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 №6

на тему

СРЕДСТВА СИНХРОНИЗАЦИИ И ВЗАИМНОГО ИСКЛЮЧЕНИЯ (WINDOWS). ИЗУЧЕНИЕ И ИСПОЛЬЗОВАНИЕ СРЕДСТВ СИНХРОНИЗАЦИИ И ВЗАИМНОГО ИСКЛЮЧЕНИЯ.

Выполнил студент гр.153502 Толстой Д.В.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc150472452)

[2 Теоретические сведения 4](#_Toc150472453)

[3 Описание функций программы 5](#_Toc150472454)

[3.1 Ввод количества обедающих философов 5](#_Toc150472455)

[3.2 Запуск обеда философов 5](#_Toc150472456)

[Список использованных источников 6](#_Toc150472457)

[Приложение А 7](#_Toc150472458)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Средства синхронизации и взаимного исключения (Windows). Изучение и использование средств синхронизации и взаимного исключения.

Реализовать алгоритм философов обедающих, где философы (потоки) соревнуются за доступ к вилкам (ресурсам) с использованием мьютексов..

# **2** ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Переменные условия — это примитивы синхронизации, которые позволяют потокам ожидать выполнения определенного условия. Переменные условия — это объекты пользовательского режима, которые нельзя совместно использовать в процессах.

Переменные условия позволяют потокам атомарно освобождать блокировку и переходить в спящее состояние. Их можно использовать с критически важными разделами или с тонкими блоками чтения и записи (*SRW*). Переменные условия поддерживают операции, которые "пробуждение одного" или "пробуждение всех" ожидающих потоков. После пробуждения поток повторно получает блокировку, освобожденную при входе потока в спящее состояние.

Обратите внимание, что вызывающий объект должен выделить *CONDITION*\_*VARIABLE* структуру и инициализировать ее путем вызова Метода *InitializeConditionVariable* (для динамической инициализации структуры) или назначения константы *CONDITION*\_*VARIABLE*\_*INIT* переменной структуры (для статической инициализации структуры).[1]

Объект мьютекса можно использовать для защиты общего ресурса от одновременного доступа нескольких потоков или процессов. Каждый поток должен дождаться владения мьютексом, прежде чем он сможет выполнить код, который обращается к общему ресурсу. Например, если несколько потоков совместно используют доступ к базе данных, потоки могут использовать объект мьютекса, чтобы разрешить запись в базу данных только одному потоку за раз.[2]

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были реализованы следующие функции.

3.1 Ввод количества обедающих философов

Пользователю требуется ввести определенное количество философов, чтобы симулировать такое же количество потоков (рисунок 1).



Рисунок 1 – Ввод количества философов

3.2 Запуск обеда философов

После указания количества философов, пользователю требуется нажать кнопку *Start* *diner,* после чего выводится вся информация о потоках(рисунок 2).

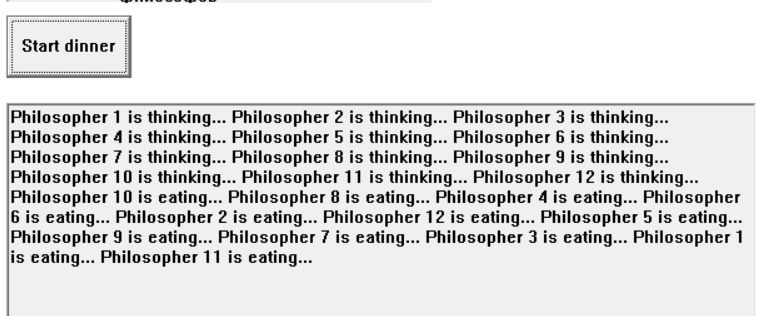


Рисунок 2 – Работа программы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Переменные условия [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/ru ru/windows/win32/sync/condition-variables>

[2] Использование объектов мьютекса – Электронные данные. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sync/using-mutex-objects>

ПРИЛОЖЕНИЕ А

Листинг кода

Файл Lab6.h

#pragma once

#include "resource.h"

#include <string>

#include <Windows.h>

#include <iostream>

#define START\_DINNER 1101

HWND hEditPhilosophersNumber;

HWND hEditLog;

HWND hButtonStartDinner;

int NUM\_PHILOSOPHERS;

HANDLE editMutex;

HANDLE\* forks;

HANDLE monitorMutex;

void PrintMessage(std::wstring message);

DWORD WINAPI philosopherThread(LPVOID param);

DWORD WINAPI startDinner(LPVOID param);

#define THINKING 1103

#define HUNGRY 1102

#define EATING 1104

int times = 1;

std::string my\_log;

class Monitor {

private:

int \*state;

CONDITION\_VARIABLE \*phcond;

CRITICAL\_SECTION condLock;

public:

Monitor();

~Monitor();

void test(int phnum);

void take\_fork(int phnum);

void put\_fork(int phnum);

};

Monitor \*phil\_object;

Файл Lab6.cpp

#include "Lab6.h"

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

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow)

{

// Register the window class.

const wchar\_t CLASS\_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

// Create the window.

HWND hwnd = CreateWindowExW(

0, // Optional window styles.

CLASS\_NAME, // Window class

L"Learn to Program Windows", // Window text

WS\_OVERLAPPEDWINDOW, // Window style

// Size and position

CW\_USEDEFAULT, CW\_USEDEFAULT, 1300, 700,

NULL, // Parent window

NULL, // Menu

hInstance, // Instance handle

NULL // Additional application data

);

if (hwnd == NULL)

{

return 0;

}

hEditPhilosophersNumber = CreateWindowExW(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

WS\_CHILD | WS\_VISIBLE | ES\_NUMBER,

10, 10, 100, 30,

hwnd,

(HMENU)1,

NULL,

NULL

);

HWND hPhilosophersNumberText = CreateWindowW(

L"STATIC",

L"Введите значение количества философов",

WS\_VISIBLE | WS\_CHILD | SS\_LEFT,

100, 10, 250, 30,

hwnd,

NULL,

hInstance,

NULL

);

if (hEditPhilosophersNumber == NULL) {

MessageBox(NULL, L"Edit Control Creation Failed!", L"Error", MB\_ICONEXCLAMATION | MB\_OK);

return 0;

}

hButtonStartDinner = CreateWindowExW(

0,

L"BUTTON",

L"Start dinner",

WS\_CHILD | WS\_VISIBLE | BS\_DEFPUSHBUTTON,

10, 50, 100, 50,

hwnd,

(HMENU)START\_DINNER,

hInstance,

NULL

);

if (hButtonStartDinner == NULL) {

MessageBox(NULL, L"Button Control Creation Failed!", L"Error", MB\_ICONEXCLAMATION | MB\_OK);

return 0;

}

hEditLog = CreateWindowExW(

WS\_EX\_CLIENTEDGE,

L"EDIT",

NULL,

WS\_CHILD | WS\_VISIBLE | ES\_MULTILINE | ES\_READONLY,

10, 120, 600, 500,

hwnd,

NULL,

NULL,

NULL

);

if (hEditLog == NULL) {

MessageBox(NULL, L"Edit Control Creation Failed!", L"Error", MB\_ICONEXCLAMATION | MB\_OK);

return -1;

}

ShowWindow(hwnd, nCmdShow);

// Run the message loop.

MSG msg = { };

while (GetMessage(&msg, NULL, 0, 0) > 0)

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

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

{

switch (uMsg)

{

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

// All painting occurs here, between BeginPaint and EndPaint.

FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 1));

EndPaint(hwnd, &ps);

}

case WM\_COMMAND:

{

int command = LOWORD(wParam);

switch (command)

{

case START\_DINNER:

{

editMutex = CreateMutex(NULL, FALSE, NULL);

wchar\_t buff[256];

GetWindowText(hEditPhilosophersNumber, buff, 256);

buff[255] = '\0';

NUM\_PHILOSOPHERS = std::stoi(buff);

if (NUM\_PHILOSOPHERS < 2)

{

MessageBox(NULL, L"Неверное количество!", L"Ошибка", MB\_ICONERROR);

break;

}

phil\_object = new Monitor();

HANDLE monitoringThread = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)startDinner, NULL, 0, NULL);

SetWindowTextW(hEditLog, L"");

if (monitoringThread == NULL) {

// Обработка ошибки

return 1;

}

break;

}

}

}

return 0;

}

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

}

DWORD WINAPI philosopherThread(LPVOID param)

{

int c = 0;

int i = \*(int\*)param;

while (c < times) {

Sleep(100);

phil\_object->take\_fork(i);

Sleep(500);

phil\_object->put\_fork(i);

c++;

}

return 0;

}

void PrintMessage(std::wstring message)

{

wchar\_t buff[3000];

GetWindowTextW(hEditLog, buff, 3000);

buff[2999] = '\0';

message = std::wstring(buff) + message + L' | ';

SetWindowTextW(hEditLog, message.c\_str());

}

DWORD WINAPI startDinner(LPVOID param)

{

HANDLE \*thread\_id = new HANDLE[NUM\_PHILOSOPHERS];

int \*phil = new int[NUM\_PHILOSOPHERS];

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

phil[i] = i;

}

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

thread\_id[i] = CreateThread(NULL, 0, philosopherThread, &phil[i], 0, NULL);

WaitForSingleObject(&editMutex, INFINITE);

PrintMessage(L"Philosopher " + std::to\_wstring(i + 1) + L" is thinking...");

ReleaseMutex(&editMutex);

}

WaitForMultipleObjects(NUM\_PHILOSOPHERS, thread\_id, TRUE, INFINITE);

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

CloseHandle(thread\_id[i]);

}

return 0;

}

Monitor::Monitor()

{

phcond = new CONDITION\_VARIABLE[NUM\_PHILOSOPHERS];

state = new int[NUM\_PHILOSOPHERS];

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

state[i] = THINKING;

InitializeConditionVariable(&phcond[i]);

}

InitializeCriticalSection(&condLock);

}

Monitor::~Monitor()

{

DeleteCriticalSection(&condLock);

}

void Monitor::test(int phnum) {

int right = (phnum + 1) % NUM\_PHILOSOPHERS;

int left = (phnum - 1) % NUM\_PHILOSOPHERS;

if (state[right] != EATING &&

state[left] != EATING &&

state[phnum] == HUNGRY) {

state[phnum] = EATING;

WakeConditionVariable(&phcond[phnum]);

}

}

void Monitor::take\_fork(int phnum) {

EnterCriticalSection(&condLock);

state[phnum] = HUNGRY;

test(phnum);

while (state[phnum] != EATING) {

SleepConditionVariableCS(&phcond[phnum], &condLock, INFINITE);

}

WaitForSingleObject(&editMutex, INFINITE);

PrintMessage(L"Philosopher " + std::to\_wstring(phnum + 1) + L" is eating...");

ReleaseMutex(&editMutex);

LeaveCriticalSection(&condLock);

}

void Monitor::put\_fork(int phnum) {

EnterCriticalSection(&condLock);

state[phnum] = THINKING;

test((phnum - 1) % NUM\_PHILOSOPHERS);

test((phnum + 1) % NUM\_PHILOSOPHERS);

LeaveCriticalSection(&condLock);

}