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"

Студент Сукочева Алис
Группа ИУ7-53Б
Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7
Тема «Реализация монитора Хоара «Читатели-писатели» под ОС Windows».

Студент: _____ Сукочева А.
подпись, дата Фамилия, И.О.

Преподаватель: _____ Рязанова Н.Ю.
подпись, дата Фамилия, И. О.

Москва — 2020 г.

Результат работы.

```
Reader with id = 2; value = 0; sleep time = 137.
Reader with id = 0; value = 0; sleep time = 183.
Reader with id = 1; value = 0; sleep time = 116.
Writer with id = 0; value = 1; sleep time = 102.
Writer with id = 1; value = 2; sleep time = 160.
Writer with id = 2; value = 3; sleep time = 134.
Writer with id = 0; value = 4; sleep time = 119.
Reader with id = 1; value = 4; sleep time = 132.
Reader with id = 2; value = 4; sleep time = 163.
Writer with id = 2; value = 5; sleep time = 103.
Writer with id = 1; value = 6; sleep time = 196.
Reader with id = 0; value = 6; sleep time = 160.
Writer with id = 0; value = 7; sleep time = 155.
Reader with id = 1; value = 7; sleep time = 190.
Writer with id = 2; value = 8; sleep time = 117.
Reader with id = 2; value = 8; sleep time = 116.
Reader with id = 0; value = 8; sleep time = 135.
Writer with id = 1; value = 9; sleep time = 102.
Writer with id = 2; value = 10; sleep time = 122.
Writer with id = 0; value = 11; sleep time = 167.
Reader with id = 2; value = 11; sleep time = 144.
Reader with id = 1; value = 11; sleep time = 197.
Writer with id = 1; value = 12; sleep time = 116.
Reader with id = 0; value = 12; sleep time = 164.
Writer with id = 2; value = 13; sleep time = 120.
Writer with id = 0; value = 14; sleep time = 129.
Reader with id = 2; value = 14; sleep time = 114.
Writer with id = 1; value = 15; sleep time = 160.
Reader with id = 0; value = 15; sleep time = 193.
Reader with id = 1; value = 15; sleep time = 172.

Ok!
```

Рисунок 0.1 — Результат работы программы.

Листинг 1 — Код программы

```
#include <windows.h>
#include <stdbool.h>
#include <stdio.h>
#include <time.h>
#include <stdbool.h>

#define OK 0

#define CREATE_MUTEX_ERROR 1
#define CREATE_EVENT_ERROR 2
#define CREATE_READER_THREAD_ERROR 3
#define CREATE_WRITER_THREAD_ERROR 3

#define MINIMUM_READER_DELAY 100
#define MINIMUM_WRITER_DELAY 100
#define MAXIMUM_READER_DELAY 200
#define MAXIMUM_WRITER_DELAY 200

#define READERS_NUMBER 3
#define WRITERS_NUMBER 3

#define ITERATIONS_NUMBER 5

HANDLE canRead;
HANDLE canWrite;
HANDLE mutex;

LONG waitingWritersCount = 0;
LONG waitingReadersCount = 0;
LONG activeReadersCount = 0;
bool writing = false;

HANDLE readerThreads[READERS_NUMBER];
HANDLE writerThreads[WRITERS_NUMBER];

int readersID[READERS_NUMBER];
int writersID[WRITERS_NUMBER];

int readersRand[READERS_NUMBER * ITERATIONS_NUMBER];
int writersRand[READERS_NUMBER * ITERATIONS_NUMBER];

int value = 0;

bool turn(HANDLE event)
{
```

```
46      // Если функция возвращает WAIT_OBJECT_0, объект свободен.
47      return WaitForSingleObject(event, 0) == WAIT_OBJECT_0;
48  }
49
50  void StartRead()
51  {
52      // Увеличиваем кол-во ждущих читателей.
53      InterlockedIncrement(&waitingReadersCount);
54
55      // Процесс читатель сможет начать работать,
56      // Если есть нет активного писателя,
57      // И нет писателей, ждущих свою очередь.
58      if (writing || turn(canWrite))
59          WaitForSingleObject(canRead, INFINITE);
60
61      WaitForSingleObject(mutex, INFINITE);
62      // Уменьшаем кол-во ждущих читателей.
63      InterlockedDecrement(&waitingReadersCount);
64      // Увеличиваем кол-во активных читателей.
65      InterlockedIncrement(&activeReadersCount);
66      // Выдаем сигнал canRead,
67      // Чтобы следующий читатель в очереди
68      // Читателей смог начать чтение
69      SetEvent(canRead);
70      ReleaseMutex(mutex);
71  }
72
73  void StopRead()
74  {
75      // Уменьшаем количество активных читателей.
76      InterlockedDecrement(&activeReadersCount);
77      // Если число читателей равно нулю,
78      // Выполняется signal(can_write),
79      // активизирующий писателя из очереди писателей.
80      if (!activeReadersCount)
81          SetEvent(canWrite);
82  }
83
84  DWORD WINAPI Reader(CONST LPVOID param)
85  {
86      int id = *(int *)param;
87      int sleepTime;
88      int begin = id * ITERATIONS_NUMBER;
89      for (int i = 0; i < ITERATIONS_NUMBER; i++)
90      {
91          sleepTime = readersRand[begin + i];
92          StartRead();
```

```
93          printf("Reader with id = %d; value = %d; sleep time = %d.\n", id,
                value, sleepTime);
94          StopRead();
95
96          // WaitForSingleObject(canRead, INFINITE);
97          // printf("Thread with id = %d, i = %d value = %d\n", id, i,
                value);
98          Sleep(sleepTime);
99      }
100 }
101
102 void StartWrite()
103 {
104     // Увеличиваем кол-во ждущих писателей.
105     InterlockedIncrement(&waitingWritersCount);
106
107     // Процесс писатель сможет начать работать,
108     // Если нет читающих процессов
109     // И нет другого активного писателя.
110     if (activeReadersCount > 0 || writing)
111         WaitForSingleObject(canWrite, INFINITE);
112
113     // Уменьшаем кол-во ждущих писателей.
114     InterlockedDecrement(&waitingWritersCount);
115     // Писатель пишет.
116     writing = true;
117 }
118
119 void StopWrite()
120 {
121     writing = false;
122     // Предпочтение отдается читателям при условии,
123     // Что очередь ждущих читателей не пуста.
124     if (waitingReadersCount)
125         SetEvent(canRead);
126     else
127         SetEvent(canWrite);
128 }
129
130 DWORD WINAPI Writer(CONST LPVOID param)
131 {
132     int id = *(int *)param;
133     int sleepTime;
134     int begin = id * ITERATIONS_NUMBER;
135     for (int i = 0; i < ITERATIONS_NUMBER; i++)
136     {
137         sleepTime = writersRand[begin + i];
```

```
138
139         StartWrite();
140         ++value;
141         printf("Writer with id = %d; value = %d; sleep time = %d.\n", id,
                value, sleepTime);
142         StopWrite();
143
144         Sleep(sleepTime);
145     }
146 }
147
148 int InitHandles()
149 {
150     // 2ой аргумент == false значит мьютекс свободный.
151     if ((mutex = CreateMutex(NULL, FALSE, NULL)) == NULL)
152     {
153         perror("CreateMutex");
154         return CREATE_MUTEX_ERROR;
155     }
156
157     // 2ой аргумент == FALSE значит автоматический сброс.
158     // 3ий аргумент == FALSE значит, что объект не в сигнальном состоянии.
159     if ((canRead = CreateEvent(NULL, FALSE, FALSE, NULL)) == NULL)
160     {
161         perror("CreateEvent (canRead)");
162         return CREATE_EVENT_ERROR;
163     }
164
165     if ((canWrite = CreateEvent(NULL, FALSE, FALSE, NULL)) == NULL)
166     {
167         perror("CreateEvent (canWrite)");
168         return CREATE_EVENT_ERROR;
169     }
170
171     return OK;
172 }
173
174 int CreateThreads()
175 {
176     DWORD id = 0;
177     for (int i = 0; i < READERS_NUMBER; i++)
178     {
179         readersID[i] = i;
180         // Параметры слева направо:
181         // NULL - Атрибуты защиты определены по умолчанию;
182         // 0 - размер стека устанавливается по умолчанию;
```

```
183         // Reader - определяет адрес функции потока, с которой следует нача
                ть выполнение потока;
184         // readersID + i - указатель на переменную, которая передается в по
                ток;
185         //  0 - исполнение потока начинается немедленно;
186         // Последний - адрес переменной типа DWORD, в которую функция возвр
                ащает идентификатор потока.
187         if ((readerThreads[i] = CreateThread(NULL, 0, &Reader, readersID +
                i, 0, &id)) == NULL)
188         {
189             perror("CreateThread (reader)");
190             return CREATE_READER_THREAD_ERROR;
191         }
192         // printf("Created reader with thread id = %d\n", id);
193     }
194
195     for (int i = 0; i < WRITERS_NUMBER; i++)
196     {
197         writersID[i] = i;
198         if ((writerThreads[i] = CreateThread(NULL, 0, &Writer, writersID +
                i, 0, &id)) == NULL)
199         {
200             perror("CreateThread (writer)");
201             return CREATE_WRITER_THREAD_ERROR;
202         }
203         // printf("Created writer with thread id = %d\n", id);
204     }
205
206     return OK;
207 }
208
209 void Close()
210 {
211     // Закрываем дескрипторы mutex, event и всех созданных потоков.
212     for (int i = 0; i < READERS_NUMBER; i++)
213         CloseHandle(readerThreads[i]);
214
215     for (int i = 0; i < WRITERS_NUMBER; i++)
216         CloseHandle(writerThreads[i]);
217
218     CloseHandle(canRead);
219     CloseHandle(canWrite);
220     CloseHandle(mutex);
221 }
222
223 void CreateRand()
224 {
```

```
225     for (int i = 0; i < READERS_NUMBER * ITERATIONS_NUMBER; i++)
226         readersRand[i] = rand() % (MAXIMUM_READER_DELAY -
                MINIMUM_READER_DELAY) + MINIMUM_READER_DELAY;
227
228     for (int i = 0; i < WRITERS_NUMBER * ITERATIONS_NUMBER; i++)
229         writersRand[i] = rand() % (MAXIMUM_WRITER_DELAY -
                MINIMUM_WRITER_DELAY) + MINIMUM_WRITER_DELAY;
230 }
231
232 int main(void)
233 {
234     setbuf(stdout, NULL);
235     srand(time(NULL));
236
237     CreateRand();
238
239     int err = InitHandles();
240     if (err)
241         return err;
242
243     err = CreateThreads();
244     if (err)
245         return err;
246
247     // READERS_NUMBER - кол-во инетерсующих нас объектов ядра.
248     // readerThreads - указатель на массив описателей объектов ядра.
249     // TRUE - функция не даст потоку возобновить свою работу, пока не освоб
            одятся все объекты.
250     // INFINITE - указывает, сколько времени поток готов ждать освобождения
             объекта.
251     WaitForMultipleObjects(READERS_NUMBER, readerThreads, TRUE, INFINITE);
252     WaitForMultipleObjects(WRITERS_NUMBER, writerThreads, TRUE, INFINITE);
253
254     Close();
255
256     printf("\nOk!\n");
257     return OK;
258 }
```