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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

О Т Ч Е Т

по лабораторной работе № 6

Название: Реализация монитора Хоара «Читатели-писатели» под ОС Windows

Дисциплина: Операционные системы

Студент ИУ7-52Б _____ Е.В. Брянская
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Н.Ю. Рязанова
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Задание 1.

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».

```
#define K_TIME          5
#define NUM_WRITERS     3
#define NUM_READERS     6

HANDLE mtx;
HANDLE can_write;
HANDLE can_read;

HANDLE writers[NUM_WRITERS];
HANDLE readers[NUM_READERS];

BOOL active_writer = FALSE;
volatile long active_readers = 0;
volatile long waiting_readers = 0, waiting_writers = 0;

int shr_var = -1;

void start_write()
{
    InterlockedIncrement(&waiting_writers);

    if (active_readers > 0 || active_writer)
        if (WaitForSingleObject(can_write, INFINITE) ==
WAIT_FAILED)
        {
            perror("WaitForSingleObject error\n");
            exit(4);
        }

    InterlockedDecrement(&waiting_writers);

    active_writer = TRUE;
}

void stop_write()
{
    active_writer = FALSE;

    if (waiting_readers > 0)
        SetEvent(can_read);
    else if (waiting_writers)
        SetEvent(can_write);
}

void start_read()
```

```
{
    InterlockedIncrement(&waiting_readers);

    if (active_writer || waiting_writers > 0)
    {
        if (WaitForSingleObject(can_read, INFINITE) ==
WAIT_FAILED)
        {
            perror("WaitForSingleObject error\n");
            exit(4);
        }
        else if (waiting_readers == 1)
            ResetEvent(can_read);
    }

    InterlockedDecrement(&waiting_readers);

    InterlockedIncrement(&active_readers);
}

void stop_read()
{
    InterlockedDecrement(&active_readers);

    if (active_readers == 0 && waiting_writers)
        SetEvent(can_write);
}

DWORD WINAPI action_writer()
{
    long cur_id = GetCurrentThreadId();
    srand(time(NULL) + cur_id);

    sleep(rand() % K_TIME);

    while (1)
    {
        start_write();

        if (WaitForSingleObject(mtx, INFINITE) == WAIT_FAILED)
        {
            perror("WaitForSingleObject error\n");
            exit(4);
        }

        shr_var++;
        printf(">>> WRITER %ld: \twrote %d\n", cur_id, shr_var);

        if (!ReleaseMutex(mtx))
        {
            perror("ReleaseMutex error\n");
            exit(5);
        }
```

```
        stop_write();

        sleep(rand() % K_TIME + 1);
    }

    return 0;
}

DWORD WINAPI action_reader()
{
    long cur_id = GetCurrentThreadId();

    srand(time(NULL) + cur_id);

    sleep(rand() % K_TIME);

    while (1)
    {
        start_read();

        if (WaitForSingleObject(mtx, INFINITE) == WAIT_FAILED)
        {
            perror("WaitForSingleObject error\n");
            exit(4);
        }

        printf(">>> READER %ld: \tread %d\n", cur_id, shr_var);

        if (!ReleaseMutex(mtx))
        {
            perror("ReleaseMutex error\n");
            exit(5);
        }

        stop_read();

        sleep(rand() % K_TIME + 1);
    }

    return 0;
}


void create_mutex()
{
    mtx = CreateMutex(NULL, FALSE, NULL);
    if (mtx == NULL)
    {
        perror("CreateMutex error\n");
        exit(1);
    }
}
```

```
void create_events()
{
    can_write = CreateEvent(NULL, FALSE, FALSE, NULL);
    can_read = CreateEvent(NULL, TRUE, FALSE, NULL);
    if (can_write == NULL || can_read == NULL)
    {
        perror("CreateEvent error\n");
        exit(2);
    }
}

void create_threads()
{
    for (int i = 0; i < NUM_WRITERS; i++)
    {
        writers[i] = CreateThread(NULL, 0, action_writer, NULL,
0, NULL);
        if (writers[i] == NULL)
        {
            perror("CreateThread error\n");
            exit(3);
        }
    }

    rand();

    for (int i = 0; i < NUM_READERS; i++)
    {
        readers[i] = CreateThread(NULL, 0, action_reader, NULL,
0, NULL);
        if (readers[i] == NULL)
        {
            perror("CreateThread error\n");
            exit(3);
        }
    }
}

int main()
{
    create_mutex();
    create_events();
    create_threads();

    if (WaitForMultipleObjects(NUM_WRITERS, writers, TRUE,
INFINITE) == WAIT_FAILED ||
            WaitForMultipleObjects(NUM_READERS, readers, TRUE,
INFINITE) == WAIT_FAILED)
    {
        perror("WaitForMultipleObjects error\n");
        exit(5);
    }
```

```
    CloseHandle(mtx);
    CloseHandle(can_read);
    CloseHandle(can_write);

    return 0;
}
```

Результат выполнения программы:

```
>>> WRITER 16228:       wrote 0
>>> READER 13544:       read 0
>>> READER 7388:        read 0
>>> WRITER 4932:        wrote 1
>>> WRITER 8420:        wrote 2
>>> READER 14780:       read 2
>>> READER 17268:       read 2
>>> READER 16492:       read 2
>>> WRITER 16228:       wrote 3
>>> READER 7660:        read 3
>>> WRITER 4932:        wrote 4
>>> READER 17268:       read 4
>>> WRITER 8420:        wrote 5
>>> READER 13544:       read 5
>>> READER 7660:        read 5
>>> READER 7388:        read 5
>>> READER 14780:       read 5
>>> READER 17268:       read 5
>>> READER 7388:        read 5
>>> WRITER 4932:        wrote 6
>>> READER 7660:        read 6
>>> READER 16492:       read 6
>>> WRITER 16228:       wrote 7
```