|  |  |
| --- | --- |
| **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»  **Студент** Шиленков А. А.  **Группа** ИУ7-55Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Рязанова Н.Ю. |  |

Москва.

2020 г.

**Цели и задачи**

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».

**Код программы:**

// Читатели писатели

#include <windows.h>

#include <stdbool.h>

#include <stdio.h>

#include <time.h>

#include <stdbool.h>

enum counts {

CNT\_ITERS = 5,

READERS\_NUMBER = 4,

WRITERS\_NUMBER = 3

};

enum errors {

OK = 0,

CREATE\_MUTEX\_ERROR =-4,

CREATE\_EVENT\_ERROR,

CREATE\_READER\_THREAD\_ERROR,

CREATE\_WRITER\_THREAD\_ERROR

};

enum delays {

MINIMUM\_READER\_DELAY = 100,

MINIMUM\_WRITER\_DELAY = 100,

MAXIMUM\_READER\_DELAY = 200,

MAXIMUM\_WRITER\_DELAY = 200

};

HANDLE canRead;

HANDLE canWrite;

HANDLE mutex;

HANDLE readerThreads[READERS\_NUMBER];

HANDLE writerThreads[WRITERS\_NUMBER];

int readersID[READERS\_NUMBER];

int writersID[WRITERS\_NUMBER];

int readersRand[READERS\_NUMBER \* CNT\_ITERS];

int writersRand[READERS\_NUMBER \* CNT\_ITERS];

int value = 0;

LONG WWCount = 0;

LONG WRCount = 0;

LONG ARCount = 0;

bool writing = false;

void StartRead()

{

InterlockedIncrement(&WRCount); // WRCount++;

// Читатель может начать, если нет активного писателя и писателей в очереди

if (writing || WaitForSingleObject(canWrite, 0) == WAIT\_OBJECT\_0)

WaitForSingleObject(canRead, INFINITE);

WaitForSingleObject(mutex, INFINITE);

InterlockedDecrement(&WRCount); // WRCount--;

InterlockedIncrement(&ARCount); // ARCount++;

SetEvent(canRead);

ReleaseMutex(mutex);

}

void StopRead()

{

InterlockedDecrement(&ARCount); // ARCount--;

if (!ARCount)

SetEvent(canWrite);

}

void StartWrite()

{

InterlockedIncrement(&WWCount); // WW++;

// Писатель может начать, если нет читающик читателей и нет активного писателя

if (ARCount > 0 || writing)

WaitForSingleObject(canWrite, INFINITE);

InterlockedDecrement(&WWCount); // WW--;

writing = true;

}

void StopWrite()

{

writing = false;

if (WRCount)

SetEvent(canRead);

else

SetEvent(canWrite);

}

DWORD WINAPI Reader(CONST LPVOID param)

{

int id = \*(int \*)param;

int sleepTime;

int begin = id \* CNT\_ITERS;

for (int i = 0; i < CNT\_ITERS; i++)

{

sleepTime = readersRand[begin + i];

StartRead();

printf("READER {ID = %d, VALUE = %d, TIME = %d}\n", id, value, sleepTime);

StopRead();

Sleep(sleepTime);

}

}

DWORD WINAPI Writer(CONST LPVOID param)

{

int id = \*(int \*)param;

int sleepTime;

int begin = id \* CNT\_ITERS;

for (int i = 0; i < CNT\_ITERS; i++)

{

sleepTime = writersRand[begin + i];

StartWrite();

++value;

printf("\t{ID = %d, VALUE = %d, TIME = %d} WRITER\n", id, value, sleepTime);

StopWrite();

Sleep(sleepTime);

}

}

int main(void)

{

setbuf(stdout, NULL);

srand(time(NULL));

for (int i = 0; i < READERS\_NUMBER \* CNT\_ITERS; i++)

readersRand[i] = rand() % (MAXIMUM\_READER\_DELAY - MINIMUM\_READER\_DELAY) + MINIMUM\_READER\_DELAY;

for (int i = 0; i < WRITERS\_NUMBER \* CNT\_ITERS; i++)

writersRand[i] = rand() % (MAXIMUM\_WRITER\_DELAY - MINIMUM\_WRITER\_DELAY) + MINIMUM\_WRITER\_DELAY;

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

{

perror("Can't create mutex!\n");

return CREATE\_MUTEX\_ERROR;

}

if ((canRead = CreateEvent(NULL, FALSE, FALSE, NULL)) == NULL)

{

perror("Can't create event (canRead)!\n");

return CREATE\_EVENT\_ERROR;

}

if ((canWrite = CreateEvent(NULL, FALSE, FALSE, NULL)) == NULL)

{

perror("Can't create event (canWrite)!\n");

return CREATE\_EVENT\_ERROR;

}

// Создаем потоки

DWORD id = 0;

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

{

readersID[i] = i;

if ((readerThreads[i] = CreateThread(NULL, 0, &Reader, readersID + i, 0, &id)) == NULL)

{

perror("Can't CreateThread (reader)");

return CREATE\_READER\_THREAD\_ERROR;

}

}

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

{

writersID[i] = i;

if ((writerThreads[i] = CreateThread(NULL, 0, &Writer, writersID + i, 0, &id)) == NULL)

{

perror("Can't CreateThread (writer)");

return CREATE\_WRITER\_THREAD\_ERROR;

}

}

WaitForMultipleObjects(READERS\_NUMBER, readerThreads, TRUE, INFINITE);

WaitForMultipleObjects(WRITERS\_NUMBER, writerThreads, TRUE, INFINITE);

// Закрываем

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

CloseHandle(readerThreads[i]);

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

CloseHandle(writerThreads[i]);

CloseHandle(canRead);

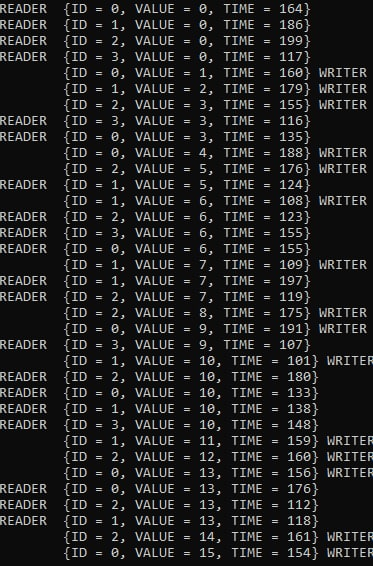
CloseHandle(canWrite);

CloseHandle(mutex);

return OK;

}

**Пример работы программы:**



*Изображение 1. – Работа программы.*