

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 Миненков Я.А.

Проверил ассистент кафедры информатики  
Гриценко Н.Ю.

Минск 2023

## СОДЕРЖАНИЕ

1 Формулировка задачи .....	3
2 Теоретические сведения .....	4
3 Описание функций программы.....	5
Список использованных источников .....	6
Приложение А (обязательное) Листинг кода .....	7

## **1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является разработка многопоточного приложения для решения задачи производителей и потребителей, используя семафоры для управления доступом к буферу.

## 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Многопоточность (*multithreading*) – свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

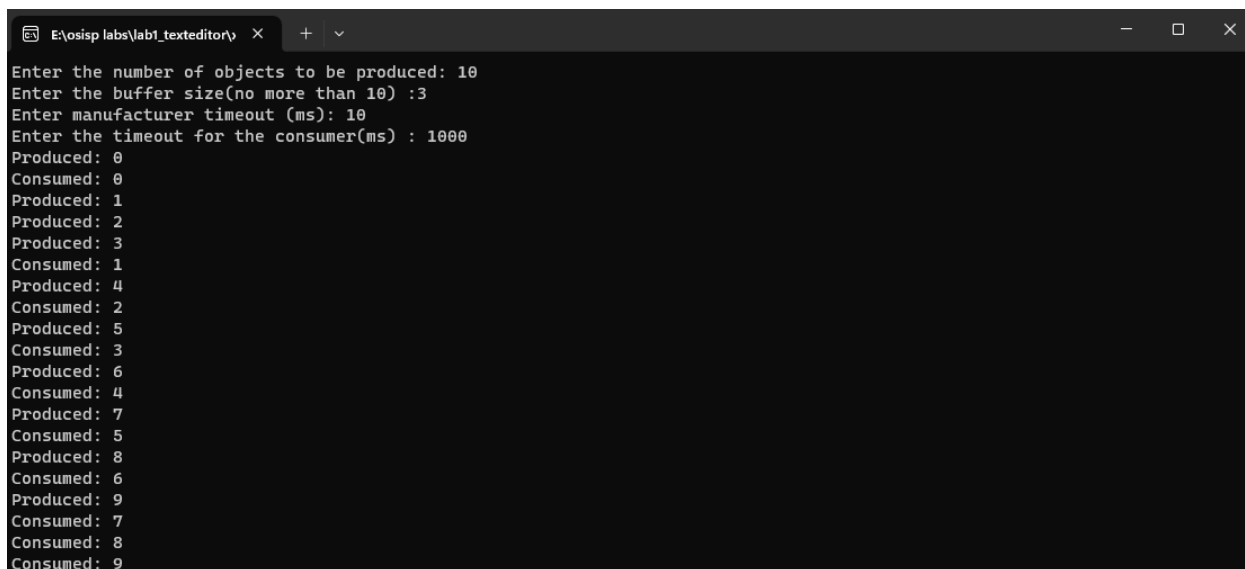
Семафор (*semaphore*) – примитив синхронизации работы процессов и потоков, в основе которого лежит счётчик, над которым можно производить две атомарные операции: увеличение и уменьшение значения на единицу, при этом операция уменьшения для нулевого значения счётчика является блокирующей. Служит для построения 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.

Мьютексные семафоры(мьютексы) являются упрощённой реализацией семафоров, аналогичной двоичным семафорам с тем отличием, что мьютексы должны отпускаться тем же потоком, который осуществляет их захват. [1]

Для реализации приложения были использованы следующие функции WinAPI: *CreateThread()* для создания нового потока внутри процесса, *CloseHandle()* для закрытия дескрипторов потока. Функции семафоров и мьютексов: *CreateSemaphore()* – создает семафор или открывает уже существующий, *ReleaseSemaphore()* – увеличивает счетчик семафора на указанное количество, *CreateMutex()* – создает мьютекс или открывает уже существующий, *ReleaseMutex()* – освобождает мьютекс, делая его доступным для других потоков. Функции контроля потоков: *WaitForSingleObject()* – ожидает, пока указанный объект не перейдет в сигнальное состояние, *Sleep()* – приостанавливает выполнение текущего потока на заданное число миллисекунд. [2]

### 3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

При запуске консольного приложения пользователь должен ввести количество производимых объектов, максимальный размер буфера, время ожидания для производителя и потребителя (Рисунок 1).



```
E:\osisp labs\lab1_texteditor\ X + v
Enter the number of objects to be produced: 10
Enter the buffer size(no more than 10) :3
Enter manufacturer timeout (ms): 10
Enter the timeout for the consumer(ms) : 1000
Produced: 0
Consumed: 0
Produced: 1
Produced: 2
Produced: 3
Consumed: 1
Produced: 4
Consumed: 2
Produced: 5
Consumed: 3
Produced: 6
Consumed: 4
Produced: 7
Consumed: 5
Produced: 8
Consumed: 6
Produced: 9
Consumed: 7
Consumed: 8
Consumed: 9
```

Рисунок 1 – Пример выполнения программы

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] *Multithreading* [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://habr.com/ru/companies/otus/articles/549814/>
- [2] *Synchapi.h header - Win32 apps* [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/synchapi/>

## ПРИЛОЖЕНИЕ А

### (обязательное)

### Листинг кода

#### Листинг 1 – Файл lab6.cpp

```
#include <windows.h>
#include <iostream>
#include <queue>

bool isClosed = false;

std::queue<int> buffer;
HANDLE semaphoreFull;
HANDLE semaphoreEmpty;
HANDLE mutexBuffer;
int buffer_size, sleepProducer, sleepConsumer, itemsToProduce;

DWORD WINAPI Producer(LPVOID);
DWORD WINAPI Consumer(LPVOID);

int main() {
    DWORD producerId, consumerId;

    std::cout << "Enter the number of objects to be produced: ";
    std::cin >> itemsToProduce;
    std::cout << "Enter the buffer size(no more than " << itemsToProduce << ") :";
    std::cin >> buffer_size;
    std::cout << "Enter manufacturer timeout (ms): ";
    std::cin >> sleepProducer;
    std::cout << "Enter the timeout for the consumer(ms) : ";
    std::cin >> sleepConsumer;

    if (buffer_size > itemsToProduce) {
        std::cout << "The buffer size cannot be greater than the number of objects produced!";
        return 1;
    }

    semaphoreFull = CreateSemaphore(NULL, 0, buffer_size, NULL);
    semaphoreEmpty = CreateSemaphore(NULL, buffer_size, buffer_size, NULL);
    mutexBuffer = CreateMutex(NULL, FALSE, NULL);

    HANDLE hProducer = CreateThread(NULL, 0, Producer, NULL, 0, &producerId);
    HANDLE hConsumer = CreateThread(NULL, 0, Consumer, NULL, 0, &consumerId);

    WaitForSingleObject(hProducer, INFINITE);
    WaitForSingleObject(hConsumer, INFINITE);

    CloseHandle(semaphoreFull);
    CloseHandle(semaphoreEmpty);
    CloseHandle(mutexBuffer);
    CloseHandle(hProducer);
    CloseHandle(hConsumer);

    return 0;
}

DWORD WINAPI Producer(LPVOID lpParam) {
    for (int i = 0; i < itemsToProduce; ++i) {
        Sleep(sleepProducer);
```

```

        WaitForSingleObject(semaphoreEmpty, INFINITE);

        WaitForSingleObject(mutexBuffer, INFINITE);
        buffer.push(i);
        std::cout << "Produced: " << i << std::endl;
        ReleaseMutex(mutexBuffer);

        ReleaseSemaphore(semaphoreFull, 1, NULL);
    }

    isClosed = true;

    return 0;
}

DWORD WINAPI Consumer(LPVOID lpParam) {
    while (true) {
        WaitForSingleObject(semaphoreFull, INFINITE);

        WaitForSingleObject(mutexBuffer, INFINITE);
        if (!buffer.empty()) {
            int item = buffer.front();
            buffer.pop();
            std::cout << "Consumed: " << item << std::endl;
            ReleaseMutex(mutexBuffer);

            ReleaseSemaphore(semaphoreEmpty, 1, NULL);
        }
        else if (isClosed) {
            ReleaseMutex(mutexBuffer);
            break;
        }
        else {
            ReleaseMutex(mutexBuffer);
        }

        Sleep(sleepConsumer);
    }

    return 0;
}

```