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 дисциплине "Операционные системы"

Тема Реализация монитора Хоара «Читатели-писатели» для Windows

Студент Романов А.В.

Группа ИУ7-53Б

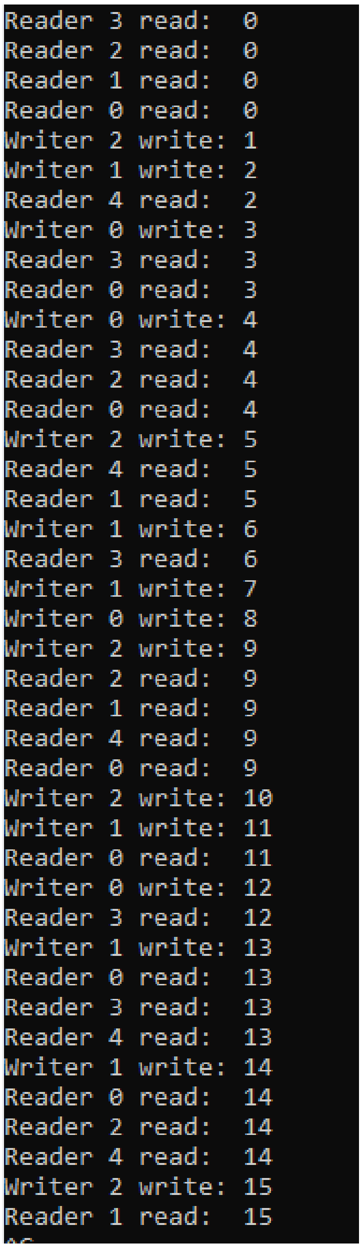
Оценка (баллы) ______________________________

Преподаватели Рязанова Н.Ю.

Москва — 2020 г.

1 Задача «Читатели-писатели»

1.1 Демонстрация работы программы



```
Reader 3 read:  0
Reader 2 read:  0
Reader 1 read:  0
Reader 0 read:  0
Writer 2 write: 1
Writer 1 write: 2
Reader 4 read:  2
Writer 0 write: 3
Reader 3 read:  3
Reader 0 read:  3
Writer 0 write: 4
Reader 3 read:  4
Reader 2 read:  4
Reader 0 read:  4
Writer 2 write: 5
Reader 4 read:  5
Reader 1 read:  5
Writer 1 write: 6
Reader 3 read:  6
Writer 1 write: 7
Writer 0 write: 8
Writer 2 write: 9
Reader 2 read:  9
Reader 1 read:  9
Reader 4 read:  9
Reader 0 read:  9
Writer 2 write: 10
Writer 1 write: 11
Reader 0 read:  11
Writer 0 write: 12
Reader 3 read:  12
Writer 1 write: 13
Reader 0 read:  13
Reader 3 read:  13
Reader 4 read:  13
Writer 1 write: 14
Reader 0 read:  14
Reader 2 read:  14
Reader 4 read:  14
Writer 2 write: 15
Reader 1 read:  15
```

Рис. 1.1: Демонстрация работы программы.

1.2 Листинги кода

В листинге 1.1 представлен исходный код реализующий монитор Хоара «Читатели-писатели» для Windows.

Листинг 1.1: Главный файл программы

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

#define OK 0
#define FALSE 0
#define TRUE 1

#define READERS_CNT 5
#define WRITERS_CNT 3

#define WITER_CNT 8
#define RITER_CNT 7

#define WRITE_TIMEOUT 300
#define READ_TIMEOUT 300
#define DIFF 4000

#define CREATE_MUTEX_FAILED 1
#define CREATE_EVENT_FAILED 2
#define CREATE_THREAD_FAILED 3

HANDLE mutex;
HANDLE can_read;
HANDLE can_write;

LONG active_readers = 0;
LONG waiting_writers = 0;
LONG waiting_readers = 0;

int active_writer = FALSE;
int value = 0;

void start_read(void) {
  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(void) {
  InterlockedDecrement(&active_readers);
  if (active_readers == 0) {
    ResetEvent(can_read);
    SetEvent(can_write);
  }
}

DWORD WINAPI run_reader(CONST LPVOID lpParams) {
  srand(time(NULL) + WRITERS_CNT);
  int sleep_time;

  for (size_t i = 0; i < RITER_CNT; i++) {
    sleep_time = READ_TIMEOUT + rand() % DIFF;
    Sleep(sleep_time);
    start_read();
    printf("Reader %d read:  %d\n", (int)lpParams;, value);
    stop_read();
  }

  return OK;
}


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 run_writer(CONST LPVOID lpParams) {
  srand(time(NULL)+ READERS_CNT);
  int sleep_time;

  for (int i = 0; i < WITER_CNT; ++i) {
    sleep_time = WRITE_TIMEOUT + rand() % DIFF;
    Sleep(sleep_time);
    start_write();

    printf("Writer %d write: %d\n", (int)lpParams;, ++value);
    stop_write();
  }

  return OK;
}

int main(void) {
  HANDLE writers_threads[WRITERS_CNT];
  HANDLE readers_threads[READERS_CNT];

  if (!(mutex = CreateMutex(NULL, FALSE, NULL))) {
    perror("Failed call of CreateMutex");
    return CREATE_MUTEX_FAILED;
  }

  if (!(can_read = CreateEvent(NULL, FALSE, FALSE, NULL)) || !(can_write =
     CreateEvent(NULL, FALSE, FALSE, NULL))) {
    perror("Failed call of CreateEvent");
    return CREATE_EVENT_FAILED;
  }

  for (int i = 0; i < READERS_CNT; ++i) {
    if (!(readers_threads[i] = CreateThread(NULL, 0, run_reader, (LPVOID)i,
       0, NULL))) {
      perror("Failed call of CreateThread");
      return CREATE_THREAD_FAILED;
    }
  }

  for (int i = 0; i < WRITERS_CNT; i++) {
    if (!(writers_threads[i] = CreateThread(NULL, 0, run_writer, (LPVOID)i,
       0, NULL))) {
      perror("Failed call of CreateThread");
      return CREATE_THREAD_FAILED;
    }
  }

```

```
    WaitForMultipleObjects(READERS_CNT, readers_threads, TRUE, INFINITE);
    WaitForMultipleObjects(WRITERS_CNT, writers_threads, TRUE, INFINITE);

    CloseHandle(mutex);
    CloseHandle(can_read);
    CloseHandle(can_write);

    return OK;
}
```