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.253505 Гимпель К.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc178780699)

[2 Краткие теоритические сведения 4](#_Toc178780700)

[3 Результат запуска программы 5](#_Toc178780701)

[3.1 Выбор процесса для запуска 5](#_Toc178780702)

[3.2 Иллюстрация запуска и работы программы 5](#_Toc178780703)

Заключение 6

Список использованных источников 7

Приложение А (обязательное) Исходный код программы 8

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является подходы, системные объекты и функции для обеспечения передачи данных между взаимодействующими процессами и/или совместной их обработки. В ходе выполнения работы необходимо изучить типичные проблемы, возникающие при организации взаимодействия, и пути их решения.

Задача заключается в реализации механизма обмена данными между процессами через разделяемую память, который должен выполнять следующие функции:

– организация обмена через разделяемую память;

– совместное использование разделяемой памяти;

– механизм защиты от коллизий;

– отображение возникающих ошибок.

Специальных требований к приложению не предъявляется; в частности, оно может быть не обязательно оконными, но также и консольным.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Взаимодействие процессов (Interprocess Communication, IPC) представляет собой совокупность методов, которые обеспечивают обмен данными или координацию действий между процессами в многозадачных системах. Одним из основных подходов к решению задач IPC является использование разделяемой памяти. Разделяемая память позволяет нескольким процессам обращаться к одному и тому же участку памяти, что обеспечивает высокую скорость обмена данными, поскольку исключается необходимость их копирования.

Для организации доступа к разделяемой памяти в операционной системе Windows используется API функции. Объект разделяемой памяти создаётся с помощью функции CreateFileMapping(), которая возвращает дескриптор объекта. Затем с помощью функции MapViewOfFile() выполняется отображение объекта разделяемой памяти в адресное пространство процесса. Процессы могут взаимодействовать с одним и тем же участком памяти, используя указатель на этот объект.

При работе с разделяемой памятью важно учитывать проблемы согласованного доступа. В случае одновременного обращения нескольких процессов к одному ресурсу могут возникнуть коллизии, что приводит к некорректной работе программы. Для предотвращения таких ситуаций используются механизмы синхронизации, такие как мьютексы. В Windows объект мьютекса создаётся с помощью функции CreateMutex() и управляется через функции WaitForSingleObject() и ReleaseMutex(), которые обеспечивают взаимное исключение при доступе к разделяемой памяти.

Примером использования разделяемой памяти является обмен данными между двумя процессами: производителем, записывающим данные в память, и потребителем, считывающим их. Такая модель обеспечивает эффективный обмен информацией при условии корректной синхронизации доступа.

Эти механизмы взаимодействия процессов и синхронизации подробно рассмотрены в книгах А. Таненбаума «Современные операционные системы» [1] и А. Сильбершаца «Операционные системы: принципы и практика» [2], где описываются теоретические аспекты и практическая реализация межпроцессного взаимодействия в современных ОС.

3 РЕЗУЛЬТАТ ЗАПУСКА ПРОГРАММЫ

## 3.1 Шаги запуска программы

Программа для взаимодействия процессов через разделяемую память была успешно разработана, скомпилирована и запущена. Основные шаги запуска включают:

– Открытие командной строки;

– Компиляцию файла producer.cpp: g++ producer.cpp -o producer.exe;

– Компиляцию файла consumer.cpp: g++ consumer.cpp -o consumer.exe;

– Запуск программы producer.exe, а затем consumer.exe.

В процессе работы программы производитель запрашивает ввод данных от пользователя, записывает их в разделяемую память и уведомляет потребителя. Потребитель считывает данные из памяти, обрабатывает их и выводит результат на экран. Этот процесс продолжается до тех пор, пока в программе производителя не будет введена команда завершения (exit).

## 3.2 Иллюстрация запуска и работы программы

На рисунке 3.1 показан консольный вывод программы производителя producer.exe и программы потребителя consumer.exe. Пользователь вводит строку данных, которая записывается в разделяемую память. После ввода программа переходит в ожидание следующей строки данных. Потребитель ожидает данные в разделяемой памяти, затем считывает их, выводит на экран и очищает память для следующего использования. Когда в программе производителя вводится команда exit, потребитель и производитель завершают свою работу.

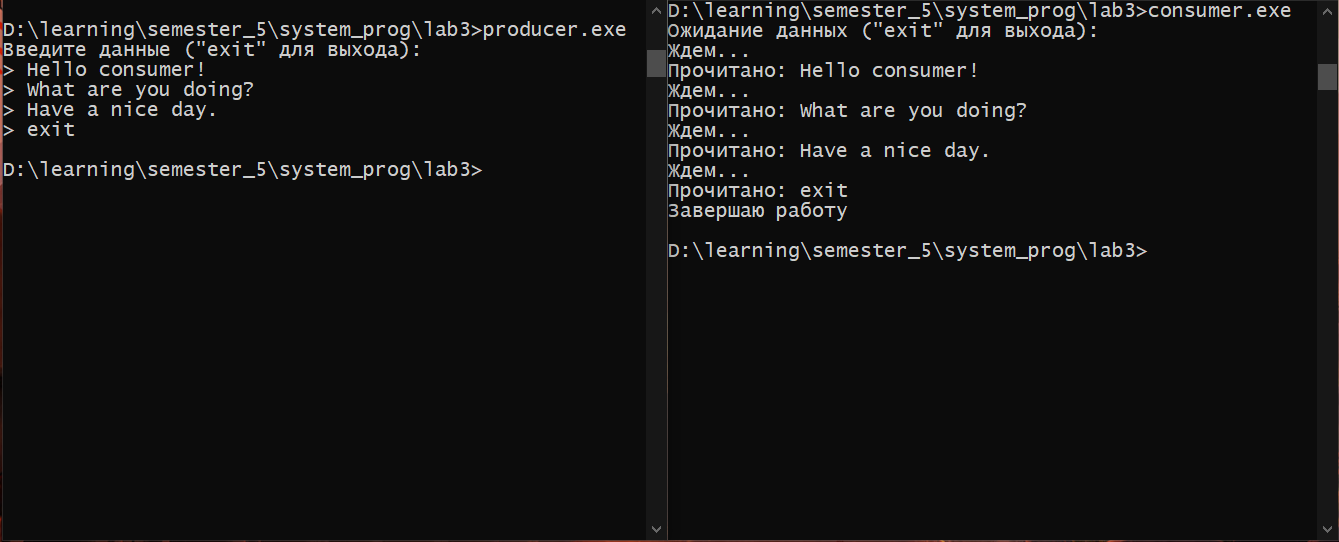


Рисунок 3.1 – Результат работы программ

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы по дисциплине «Операционные системы и системное программирование» была разработана программа для взаимодействия процессов через разделяемую память. Основная задача заключалась в организации обмена данными между процессами с использованием разделяемой памяти и защиты от коллизий с помощью мьютексов.

В процессе выполнения работы были изучены и применены механизмы взаимодействия процессов на платформе Windows. Были реализованы и протестированы средства синхронизации доступа к разделяемым ресурсам, что позволило избежать конфликтов при одновременной работе нескольких процессов.

Лабораторная работа не только углубила теоретические знания о способах взаимодействия процессов, но и предоставила практические навыки, которые являются основой для разработки надёжных и эффективных систем многозадачности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Таненбаум Э. С. Современные операционные системы. — 4-е изд. — СПб.: Питер, 2018. — 1120 с.

[2] Сильбершац А., Галвин П., Гейдж Г. Операционные системы: принципы и практика. — 9-е изд. — М.: Вильямс, 2015. — 1056 с.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <windows.h>

#include <stdio.h>

#define SHARED\_MEMORY\_NAME L"SharedMemoryExample"

#define MUTEX\_NAME L"SharedMemoryMutex"

#define BUFFER\_SIZE 256

int main() {

SetConsoleOutputCP(CP\_UTF8);

HANDLE hMapFile;

HANDLE hMutex;

LPVOID pBuf;

hMapFile = CreateFileMappingW(

INVALID\_HANDLE\_VALUE, // Использовать файл подкачки

NULL, // Стандартный атрибут безопасности

PAGE\_READWRITE, // Доступ для чтения/записи

0, // Высокий DWORD размера

BUFFER\_SIZE, // Низкий DWORD размера

SHARED\_MEMORY\_NAME // Имя разделяемой памяти

);

if (hMapFile == NULL) {

printf("Не удалось создать разделяемую память. Код ошибки: %d\n", GetLastError());

return 1;

}

hMutex = CreateMutexW(NULL, FALSE, MUTEX\_NAME);

if (hMutex == NULL) {

printf("Не удалось создать мьютекс. Код ошибки: %d\n", GetLastError());

CloseHandle(hMapFile);

return 1;

}

pBuf = MapViewOfFile(

hMapFile, // Дескриптор разделяемой памяти

FILE\_MAP\_ALL\_ACCESS, // Доступ для чтения/записи

0, // Высокий DWORD смещения

0, // Низкий DWORD смещения

BUFFER\_SIZE // Число отображаемых байт

);

if (pBuf == NULL) {

printf("Не удалось отобразить разделяемую память. Код ошибки: %d\n", GetLastError());

CloseHandle(hMapFile);

CloseHandle(hMutex);

return 1;

}

printf("Введите данные (\"exit\" для выхода):\n");

char input[BUFFER\_SIZE];

while (1) {

printf("> ");

fgets(input, BUFFER\_SIZE, stdin);

input[strcspn(input, "\n")] = 0;

WaitForSingleObject(hMutex, INFINITE);

memcpy(pBuf, input, BUFFER\_SIZE);

ReleaseMutex(hMutex);

if (strcmp(input, "exit") == 0) break;

}

UnmapViewOfFile(pBuf);

CloseHandle(hMapFile);

CloseHandle(hMutex);

return 0;

}

#include <windows.h>

#include <stdio.h>

#define SHARED\_MEMORY\_NAME L"SharedMemoryExample"

#define MUTEX\_NAME L"SharedMemoryMutex"

#define BUFFER\_SIZE 256

int main() {

SetConsoleOutputCP(CP\_UTF8);

HANDLE hMapFile;

HANDLE hMutex;

LPVOID pBuf;

hMapFile = OpenFileMappingW(

FILE\_MAP\_ALL\_ACCESS, // Доступ для чтения/записи

FALSE, // Унаследование дескриптора

SHARED\_MEMORY\_NAME // Имя разделяемой памяти

);

if (hMapFile == NULL) {

printf("Не удалось открыть разделяемую память. Код ошибки: %d\n", GetLastError());

return 1;

}

hMutex = OpenMutexW(

MUTEX\_ALL\_ACCESS, // Доступ полный

FALSE, // Унаследование дескриптора

MUTEX\_NAME // Имя мьютекса

);

if (hMutex == NULL) {

printf("Не удалось открыть мьютекс. Код ошибки: %d\n", GetLastError());

CloseHandle(hMapFile);

return 1;

}

pBuf = MapViewOfFile(

hMapFile, // Дескриптор разделяемой памяти

FILE\_MAP\_ALL\_ACCESS, // Доступ для чтения/записи

0, // Высокий DWORD смещения

0, // Низкий DWORD смещения

BUFFER\_SIZE // Число отображаемых байт

);

if (pBuf == NULL) {

printf("Не удалось отобразить разделяемую память. Код ошибки: %d\n", GetLastError());

CloseHandle(hMapFile);

CloseHandle(hMutex);

return 1;

}

printf("Ожидание данных (\"exit\" для выхода):\n");

char buffer[BUFFER\_SIZE];

bool isWait = false;

while (1) {

WaitForSingleObject(hMutex, INFINITE);

memcpy(buffer, pBuf, BUFFER\_SIZE);

memset(pBuf, 0, BUFFER\_SIZE);

ReleaseMutex(hMutex);

bool isEmpty = strcmp(buffer, "") == 0;

if (!isWait && isEmpty) {

printf("Ждем...\n");

isWait = true;

}

else if (!isEmpty) {

printf("Прочитано: %s\n", buffer);

isWait = false;

}

if (strcmp(buffer, "exit") == 0) {

printf("Завершаю работу\n");

break;

}

}

UnmapViewOfFile(pBuf);

CloseHandle(hMapFile);

CloseHandle(hMutex);

return 0;

}