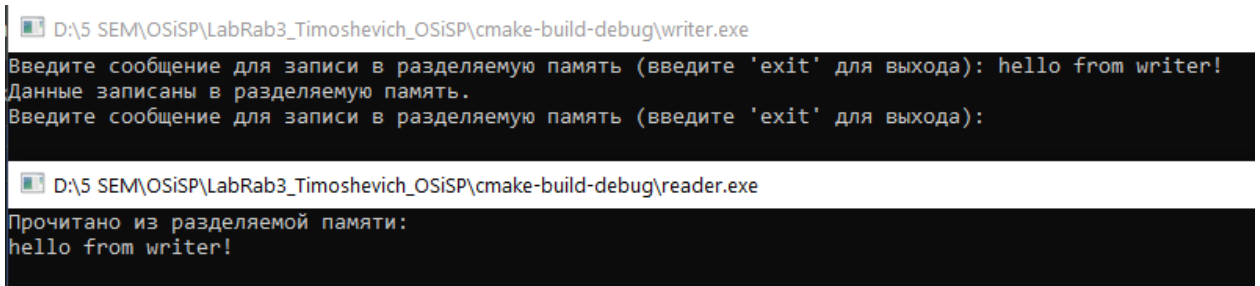
### 2.3 Синхронизация через мьютексы

Для предотвращения одновременного доступа к разделяемой памяти из разных процессов используется мьютекс. *writer* блокирует мьютекс при записи данных, а *reader* блокирует мьютекс при чтении, что исключает ситуации гонок. Мьютекс освобождается сразу после завершения операции, что позволяет другому процессу получить доступ к памяти.

## 3 ХОД ВЫПОЛНЕНИЯ ПРОГРАММЫ

### 3.1 Примеры выполнения задания

На рисунке 3.1 представлен результат работы программы *writer* и *reader* при вводе первой строки, использующих разделяемую память для обмена данными. В процессе выполнения задания модуль *writer* записывает введенные пользователем строки в разделяемую память, которая защищена мьютексом для исключения одновременного доступа. При вводе команды «exit» программы завершают свою работу, освобождая ресурсы.

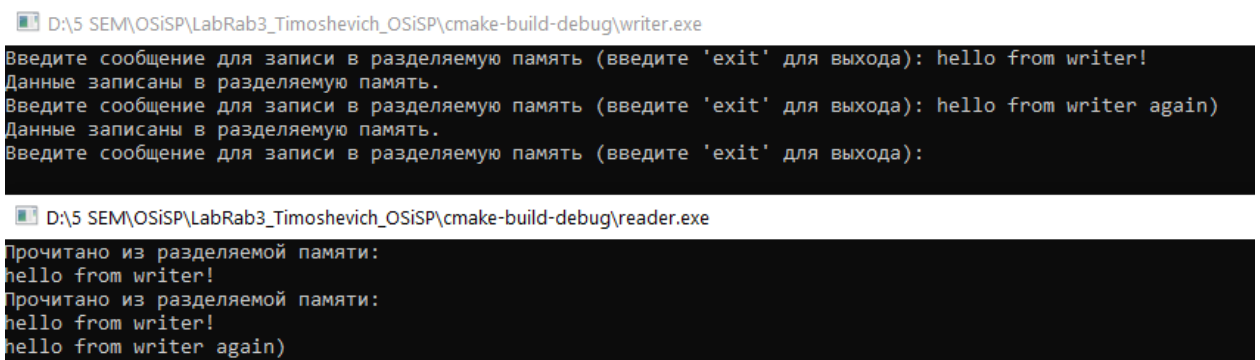


```
D:\5 SEM\OSiSP\LabRab3_Timoshevich_OSiSP\cmake-build-debug\writer.exe
Введите сообщение для записи в разделяемую память (введите 'exit' для выхода): hello from writer!
Данные записаны в разделяемую память.
Введите сообщение для записи в разделяемую память (введите 'exit' для выхода):

D:\5 SEM\OSiSP\LabRab3_Timoshevich_OSiSP\cmake-build-debug\reader.exe
Прочитано из разделяемой памяти:
hello from writer!
```

Рисунок 3.1 - Результат работы после ввода первой строки

На рисунке 3.2 представлен результат работы программы после ввода второй строки и чтения программой *reader* всего записанного текста из разделяемой памяти. На первом шаге был показан вывод первой строки, записанной *writer* в память и успешно считанной *reader*. После ввода второй строки модулем *writer*, *reader* выводит обе строки, подтверждая, что он корректно считывает все содержимое памяти, включая ранее записанные данные.



```
D:\5 SEM\OSiSP\LabRab3_Timoshevich_OSiSP\cmake-build-debug\writer.exe
Введите сообщение для записи в разделяемую память (введите 'exit' для выхода): hello from writer!
Данные записаны в разделяемую память.
Введите сообщение для записи в разделяемую память (введите 'exit' для выхода): hello from writer again)
Данные записаны в разделяемую память.
Введите сообщение для записи в разделяемую память (введите 'exit' для выхода):

D:\5 SEM\OSiSP\LabRab3_Timoshevich_OSiSP\cmake-build-debug\reader.exe
Прочитано из разделяемой памяти:
hello from writer!
Прочитано из разделяемой памяти:
hello from writer!
hello from writer again)
```

Рисунок 3.2 - Результат работы после ввода последующей строки

## ВЫВОД

В ходе выполнения лабораторной работы была реализована система взаимодействия между процессами с использованием разделяемой памяти и механизмов синхронизации. Были разработаны две программы: *writer*, которая записывает данные в разделяемую память, и *reader*, которая считывает эти данные. Для синхронизации доступа к разделяемой памяти использовался мьютекс, предотвращающий конкурентный доступ к ресурсу.

Результаты тестирования показали, что механизм синхронизации с помощью мьютекса позволяет эффективно организовать безопасный доступ к разделяемой памяти между процессами. Программа корректно обрабатывает данные, обеспечивая сохранность записанной информации и её последовательную передачу между процессами. Этот подход может быть полезен в сценариях межпроцессного взаимодействия, где необходим обмен данными между независимыми процессами.

Таким образом, лабораторная работа продемонстрировала практическое применение технологий работы с разделяемой памятью и механизмов синхронизации в операционной системе *Windows*.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] CreateFileMapping function [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/winbase/nf-winbase-createfilemapping> – Дата доступа: 10.10.2024.

[2] WinAPI: CreateMutex [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-createmutex>. – Дата доступа: 12.10.2024.

[3] API Win32 MapViewOfFile [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile>. – Дата доступа: 12.10.2024.

[4] WaitForSingleObject [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject>. – Дата доступа: 12.10.2024.

## ПРИЛОЖЕНИЕ А

### (справочное)

### Исходный код

Листинг файла writer.cpp:

```
#include <windows.h>
#include <iostream>

const int BUF_SIZE = 256;

int main() {
    SetConsoleOutputCP(CP_UTF8);

    HANDLE hMapFile = CreateFileMappingW(
        INVALID_HANDLE_VALUE,
        NULL,
        PAGE_READWRITE,
        0,
        BUF_SIZE,
        L"Local\\MySharedMemory"
    );

    if (hMapFile == NULL) {
        std::cerr << "Не удалось создать разделяемую память: " <<
        GetLastError() << std::endl;
        return 1;
    }

    LPCTSTR pBuf = (LPTSTR)MapViewOfFile(
        hMapFile,
        FILE_MAP_ALL_ACCESS,
        0,
        0,
        BUF_SIZE
    );

    if (pBuf == NULL) {
        std::cerr << "Не удалось отобразить разделяемую память: " <<
        GetLastError() << std::endl;
        CloseHandle(hMapFile);
        return 1;
    }

    HANDLE hMutex = CreateMutexW(
        NULL,
        FALSE,
        L"Local\\MyMutex"
    );

    if (hMutex == NULL) {
        std::cerr << "Не удалось создать мьютекс: " << GetLastError() <<
        std::endl;
        UnmapViewOfFile(pBuf);
        CloseHandle(hMapFile);
        return 1;
    }

    while (true) {
        WaitForSingleObject(hMutex, INFINITE);
```



```

        std::cout << "Введите сообщение для записи в разделяемую память
(введите 'exit' для выхода): ";
        std::string input;
        std::getline(std::cin, input);

        if (input == "exit") {
            CopyMemory((PVOID)pBuf, "exit", 5 * sizeof(char));
            break;
        }

        std::string currentData = (char*)pBuf;
        if (!currentData.empty()) {
            currentData += "\n";
        }
        currentData += input;
        CopyMemory((PVOID)pBuf, currentData.c_str(), (currentData.size() + 1) *
sizeof(char));
        std::cout << "Данные записаны в разделяемую память.\n";

        ReleaseMutex(hMutex);
    }

    UnmapViewOfFile(pBuf);
    CloseHandle(hMapFile);
    CloseHandle(hMutex);

    return 0;
}

```

## Листинг файла reader.cpp

```

#include <windows.h>
#include <iostream>

const int BUF_SIZE = 256;

int main() {
    SetConsoleOutputCP(CP_UTF8);

    HANDLE hMapFile = OpenFileMappingW(
        FILE_MAP_ALL_ACCESS,
        FALSE,
        L"Local\\MySharedMemory"
    );

    if (hMapFile == NULL) {
        std::cerr << "Не удалось открыть разделяемую память: " <<
GetLastError() << std::endl;
        return 1;
    }

    LPCTSTR pBuf = (LPTSTR)MapViewOfFile(
        hMapFile,
        FILE_MAP_ALL_ACCESS,
        0,
        0,
        BUF_SIZE
    );

    if (pBuf == NULL) {
        std::cerr << "Не удалось отобразить разделяемую память: " <<
GetLastError() << std::endl;
        CloseHandle(hMapFile);
    }
}

```

```

        return 1;
    }

    HANDLE hMutex = OpenMutexW(
        MUTEX_ALL_ACCESS,
        FALSE,
        L"Local\\MyMutex"
    );

    if (hMutex == NULL) {
        std::cerr << "Не удалось открыть мьютекс: " << GetLastError() <<
std::endl;
        UnmapViewOfFile(pBuf);
        CloseHandle(hMapFile);
        return 1;
    }

    while (true) {
        WaitForSingleObject(hMutex, INFINITE);
        std::cout << "Прочитано из разделяемой памяти:" << std::endl;
        if (strcmp((const char*)pBuf, "exit") == 0) {
            std::cout << "Получен сигнал на завершение. Закрытие Reader." <<
std::endl;
            ReleaseMutex(hMutex);
            break;
        }
        std::cout << pBuf << std::endl;

        ReleaseMutex(hMutex);
        Sleep(100);
    }

    UnmapViewOfFile(pBuf);
    CloseHandle(hMapFile);
    CloseHandle(hMutex);

    return 0;
}

```