|  |  |
| --- | --- |
| 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»

**Лабораторная работа 6**

|  |  |
| --- | --- |
| **Тема: «Реализация монитора Хоара «Читатели-писатели» под ОС Windows»**  **Студент** Гарасев Н.А.  **Группа** ИУ7-52Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Рязанова Н.Ю. |  |

Москва.

2021 г.

**В лабораторной работе необходимо разработать многопоточное приложение, используя API ОС Windows такие как, потоки, события (event) и мьютексы (mutex). Потоки разделяют единственную глобальную переменную. Приложение реализует монитор Хоара «Читатели-писатели».**

#include <stdio.h>

#include <stdbool.h>

#include <windows.h>

#define READERS 5

#define WRITERS 3

#define ITERS 7

HANDLE mutex;

HANDLE can\_read;

HANDLE can\_write;

int waiting\_writers = 0;

int waiting\_readers = 0;

int active\_readers = 0;

bool is\_writer\_active = false;

int value = 0;

void start\_read()

{

    InterlockedIncrement(&waiting\_readers);

    if (waiting\_writers > 0 || is\_writer\_active)

    {

        WaitForSingleObject(can\_read, INFINITE);

    }

    WaitForSingleObject(mutex, INFINITE);

    InterlockedDecrement(&waiting\_readers);

    InterlockedIncrement(&active\_readers);

    SetEvent(can\_read);

    ReleaseMutex(mutex);

}

void stop\_read()

{

    InterlockedDecrement(&active\_readers);

    if (waiting\_readers == 0)

    {

        SetEvent(can\_write);

    }

}

void start\_write()

{

    InterlockedIncrement(&waiting\_writers);

    if (active\_readers > 0 || is\_writer\_active)

    {

        WaitForSingleObject(can\_write, INFINITE);

    }

    InterlockedDecrement(&waiting\_writers);

    is\_writer\_active = true;

    ResetEvent(can\_write);

}

void stop\_write(void)

{

    is\_writer\_active = false;

    if (!waiting\_writers)

    {

        SetEvent(can\_read);

    }

    else

    {

        SetEvent(can\_write);

    }

}

DWORD WINAPI reader()

{

    while (value < WRITERS \* ITERS)

    {

        start\_read();

        printf("Reader #%ld read value: %d\n", GetCurrentThreadId(), value);

        stop\_read();

        Sleep(1000);

    }

    return 0;

}

DWORD WINAPI writer()

{

    for (int i = 0; i < ITERS; i++)

    {

        start\_write();

        value++;

        printf("Writer #%ld wrote value: %d\n", GetCurrentThreadId(), value);

        stop\_write();

        Sleep(2000);

    }

    return 0;

}

int main(void)

{

    HANDLE wThread[WRITERS];

    HANDLE rThread[READERS];

    if ((mutex = CreateMutex(NULL, FALSE, NULL)) == NULL)

    {

        perror("CreateMutex");

        return EXIT\_FAILURE;

    }

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

    {

        perror("CreateEvent can\_read");

        return EXIT\_FAILURE;

    }

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

    {

        perror("CreateEvent can write");

        return EXIT\_FAILURE;

    }

    for (int i = 0; i < WRITERS; i++)

    {

        wThread[i] = CreateThread(NULL, 0, &writer, NULL, 0, NULL);

        if (wThread[i] == NULL)

        {

            perror("Couldn't create thread");

            return EXIT\_FAILURE;

        }

    }

    for (int i = 0; i < READERS; i++)

    {

        rThread[i] = CreateThread(NULL, 0, &reader, NULL, 0, NULL);

        if (rThread[i] == NULL)

        {

            perror("Couldn't create thread");

            return EXIT\_FAILURE;

        }

    }

    WaitForMultipleObjects(WRITERS, wThread, TRUE, INFINITE);

    WaitForMultipleObjects(READERS, rThread, TRUE, INFINITE);

    CloseHandle(mutex);

    CloseHandle(can\_read);

    CloseHandle(can\_write);

    return EXIT\_SUCCESS;

}





