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 Богомолов 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 разработка многопоточного приложение для решения задачи производителей и потребителей, используя семафоры для управления доступом к буферу.

# **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 *WaitForSingleObject*, который был описан ранее, и метод *ReleaseMutex*.

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

# **3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

При запуске консольного приложения пользователь должен ввести количество производимых товаров, максимальный размер буфера производителя, время между выпусками единицы товара для производителя и время между потреблениями единицы товара для потребителя. Результат выполнения программы (Рисунок 1).

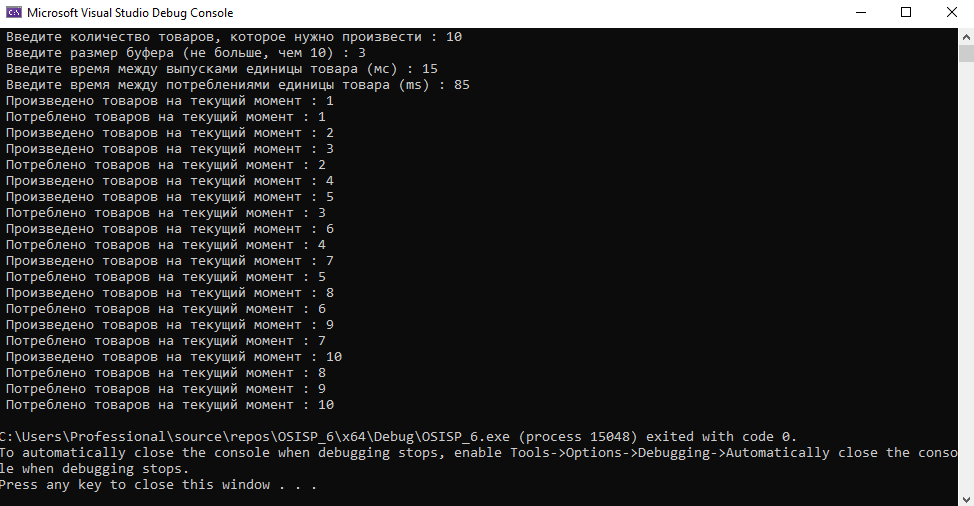


Рисунок 1 – Результат выполнения программы

# **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 OSISP\_6.cpp

#include <windows.h>

#include <iostream>

#include <queue>

std::queue<int> Buffer;

HANDLE semaphoreForConsumer;

HANDLE semaphoreForProducer;

HANDLE mutexForBuffer;

int B\_SIZE, P\_SLEEP, C\_SLEEP, I\_COUNT;

int remainingItems;

DWORD WINAPI ToProduce(LPVOID);

DWORD WINAPI ToConsume(LPVOID);

int main() {

setlocale(LC\_ALL, "Russian");

DWORD ProducerThreadId, ConsumerThreadId;

std::cout << "Введите количество товаров, которое нужно произвести : ";

std::cin >> I\_COUNT;

std::cout << "Введите размер буфера (не больше, чем " << I\_COUNT << ") : ";

std::cin >> B\_SIZE;

std::cout << "Введите время выпусками единицы товара (мс) : ";

std::cin >> P\_SLEEP;

std::cout << "Введите время между потреблениями единицы товара (ms) : ";

std::cin >> C\_SLEEP;

if (B\_SIZE > I\_COUNT) {

std::cout << "Размер буфера не может быть больше количества вещей, которое нужно произвести.";

return 1;

}

semaphoreForConsumer = CreateSemaphore(NULL, 0, B\_SIZE, NULL);

semaphoreForProducer = CreateSemaphore(NULL, B\_SIZE, B\_SIZE, NULL);

mutexForBuffer = CreateMutex(NULL, FALSE, NULL);

remainingItems = I\_COUNT;

HANDLE hProducer = CreateThread(NULL, 0, ToProduce, NULL, 0, &ProducerThreadId);

HANDLE hConsumer = CreateThread(NULL, 0, ToConsume, NULL, 0, &ConsumerThreadId);

WaitForSingleObject(hProducer, INFINITE);

WaitForSingleObject(hConsumer, INFINITE);

CloseHandle(semaphoreForConsumer);

CloseHandle(semaphoreForProducer);

CloseHandle(mutexForBuffer);

CloseHandle(hProducer);

CloseHandle(hConsumer);

return 0;

}

DWORD WINAPI ToProduce(LPVOID lpParam) {

for (int i = 0; i < I\_COUNT; ++i) {

WaitForSingleObject(semaphoreForProducer, INFINITE);

WaitForSingleObject(mutexForBuffer, INFINITE);

Buffer.push(i);

std::cout << "Произведено товаров на текущий момент : " << i + 1 << std::endl;

ReleaseMutex(mutexForBuffer);

ReleaseSemaphore(semaphoreForConsumer, 1, NULL);

Sleep(P\_SLEEP);

}

return 0;

}

DWORD WINAPI ToConsume(LPVOID lpParam) {

while (true) {

WaitForSingleObject(semaphoreForConsumer, INFINITE);

WaitForSingleObject(mutexForBuffer, INFINITE);

if (!Buffer.empty()) {

int item = Buffer.front();

Buffer.pop();

std::cout << "Потреблено товаров на текущий момент : " << item + 1 << std::endl;

remainingItems--;

ReleaseMutex(mutexForBuffer);

ReleaseSemaphore(semaphoreForProducer, 1, NULL);

}

else {

ReleaseMutex(mutexForBuffer);

}

Sleep(C\_SLEEP);

if (remainingItems == 0) {

break;

}

}

return 0;

}