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 Матвеев М.А.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc148046360)

[2 Теоретические сведения 4](#_Toc148046361)

[3 Описание функций программы 5](#_Toc148046362)

[Список использованных источников 7](#_Toc148046363)

[Приложение А (обязательное) Листинг кода 8](#_Toc148046364)

# **1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является реализация алгоритма обедающих философов, где философы (потоки) соревнуются за доступ к вилкам (ресурсам) с использованием мьютексов и семофора.

# **2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Семафор (англ. *semaphore*) – примитив синхронизации работы процессов и потоков, в основе которого лежит счётчик, над которым можно производить две атомарные операции: увеличение и уменьшение значения на единицу, при этом операция уменьшения для нулевого значения счётчика является блокирующейся. Служит для построения более сложных механизмов синхронизации и используется для синхронизации параллельно работающих задач, для защиты передачи данных через разделяемую память, для защиты критических секций, а также для управления доступом к аппаратному обеспечению [1].

В разработанной программе для синхронизации доступа потоков к буферу товаров используются методы *WaitForSingleObject* и *ReleaseSemaphore*.

Функция *WaitForSingleObject* (*synchapi.h*) ожидает, пока указанный объект не перейдет в состояние сигнала или не истекает интервал времени ожидания [2].

Функция *ReleaseSemaphore* (*synchapi.h*) увеличивает количество указанного объекта семафора на указанную величину [3].

Для исключения доступа к буферу одновременно нескольких потоков используется мьютекс.

Мьютекс (англ. *mutex*, от *mutual exclusion* – взаимное исключение) – примитив синхронизации, обеспечивающий взаимное исключение исполнения критических участков кода [4].

Для управления мьютексами в разработанной программе используются метод *WaitForMultipleObjects*, который был описан ранее, и метод *ReleaseMutex*.

Функция *ReleaseMutex* (*synchapi.h*) освобождает владение указанным объектом мьютекса [5].

# **3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

При реализации алгоритма в качестве философов использовались потоки, которые ожидают какое-то количество времени (в это время философ думает), которое определяет генератор случайных чисел. После процесса мышления философы хотят есть. Для этого им нужно взять обе вилки (получить доступ к двум мьютексам). Для того чтобы избежать взаимной блокировки используется семафор, который ограничивает максимальное количество мьютексов, которыми в одно и то же время могут владеть потоки. Для того чтобы отслеживать процесс выполнения алгоритма использовался таймер, по истечению которого в консоль выводится текущее состояние каждого философа.

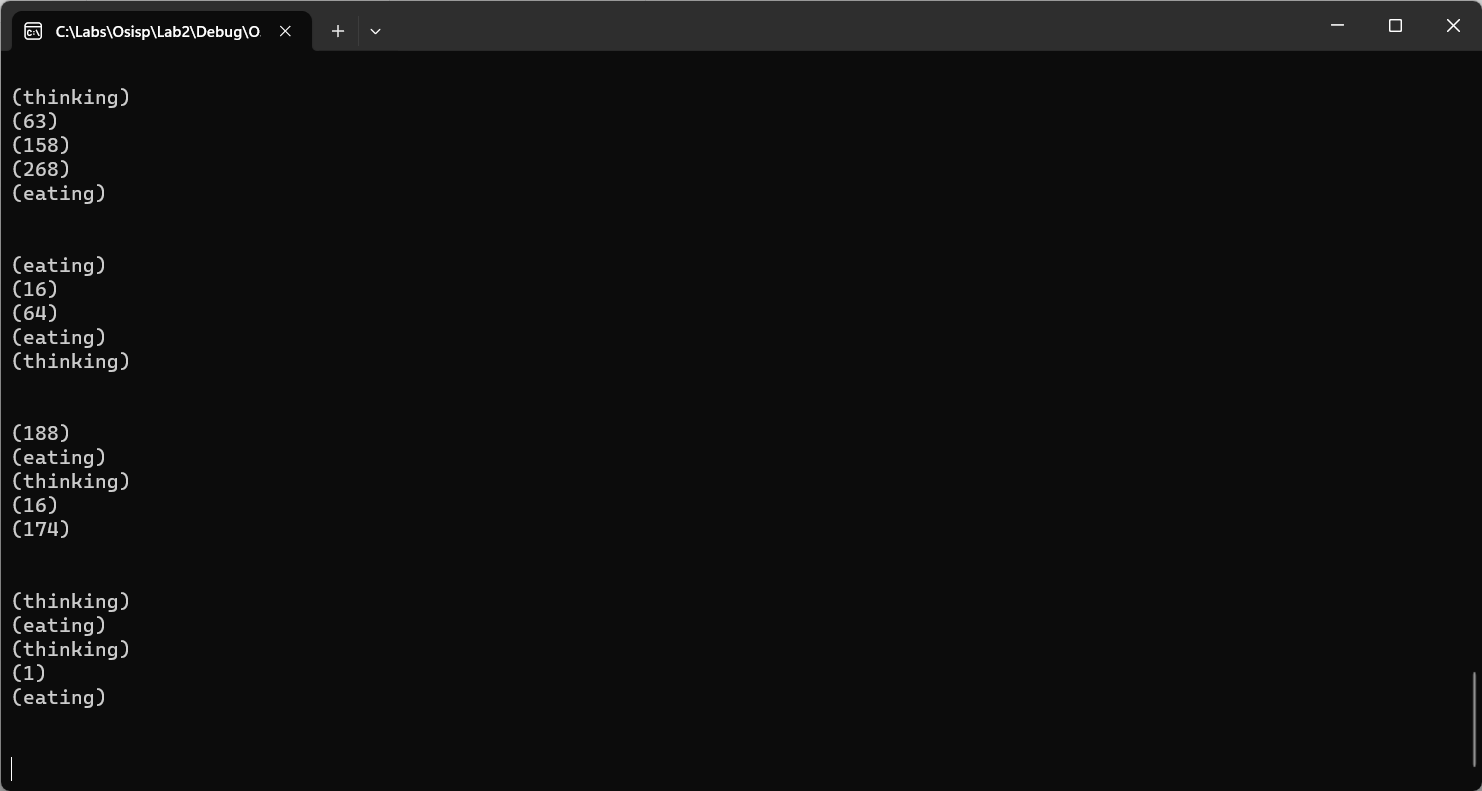


Рисунок 1 – Результат выполнения программы

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Семафор (программирование) [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Семафор_(программирование)> – Дата доступа: 11.11.2023.

[2] Функция WaitForSingleObject (synchapi.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject> – Дата доступа: 11.11.2023.

[3] Функция ReleaseSemaphore (synchapi.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/synchapi/nf-synchapi-releasesemaphore> – Дата доступа: 11.11.2023.

[4] Мьютекс [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Мьютекс> – Дата доступа: 12.11.2023.

[5] Функция ReleaseMutex (synchapi.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/synchapi/nf-synchapi-releasemutex> – Дата доступа: 12.11.2023.

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг 1 – Файл main.cpp

#include <windows.h>

#include <Windowsx.h>

#include <d2d1.h>

#include <Winuser.h>

#include <list>

#include <memory>

using namespace std;

#pragma comment(lib, "d2d1")

#pragma comment(lib, "Ntdll")

#pragma comment(lib, "Comctl32")

#include <objidl.h>

#include <gdiplus.h>

#include <winternl.h>

#include <CommCtrl.h>

#include <ctime>

#include <cstdlib>

#include <string>

#include <iostream>

HANDLE forksSemathore;

enum State

{

eating,

waiting,

thinking

};

struct philosopher

{

bool leftForkOwner;

HANDLE leftForkMutex;

State state;

clock\_t inactiveWaitingStart;

};

const long long MAX\_THINKING\_TIME = 1000000;

const long long MAX\_EATING\_TIME = 1000000;

LPCWSTR timer\_name = L"ttimer";

LPCWSTR semathore\_name = L"semathore";

size\_t PHILOSOPHERS\_COUNT = 5;

philosopher\* PHILOSOPHERS = new philosopher[PHILOSOPHERS\_COUNT]();

DWORD WINAPI MyThreadFunction(LPVOID lpParam)

{

size\_t id = (size\_t)lpParam;

DWORD thread\_id = GetCurrentThreadId();

philosopher& phil = PHILOSOPHERS[id];

philosopher& nextPhil = PHILOSOPHERS[(id + 1) % PHILOSOPHERS\_COUNT];

HANDLE timer = CreateWaitableTimerW(NULL, FALSE, (std::to\_wstring(id) + timer\_name).c\_str());

while (true)

{

phil.state = thinking;

LARGE\_INTEGER liDueTime;

liDueTime.QuadPart = -(long long)(rand()\*777 % MAX\_THINKING\_TIME);

SetWaitableTimer(timer, &liDueTime, NULL, NULL, NULL, FALSE);

WaitForSingleObject(timer, INFINITE);

phil.state = waiting;

phil.inactiveWaitingStart = clock();

CONST HANDLE handles[] = { forksSemathore, phil.leftForkMutex };

WaitForMultipleObjects(2, handles, TRUE, INFINITE);

phil.leftForkOwner = TRUE;

WaitForSingleObject(nextPhil.leftForkMutex, INFINITE);

nextPhil.leftForkOwner = FALSE;

phil.state = eating;

liDueTime.QuadPart = -(long long)(rand()\*777 % MAX\_EATING\_TIME);

SetWaitableTimer(timer, &liDueTime, NULL, NULL, NULL, FALSE);

WaitForSingleObject(timer, INFINITE);

phil.state = thinking;

ReleaseSemaphore(forksSemathore, 1, NULL);

ReleaseMutex(phil.leftForkMutex);

ReleaseMutex(nextPhil.leftForkMutex);

phil.leftForkOwner = FALSE;

}

CloseHandle(timer);

return 0;

}

void CALLBACK EverySecond(LPVOID lpArgToCompletionRoutine, DWORD dwTimerLowValue, DWORD dwTimerHighValue)

{

for (size\_t i = 0; i < PHILOSOPHERS\_COUNT; i++)

{

philosopher& phil = PHILOSOPHERS[i];

if (phil.state == waiting)

std::cout << "(" << clock() - phil.inactiveWaitingStart << ")";

else if (phil.state == thinking)

std::cout << "(thinking)";

else

std::cout << "(eating)";

std::cout << std::endl;

}

std::cout << std::endl << std::endl;

}

int main(int argc, char\*\* argv)

{

srand(time(NULL));

forksSemathore = CreateSemaphore(NULL, PHILOSOPHERS\_COUNT - 1, PHILOSOPHERS\_COUNT - 1, semathore\_name);

for (size\_t i = 0; i < PHILOSOPHERS\_COUNT; i++)

{

PHILOSOPHERS[i].state = thinking;

PHILOSOPHERS[i].leftForkMutex = CreateMutexW(NULL, NULL, std::to\_wstring(i).c\_str());

}

for (size\_t i = 0; i < PHILOSOPHERS\_COUNT; i++)

{

CreateThread(NULL, NULL, &MyThreadFunction, (LPVOID)i, NULL, NULL);

}

HANDLE timer = CreateWaitableTimerW(NULL, FALSE, timer\_name);

LARGE\_INTEGER liDueTime;

liDueTime.QuadPart = -1000LL;

SetWaitableTimer(timer, &liDueTime, 1000, &EverySecond, NULL, FALSE);

while (true)

{

SleepEx(INFINITE, TRUE);

}

return 0;

}