|  |  |
| --- | --- |
| 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»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ДИСЦИПЛИНА «Операционные системы»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 6**

|  |  |
| --- | --- |
| **Тема** «Реализация монитора Хоара «Читатели-писатели» под ОС Windows»  **Студент** Садулаева Т. Р.  **Группа** ИУ7-54Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Рязанова Н.Ю. |  |

Москва.

2020 г.

**Задание**

Написать программу, реализующую задачу «Читатели – писатели» по монитору Хоара с четырьмя функциями: Начать\_чтение, Закончить\_чтение, Начать\_запись, Закончить\_запись. В программе всеми процессами разделяется одно единственное значение в разделяемой памяти. Писатели ее только инкрементируют, читатели могут только читать значение.

**Код программы**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <windows.h>

#define READERS\_COUNT 5

#define WRITERS\_COUNT 3

#define READ\_ITERS 7

#define WRITE\_ITERS 8

HANDLE mutex;

HANDLE can\_read;

HANDLE can\_write;

LONG waiting\_writers = 0;

LONG waiting\_readers = 0;

LONG active\_readers = 0;

bool active\_writer = false;

int val = 0;

void start\_read()

{

InterlockedIncrement(&waiting\_readers);

if (active\_writer || (WaitForSingleObject(can\_write, 0) == WAIT\_OBJECT\_0 && waiting\_writers))

{

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 (active\_readers == 0)

{

ResetEvent(can\_read);

SetEvent(can\_write);

}

}

void start\_write(void)

{

InterlockedIncrement(&waiting\_writers);

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

{

WaitForSingleObject(can\_write, INFINITE);

}

InterlockedDecrement(&waiting\_writers);

active\_writer = true;

}

void stop\_write(void)

{

active\_writer = false;

if (waiting\_readers)

{

SetEvent(can\_read);

}

else

{

SetEvent(can\_write);

}

}

DWORD WINAPI rr\_run(CONST LPVOID lpParams)

{

int r\_id = (int)lpParams;

for (size\_t i = 0; i < READ\_ITERS; i++)

{

start\_read();

printf("?Reader #%d read: %3d", r\_id, val);

stop\_read();

}

return 0;

}

DWORD WINAPI wr\_run(CONST LPVOID lpParams)

{

int w\_id = (int)lpParams;

for (size\_t i = 0; i < WRITE\_ITERS; ++i)

{

start\_write();

++val;

printf("!Writer #%d wrote: %3d", w\_id, val);

stop\_write();

}

return 0;

}

int **main()**

{

setbuf(stdout, NULL);

HANDLE readers\_threads[READERS\_COUNT];

HANDLE writers\_threads[WRITERS\_COUNT];

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

{

perror("Failed call of CreateMutex");

return -1;

}

can\_read = CreateEvent(NULL, FALSE, FALSE, NULL);

can\_write = CreateEvent(NULL, FALSE, FALSE, NULL);

if (can\_read == NULL || can\_write == NULL)

{

perror("Failed call of CreateEvent");

return -1;

}

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

{

readers\_threads[i] = CreateThread(NULL, 0, rr\_run, (LPVOID)i, 0, NULL);

if (readers\_threads[i] == NULL)

{

perror("Failed call of CreateThread");

return -1;

}

}

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

{

writers\_threads[i] = CreateThread(NULL, 0, wr\_run, (LPVOID)i, 0, NULL);

if (writers\_threads[i] == NULL)

{

perror("Failed call of CreateThread");

return -1;

}

}

WaitForMultipleObjects(READERS\_COUNT, readers\_threads, TRUE, INFINITE);

WaitForMultipleObjects(WRITERS\_COUNT, writers\_threads, TRUE, INFINITE);

CloseHandle(mutex);

CloseHandle(can\_read);

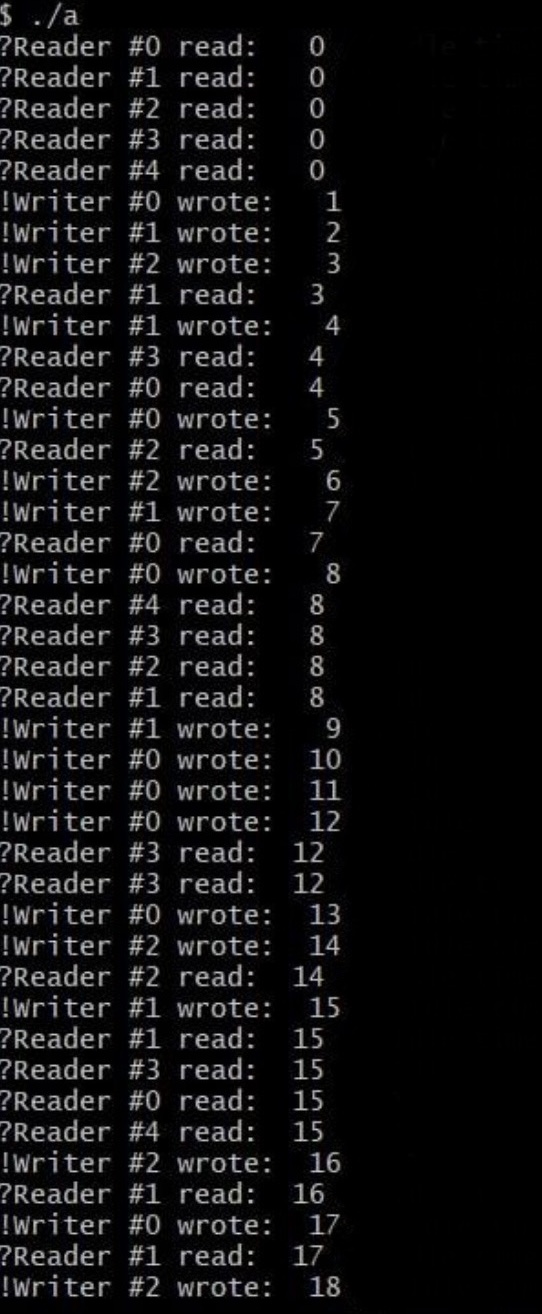
CloseHandle(can\_write);

return 0;

}

**Пример работы программы**

Пример работы первой программы представлен на рисунке 1 ниже.

**

*Рисунок 1. – Пример работы программы.*