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

ОТЧЕТ

к лабораторной работе №6

на тему

Взаимодействие процессов (потоков): взаимное исключение и синхронизация

Студент П. С. Павлюткин

Преподаватель С. И. Сиротко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Цель работы изучить и применить на практике подходы, системные объекты и функции для обеспечения синхронизации и передачи управления между взаимодействующими процессами, научиться решать типичные проблемы, возникающие при организации взаимодействия, модели для их описания, пути их решения.Для достижения цели будет создано приложение, отображающее модель ‘обедающие философы’.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Задачи взаимного исключения (предотвращения некорректного совместного доступа к критическим ресурсам) и синхронизации – в числе основных задач взаимодействия. Критический ресурс и критическая секция. Проблема атомарности доступа для проверки и модификации глобальных объектов, используемых для синхронизации. Теоретические модели для описания ситуаций синхронизации. Механизм *CriticalSection*; функции ожидания *WaitFor*() и объекты ожидания: *Mutex*, *Semaphore*, *Event*, *WaitableTimer*; барьеры; спин и блокировки и т.д.

Взаимодействие процессов - это ключевой аспект в современных операционных системах и программировании. Процессы могут быть независимыми сущностями, работающими параллельно, и взаимодействие между ними часто необходимо для решения различных задач. Одним из основных методов взаимодействия процессов является обмен данными. Вот некоторые теоретические аспекты этой темы:

Процессы могут требовать обмена данными для передачи информации, синхронизации, координации и выполнения распределенных задач. При обмене данными между процессами важно обеспечивать безопасность и конфиденциальность передаваемой информации, особенно в распределенных системах. Обеспечивают двунаправленную связь между процессами, работающими на одной машине.

Позволяют взаимодействовать между процессами, работающими на разных машинах в сети. Используются для синхронизации доступа к ресурсам между процессами. Позволяют посылать сообщения между процессами. Позволяет нескольким процессам разделять общую область памяти. Процессы в операционных системах имеют свои адресные пространства, и для взаимодействия между ними требуются специальные механизмы. Операционные системы обеспечивают множество средств для межпроцессорного взаимодействия, включая средства синхронизации, обмена сообщениями и разделяемую память.

Возникают, когда два или более процесса пытаются изменить общий ресурс одновременно. Ситуации, когда несколько процессов блокируют друг друга, ожидая освобождения ресурсов. Задача предотвращения одновременного доступа нескольких процессов к общему ресурсу. Увеличение производительности, повышение отказоустойчивости и повышение масштабируемости за счет использования ресурсов нескольких вычислительных узлов. Распределенные базы данных, облачные вычисления, распределенные вычислительные сети.

Взаимодействие между процессами и обмен данными - это обширная и важная тема, и выбор методов зависит от конкретных требований приложения. Он играет ключевую роль в разработке многозадачных, многопоточных и распределенных систем.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате выполнения лабораторной работы было создано приложение, отображающее модель ‘обедающие философы’ (Рисунок 1).

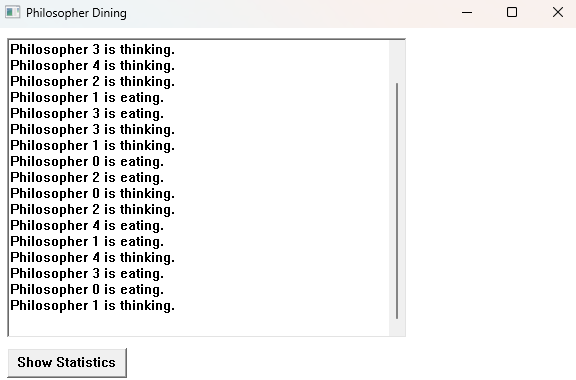


Рисунок 1 – Основное окно приложения

Данная программа моделирует проблему обедающих философов с использованием графического интерфейса пользователя (*GUI*) с помощью *Windows API*. Проблема обедающих философов - это классическая задача синхронизации и параллелизма, где группа философов сидит вокруг стола. Каждый философ чередует фазы размышления и приема пищи. Однако они разделяют ограниченное количество вилок, и для приема пищи философ должен взять обе вилки с обеих сторон.

Программа инициализирует графическое окно с использованием *Windows API*. Создается элемент управления текстовым редактором (`*EDIT*`), чтобы отображать информацию о деятельности философов.

Пять философов (`*num\_philosophers*`) представлены потоками. Вилки представлены объектами `*std::mutex*`.

Каждый философ проходит цикл размышления и еды до тех пор, пока не достигнет максимального количества приемов пищи (`*max\_meals*`). Функции `*think*` и `*eat*` моделируют действия философа, включая отображение сообщений в элементе управления текстовым редактором.

Для представления требования, что философ должен взять обе вилки для еды, программа использует `*std::unique\_lock*` для блокировки двух смежных вилок, когда философ решает поесть. Это предотвращает дедлоки, так как каждому философу нужны две вилки, и они не могут быть захвачены одновременно двумя соседними философами.

Программа предоставляет кнопку с надписью "Показать статистику". При нажатии этой кнопки вызывается функция `*printStatistics*`, отображающая информацию о количестве приемов пищи каждым философом и общем количестве приемов пищи.

Функция `*WindowProc*` обрабатывает различные сообщения *Windows*. Она создает компоненты *GUI*, запускает потоки философов и обрабатывает нажатия кнопок.

Основная точка входа (`*WinMain*`) инициализирует программу, создает главное окно и входит в цикл обработки сообщений *Windows*.

Таким образом, программа демонстрирует базовое решение проблемы обедающих философов с графическим интерфейсом, используя *Windows API*. *GUI* позволяет визуализировать действия философов и просматривать статистику их приемов пищи. Эта программа представляет собой простой пример распределенной обработки данных, где процесс-диспетчер выдает задание процессу-рабочему, а тот возвращает результат обратно. В реальных приложениях это может использоваться, например, для распределенных вычислений, обработки данных в разных частях системы, или для организации взаимодействия между компонентами программы, работающими в разных процессах (Рисунок 2).

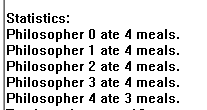


Рисунок 2– Отображение окна со статистикой

Наше приложение демонстрирует то, как можно отобразить модель ‘обедающие философы’.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения настоящей лабораторной работы были исследованы и применены методы взаимодействия между процессами с использованием технологии именованных объектов в среде Windows, а также библиотеки WinAPI, реализованные на языке программирования C++. Этот процесс включал в себя моделирование взаимодействия конкурирующих параллельных процессов, сфокусированных на решении задачи обедающих философов.

Проект охватывает два основных типа процессов: процесс-диспетчер и рабочий процесс. Процесс-диспетчер отвечает за управление объектами IPC (в данном случае, оконным приложением), выдачу задач рабочим процессам и сбор результатов. Рабочий процесс, в свою очередь, принимает задание, выполняет его и возвращает результат.

В качестве средства взаимодействия были использованы именованные каналы для обеспечения двустороннего обмена данными между процессами. Особое внимание было уделено обработке ошибок, и для подробного контроля и обеспечения безопасности использовались диалоговые окна с сообщениями об ошибках и информацией о ходе выполнения программы.

Эта лабораторная работа предоставила практический опыт в реализации взаимодействия между процессами в операционной системе Windows. Она также позволила применить знания WinAPI для создания и управления процессами, обеспечивая эффективный обмен данными и координацию между ними. Полученные навыки являются важными в контексте разработки распределенных систем и приложений, требующих взаимодействия между отдельными компонентами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. *Win32 API*. Разработка приложений для *Windows*. ─ СПБ: Питер, 2008. ─ 592 с.: ип.
2. Создание классических приложений для *Windows* с использованием *API Win32* [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл Lab6.cpp

#include <windows.h>

#include <thread>

#include <vector>

#include <mutex>

#include <sstream>

#include <iomanip>

const int num\_philosophers = 5;

std::vector<std::mutex> forks(num\_philosophers);

std::vector<std::thread> philosopher\_threads;

HWND hwndEdit; // Дескриптор текстового окна для вывода информации

int num\_meals = 0;

int max\_meals = 10; // Максимальное количество приемов пищи для каждого философа

std::vector<int> meals\_eaten(num\_philosophers, 0);

void think(int philosopher\_id) {

std::wstringstream ss;

ss << L"Philosopher " << philosopher\_id << L" is thinking.\r\n";

SendMessageW(hwndEdit, EM\_SETSEL, -1, -1);

SendMessageW(hwndEdit, EM\_REPLACESEL, 0, reinterpret\_cast<LPARAM>(ss.str().c\_str()));

std::this\_thread::sleep\_for(std::chrono::milliseconds(rand() % 1000));

}

void eat(int philosopher\_id) {

std::wstringstream ss;

ss << L"Philosopher " << philosopher\_id << L" is eating.\r\n";

SendMessageW(hwndEdit, EM\_SETSEL, -1, -1);

SendMessageW(hwndEdit, EM\_REPLACESEL, 0, reinterpret\_cast<LPARAM>(ss.str().c\_str()));

// Счетчик приемов пищи

meals\_eaten[philosopher\_id]++;

num\_meals++;

std::this\_thread::sleep\_for(std::chrono::milliseconds(rand() % 1000));

}

void philosopher(int philosopher\_id) {

while (meals\_eaten[philosopher\_id] < max\_meals) {

think(philosopher\_id);

std::unique\_lock<std::mutex> left\_fork(forks[philosopher\_id]);

std::unique\_lock<std::mutex> right\_fork(forks[(philosopher\_id + 1) % num\_philosophers]);

eat(philosopher\_id);

}

}

void printStatistics() {

std::wstringstream ss;

ss << L"\r\nStatistics:\r\n";

for (int i = 0; i < num\_philosophers; ++i) {

ss << L"Philosopher " << i << L" ate " << meals\_eaten[i] << L" meals.\r\n";

}

ss << L"Total meals eaten: " << num\_meals << L"\r\n";

SendMessageW(hwndEdit, EM\_SETSEL, -1, -1);

SendMessageW(hwndEdit, EM\_REPLACESEL, 0, reinterpret\_cast<LPARAM>(ss.str().c\_str()));

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg) {

case WM\_CREATE:

hwndEdit = CreateWindowExW(

WS\_EX\_CLIENTEDGE, L"EDIT", L"",

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL,

10, 10, 400, 300,

hwnd, nullptr, GetModuleHandle(nullptr), nullptr);

for (int i = 0; i < num\_philosophers; ++i) {

philosopher\_threads.emplace\_back(philosopher, i);

}

// Добавление кнопки для вывода статистики

CreateWindowW(

L"BUTTON", L"Show Statistics", WS\_VISIBLE | WS\_CHILD,

10, 320, 120, 30,

hwnd, reinterpret\_cast<HMENU>(1), GetModuleHandle(nullptr), nullptr);

break;

case WM\_COMMAND:

if (LOWORD(wParam) == 1) {

// Обработка нажатия кнопки вывода статистики

printStatistics();

}

break;

case WM\_DESTROY:

for (auto& thread : philosopher\_threads) {

thread.join();

}

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

srand(static\_cast<unsigned>(time(nullptr)));

WNDCLASSW wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = L"PhilosopherClass";

RegisterClassW(&wc);

HWND hwnd = CreateWindowExW(

0,

L"PhilosopherClass",

L"Philosopher Dining",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT,

CW\_USEDEFAULT,

600,

400,

nullptr,

nullptr,

hInstance,

nullptr);

ShowWindow(hwnd, nCmdShow);

MSG msg;

while (GetMessageW(&msg, nullptr, 0, 0)) {

TranslateMessage(&msg);

DispatchMessageW(&msg);

}

return 0;

}