|  |  |
| --- | --- |
| **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**

**по лабораторной работе № 6**

**по курсу «Операционные системы»**

|  |  |
| --- | --- |
| **Тема** “Реализация монитора Хоара «Читатели-писатели» под ОС Windows”  **Студент** Блохин Д. М.  **Группа** ИУ7-52Б  **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.

**Исходный код:**

#include <stdio.h>

#include <stdbool.h>

#include <windows.h>

#define MILLISECONDS 1E3

#define COUNT\_OF\_READERS 5

#define COUNT\_OF\_WRITERS 3

#define MAX\_NUMBER 15

#define RANDOM\_SLEEPING\_READER ((rand() % 4) + 1)

#define RANDOM\_SLEEPING\_WRITER ((rand() % 3) + 1)

HANDLE writers[COUNT\_OF\_WRITERS];

HANDLE readers[COUNT\_OF\_READERS];

volatile int shared\_value = 0;

HANDLE mutex;

HANDLE can\_read;

HANDLE can\_write;

volatile LONG waiting\_writers = 0;

volatile LONG waiting\_readers = 0;

volatile LONG active\_readers = 0;

bool active\_writer = false;

void start\_writing(void) {

InterlockedIncrement(&waiting\_writers);

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

WaitForSingleObject(can\_write, INFINITE);

}

InterlockedDecrement(&waiting\_writers);

active\_writer = true;

ResetEvent(can\_write);

}

void stop\_writing(void) {

active\_writer = false;

if (!waiting\_writers) { SetEvent(can\_read); }

else { SetEvent(can\_write); }

}

DWORD WINAPI writer\_function(LPVOID reader\_id) {

bool is\_over = false;

while(!is\_over) {

tart\_writing();

if (shared\_value != MAX\_NUMBER) {

shared\_value++;

printf("Writer #%ld \t wrote value: \t %ld\n", (long) reader\_id, shared\_value);

} else { is\_over = true; }

stop\_writing();

Sleep(RANDOM\_SLEEPING\_READER \* MILLISECONDS);

}

return EXIT\_SUCCESS;

}

void start\_reading(void) {

InterlockedIncrement(&waiting\_readers);

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

WaitForSingleObject(can\_read, INFINITE);

}

WaitForSingleObject(mutex, INFINITE);

InterlockedDecrement(&waiting\_readers);

InterlockedIncrement(&active\_readers);

SetEvent(can\_read);

ReleaseMutex(mutex);

}

void stop\_reading(void) {

InterlockedDecrement(&active\_readers);

if (waiting\_readers == 0) {

SetEvent(can\_write);

}

}

DWORD WINAPI reader\_function(LPVOID reader\_id) {

bool is\_over = false;

while (!is\_over) {

start\_reading();

printf("Reader #%ld \t read value: \t %d\n", (long) reader\_id, shared\_value);

if (shared\_value == MAX\_NUMBER) {

is\_over = true;

}

stop\_reading();

Sleep(RANDOM\_SLEEPING\_READER \* MILLISECONDS);

}

return EXIT\_SUCCESS;

}

int init\_threads(HANDLE \*threads, int threads\_count, DWORD (\*thread\_function)(LPVOID)) {

for (long i = 0; i < threads\_count; ++i) {

if ((threads[i] = CreateThread(NULL, 0, thread\_function, (LPVOID) i, 0, NULL)) == NULL) {

fprintf(stderr, "init\_threads::CreateThread Error!\n");

return EXIT\_FAILURE;

}

}

return EXIT\_SUCCESS;

}

int main(void) {

setbuf(stdout, NULL);

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

perror("CreateMutex");

return EXIT\_FAILURE;

}

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

perror("CreateEvent");

return EXIT\_FAILURE;

}

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

perror("CreateEvent");

return EXIT\_FAILURE;

}

init\_threads(writers, COUNT\_OF\_WRITERS, writer\_function);

init\_threads(readers, COUNT\_OF\_READERS, reader\_function);

WaitForMultipleObjects(COUNT\_OF\_WRITERS, writers, TRUE, INFINITE);

WaitForMultipleObjects(COUNT\_OF\_READERS, readers, TRUE, INFINITE);

CloseHandle(mutex);

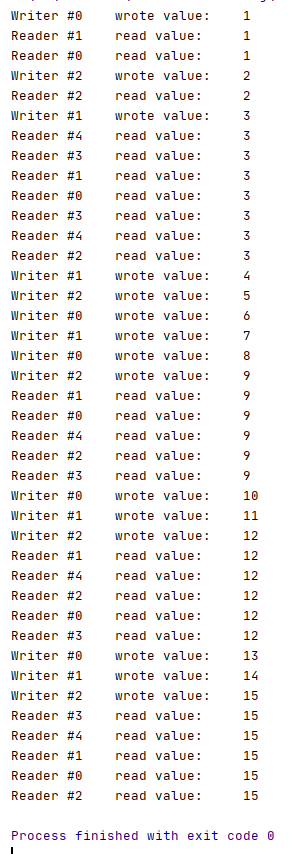
CloseHandle(can\_read);

CloseHandle(can\_write);

return 0;

}

**Пример работы программы:**

****