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»

Студент Жигалкин Д.Р

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватель Рязанова Н.Ю

Москва.
2020 г.

Задание на лабораторную работу:

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».

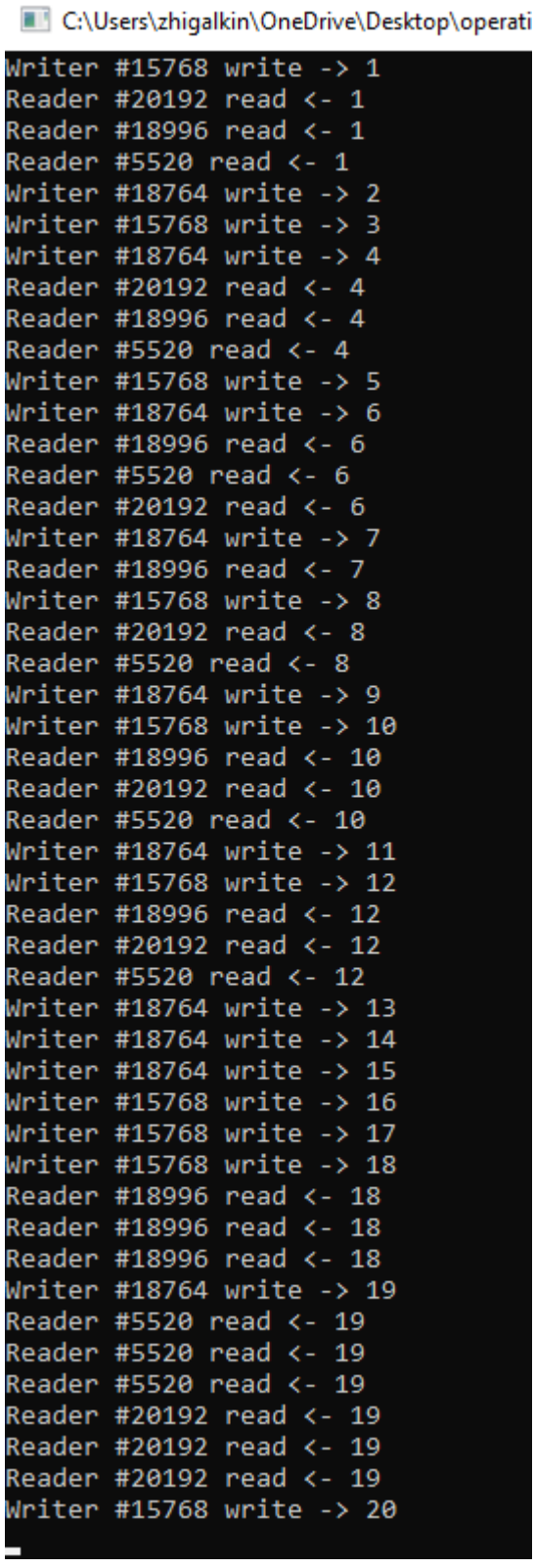
Листинг 1.1 — Реализация монитора Хоара «Читатели-писатели»

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

#define _CRT_SECURE_NO_WARNINGS
#define COUNT_READERS 3
#define COUNT_WRITERS 2

struct handle_information
{
	LONG active_readers = 0;
	LONG waiting_readers = 0;
	LONG waiting_writers = 0;
	bool active_writer = false;

	HANDLE mutex;
	HANDLE can_read;
	HANDLE can_write;

	int *buffer;
};

void start_write(handle_information* my_handle)
{
	InterlockedIncrement(&my_handle->waiting_writers);

	// обеспечение монопольного доступа писателя
	if (my_handle->active_writer || my_handle->active_readers > 0)
	{
		WaitForSingleObject(my_handle->can_write, INFINITE);
	}

	WaitForSingleObject(my_handle->mutex, INFINITE);
	InterlockedDecrement(&my_handle->waiting_writers);
	my_handle->active_writer = true;

	// сброс "в ручную"
	ResetEvent(my_handle->can_write);

	ReleaseMutex(my_handle->mutex);
}

void start_read(handle_information* my_handle)
{
	InterlockedIncrement(&my_handle->waiting_readers);
	if (my_handle->active_writer || my_handle->waiting_writers > 0)
	{
		WaitForSingleObject(my_handle->can_read, INFINITE);
	}

	InterlockedDecrement(&my_handle->waiting_readers);
	InterlockedIncrement(&my_handle->active_readers);

	// чтобы следующий читатель в очереди читателей смог начать чтение
	SetEvent(my_handle->can_read);
}

void stop_write(handle_information* my_handle)
{
	my_handle->active_writer = false;
	if (WaitForSingleObject(my_handle->can_read, 0) == WAIT_OBJECT_0)
	{
		SetEvent(my_handle->can_read);
	}
	else
	{
		SetEvent(my_handle->can_write);
	}
}

```

```
70:  void stop_read(handle_information* my_handle)
71:  {
72:          // уменьшение количества активных писаталей
73:          InterlockedDecrement(&my_handle->active_readers);
74:          if (my_handle->active_readers == 0)
75:          {
76:                  // активизация писателя из очереди писателей
77:                  SetEvent(my_handle->can_write);
78:          }
79:  }
80:
81:  DWORD reader(handle_information* my_handle)
82:  {
83:          while (true)
84:          {
85:                  start_read(my_handle);
86:                  printf("Reader #%ld read <- %d\n", GetCurrentThreadId(), *my_handle-
     >buffer);
87:                  stop_read(my_handle);
88:
89:                  Sleep(1000 * (rand() % 3));
90:          }
91:
92:          return 0;
93:  }
94:
95:  DWORD writer(handle_information* my_handle)
96:  {
97:          while (true)
98:          {
99:                  start_write(my_handle);
100:                 printf("Writer #%ld write -> %ld\n", GetCurrentThreadId(), ++(*my_handle-
     >buffer));
101:                 stop_write(my_handle);
102:
103:                 Sleep(900 * (rand() % 3));
104:         }
105:
106:         return 0;
107: }
108:
109: handle_information* Initialization(int *buffer)
110: {
111:         // создает новый своободный мьютекс
112:         HANDLE mutex = CreateMutex(NULL, FALSE, NULL);
113:         if (mutex == NULL)
114:         {
115:                 perror("Can't create mutex");
116:                 exit(1);
117:         }
118:
119:         // создание Event, который переключается "в ручную", объект в сигнальном
     состоянии
120:         HANDLE can_write = CreateEvent(NULL, TRUE, TRUE, NULL);
121:         if (can_write == NULL)
122:         {
123:                 perror("Can't create event can write");
124:                 exit(1);
125:
126:         }
127:
128:         // создание Event, который переключается автоматически, объект в сигнальном
     состоянии
129:         HANDLE can_read = CreateEvent(NULL, FALSE, TRUE, NULL);
130:         if (can_read == NULL)
131:         {
132:                 perror("Can't create event can read");
133:                 exit(1);
134:
135:         }
```

```
136:
137:         handle_information* my_handle = new handle_information();
138:
139:         my_handle->mutex = mutex;
140:         my_handle->can_read = can_read;
141:         my_handle->can_write = can_write;
142:         my_handle->buffer = buffer;
143:
144:         return my_handle;
145: }
146:
147: int main()
148: {
149:         HANDLE writers[COUNT_WRITERS];
150:         HANDLE readers[COUNT_READERS];
151:
152:         int buffer = 0;
153:
154:         handle_information* my_handle = Initialization(&buffer);
155:
156:         for (int i = 0; i < COUNT_WRITERS; i++)
157:         {
158:                 writers[i] = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)&writer,
    my_handle, 0, NULL);
159:                 if (writers[i] == NULL)
160:                 {
161:                         perror("Can't create writer");
162:                         return 1;
163:                 }
164:         }
165:
166:         for (int i = 0; i < COUNT_READERS; i++)
167:         {
168:                 readers[i] = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)&reader,
    my_handle, 0, NULL);
169:                 if (readers[i] == NULL)
170:                 {
171:                         perror("Can't create reader");
172:                         return 1;
173:                 }
174:         }
175:
176:         // ожидание освобождения
177:         WaitForMultipleObjects(COUNT_WRITERS, writers, TRUE, INFINITE);
178:         WaitForMultipleObjects(COUNT_READERS, readers, TRUE, INFINITE);
179:
180:         // освобождение ресурсов
181:         CloseHandle(my_handle->mutex);
182:         CloseHandle(my_handle->can_read);
183:         CloseHandle(my_handle->can_write);
184:
185:         return 0;
186: }
187:
```

Пример работы реализации:


C:\Users\zhigalkin\OneDrive\Desktop\operati
Writer #15768 write -> 1
Reader #20192 read <- 1
Reader #18996 read <- 1
Reader #5520 read <- 1
Writer #18764 write -> 2
Writer #15768 write -> 3
Writer #18764 write -> 4
Reader #20192 read <- 4
Reader #18996 read <- 4
Reader #5520 read <- 4
Writer #15768 write -> 5
Writer #18764 write -> 6
Reader #18996 read <- 6
Reader #5520 read <- 6
Reader #20192 read <- 6
Writer #18764 write -> 7
Reader #18996 read <- 7
Writer #15768 write -> 8
Reader #20192 read <- 8
Reader #5520 read <- 8
Writer #18764 write -> 9
Writer #15768 write -> 10
Reader #18996 read <- 10
Reader #20192 read <- 10
Reader #5520 read <- 10
Writer #18764 write -> 11
Writer #15768 write -> 12
Reader #18996 read <- 12
Reader #20192 read <- 12
Reader #5520 read <- 12
Writer #18764 write -> 13
Writer #18764 write -> 14
Writer #18764 write -> 15
Writer #15768 write -> 16
Writer #15768 write -> 17
Writer #15768 write -> 18
Reader #18996 read <- 18
Reader #18996 read <- 18
Reader #18996 read <- 18
Writer #18764 write -> 19
Reader #5520 read <- 19
Reader #5520 read <- 19
Reader #5520 read <- 19
Reader #20192 read <- 19
Reader #20192 read <- 19
Reader #20192 read <- 19
Writer #15768 write -> 20