|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отчет**

***к лабораторной работе №6 по курсу «Операционные системы»***

***по теме «Реализация монитора Хоара “Читатели-писатели” под ОС Windows»***

|  |  |
| --- | --- |
| **Студент: Русинова Д. Э.**  **Группа: ИУ7-55б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель: Рязанова Н. Ю.** |  |

Москва.

1. г.

Код реализация монитора Хоара “Читатели-писатели” под ОС Windows.

#include <stdio.h>

#include <stdbool.h>

#include <windows.h>

// кол-во читателей и писателей

#define WRITERS\_CNT 3

#define READERS\_CNT 5

// глобальная переменная, разделяемая между потоками

volatile int resource = 0;

// мьютекс, чтобы обеспечить

// монопольный доступ активного писателя

// к разд. переменной

HANDLE resource\_mutex;

// активный писатель - логическая переменная

volatile bool ACTIVE\_WRITER = false;

// счетчик активных читателей

volatile int ACTIVE\_READERS = 0;

// счетчик ожидающих писателей

volatile int WAITING\_WRITERS = 0;

// счетчик ожидающих читателй

volatile int WAITING\_READERS = 0;

// объекты событий

HANDLE can\_read\_event;

HANDLE can\_write\_event;

// массив указателей на объекты потоков-читателй

// передается в аргументы WaitForMultipleObjects

HANDLE readers[READERS\_CNT];

// массив указателей на объекты потоков-писателей

// передается в аргументы WaitForMultipleObjects

HANDLE writers[WRITERS\_CNT];

void start\_writing()

{

    // запрашиваем монопольный доступ к разделяемому ресурсу

    // программа работает и без мьютекса, он искуственно включен в

    // программу для ознакомления с функцией мьютекса

    WaitForSingleObject(resource\_mutex, INFINITE);

    // писатель становится ожидающим

    InterlockedIncrement(&WAITING\_WRITERS);

    // если есть активный писатель или активные читатели,

    // поток блокируется на событии

    if (ACTIVE\_WRITER || ACTIVE\_READERS)

        WaitForSingleObject(can\_write\_event, INFINITE);

    // писатель стал активным

    InterlockedDecrement(&WAITING\_WRITERS);

    ACTIVE\_WRITER = true;

    // другие писатели не могут писать

    ResetEvent(can\_write\_event);

    ReleaseMutex(resource\_mutex);

}

void stop\_writing()

{

    ACTIVE\_WRITER = false;

    if (WAITING\_WRITERS)

    {

        // освобождаем событие, на котором

        // заблокированы ожидающие писатели

        SetEvent(can\_write\_event);

    }

    else

    {

        // иначе осв событие, на кототором

        // заблок. читатели

        SetEvent(can\_read\_event);

    }

}

void start\_reading()

{

    // неделимый инкремент счетчика ожидающих писателей

    InterlockedIncrement(&WAITING\_READERS);

    // если есть активный писатель или ожидающие писатели

    // поток блокируется на событии

    if (ACTIVE\_WRITER || WAITING\_WRITERS)

        WaitForSingleObject(can\_read\_event, INFINITE);

    // декремент ожидающих читателей

    InterlockedDecrement(&WAITING\_READERS);

    // инкремент активных читателей

    InterlockedIncrement(&ACTIVE\_READERS);

}

void stop\_reading()

{

    // Когда число читателей равно 0,

    // процесс писатель получит возможность начать работу

    InterlockedDecrement(&ACTIVE\_READERS);

    if (ACTIVE\_READERS == 0)

        SetEvent(can\_write\_event);

}

DWORD WINAPI writer(LPVOID lpParams)

{

    while(resource < 25)

    {

        start\_writing();

        resource++;

        printf("\t\t\tWriter # %d increased resource: %d\n", (int) lpParams, resource);

        stop\_writing();

        Sleep(200);

    }

    return EXIT\_SUCCESS;

}

DWORD WINAPI reader(LPVOID lpParams)

{

    while (resource <= 25)

    {

        start\_reading();

        printf("Reader # %d read resource: %d\n", (int) lpParams, resource);

        stop\_reading();

        if (resource == 25)

            return EXIT\_SUCCESS;

        Sleep(200);

    }

    return EXIT\_SUCCESS;

}

int main(void)

{

    // создать мьютекс

    if ((resource\_mutex = CreateMutex(NULL, FALSE, NULL)) == NULL)

    {

        printf("Failed to create resource\_mutex");

        return EXIT\_FAILURE;

    }

    // создать событие can\_read\_event

    if ((can\_read\_event = CreateEvent(NULL, FALSE, TRUE, NULL)) == NULL)

    {

        CloseHandle(resource\_mutex);

        printf("Failed to create can\_read\_event event");

        return EXIT\_FAILURE;

    }

    // создать событие can\_write\_event

    if ((can\_write\_event = CreateEvent(NULL, TRUE, TRUE, NULL)) == NULL)

    {

        CloseHandle(resource\_mutex);

        CloseHandle(can\_read\_event);

        printf("Failed to create can\_write\_event event");

        return EXIT\_FAILURE;

    }

    // создать потоки-писатели

    for (int i = 0; i < WRITERS\_CNT; i++)

    {

        if ((writers[i] = CreateThread(NULL, 0, writer, (LPVOID) i, 0, NULL)) == NULL)

        {

            printf("Failed to create writer thread");

            return EXIT\_FAILURE;

        }

    }

    // создать потоки-читатели

    for (int i = 0; i < READERS\_CNT; i++)

    {

        if ((readers[i] = CreateThread(NULL, 0, reader, (LPVOID) i, 0, NULL)) == NULL)

        {

            printf("Failed to create reader thread");

            return EXIT\_FAILURE;

        }

    }

    // Ожидаем завершения потоков

    // в параметрах передаем кол-во объектов HANDLE,

    // указатель на массив объектов HANDLE,

    // флаг: TRUE=ожидать всех, FALSE=ожидать хотя бы одного

    // временной тайм-аут в миллисекундах: INFINITE=тайм аут не определен

    WaitForMultipleObjects(WRITERS\_CNT, writers, TRUE, INFINITE);

    WaitForMultipleObjects(READERS\_CNT, readers, TRUE, INFINITE);

    // освобождаем объекты ядра

    CloseHandle(resource\_mutex);

    CloseHandle(can\_read\_event);

    CloseHandle(can\_write\_event);

    return EXIT\_SUCCESS;

}

Демонстрация работы программы приведена на следующем рисунке.

