

Санкт-Петербургский государственный политехнический университет
Институт Информационных Технологий и Управления
Кафедра компьютерных систем и программных технологий

Отчёт по расчётной работе № 3
по предмету «Системное программное обеспечение»

ПРИМИТИВЫ синхронизации в ОС WINDOWS

Работу выполнил студент гр. 53501/3 _____ Мартынов С. А.

Работу принял преподаватель _____ Душутина Е. В.

Санкт-Петербург
2014

Содержание

Постановка задачи	3
Введение	5
1 Прimitives синхронизации	6
1.1 Использование мьютексов	12
1.2 Использование семафоров	17
1.3 Критические секции	22
1.4 Объекты-события в качестве средства синхронизации	27
1.5 Условные переменные	32
1.6 Задача читателя-писателя (для потоков одного процесса)	38
1.7 Задача читателя-писателя (для потоков разных процессов)	46
2 Модификация задачи читателя-писателя	57
Заключение	68

Постановка задачи

В рамках данной работы необходимо ознакомиться с основными примитивами синхронизации в ОС Windows, и выполнить следующие задачи:

Потоки разделяют целочисленный массив, в который заносятся производимые и извлекаются потребляемые данные. Для наглядности и контроля за происходящим в буфер помещается нарастающее значение, однозначно идентифицирующее производителя и номер его очередной отправки.

Код должен удовлетворять трем требованиям:

- потребитель не должен пытаться извлечь значение из буфера, если буфер пуст;
- производитель не должен пытаться поместить значение в буфер, если буфер полон;
- состояние буфера должно описываться общими переменными (индексами, счётчиками, указателями связанных списков и т.д.).

Задание необходимо выполнить различными способами, применив следующие средства синхронизации доступа к разделяемому ресурсу:

- Мьютексы;
- Семафоры;
- Критические секции;
- Объекты события;
- Условные переменные;
- Функции ожидания.

Создать аналогичные программы для множества потоков, количество которых можно задать из командной строки.

Программы должны предоставлять возможность завершения по таймеру либо по команде оператора.

Отчёт должен содержать:

1. Результаты выполнения предложенных в методическом пособии программ и их анализ.

2. Решение задачи читатели-писатели таким образом, чтобы читатели не имели доступа к памяти по записи.
3. Более рациональное решение задачи читатели-писатель, используя другие средства синхронизации или их сочетание. Объяснить и подтвердить экспериментально улучшение характеристик взаимодействия.
4. Клиент-серверное приложение для полной задачи читатели-писатели с собственной системой ограничений на доступ каждого читателя к информации.
5. Программу читатели-писатели для сетевого функционирования (для этого необходимо выбрать подходящие средства IPC и синхронизации).
6. Решение задачи производители-потребители (разница с предыдущей задачей в возможности модификации считываемых данных).
7. Задачу "обедающие философы" с обоснованием выбранных средств синхронизации.

Введение

Исходный код всех представленных листингов доступен по адресу
https://github.com/SemenMartynov/SPbPU_SystemProgramming.

1 Прimitives синхронизации

Код задач в данном разделе разбит на файлы. Некоторые файлы (такие как система логирования) в разных проектах содержат одинаковый код. Для простоты восприятия информации, они вынесены в этот раздел (полную версию исходных кодов можно получить по ссылке на гитхаб, приведённой во введении).

Листинг 1: Реализация класса логера

```
1 #include "Logger.h"
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <tchar.h>
6 #include <stdarg.h>
7 #include <time.h>
8 #include <Windows.h>
9
10 Logger::Logger(const _TCHAR* prog, int tid /* = -1 */) {
11     _TCHAR logname[255];
12
13     if (tid > 0)
14         swprintf_s(logname, _T("%s.%d.log"), prog, tid);
15     else
16         swprintf_s(logname, _T("%s.log"), prog);
17
18     // Try to open log file for append
19     if (_wfopen_s(&logfile, logname, _T("a+"))) {
20         _wprintf(_T("The following error occurred"));
21         _tprintf(_T("Can't open log file %s\n"), logname);
22         exit(-1);
23     }
24     quietlog(_T("%s is starting."), prog);
25 }
26
27 Logger::~Logger() {
28     quietlog(_T("Shutting down.\n"));
29     fclose(logfile);
30 }
31
32 void Logger::quietlog(_TCHAR* format, ...) {
33     _TCHAR buf[255];
34     va_list ap;
35     struct tm newtime;
36     __time64_t long_time;
```

```

37 // Get time as 64-bit integer.
38 _time64(&long_time);
39 // Convert to local time.
40 _localtime64_s(&newtime, &long_time);
41 // Convert to normal representation.
42 swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
43     newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
44     newtime.tm_min, newtime.tm_sec);
45 // Write date and time
46 fwprintf(logfile, _T("%s"), buf);
47 // Write all params
48 va_start(ap, format);
49 _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
50 fwprintf(logfile, _T("%s"), buf);
51 va_end(ap);
52 // New sting
53 fwprintf(logfile, _T("\n"));
54 }
55
56 void Logger::loudlog(_TCHAR* format, ...) {
57     _TCHAR buf[255];
58     va_list ap;
59     struct tm newtime;
60     __time64_t long_time;
61     // Get time as 64-bit integer.
62     _time64(&long_time);
63     // Convert to local time.
64     _localtime64_s(&newtime, &long_time);
65     // Convert to normal representation.
66     swprintf_s(buf, _T("[%d/%d/%d %d:%d:%d] "), newtime.tm_mday,
67         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
68         newtime.tm_min, newtime.tm_sec);
69     // Write date and time
70     fwprintf(logfile, _T("%s"), buf);
71     // Write all params
72     va_start(ap, format);
73     _vsnwprintf_s(buf, sizeof(buf) - 1, format, ap);
74     fwprintf(logfile, _T("%s"), buf);
75     _tprintf(_T("%s"), buf);
76     va_end(ap);
77     // New sting
78     fwprintf(logfile, _T("\n"));
79     _tprintf(_T("\n"));
80 }

```

Листинг 2: Сервисные функции

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "thread.h"
6 #include "utils.h"
7 #include "Logger.h"
8
9 //создание, установка и запуск таймера
10 HANDLE CreateAndStartWaitableTimer(int sec) {
11     __int64 end_time;
12     LARGE_INTEGER end_time2;
13     HANDLE tm = CreateWaitableTimer(NULL, false, _T("Timer!"));
14     end_time = -1 * sec * 10000000;
15     end_time2.LowPart = (DWORD)(end_time & 0xFFFFFFFF);
16     end_time2.HighPart = (LONG)(end_time >> 32);
17     SetWaitableTimer(tm, &end_time2, 0, NULL, NULL, false);
18     return tm;
19 }
20
21 //создание всех потоков
22 void CreateAllThreads(struct Configuration* config, Logger* log) {
23     extern HANDLE *allhandlers;
24
25     int total = config->numOfReaders + config->numOfWriters + 1;
26     log->quietlog(_T("Total num of threads is %d"), total);
27     allhandlers = new HANDLE[total];
28     int count = 0;
29
30     //создаем потоки-читатели
31     log->loudlog(_T("Create readers"));
32     for (int i = 0; i != config->numOfReaders; ++i, ++count) {
33         log->loudlog(_T("Count = %d"), count);
34         //создаем потоки-читатели, которые пока не стартуют
35         if ((allhandlers[count] = CreateThread(NULL, 0, ThreadReaderHandler, (
36             LPVOID)i, CREATE_SUSPENDED, NULL)) == NULL) {
37             log->loudlog(_T("Impossible to create thread-reader, GLE = %d"),
38                 GetLastError());
39             exit(8000);
40         }
41     }
42
43     //создаем потоки-писатели
44     log->loudlog(_T("Create writers"));

```



```

43 for (int i = 0; i != config->numOfWriters; ++i, ++count) {
44     log->loudlog(_T("count = %d"), count);
45     //создаем потоки-писатели, которые пока не стартуют
46     if ((allhandlers[count] = CreateThread(NULL, 0, ThreadWriterHandler, (
         LPVOID)i, CREATE_SUSPENDED, NULL)) == NULL) {
47         log->loudlog(_T("Impossible to create thread-writer, GLE = %d"),
            GetLastError());
48         exit(8001);
49     }
50 }
51
52 //создаем поток TimeManager
53 log->loudlog(_T("Create TimeManager"));
54 log->loudlog(_T("Count = %d"), count);
55 //создаем потоки-читатели, которые пока не стартуют
56 if ((allhandlers[count] = CreateThread(NULL, 0, ThreadTimeManagerHandler,
    (LPVOID)config->t1, CREATE_SUSPENDED, NULL)) == NULL) {
57     log->loudlog(_T("impossible to create thread-reader, GLE = %d"),
        GetLastError());
58     exit(8002);
59 }
60 log->loudlog(_T("Successfully created threads!"));
61 }
62
63 //функция установки конфигурации
64 void SetConfig(_TCHAR* path, struct Configuration* config, Logger* log) {
65     _TCHAR filename[255];
66     wcscpy_s(filename, path);
67     log->quietlog(_T("Using config from %s"), filename);
68
69     FILE *confsource;
70     int numOfReaders;
71     int numOfWriters;
72     int readersDelay;
73     int writersDelay;
74     int sizeOfQueue;
75     int t1;
76     _TCHAR trash[30];
77
78     if (!_wopen_s(&confsource, filename, _T("r"))) {
79         _wpperror(_T("The following error occurred"));
80         log->loudlog(_T("impossible open config file %s\n"), filename);
81         exit(1000);
82     }
83

```

```

84 //начинаем читать конфигурацию
85 fscanf_s(confsource, "%s %d", trash, _countof(trash), &numOfReaders); //чи
    сло потоков-читателей
86 fscanf_s(confsource, "%s %d", trash, _countof(trash), &readersDelay); //за
    держки потоков-читателей
87 fscanf_s(confsource, "%s %d", trash, _countof(trash), &numOfWriters); //чи
    сло потоков-писателей
88 fscanf_s(confsource, "%s %d", trash, _countof(trash), &writersDelay); //за
    держки потоков-писателей
89 fscanf_s(confsource, "%s %d", trash, _countof(trash), &sizeOfQueue); //раз
    мер очереди
90 fscanf_s(confsource, "%s %d", trash, _countof(trash), &ttl); //время жизни
91
92 if (numOfReaders <= 0 || numOfWriters <= 0) {
93     log->loudlog(_T("Incorrect num of Readers or writers"));
94     exit(500);
95 }
96 else if (readersDelay <= 0 || writersDelay <= 0) {
97     log->loudlog(_T("Incorrect delay of Readers or writers"));
98     exit(501);
99 }
100 else if (sizeOfQueue <= 0) {
101     log->loudlog(_T("Incorrect size of queue"));
102     exit(502);
103 }
104 else if (ttl == 0) {
105     log->loudlog(_T("Incorrect ttl"));
106     exit(503);
107 }
108 fclose(confsource);
109
110 config->numOfReaders = numOfReaders;
111 config->readersDelay = readersDelay;
112 config->numOfWriters = numOfWriters;
113 config->writersDelay = writersDelay;
114 config->sizeOfQueue = sizeOfQueue;
115 config->ttl = ttl;
116
117 log->quietlog(_T("Config:\n\tNumOfReaders = %d\n\tReadersDelay = %d\n\t
    tNumOfWriters = %d\n\tWritersDelay = %d\n\tSizeOfQueue = %d\n\tttl = %d
    "),
118     config->numOfReaders, config->readersDelay, config->numOfWriters, config
    ->writersDelay, config->sizeOfQueue, config->ttl);
119 }
120

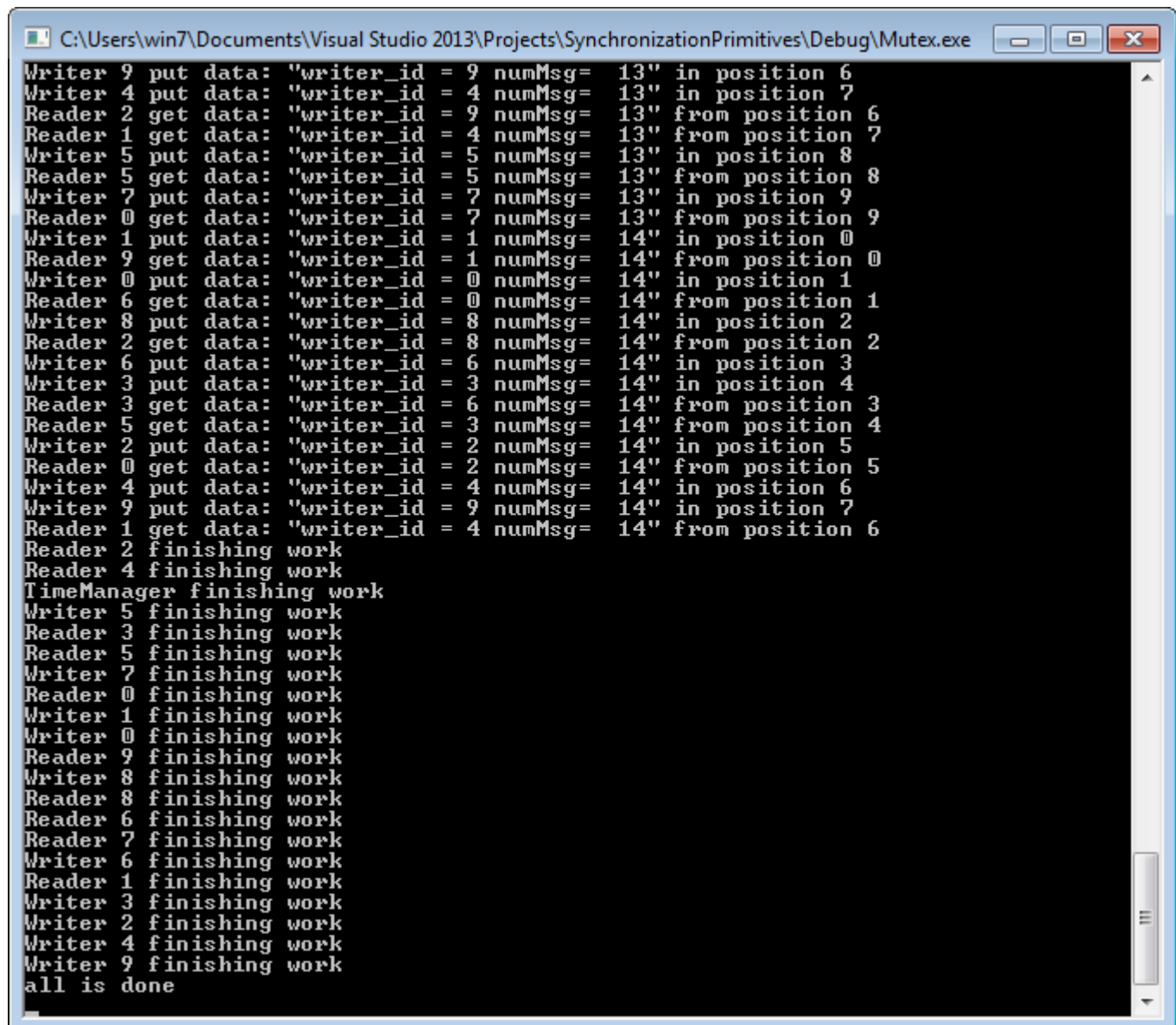
```

```

121 void SetDefaultConfig(struct Configuration* config, Logger* log) {
122     log->quietlog(_T("Using default config"));
123     //Вуд конфигурационного файла:
124     //     NumOfReaders= 10
125     //     ReadersDelay= 100
126     //     NumOfWriters= 10
127     //     WritersDelay= 200
128     //     SizeOfQueue= 10
129     //     ttl= 3
130
131     config->numOfReaders = 10;
132     config->readersDelay = 100;
133     config->numOfWriters = 10;
134     config->writersDelay = 200;
135     config->sizeOfQueue = 10;
136     config->ttl = 3;
137
138     log->quietlog(_T("Config:\n\tNumOfReaders = %d\n\tReadersDelay = %d\n\t
        \tNumOfWriters = %d\n\tWritersDelay = %d\n\tSizeOfQueue = %d\n\tttl = %d
        "),
139         config->numOfReaders, config->readersDelay, config->numOfWriters, config
            ->writersDelay, config->sizeOfQueue, config->ttl);
140 }

```

1.1 Использование мьютексов



```
C:\Users\win7\Documents\Visual Studio 2013\Projects\SynchronizationPrimitives\Debug\Mutex.exe
Writer 9 put data: "writer_id = 9 numMsg= 13" in position 6
Writer 4 put data: "writer_id = 4 numMsg= 13" in position 7
Reader 2 get data: "writer_id = 9 numMsg= 13" from position 6
Reader 1 get data: "writer_id = 4 numMsg= 13" from position 7
Writer 5 put data: "writer_id = 5 numMsg= 13" in position 8
Reader 5 get data: "writer_id = 5 numMsg= 13" from position 8
Writer 7 put data: "writer_id = 7 numMsg= 13" in position 9
Reader 0 get data: "writer_id = 7 numMsg= 13" from position 9
Writer 1 put data: "writer_id = 1 numMsg= 14" in position 0
Reader 9 get data: "writer_id = 1 numMsg= 14" from position 0
Writer 0 put data: "writer_id = 0 numMsg= 14" in position 1
Reader 6 get data: "writer_id = 0 numMsg= 14" from position 1
Writer 8 put data: "writer_id = 8 numMsg= 14" in position 2
Reader 2 get data: "writer_id = 8 numMsg= 14" from position 2
Writer 6 put data: "writer_id = 6 numMsg= 14" in position 3
Writer 3 put data: "writer_id = 3 numMsg= 14" in position 4
Reader 3 get data: "writer_id = 6 numMsg= 14" from position 3
Reader 5 get data: "writer_id = 3 numMsg= 14" from position 4
Writer 2 put data: "writer_id = 2 numMsg= 14" in position 5
Reader 0 get data: "writer_id = 2 numMsg= 14" from position 5
Writer 4 put data: "writer_id = 4 numMsg= 14" in position 6
Writer 9 put data: "writer_id = 9 numMsg= 14" in position 7
Reader 1 get data: "writer_id = 4 numMsg= 14" from position 6
Reader 2 finishing work
Reader 4 finishing work
TimeManager finishing work
Writer 5 finishing work
Reader 3 finishing work
Reader 5 finishing work
Writer 7 finishing work
Reader 0 finishing work
Writer 1 finishing work
Writer 0 finishing work
Reader 9 finishing work
Writer 8 finishing work
Reader 8 finishing work
Reader 6 finishing work
Reader 7 finishing work
Writer 6 finishing work
Reader 1 finishing work
Writer 3 finishing work
Writer 2 finishing work
Writer 4 finishing work
Writer 9 finishing work
all is done
```

Рис. 1: Использование мьютексов.

Листинг 3: Основной файл

```
1 #include <windows.h>
2 #include <string.h>
3 #include <stdio.h>
4 #include <conio.h>
5 #include <tchar.h>
6
7 #include "thread.h"
8 #include "utils.h"
9 #include "Logger.h"
10
```

```

11 //глобальные переменные:
12 struct FIFOQueue queue; //структура очереди
13 struct Configuration config; //конфигурация программы
14 bool isDone = false; //Признак завершения
15 HANDLE *allhandlers; //массив всех создаваемых потоков
16 HANDLE mutex; // описатель мьютекса
17
18 int _tmain(int argc, _TCHAR* argv[]) {
19     Logger log(_T("Mutex"));
20
21     if (argc < 2)
22         // Используем конфигурацию по-умолчанию
23         SetDefaultConfig(&config, &log);
24     else
25         // Загрузка конфига из файла
26         SetConfig(argv[1], &config, &log);
27
28     //создаем необходимые потоки без их запуска
29     CreateAllThreads(&config, &log);
30
31     //Инициализируем очередь
32     queue.full = 0;
33     queue.readindex = 0;
34     queue.writeindex = 0;
35     queue.size = config.sizeOfQueue;
36     queue.data = new _TCHAR*[config.sizeOfQueue];
37     //инициализируем средство синхронизации
38     mutex = CreateMutex(NULL, FALSE, L "");
39     //     NULL - параметры безопасности
40     //     FALSE - создаваемый мьютекс никому изначально не принадлежит
41     //     "" - имя мьютекса
42
43     //запускаем потоки на исполнение
44     for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
45         ResumeThread(allhandlers[i]);
46
47     //ожидаем завершения всех потоков
48     WaitForMultipleObjects(config.numOfReaders + config.numOfWriters + 1,
49         allhandlers, TRUE, INFINITE);
50     //закрываем handle потоков
51     for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
52         CloseHandle(allhandlers[i]);
53     //удаляем объект синхронизации
54     CloseHandle(mutex);
55

```

```

56 // Очистка памяти
57 for (size_t i = 0; i != config.sizeOfQueue; ++i)
58     if (queue.data[i])
59         free(queue.data[i]); // _wcsdup использует calloc
60 delete[] queue.data;
61
62 // Завершение работы
63 log.loudlog(_T("All is done!"));
64 _getch();
65 return 0;
66 }

```

Листинг 4: Поток писателя

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadWriterHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("ThreadsReaderWriter.ThreadWriter"), myid);
11    extern bool isDone;
12    extern struct FIFOQueue queue;
13    extern struct Configuration config;
14    extern HANDLE mutex;
15
16    _TCHAR tmp[50];
17    int msgnum = 0; //номер передаваемого сообщения
18    while (isDone != true) {
19        //Захват синхронизирующего объекта
20        log.quietlog(_T("Waiting for Mutex"));
21        WaitForSingleObject(mutex, INFINITE);
22        log.quietlog(_T("Get Mutex"));
23
24        //если в очереди есть место
25        if (queue.readindex != queue.writeindex || !queue.full == 1) {
26            //записываем в очередь данные
27            swprintf_s(tmp, _T("writer_id = %d numMsg= %3d"), myid, msgnum);
28            queue.data[queue.writeindex] = _wcsdup(tmp);
29            msgnum++;
30
31            //печатаем принятые данные

```

```

32     log.loudlog(_T("Writer %d put data: \"%s\" in position %d"), myid,
33         queue.data[queue.writeindex], queue.writeindex);
34     queue.writeindex = (queue.writeindex + 1) % queue.size;
35     //если очередь заполнилась
36     queue.full = queue.writeindex == queue.readindex ? 1 : 0;
37 }
38 //освобождение объекта синхронизации
39 log.quietlog(_T("Release Mutex"));
40 ReleaseMutex(mutex);
41
42 //задержка
43 Sleep(config.writersDelay);
44 }
45 log.loudlog(_T("Writer %d finishing work"), myid);
46 return 0;
47 }

```

Листинг 5: Потоки читатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadReaderHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("Mutex.ThreadReader"), myid);
11    extern bool isDone;
12    extern struct FIFOQueue queue;
13    extern struct Configuration config;
14    extern HANDLE mutex;
15
16    while (isDone != true) {
17        //Захват объекта синхронизации
18        log.quietlog(_T("Waiting for Mutex"));
19        WaitForSingleObject(mutex, INFINITE);
20        log.quietlog(_T("Get Mutex"));
21
22        //если в очереди есть данные
23        if (queue.readindex != queue.writeindex || queue.full == 1) {
24            //взяли данные, значит очередь не пуста
25            queue.full = 0;
26            //печатаем принятые данные

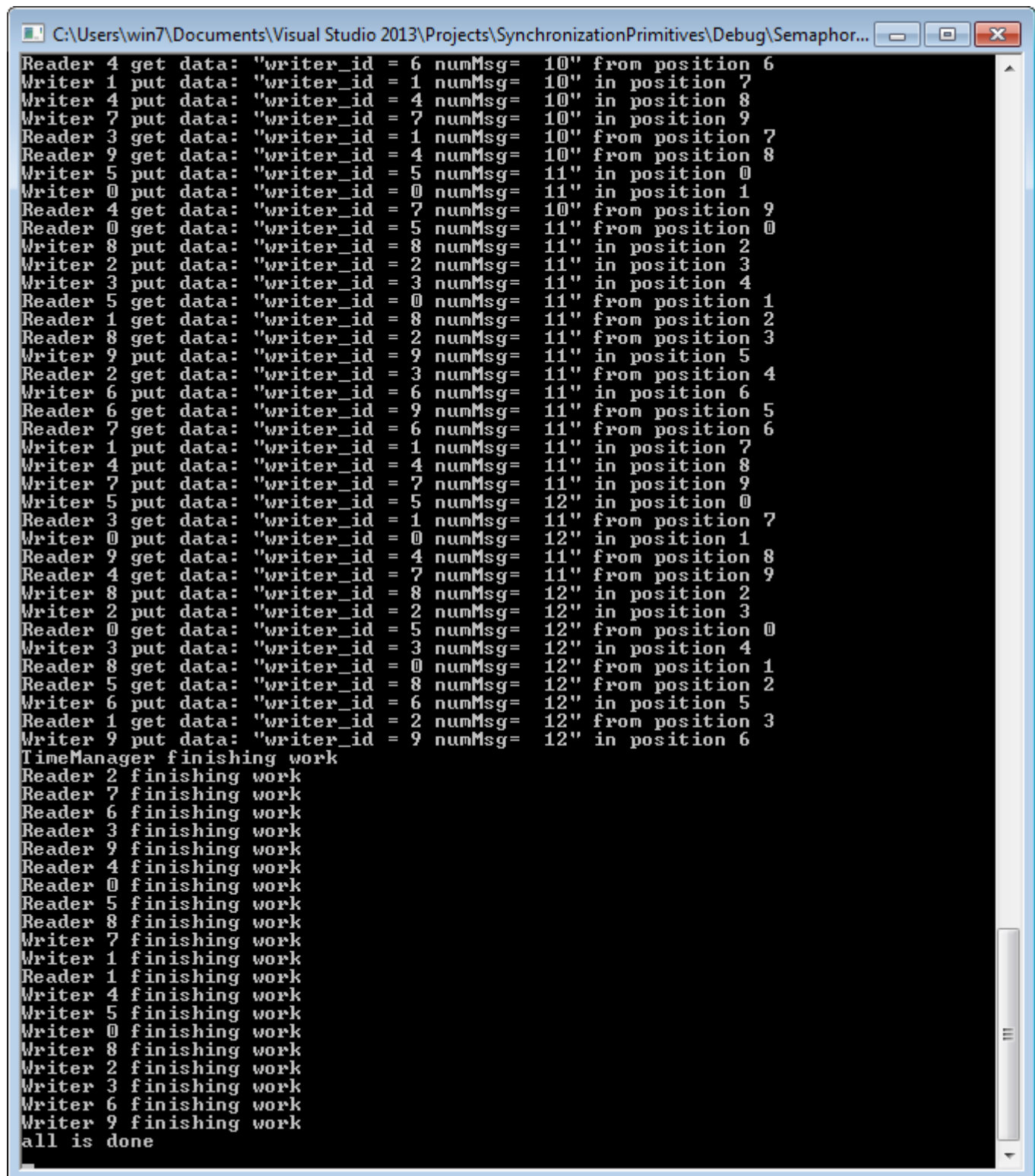
```

```

27     log.loudlog(_T("Reader %d get data: \"%s\" from position %d"), myid,
28         queue.data[queue.readindex], queue.readindex);
29     free(queue.data[queue.readindex]); //очищаем очередь от данных
30     queue.data[queue.readindex] = NULL;
31     queue.readindex = (queue.readindex + 1) % queue.size;
32 }
33 //Освобождение объекта синхронизации
34 log.quietlog(_T("Release Mutex"));
35 ReleaseMutex(mutex);
36
37 //задержка
38 Sleep(config.readersDelay);
39 }
40 log.loudlog(_T("Reader %d finishing work"), myid);
41 return 0;
42 }

```


1.2 Использование семафоров



```
C:\Users\win7\Documents\Visual Studio 2013\Projects\SynchronizationPrimitives\Debug\Semaphor...
Reader 4 get data: "writer_id = 6 numMsg= 10" from position 6
Writer 1 put data: "writer_id = 1 numMsg= 10" in position 7
Writer 4 put data: "writer_id = 4 numMsg= 10" in position 8
Writer 7 put data: "writer_id = 7 numMsg= 10" in position 9
Reader 3 get data: "writer_id = 1 numMsg= 10" from position 7
Reader 9 get data: "writer_id = 4 numMsg= 10" from position 8
Writer 5 put data: "writer_id = 5 numMsg= 11" in position 0
Writer 0 put data: "writer_id = 0 numMsg= 11" in position 1
Reader 4 get data: "writer_id = 7 numMsg= 10" from position 9
Reader 0 get data: "writer_id = 5 numMsg= 11" from position 0
Writer 8 put data: "writer_id = 8 numMsg= 11" in position 2
Writer 2 put data: "writer_id = 2 numMsg= 11" in position 3
Writer 3 put data: "writer_id = 3 numMsg= 11" in position 4
Reader 5 get data: "writer_id = 0 numMsg= 11" from position 1
Reader 1 get data: "writer_id = 8 numMsg= 11" from position 2
Reader 8 get data: "writer_id = 2 numMsg= 11" from position 3
Writer 9 put data: "writer_id = 9 numMsg= 11" in position 5
Reader 2 get data: "writer_id = 3 numMsg= 11" from position 4
Writer 6 put data: "writer_id = 6 numMsg= 11" in position 6
Reader 6 get data: "writer_id = 9 numMsg= 11" from position 5
Reader 7 get data: "writer_id = 6 numMsg= 11" from position 6
Writer 1 put data: "writer_id = 1 numMsg= 11" in position 7
Writer 4 put data: "writer_id = 4 numMsg= 11" in position 8
Writer 7 put data: "writer_id = 7 numMsg= 11" in position 9
Writer 5 put data: "writer_id = 5 numMsg= 12" in position 0
Reader 3 get data: "writer_id = 1 numMsg= 11" from position 7
Writer 0 put data: "writer_id = 0 numMsg= 12" in position 1
Reader 9 get data: "writer_id = 4 numMsg= 11" from position 8
Reader 4 get data: "writer_id = 7 numMsg= 11" from position 9
Writer 8 put data: "writer_id = 8 numMsg= 12" in position 2
Writer 2 put data: "writer_id = 2 numMsg= 12" in position 3
Reader 0 get data: "writer_id = 5 numMsg= 12" from position 0
Writer 3 put data: "writer_id = 3 numMsg= 12" in position 4
Reader 8 get data: "writer_id = 0 numMsg= 12" from position 1
Reader 5 get data: "writer_id = 8 numMsg= 12" from position 2
Writer 6 put data: "writer_id = 6 numMsg= 12" in position 5
Reader 1 get data: "writer_id = 2 numMsg= 12" from position 3
Writer 9 put data: "writer_id = 9 numMsg= 12" in position 6
TimeManager finishing work
Reader 2 finishing work
Reader 7 finishing work
Reader 6 finishing work
Reader 3 finishing work
Reader 9 finishing work
Reader 4 finishing work
Reader 0 finishing work
Reader 5 finishing work
Reader 8 finishing work
Writer 7 finishing work
Writer 1 finishing work
Reader 1 finishing work
Writer 4 finishing work
Writer 5 finishing work
Writer 0 finishing work
Writer 8 finishing work
Writer 2 finishing work
Writer 3 finishing work
Writer 6 finishing work
Writer 9 finishing work
all is done
```

Рис. 2: Использование семафоров.

Листинг 6: Основной файл

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

```

2 #include <string.h>
3 #include <stdio.h>
4 #include <conio.h>
5 #include <tchar.h>
6
7 #include "thread.h"
8 #include "utils.h"
9 #include "Logger.h"
10
11 //глобальные переменные:
12 struct FIFOQueue queue; //структура очереди
13 struct Configuration config; //конфигурация программы
14 bool isDone = false; //Признак завершения
15 HANDLE *allhandlers; //массив всех создаваемых потоков
16 HANDLE sem; // описатель семафора
17
18 int _tmain(int argc, _TCHAR* argv[]) {
19     Logger log(_T("Semaphore"));
20
21     if (argc < 2)
22         // Используем конфигурацию по-умолчанию
23         SetDefaultConfig(&config, &log);
24     else
25         // Загрузка конфига из файла
26         SetConfig(argv[1], &config, &log);
27
28     //создаем необходимые потоки без их запуска
29     CreateAllThreads(&config, &log);
30
31     //Инициализируем очередь
32     queue.full = 0;
33     queue.readindex = 0;
34     queue.writeindex = 0;
35     queue.size = config.sizeOfQueue;
36     queue.data = new _TCHAR*[config.sizeOfQueue];
37     //инициализируем средство синхронизации
38     sem = CreateSemaphore(NULL, 1, 1, L ""); // изначально семафор свободен
39     //     NULL - атрибуты безопасности
40     //     1 - Сколько свободно ресурсов в начале
41     //     1 - Сколько ресурсов всего
42     //     "" - Имя
43
44     //запускаем потоки на исполнение
45     for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
46         ResumeThread(allhandlers[i]);

```

```

47
48 //ожидаем завершения всех потоков
49 WaitForMultipleObjects(config.numOfReaders + config.numOfWriters + 1,
50     allhandlers, TRUE, INFINITE);
51 //закрываем handle потоков
52 for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
53     CloseHandle(allhandlers[i]);
54 //удаляем объект синхронизации
55 CloseHandle(sem);
56
57 // Очистка памяти
58 for (size_t i = 0; i != config.sizeOfQueue; ++i)
59     if (queue.data[i])
60         free(queue.data[i]); // _wcsdup использует calloc
61 delete[] queue.data;
62
63 // Завершение работы
64 log.loudlog(_T("All is done!"));
65 _getch();
66 return 0;
67 }

```

Листинг 7: Потоки писатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadWriterHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("Semaphore.ThreadWriter"), myid);
11    extern bool isDone;
12    extern struct FIFOQueue queue;
13    extern struct Configuration config;
14    extern HANDLE sem;
15
16    _TCHAR tmp[50];
17    int msgnum = 0; //номер передаваемого сообщения
18    while (isDone != true) {
19        //Захват синхронизирующего объекта
20        log.quietlog(_T("Waiting for Semaphore"));
21        WaitForSingleObject(sem, INFINITE);

```

```

22     log.quietlog(_T("Get Semaphore"));
23
24     //если в очереди есть место
25     if (queue.readindex != queue.writeindex || !queue.full == 1) {
26         //заношим в очередь данные
27         swprintf_s(tmp, _T("writer_id = %d numMsg= %3d"), myid, msgnum);
28         queue.data[queue.writeindex] = _wcsdup(tmp);
29         msgnum++;
30
31         //печатаем принятые данные
32         log.loudlog(_T("Writer %d put data: \"%s\" in position %d"), myid,
33             queue.data[queue.writeindex], queue.writeindex);
34         queue.writeindex = (queue.writeindex + 1) % queue.size;
35         //если очередь заполнилась
36         queue.full = queue.writeindex == queue.readindex ? 1 : 0;
37     }
38     //освобождение объекта синхронизации
39     log.quietlog(_T("Release Semaphore"));
40     ReleaseSemaphore(sem, 1, NULL);
41
42     //задержка
43     Sleep(config.writersDelay);
44 }
45 log.loudlog(_T("Writer %d finishing work"), myid);
46 return 0;
47 }

```

Листинг 8: Поток читателя

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadReaderHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("Semaphore.ThreadReader"), myid);
11    extern bool isDone;
12    extern struct FIFOQueue queue;
13    extern struct Configuration config;
14    extern HANDLE sem;
15
16    while (isDone != true) {

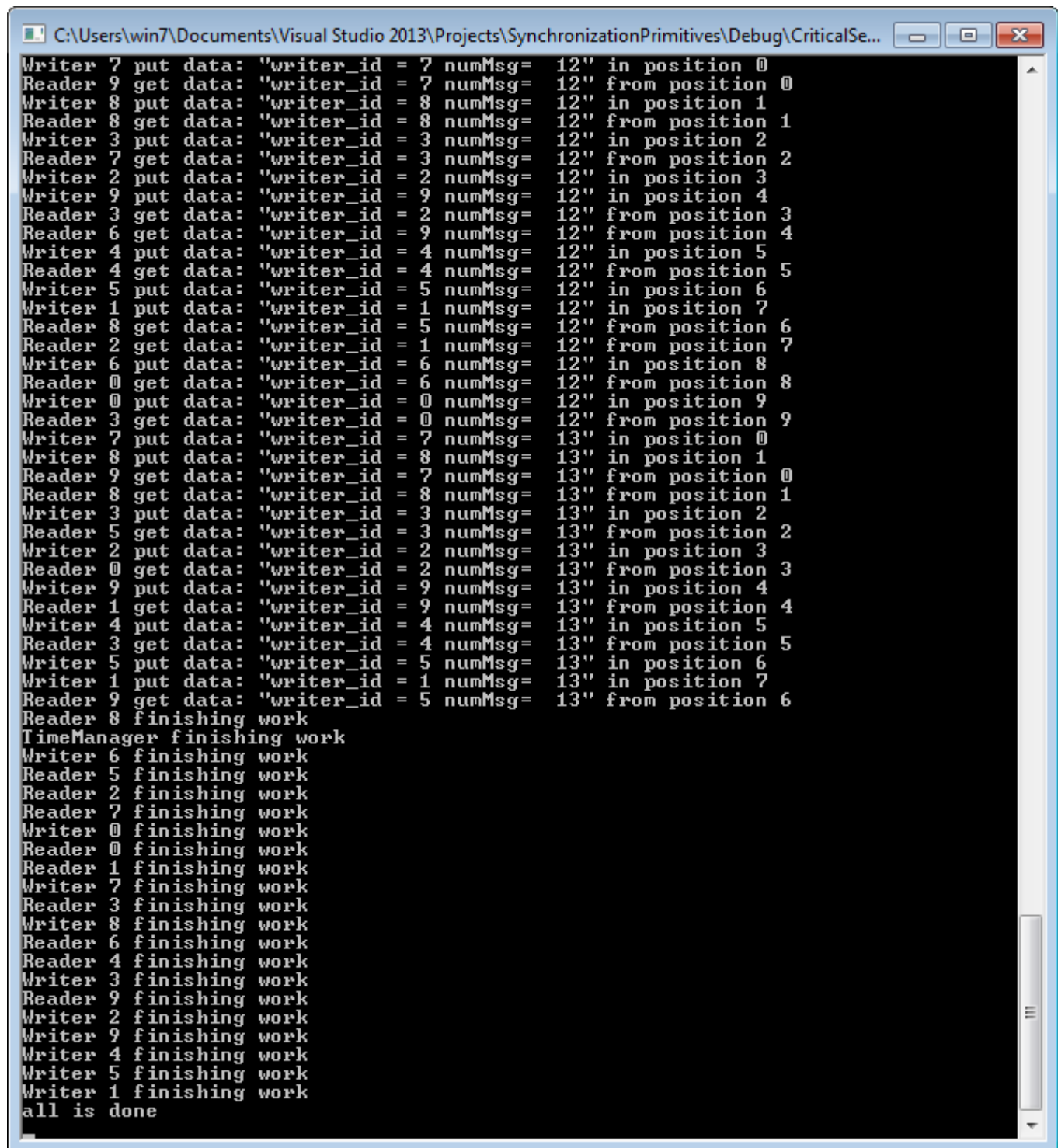
```

```

17 //Захват объекта синхронизации
18 log.quietlog(_T("Waiting for Semaphore"));
19 WaitForSingleObject(sem, INFINITE);
20 log.quietlog(_T("Get Semaphore"));
21
22 //если в очереди есть данные
23 if (queue.readindex != queue.writeindex || queue.full == 1) {
24     //взяли данные, значит очередь не пуста
25     queue.full = 0;
26     //печатаем принятые данные
27     log.loudlog(_T("Reader %d get data: \"%s\" from position %d"), myid,
28         queue.data[queue.readindex], queue.readindex);
29     free(queue.data[queue.readindex]); //очищаем очередь от данных
30     queue.data[queue.readindex] = NULL;
31     queue.readindex = (queue.readindex + 1) % queue.size;
32 }
33 //Освобождение объекта синхронизации
34 log.quietlog(_T("Release Semaphore"));
35 ReleaseSemaphore(sem, 1, NULL);
36
37 //задержка
38 Sleep(config.readersDelay);
39 }
40 log.loudlog(_T("Reader %d finishing work"), myid);
41 return 0;
42 }

```

1.3 Критические секции



```
C:\Users\win7\Documents\Visual Studio 2013\Projects\SynchronizationPrimitives\Debug\CriticalSe...
Writer 7 put data: "writer_id = 7 numMsg= 12" in position 0
Reader 9 get data: "writer_id = 7 numMsg= 12" from position 0
Writer 8 put data: "writer_id = 8 numMsg= 12" in position 1
Reader 8 get data: "writer_id = 8 numMsg= 12" from position 1
Writer 3 put data: "writer_id = 3 numMsg= 12" in position 2
Reader 7 get data: "writer_id = 3 numMsg= 12" from position 2
Writer 2 put data: "writer_id = 2 numMsg= 12" in position 3
Writer 9 put data: "writer_id = 9 numMsg= 12" in position 4
Reader 3 get data: "writer_id = 2 numMsg= 12" from position 3
Reader 6 get data: "writer_id = 9 numMsg= 12" from position 4
Writer 4 put data: "writer_id = 4 numMsg= 12" in position 5
Reader 4 get data: "writer_id = 4 numMsg= 12" from position 5
Writer 5 put data: "writer_id = 5 numMsg= 12" in position 6
Writer 1 put data: "writer_id = 1 numMsg= 12" in position 7
Reader 8 get data: "writer_id = 5 numMsg= 12" from position 6
Reader 2 get data: "writer_id = 1 numMsg= 12" from position 7
Writer 6 put data: "writer_id = 6 numMsg= 12" in position 8
Reader 0 get data: "writer_id = 6 numMsg= 12" from position 8
Writer 0 put data: "writer_id = 0 numMsg= 12" in position 9
Reader 3 get data: "writer_id = 0 numMsg= 12" from position 9
Writer 7 put data: "writer_id = 7 numMsg= 13" in position 0
Writer 8 put data: "writer_id = 8 numMsg= 13" in position 1
Reader 9 get data: "writer_id = 7 numMsg= 13" from position 0
Reader 8 get data: "writer_id = 8 numMsg= 13" from position 1
Writer 3 put data: "writer_id = 3 numMsg= 13" in position 2
Reader 5 get data: "writer_id = 3 numMsg= 13" from position 2
Writer 2 put data: "writer_id = 2 numMsg= 13" in position 3
Reader 0 get data: "writer_id = 2 numMsg= 13" from position 3
Writer 9 put data: "writer_id = 9 numMsg= 13" in position 4
Reader 1 get data: "writer_id = 9 numMsg= 13" from position 4
Writer 4 put data: "writer_id = 4 numMsg= 13" in position 5
Reader 3 get data: "writer_id = 4 numMsg= 13" from position 5
Writer 5 put data: "writer_id = 5 numMsg= 13" in position 6
Writer 1 put data: "writer_id = 1 numMsg= 13" in position 7
Reader 9 get data: "writer_id = 5 numMsg= 13" from position 6
Reader 8 finishing work
TimeManager finishing work
Writer 6 finishing work
Reader 5 finishing work
Reader 2 finishing work
Reader 7 finishing work
Writer 0 finishing work
Reader 0 finishing work
Reader 1 finishing work
Writer 7 finishing work
Reader 3 finishing work
Writer 8 finishing work
Reader 6 finishing work
Reader 4 finishing work
Writer 3 finishing work
Reader 9 finishing work
Writer 2 finishing work
Writer 9 finishing work
Writer 4 finishing work
Writer 5 finishing work
Writer 1 finishing work
all is done
```

Рис. 3: Критические секции.

Листинг 9: Основной файл

```
1 #include <windows.h>
2 #include <string.h>
3 #include <stdio.h>
```

```

4 #include <conio.h>
5 #include <tchar.h>
6
7 #include "thread.h"
8 #include "utils.h"
9 #include "Logger.h"
10
11 //глобальные переменные:
12 struct FIFOQueue queue; //структура очереди
13 struct Configuration config; //конфигурация программы
14 bool isDone = false; //Признак завершения
15 HANDLE *allhandlers; //массив всех создаваемых потоков
16 CRITICAL_SECTION crs; // Объявление критической секции
17
18 int _tmain(int argc, _TCHAR* argv[]) {
19     Logger log(_T("CriticalSection"));
20
21     if (argc < 2)
22         // Используем конфигурацию по-умолчанию
23         SetDefaultConfig(&config, &log);
24     else
25         // Загрузка конфига из файла
26         SetConfig(argv[1], &config, &log);
27
28     //создаем необходимые потоки без их запуска
29     CreateAllThreads(&config, &log);
30
31     //Инициализируем очередь
32     queue.full = 0;
33     queue.readindex = 0;
34     queue.writeindex = 0;
35     queue.size = config.sizeOfQueue;
36     queue.data = new _TCHAR*[config.sizeOfQueue];
37     //инициализируем средство синхронизации
38     InitializeCriticalSection(&crs);
39
40     //запускаем потоки на исполнение
41     for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
42         ResumeThread(allhandlers[i]);
43
44     //ожидаем завершения всех потоков
45     WaitForMultipleObjects(config.numOfReaders + config.numOfWriters + 1,
46         allhandlers, TRUE, INFINITE);
47     //закрываем handle потоков
48     for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)

```

```

49     CloseHandle(allhandlers[i]);
50     //удаляем объект синхронизации
51     DeleteCriticalSection(&crs);
52
53     // Очистка памяти
54     for (size_t i = 0; i != config.sizeOfQueue; ++i)
55         if (queue.data[i])
56             free(queue.data[i]); // _wcsdup используем calloc
57     delete[] queue.data;
58
59     // Завершение работы
60     log.loudlog(_T("All is done!"));
61     _getch();
62     return 0;
63 }

```

Листинг 10: Потоки писатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadWriterHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("CriticalSection.ThreadWriter"), myid);
11    extern bool isDone;
12    extern struct FIFOQueue queue;
13    extern struct Configuration config;
14    extern CRITICAL_SECTION crs;
15
16    _TCHAR tmp[50];
17    int msgnum = 0; //номер передаваемого сообщения
18    while (isDone != true) {
19        //Захват синхронизирующего объекта
20        log.quietlog(_T("Waiting for Critical Section"));
21        EnterCriticalSection(&crs);
22        log.quietlog(_T("Get Critical Section"));
23
24        //если в очереди есть место
25        if (queue.readindex != queue.writeindex || !queue.full == 1) {
26            //записываем в очередь данные
27            swprintf_s(tmp, _T("writer_id = %d numMsg = %3d"), myid, msgnum);

```



```

28     queue.data[queue.writeindex] = _wcsdup(tmp);
29     msgnum++;
30
31     //печатаем принятые данные
32     log.loudlog(_T("Writer %d put data: \"%s\" in position %d"), myid,
33         queue.data[queue.writeindex], queue.writeindex);
34     queue.writeindex = (queue.writeindex + 1) % queue.size;
35     //если очередь заполнилась
36     queue.full = queue.writeindex == queue.readindex ? 1 : 0;
37 }
38 //освобождение объекта синхронизации
39 log.quietlog(_T("Leave Critical Section"));
40 LeaveCriticalSection(&crs);
41
42 //задержка
43 Sleep(config.writersDelay);
44 }
45 log.loudlog(_T("Writer %d finishing work"), myid);
46 return 0;
47 }

```

Листинг 11: Потоки читатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadReaderHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("CriticalSection.ThreadReader"), myid);
11    extern bool isDone;
12    extern struct FIFOQueue queue;
13    extern struct Configuration config;
14    extern CRITICAL_SECTION crs;
15
16    while (isDone != true) {
17        //Захват объекта синхронизации
18        log.quietlog(_T("Waiting for Critical Section"));
19        EnterCriticalSection(&crs);
20        log.quietlog(_T("Get Critical Section"));
21
22        //если в очереди есть данные

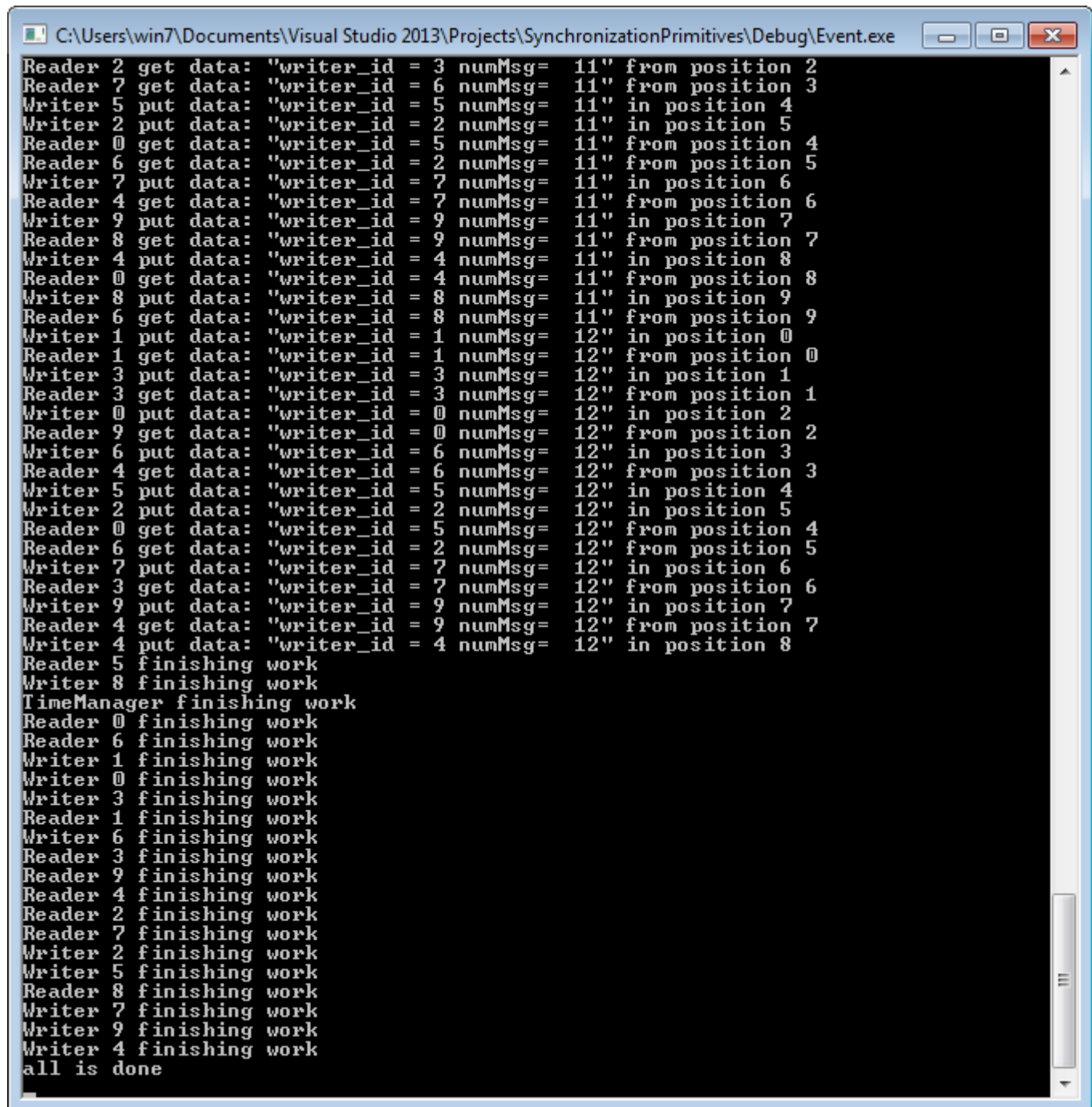
```

```

23     if (queue.readindex != queue.writeindex || queue.full == 1) {
24         //взяли данные, значит очередь не пуста
25         queue.full = 0;
26         //печатаем принятые данные
27         log.loudlog(_T("Reader %d get data: \"%s\" from position %d"), myid,
28             queue.data[queue.readindex], queue.readindex);
29         free(queue.data[queue.readindex]); //очищаем очередь от данных
30         queue.data[queue.readindex] = NULL;
31         queue.readindex = (queue.readindex + 1) % queue.size;
32     }
33     //Освобождение объекта синхронизации
34     log.quietlog(_T("Leave Critical Section"));
35     LeaveCriticalSection(&crs);
36
37     //задержка
38     Sleep(config.readersDelay);
39 }
40 log.loudlog(_T("Reader %d finishing work"), myid);
41 return 0;
42 }

```

1.4 Объекты-события в качестве средства синхронизации



```
C:\Users\win7\Documents\Visual Studio 2013\Projects\SynchronizationPrimitives\Debug\Event.exe
Reader 2 get data: "writer_id = 3 numMsg= 11" from position 2
Reader 7 get data: "writer_id = 6 numMsg= 11" from position 3
Writer 5 put data: "writer_id = 5 numMsg= 11" in position 4
Writer 2 put data: "writer_id = 2 numMsg= 11" in position 5
Reader 0 get data: "writer_id = 5 numMsg= 11" from position 4
Reader 6 get data: "writer_id = 2 numMsg= 11" from position 5
Writer 7 put data: "writer_id = 7 numMsg= 11" in position 6
Reader 4 get data: "writer_id = 7 numMsg= 11" from position 6
Writer 9 put data: "writer_id = 9 numMsg= 11" in position 7
Reader 8 get data: "writer_id = 9 numMsg= 11" from position 7
Writer 4 put data: "writer_id = 4 numMsg= 11" in position 8
Reader 0 get data: "writer_id = 4 numMsg= 11" from position 8
Writer 8 put data: "writer_id = 8 numMsg= 11" in position 9
Reader 6 get data: "writer_id = 8 numMsg= 11" from position 9
Writer 1 put data: "writer_id = 1 numMsg= 12" in position 0
Reader 1 get data: "writer_id = 1 numMsg= 12" from position 0
Writer 3 put data: "writer_id = 3 numMsg= 12" in position 1
Reader 3 get data: "writer_id = 3 numMsg= 12" from position 1
Writer 0 put data: "writer_id = 0 numMsg= 12" in position 2
Reader 9 get data: "writer_id = 0 numMsg= 12" from position 2
Writer 6 put data: "writer_id = 6 numMsg= 12" in position 3
Reader 4 get data: "writer_id = 6 numMsg= 12" from position 3
Writer 5 put data: "writer_id = 5 numMsg= 12" in position 4
Writer 2 put data: "writer_id = 2 numMsg= 12" in position 5
Reader 0 get data: "writer_id = 5 numMsg= 12" from position 4
Reader 6 get data: "writer_id = 2 numMsg= 12" from position 5
Writer 7 put data: "writer_id = 7 numMsg= 12" in position 6
Reader 3 get data: "writer_id = 7 numMsg= 12" from position 6
Writer 9 put data: "writer_id = 9 numMsg= 12" in position 7
Reader 4 get data: "writer_id = 9 numMsg= 12" from position 7
Writer 4 put data: "writer_id = 4 numMsg= 12" in position 8
Reader 5 finishing work
Writer 8 finishing work
TimeManager finishing work
Reader 0 finishing work
Reader 6 finishing work
Writer 1 finishing work
Writer 0 finishing work
Writer 3 finishing work
Reader 1 finishing work
Writer 6 finishing work
Reader 3 finishing work
Reader 9 finishing work
Reader 4 finishing work
Reader 2 finishing work
Reader 7 finishing work
Writer 2 finishing work
Writer 5 finishing work
Reader 8 finishing work
Writer 7 finishing work
Writer 9 finishing work
Writer 4 finishing work
all is done
```

Рис. 4: Объекты-события в качестве средства синхронизации.

Листинг 12: Основной файл

```
1 #include <windows.h>
2 #include <string.h>
3 #include <stdio.h>
4 #include <conio.h>
5 #include <tchar.h>
```

```

6
7 #include "thread.h"
8 #include "utils.h"
9 #include "Logger.h"
10
11 //глобальные переменные:
12 struct FIFOQueue queue; //структура очереди
13 struct Configuration config; //конфигурация программы
14 bool isDone = false; //Признак завершения
15 HANDLE *allhandlers; //массив всех создаваемых потоков
16 HANDLE event; // объект-событие
17
18 int _tmain(int argc, _TCHAR* argv[]) {
19     Logger log(_T("Event"));
20
21     if (argc < 2)
22         // Используем конфигурацию по-умолчанию
23         SetDefaultConfig(&config, &log);
24     else
25         // Загрузка конфига из файла
26         SetConfig(argv[1], &config, &log);
27
28     //создаем необходимые потоки без их запуска
29     CreateAllThreads(&config, &log);
30
31     //Инициализируем очередь
32     queue.full = 0;
33     queue.readindex = 0;
34     queue.writeindex = 0;
35     queue.size = config.sizeOfQueue;
36     queue.data = new _TCHAR*[config.sizeOfQueue];
37     //инициализируем средство синхронизации
38     event = CreateEvent(NULL, false, true, L "");
39     //     NULL - атрибуты защиты
40     //     false - режим переключения (без автосброса, после захвата
41     //             потоком события, оно его нужно сделать занятым)
42     //     true - начальное состояние события (свободное)
43     //     "" - имя
44
45     //запускаем потоки на исполнение
46     for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
47         ResumeThread(allhandlers[i]);
48
49     //ожидаем завершения всех потоков
50     WaitForMultipleObjects(config.numOfReaders + config.numOfWriters + 1,

```

```

51     allhandlers, TRUE, INFINITE);
52     //закрываем handle потоков
53     for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
54         CloseHandle(allhandlers[i]);
55     //удаляем объект синхронизации
56     CloseHandle(event);
57
58     // Очистка памяти
59     for (size_t i = 0; i != config.sizeOfQueue; ++i)
60         if (queue.data[i])
61             free(queue.data[i]); // _wcsdup использует calloc
62     delete[] queue.data;
63
64     // Завершение работы
65     log.loudlog(_T("All is done!"));
66     _getch();
67     return 0;
68 }

```

Листинг 13: Потоки писатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadWriterHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("Event.ThreadWriter"), myid);
11    extern bool isDone;
12    extern struct FIFOQueue queue;
13    extern struct Configuration config;
14    extern HANDLE event;
15
16    _TCHAR tmp[50];
17    int msgnum = 0; //номер передаваемого сообщения
18    while (isDone != true) {
19        //Захват синхронизирующего объекта
20        log.quietlog(_T("Waiting for Event"));
21        WaitForSingleObject(event, INFINITE);
22        log.quietlog(_T("Get Event"));
23
24        //если в очереди есть место

```

```

25     if (queue.readindex != queue.writeindex || !queue.full == 1) {
26         //заносим в очередь данные
27         swprintf_s(tmp, _T("writer_id = %d numMsg= %3d"), myid, msgnum);
28         queue.data[queue.writeindex] = _wcsdup(tmp);
29         msgnum++;
30
31         //печатаем принятые данные
32         log.loudlog(_T("Writer %d put data: \"%s\" in position %d"), myid,
33             queue.data[queue.writeindex], queue.writeindex);
34         queue.writeindex = (queue.writeindex + 1) % queue.size;
35         //если очередь заполнилась
36         queue.full = queue.writeindex == queue.readindex ? 1 : 0;
37     }
38     //освобождение объекта синхронизации
39     log.quietlog(_T("Set Event"));
40     SetEvent(event);
41
42     //задержка
43     Sleep(config.writersDelay);
44 }
45 log.loudlog(_T("Writer %d finishing work"), myid);
46 return 0;
47 }

```

Листинг 14: Поток читателя

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadReaderHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("Event.ThreadReader"), myid);
11    extern bool isDone;
12    extern struct FIFOQueue queue;
13    extern struct Configuration config;
14    extern HANDLE event;
15
16    while (isDone != true) {
17        //Захват объекта синхронизации
18        log.quietlog(_T("Waiting for Event"));
19        WaitForSingleObject(event, INFINITE);

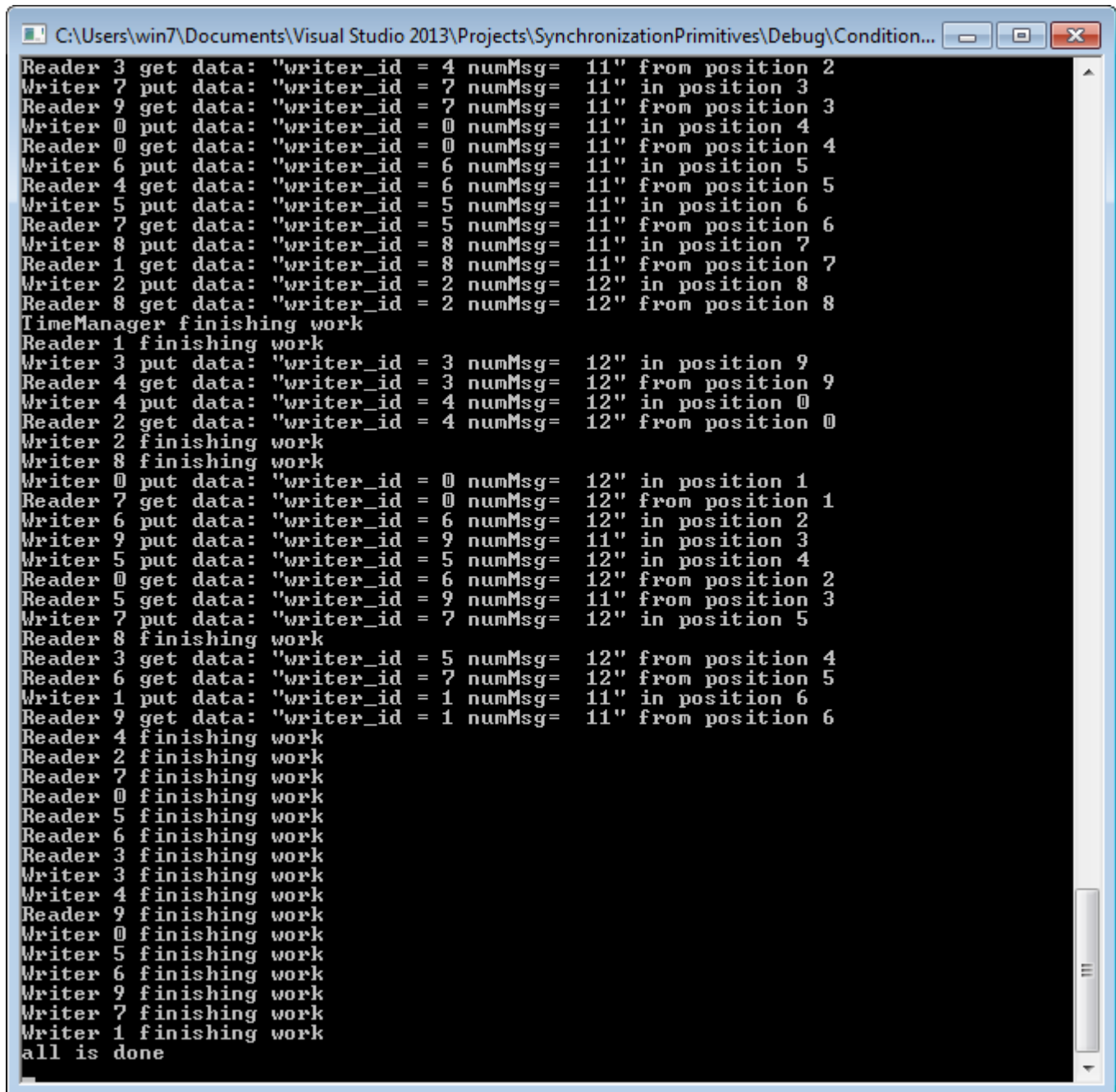
```

```

20     log.quietlog(_T("Get Event"));
21
22     //если в очереди есть данные
23     if (queue.readindex != queue.writeindex || queue.full == 1) {
24         //взяли данные, значит очередь не пуста
25         queue.full = 0;
26         //печатаем принятые данные
27         log.loudlog(_T("Reader %d get data: \"%s\" from position %d"), myid,
28             queue.data[queue.readindex], queue.readindex);
29         free(queue.data[queue.readindex]); //очищаем очередь от данных
30         queue.data[queue.readindex] = NULL;
31         queue.readindex = (queue.readindex + 1) % queue.size;
32     }
33     //Освобождение объекта синхронизации
34     log.quietlog(_T("Release Event"));
35     SetEvent(event);
36
37     //задержка
38     Sleep(config.readersDelay);
39 }
40 log.loudlog(_T("Reader %d finishing work"), myid);
41 return 0;
42 }

```

1.5 Условные переменные



```
C:\Users\win7\Documents\Visual Studio 2013\Projects\SynchronizationPrimitives\Debug\Condition...
Reader 3 get data: "writer_id = 4 numMsg= 11" from position 2
Writer 7 put data: "writer_id = 7 numMsg= 11" in position 3
Reader 9 get data: "writer_id = 7 numMsg= 11" from position 3
Writer 0 put data: "writer_id = 0 numMsg= 11" in position 4
Reader 0 get data: "writer_id = 0 numMsg= 11" from position 4
Writer 6 put data: "writer_id = 6 numMsg= 11" in position 5
Reader 4 get data: "writer_id = 6 numMsg= 11" from position 5
Writer 5 put data: "writer_id = 5 numMsg= 11" in position 6
Reader 7 get data: "writer_id = 5 numMsg= 11" from position 6
Writer 8 put data: "writer_id = 8 numMsg= 11" in position 7
Reader 1 get data: "writer_id = 8 numMsg= 11" from position 7
Writer 2 put data: "writer_id = 2 numMsg= 12" in position 8
Reader 8 get data: "writer_id = 2 numMsg= 12" from position 8
TimeManager finishing work
Reader 1 finishing work
Writer 3 put data: "writer_id = 3 numMsg= 12" in position 9
Reader 4 get data: "writer_id = 3 numMsg= 12" from position 9
Writer 4 put data: "writer_id = 4 numMsg= 12" in position 0
Reader 2 get data: "writer_id = 4 numMsg= 12" from position 0
Writer 2 finishing work
Writer 8 finishing work
Writer 0 put data: "writer_id = 0 numMsg= 12" in position 1
Reader 7 get data: "writer_id = 0 numMsg= 12" from position 1
Writer 6 put data: "writer_id = 6 numMsg= 12" in position 2
Writer 9 put data: "writer_id = 9 numMsg= 11" in position 3
Writer 5 put data: "writer_id = 5 numMsg= 12" in position 4
Reader 0 get data: "writer_id = 6 numMsg= 12" from position 2
Reader 5 get data: "writer_id = 9 numMsg= 11" from position 3
Writer 7 put data: "writer_id = 7 numMsg= 12" in position 5
Reader 8 finishing work
Reader 3 get data: "writer_id = 5 numMsg= 12" from position 4
Reader 6 get data: "writer_id = 7 numMsg= 12" from position 5
Writer 1 put data: "writer_id = 1 numMsg= 11" in position 6
Reader 9 get data: "writer_id = 1 numMsg= 11" from position 6
Reader 4 finishing work
Reader 2 finishing work
Reader 7 finishing work
Reader 0 finishing work
Reader 5 finishing work
Reader 6 finishing work
Reader 3 finishing work
Writer 3 finishing work
Writer 4 finishing work
Reader 9 finishing work
Writer 0 finishing work
Writer 5 finishing work
Writer 6 finishing work
Writer 9 finishing work
Writer 7 finishing work
Writer 1 finishing work
all is done
```

Рис. 5: Условные переменные.

Листинг 15: Основной файл

```
1 #include <windows.h>
2 #include <string.h>
3 #include <stdio.h>
4 #include <conio.h>
5 #include <tchar.h>
6
```



```

7 #include "thread.h"
8 #include "utils.h"
9 #include "Logger.h"
10
11 //глобальные переменные:
12 struct FIFOQueue queue; //структура очереди
13 struct Configuration config; //конфигурация программы
14 bool isDone = false; //Признак завершения
15 HANDLE *allhandlers; //массив всех создаваемых потоков
16
17 //критическая секция общая и для писателей и для читателей
18 CRITICAL_SECTION crs;
19 //условная переменная для потоков-писателей
20 CONDITION_VARIABLE condread;
21 //условная переменная для потоков-читателей
22 CONDITION_VARIABLE condwrite;
23
24 int _tmain(int argc, _TCHAR* argv[]) {
25     Logger log(_T("ConditionVariable"));
26
27     if (argc < 2)
28         // Используем конфигурацию по-умолчанию
29         SetDefaultConfig(&config, &log);
30     else
31         // Загрузка конфига из файла
32         SetConfig(argv[1], &config, &log);
33
34     //создаем необходимые потоки без их запуска
35     CreateAllThreads(&config, &log);
36
37     //Инициализируем очередь
38     queue.full = 0;
39     queue.readindex = 0;
40     queue.writeindex = 0;
41     queue.size = config.sizeOfQueue;
42     queue.data = new _TCHAR*[config.sizeOfQueue];
43     //инициализируем средство синхронизации
44     InitializeCriticalSection(&crs);
45     InitializeConditionVariable(&condread);
46     InitializeConditionVariable(&condwrite);
47
48     //запускаем потоки на исполнение
49     for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
50         ResumeThread(allhandlers[i]);
51

```

```

52 //ожидаем завершения всех потоков
53 WaitForMultipleObjects(config.numOfReaders + config.numOfWriters + 1,
54     allhandlers, TRUE, 5000);
55
56 //закрываем handle потоков
57 for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
58     CloseHandle(allhandlers[i]);
59 //удаляем объект синхронизации
60 DeleteCriticalSection(&crs);
61
62 // Очистка памяти
63 for (size_t i = 0; i != config.sizeOfQueue; ++i)
64     if (queue.data[i])
65         free(queue.data[i]); // _wcsdup использует calloc
66 delete[] queue.data;
67
68 // Завершение работы
69 log.loudlog(_T("All is done!"));
70 _getch();
71 return 0;
72 }

```

Листинг 16: Потоки писатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadWriterHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("ConditionVariable.ThreadWriter"), myid);
11    extern bool isDone;
12    extern struct FIFOQueue queue;
13    extern struct Configuration config;
14    extern CRITICAL_SECTION crs;
15    extern CONDITION_VARIABLE condread;
16    extern CONDITION_VARIABLE condwrite;
17
18    _TCHAR tmp[50];
19    int msgnum = 0; //номер передаваемого сообщения
20    while (isDone != true) {
21        //Захват синхронизирующего объекта

```

```

22     log.quietlog(_T("Waining for Critical Section"));
23     EnterCriticalSection(&crs);
24     log.quietlog(_T("Get Critical Section"));
25
26     log.quietlog(_T("Waining for empty space in the queue"));
27     while (!(queue.readindex != queue.writeindex || !queue.full == 1))
28         //сним пока в очереди не освободится место
29         SleepConditionVariableCS(&condwrite, &crs, INFINITE);
30     log.quietlog(_T("Get space in the queue"));
31
32     //вносим в очередь данные
33     swprintf_s(tmp, _T("writer_id = %d numMsg= %3d"), myid, msgnum);
34     queue.data[queue.writeindex] = _wcsdup(tmp);
35     msgnum++;
36
37     //печатаем принятые данные
38     log.loudlog(_T("Writer %d put data: \"%s\" in position %d"), myid,
39         queue.data[queue.writeindex], queue.writeindex);
40     queue.writeindex = (queue.writeindex + 1) % queue.size;
41     //если очередь заполнилась
42     queue.full = queue.writeindex == queue.readindex ? 1 : 0;
43
44     if (queue.full == 1)
45         log.loudlog(_T("Queue is full"));
46     //шлем сигнал потокам-читателям
47     log.quietlog(_T("Wake Condition Variable"));
48     WakeConditionVariable(&condread);
49     //освобождение синхронизируемого объекта
50     log.quietlog(_T("Leave Critical Section"));
51     LeaveCriticalSection(&crs);
52
53     //задержка
54     Sleep(config.writersDelay);
55 }
56 log.loudlog(_T("Writer %d finishing work"), myid);
57 return 0;
58 }

```

Листинг 17: Потоки читатели

```

1 #include<windows.h>
2 #include<stdio.h>
3
4 #include"utils.h"
5

```

```

6 DWORD WINAPI ThreadReaderHandler(LPVOID prm) {
7     int myid = (int)prm;
8
9     Logger log(_T("ConditionVariable.ThreadReader"), myid);
10    extern bool isDone;
11    extern struct FIFOQueue queue;
12    extern struct Configuration config;
13    extern CRITICAL_SECTION crs;
14    extern CONDITION_VARIABLE condread;
15    extern CONDITION_VARIABLE condwrite;
16
17    while (isDone != true) {
18        //Захват объекта синхронизации
19        log.quietlog(_T("Waining for Critical Section"));
20        EnterCriticalSection(&crs);
21        log.quietlog(_T("Get Critical Section"));
22
23        log.quietlog(_T("Waining for empty space in the queue"));
24        while (!(queue.readindex != queue.writeindex || queue.full == 1))
25            //сним пока в очереди не появятся данные
26            SleepConditionVariableCS(&condread, &crs, INFINITE);
27        log.quietlog(_T("Get space in the queue"));
28
29        //взяли данные, значит очередь не пуста
30        queue.full = 0;
31        //печатаем принятые данные
32        log.loudlog(_T("Reader %d get data: \"%s\" from position %d"), myid,
33            queue.data[queue.readindex], queue.readindex);
34        free(queue.data[queue.readindex]); //очищаем очередь от данных
35        queue.data[queue.readindex] = NULL;
36        queue.readindex = (queue.readindex + 1) % queue.size;
37
38        //шлем сигнал потокам-читателям
39        log.quietlog(_T("Wake Condition Variable"));
40        WakeConditionVariable(&condwrite);
41        //освобождение синхронизируемого объекта
42        log.quietlog(_T("Leave Critical Section"));
43        LeaveCriticalSection(&crs);
44
45        //задержка
46        Sleep(config.readersDelay);
47    }
48    log.loudlog(_T("Reader %d finishing work"), myid);
49    return 0;
50 }

```

1.6 Задача читателя-писателя (для потоков одного процесса)

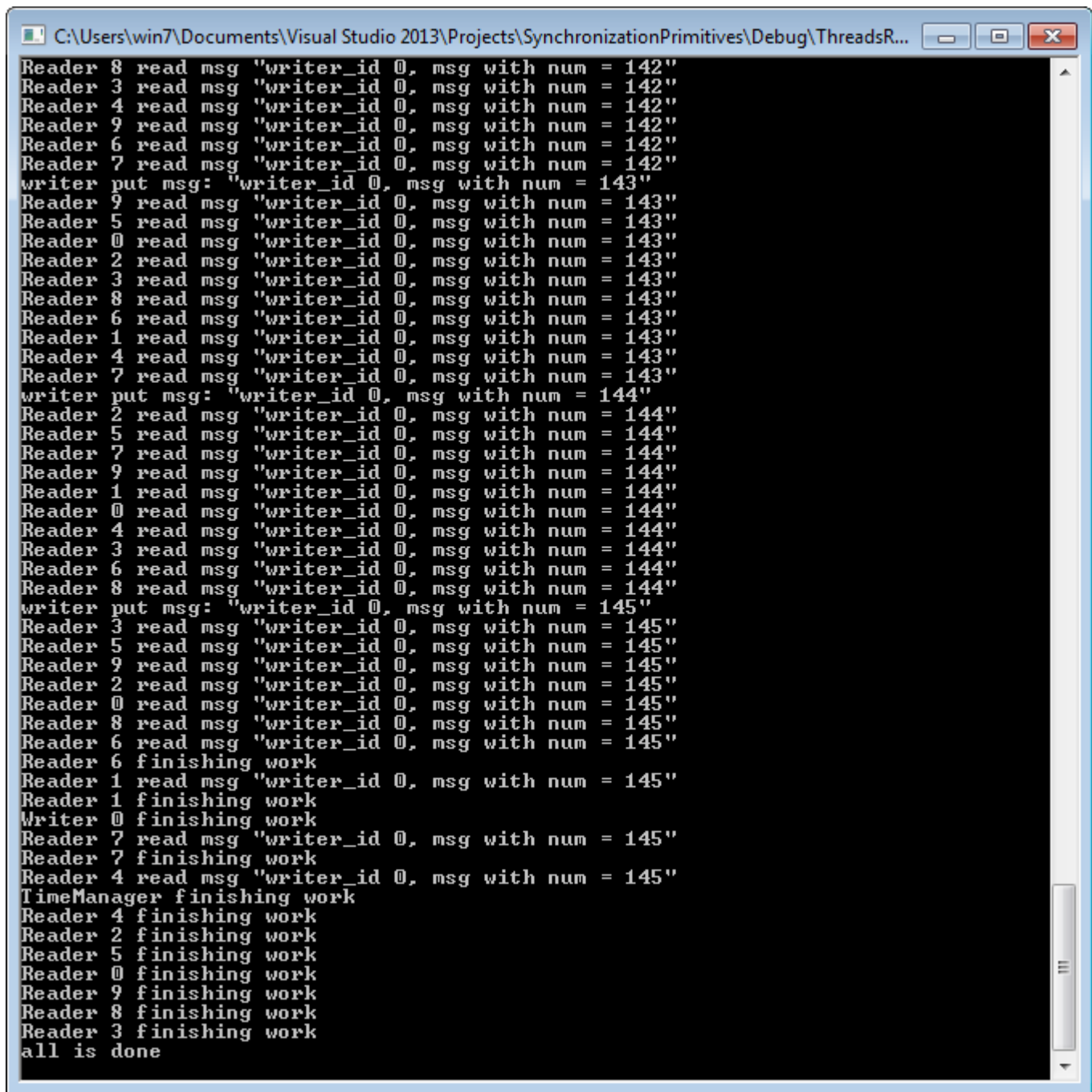
Рассмотрим частный случай этой задачи для демонстрации использования объектов-событий для синхронизации доступа к памяти.

Задание: необходимо решить задачу одного писателя и N читателей. Для синхронизации разрешено использовать только объекты-события, в качестве разделяемого ресурса – разделяемую память (share memory). Писатель пишет в share memory сообщение и ждет, пока все читатели не прочитают данное сообщение.

Задача должна быть решена сначала для потоков, принадлежащих одному процессу, а затем – разным независимым процессам.

Листинг 18: Основной файл

```
1 #include <windows.h>
2 #include <string.h>
3 #include <stdio.h>
4 #include <conio.h>
5 #include <tchar.h>
6
7 #include "thread.h"
8 #include "utils.h"
9 #include "Logger.h"
10
11 //глобальные переменные:
12 struct Configuration config; //конфигурация программы
13 bool isDone = false; //флаг завершения
14 HANDLE *allhandlers; //массив всех создаваемых потоков
15
16 //события для синхронизации:
17 HANDLE canReadEvent; //писатель записал сообщение (ручной сброс);
18 HANDLE canWriteEvent; //все читатели готовы к приему следующего (автосброс);
19 HANDLE allReadEvent; //все читатели прочитали сообщение (ручной сброс);
20 HANDLE changeCountEvent; //разрешение работы со счетчиком (автосброс);
21 HANDLE exitEvent; //завершение программы (ручной сброс);
22
23 //переменные для синхронизации работы потоков:
24 int countread = 0; //число потоков, которое уже прочитали данные
25 //                               (устанавливается писателем и изменяется
26 //                               читателями после прочтения сообщения)
27 int countready = 0; //число потоков, готовых для чтения сообщения
28 //                               (ожидаящих сигнала от писателя)
29
30 //имя разделяемой памяти
31 wchar_t shareFileName[] = L"$MyVerySpecialShareFileName$";
32
```



```
C:\Users\win7\Documents\Visual Studio 2013\Projects\SynchronizationPrimitives\Debug\ThreadsR...
Reader 8 read msg "writer_id 0, msg with num = 142"
Reader 3 read msg "writer_id 0, msg with num = 142"
Reader 4 read msg "writer_id 0, msg with num = 142"
Reader 9 read msg "writer_id 0, msg with num = 142"
Reader 6 read msg "writer_id 0, msg with num = 142"
Reader 7 read msg "writer_id 0, msg with num = 142"
writer put msg: "writer_id 0, msg with num = 143"
Reader 9 read msg "writer_id 0, msg with num = 143"
Reader 5 read msg "writer_id 0, msg with num = 143"
Reader 0 read msg "writer_id 0, msg with num = 143"
Reader 2 read msg "writer_id 0, msg with num = 143"
Reader 3 read msg "writer_id 0, msg with num = 143"
Reader 8 read msg "writer_id 0, msg with num = 143"
Reader 6 read msg "writer_id 0, msg with num = 143"
Reader 1 read msg "writer_id 0, msg with num = 143"
Reader 4 read msg "writer_id 0, msg with num = 143"
Reader 7 read msg "writer_id 0, msg with num = 143"
writer put msg: "writer_id 0, msg with num = 144"
Reader 2 read msg "writer_id 0, msg with num = 144"
Reader 5 read msg "writer_id 0, msg with num = 144"
Reader 7 read msg "writer_id 0, msg with num = 144"
Reader 9 read msg "writer_id 0, msg with num = 144"
Reader 1 read msg "writer_id 0, msg with num = 144"
Reader 0 read msg "writer_id 0, msg with num = 144"
Reader 4 read msg "writer_id 0, msg with num = 144"
Reader 3 read msg "writer_id 0, msg with num = 144"
Reader 6 read msg "writer_id 0, msg with num = 144"
Reader 8 read msg "writer_id 0, msg with num = 144"
writer put msg: "writer_id 0, msg with num = 145"
Reader 3 read msg "writer_id 0, msg with num = 145"
Reader 5 read msg "writer_id 0, msg with num = 145"
Reader 9 read msg "writer_id 0, msg with num = 145"
Reader 2 read msg "writer_id 0, msg with num = 145"
Reader 0 read msg "writer_id 0, msg with num = 145"
Reader 8 read msg "writer_id 0, msg with num = 145"
Reader 6 read msg "writer_id 0, msg with num = 145"
Reader 6 finishing work
Reader 1 read msg "writer_id 0, msg with num = 145"
Reader 1 finishing work
Writer 0 finishing work
Reader 7 read msg "writer_id 0, msg with num = 145"
Reader 7 finishing work
Reader 4 read msg "writer_id 0, msg with num = 145"
TimeManager finishing work
Reader 4 finishing work
Reader 2 finishing work
Reader 5 finishing work
Reader 0 finishing work
Reader 9 finishing work
Reader 8 finishing work
Reader 3 finishing work
all is done
```

Рис. 6: Задача читатели и писатели.

```
33 HANDLE hFileMapping; //объект-отображение файла
34 // указатели на отображаемую память
35 LPVOID lpFileMapForWriters;
36 LPVOID lpFileMapForReaders;
37
38 int _tmain(int argc, _TCHAR* argv[]) {
39     Logger log(_T("ThreadsReaderWriter"));
40
41     if (argc < 2)
```

```

42 // Используем конфигурацию по-умолчанию
43 SetDefaultConfig(&config, &log);
44 else
45 // Загрузка конфига из файла
46 SetConfig(argv[1], &config, &log);
47
48 //создаем необходимые потоки без их запуска
49 CreateAllThreads(&config, &log);
50
51 //Инициализируем ресурс (share memory): создаем объект "отображаемый файл"
52 // будет использован системный файл подкачки (на диске файл создаваться
53 // не будет), т.к. в качестве дескриптора файла использовано значение
54 // равное 0xFFFFFFFF (его эквивалент - символическая константа
    INVALID_HANDLE_VALUE)
55 if ((hFileMapping = CreateFileMapping(INVALID_HANDLE_VALUE, NULL,
56 PAGE_READWRITE, 0, 1500, shareFileName)) == NULL) {
57 // INVALID_HANDLE_VALUE - дескриптор открытого файла
58 // (INVALID_HANDLE_VALUE - файл подкачки)
59 // NULL - атрибуты защиты объекта-отображения
60 // PAGE_READWRITE - возможности доступа к представлению файла при
61 // отображении (PAGE_READWRITE - чтение/запись)
62 // 0, 1500 - старшая и младшая части значения максимального
63 // размера объекта отображения файла
64 // shareFileName - имя объекта-отображения.
65 log.loudlog(_T("Impossible to create shareFile, GLE = %d"),
66 GetLastError());
67 ExitProcess(10000);
68 }
69 //отображаем файл на адресное пространство нашего процесса для потока-писа
    теля
70 lpFileMapForWriters = MapViewOfFile(hFileMapping, FILE_MAP_WRITE, 0, 0, 0)
    ;
71 // hFileMapping - дескриптор объекта-отображения файла
72 // FILE_MAP_WRITE - доступа к файлу
73 // 0, 0 - старшая и младшая части смещения начала отображаемого участка в
    файле
74 // (0 - начало отображаемого участка совпадает с началом файла)
75 // 0 - размер отображаемого участка файла в байтах (0 - весь файл)
76
77 //отображаем файл на адресное пространство нашего процесса для потоков-чит
    ателей
78 lpFileMapForReaders = MapViewOfFile(hFileMapping, FILE_MAP_READ, 0, 0, 0);
79
80 //инициализируем средства синхронизации
81 // (атрибуты защиты, автосброс, начальное состояние, имя):

```



```

82 //событие "окончание записи" (можно читать), ручной сброс, изначально заня
    то
83 canReadEvent = CreateEvent(NULL, true, false, L "");
84 //событие - "можно писать", автосброс(разрешаем писать только одному), изна
    чально свободно
85 canWriteEvent = CreateEvent(NULL, false, false, L "");
86 //событие "все прочитали"
87 allReadEvent = CreateEvent(NULL, true, true, L "");
88 //событие для изменения счетчика (сколько клиентов еще не прочитало сообще
    ние)
89 changeCountEvent = CreateEvent(NULL, false, true, L "");
90 //событие "завершение работы программы", ручной сброс, изначально занято
91 exitEvent = CreateEvent(NULL, true, false, L "");
92
93 //запускаем потоки-писатели и поток-планировщик на исполнение
94 for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
95     ResumeThread(allhandlers[i]);
96
97 //ожидаем завершения всех потоков
98 WaitForMultipleObjects(config.numOfReaders + config.numOfWriters + 1,
99     allhandlers, TRUE, INFINITE);
100
101 //закрываем handle потоков
102 for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
103     CloseHandle(allhandlers[i]);
104
105 //закрываем описатели объектов синхронизации
106 CloseHandle(canReadEvent);
107 CloseHandle(canWriteEvent);
108 CloseHandle(allReadEvent);
109 CloseHandle(changeCountEvent);
110 CloseHandle(exitEvent);
111
112 //закрываем handle общего ресурса
113 UnmapViewOfFile(lpFileMapForReaders);
114 UnmapViewOfFile(lpFileMapForWriters);
115
116 //закрываем объект "отображаемый файл"
117 CloseHandle(hFileMapping);
118
119 // Завершение работы
120 log.loudlog(_T("All is done!"));
121 _getch();
122 return 0;
123 }

```

Листинг 19: Потоки писатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadWriterHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("ThreadsReaderWriter.ThreadWriter"), myid);
11    extern bool isDone;
12    extern struct Configuration config;
13
14    extern HANDLE canReadEvent;
15    extern HANDLE canWriteEvent;
16    extern HANDLE exitEvent;
17
18    extern int countread;
19    extern LPVOID lpFileMapForWriters;
20
21    int msgnum = 0;
22    HANDLE writerhandlers[2];
23    writerhandlers[0] = exitEvent;
24    writerhandlers[1] = canWriteEvent;
25
26    while (isDone != true) {
27        log.quietlog(_T("Waining for multiple objects"));
28        DWORD dwEvent = WaitForMultipleObjects(2, writerhandlers, false,
29            INFINITE);
30        // 2 - следим за 2-я параметрами
31        // writerhandlers - из массива writerhandlers
32        // false - ждём, когда освободится хотя бы один
33        // INFINITE - ждать бесконечно
34        switch (dwEvent) {
35            case WAIT_OBJECT_0: //сработало событие exit
36                log.quietlog(_T("Get exitEvent"));
37                log.loudlog(_T("Writer %d finishing work"), myid);
38                return 0;
39            case WAIT_OBJECT_0 + 1: // сработало событие на возможность записи
40                log.quietlog(_T("Get canWriteEvent"));
41                //увеличиваем номер сообщения
42                msgnum++;

```

```

43     //число потоков которые должны прочитать сообщение
44     countread = config.numOfReaders;
45     // Запись сообщения
46     swprintf_s((_TCHAR *)lpFileMapForWriters, 1500,
47         _T("writer_id %d, msg with num = %d"), myid, msgnum);
48     log.loudlog(_T("writer put msg: \"%s\""), lpFileMapForWriters);
49     //разрешаем читателям прочитать сообщение и опять ставим событие в зан
        ятое
50     log.quietlog(_T("Set Event canReadEvent"));
51     SetEvent(canReadEvent);
52     break;
53     default:
54         log.loudlog(_T("Error with func WaitForMultipleObjects in writerHandle
        , GLE = %d"), GetLastError());
55         ExitProcess(1000);
56     }
57 }
58 log.loudlog(_T("Writer %d finishing work"), myid);
59 return 0;
60 }

```

Листинг 20: Потоки читатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadReaderHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("Mutex.ThreadReader"), myid);
11    extern bool isDone;
12    extern struct Configuration config;
13
14    extern HANDLE canReadEvent;
15    extern HANDLE canWriteEvent;
16    extern HANDLE allReadEvent;
17    extern HANDLE changeCountEvent;
18    extern HANDLE exitEvent;
19
20    extern int countread;
21    extern int countready;
22    extern LPVOID lpFileMapForReaders;

```

```

23
24 HANDLE readerhandlers[2];
25 readerhandlers[0] = exitEvent;
26 readerhandlers[1] = canReadEvent;
27
28 while (isDone != true) {
29     //ждем, пока все прочитают
30     log.quietlog(_T("Waining for allReadEvent"));
31     WaitForSingleObject(allReadEvent, INFINITE);
32     //узнаем, сколько потоков-читателей прошло данную границу
33     log.quietlog(_T("Waining for changeCountEvent"));
34     WaitForSingleObject(changeCountEvent, INFINITE);
35     countready++;
36     //если все прошли, то "закрываем за собой дверь" и разрешаем писать
37     if (countready == config.numOfReaders) {
38         countready = 0;
39         log.quietlog(_T("Reset Event allReadEvent"));
40         ResetEvent(allReadEvent);
41         log.quietlog(_T("Set Event canWriteEvent"));
42         SetEvent(canWriteEvent);
43     }
44
45     //разрешаем изменять счетчик
46     log.quietlog(_T("Set Event changeCountEvent"));
47     SetEvent(changeCountEvent);
48
49     log.quietlog(_T("Waining for multiple objects"));
50     DWORD dwEvent = WaitForMultipleObjects(2, readerhandlers, false,
51     INFINITE);
52     // 2 - следим за 2-я параметрами
53     // readerhandlers - из массива readerhandlers
54     // false - ждём, когда освободится хотя бы один
55     // INFINITE - ждать бесконечно
56     switch (dwEvent) {
57     case WAIT_OBJECT_0: //сработало событие exit
58         log.quietlog(_T("Get exitEvent"));
59         log.loudlog(_T("Reader %d finishing work"), myid);
60         return 0;
61     case WAIT_OBJECT_0 + 1: // сработало событие на возможность чтения
62         log.quietlog(_T("Get canReadEvent"));
63         //читаем сообщение
64         log.loudlog(_T("Reader %d read msg \"%s\\\""), myid,
65             (_TCHAR *)lpFileMapForReaders);
66
67         //необходимо уменьшить счетчик количества читателей, которые прочитать

```

```

        еще не успели
68     log.quietlog(_T("Waiting for changeCountEvent"));
69     WaitForSingleObject(changeCountEvent, INFINITE);
70     countread--;
71
72     // если мы последние читали, то запрещаем читать и открываем границу
73     if (countread == 0) {
74         log.quietlog(_T("Reset Event canReadEvent"));
75         ResetEvent(canReadEvent);
76         log.quietlog(_T("Set Event allReadEvent"));
77         SetEvent(allReadEvent);
78     }
79
80     //разрешаем изменять счетчик
81     log.quietlog(_T("Set Event changeCountEvent"));
82     SetEvent(changeCountEvent);
83     break;
84 default:
85     log.loudlog(_T("Error with func WaitForMultipleObjects in readerHandle
        , GLE = %d"), GetLastError());
86     ExitProcess(1001);
87 }
88 }
89 log.loudlog(_T("Reader %d finishing work"), myid);
90 return 0;
91 }

```

1.7 Задача читатели-писатели (для потоков разных процессов)

В данной программе главный поток и поток-писатель будут принадлежать одному процессу, потоки-читатели – разным. Главный процесс создает процессы-читатели и 2 потока: писатель и планировщик. Для наглядности каждый процесс-читатель связан со своей консолью.

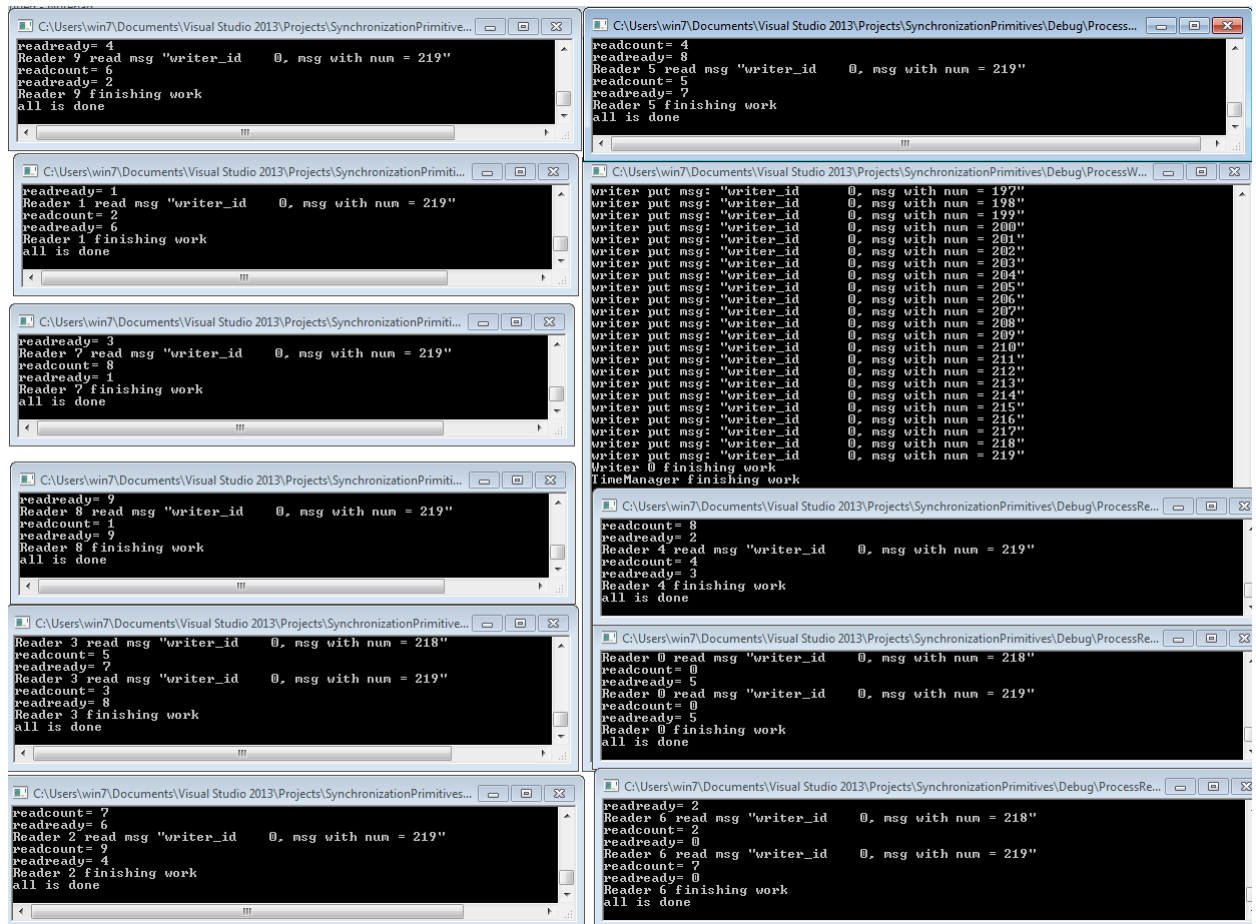


Рис. 7: Решение задачи читатели-писатели для потоков.

Листинг 21: Основной файл

```
1 #include <windows.h>
2 #include <string.h>
3 #include <stdio.h>
4 #include <conio.h>
5 #include <tchar.h>
6
7 #include "thread.h"
8 #include "utils.h"
9 #include "Logger.h"
10
11 //глобальные переменные
```

```

12 struct Configuration config; //конфигурация программы
13 bool isDone = false; //флаг завершения
14 HANDLE *allhandlers; //массив всех создаваемых потоков
15
16 //события для синхронизации:
17 HANDLE canReadEvent; //писатель записал сообщение (ручной сброс);
18 HANDLE canWriteEvent; //все читатели готовы к приему следующего (автосброс);
19 HANDLE allReadEvent; //все читатели прочитали сообщение (ручной сброс);
20 HANDLE changeCountEvent; //разрешение работы со счетчиком (автосброс);
21 HANDLE exitEvent; //завершение программы (ручной сброс);
22
23 //переменные для синхронизации работы потоков:
24 int countread = 0; //число потоков, которое уже прочитали данные
25 //                               (устанавливается писателем и изменяется
26 //                               читателями после прочтения сообщения)
27 int countready = 0; //число потоков, готовых для чтения сообщения
28 //                               (ожидаящих сигнала от писателя)
29
30 //имя разделяемой памяти
31 wchar_t shareFileName[] = L"$MyVerySpecialShareFileName$";
32
33 HANDLE hFileMapping; //объект-отображение файла
34 LPVOID lpFileMapForWriters; // указатели на отображаемую память
35
36 int _tmain(int argc, _TCHAR* argv[]) {
37     Logger log(_T("ProcessWriter"));
38
39     if (argc < 2)
40         // Используем конфигурацию по-умолчанию
41         SetDefaultConfig(&config, &log);
42     else
43         // Загрузка конфига из файла
44         SetConfig(argv[1], &config, &log);
45
46     //создаем необходимые потоки без их запуска
47     //потоки-читатели запускаются сразу (чтобы они успели дойти до функции ожи
48         дания)
49
50     CreateAllThreads(&config, &log);
51
52     //Инициализируем ресурс (share memory): создаем объект "отображаемый файл"
53     // будет использован системный файл подкачки (на диске файл создаваться
54     // не будет), т.к. в качестве дескриптора файла использовано значение
55     // равное 0xFFFFFFFF (его эквивалент - символическая константа
56         INVALID_HANDLE_VALUE)
57
58     if ((hFileMapping = CreateFileMapping(INVALID_HANDLE_VALUE, NULL,

```

```

55 PAGE_READWRITE, 0, 1500, shareFileName)) == NULL) {
56 // INVALID_HANDLE_VALUE - дескриптор открытого файла
57 // (INVALID_HANDLE_VALUE - файл подкачки)
58 // NULL - атрибуты защиты объекта-отображения
59 // PAGE_READWRITE - возможности доступа к представлению файла при
60 // отображении (PAGE_READWRITE - чтение/запись)
61 // 0, 1500 - старшая и младшая части значения максимального
62 // размера объекта отображения файла
63 // shareFileName - имя объекта-отображения.
64 log.loudlog(_T("Impossible to create shareFile, GLE = %d"),
65 GetLastError());
66 ExitProcess(10000);
67 }
68 //отображаем файл на адресное пространство нашего процесса для потока-писа
    теля
69 lpFileMapForWriters = MapViewOfFile(hFileMapping, FILE_MAP_WRITE, 0, 0, 0)
    ;
70 // hFileMapping - дескриптор объекта-отображения файла
71 // FILE_MAP_WRITE - доступа к файлу
72 // 0, 0 - старшая и младшая части смещения начала отображаемого участка в
    файле
73 // (0 - начало отображаемого участка совпадает с началом файла)
74 // 0 - размер отображаемого участка файла в байтах (0 - весь файл)
75
76 //инициализируем 2 переменные в общей памяти (readready и readcount)
77 *((int *)lpFileMapForWriters) = 0;
78 *(((int *)lpFileMapForWriters) + 1) = config.numOfReaders;
79
80 //инициализируем средства синхронизации
81 // (атрибуты защиты, автосброс, начальное состояние, имя):
82 //событие "окончание записи" (можно читать), ручной сброс, изначально заня
    то
83 canReadEvent = CreateEvent(NULL, true, false, L"$$My_canReadEvent$$");
84 //событие - "можно писать", автосброс(разрешаем писать только одному), изна
    чально свободно
85 canWriteEvent = CreateEvent(NULL, false, false, L"$$My_canWriteEvent$$");
86 //событие "все прочитали"
87 allReadEvent = CreateEvent(NULL, true, true, L"$$My_allReadEvent$$");
88 //событие для изменения счетчика (сколько клиентов еще не прочитало сообще
    ние)
89 changeCountEvent = CreateEvent(NULL, false, true, L"
    $$My_changeCountEvent$$");
90 //событие "завершение работы программы", ручной сброс, изначально занято
91 exitEvent = CreateEvent(NULL, true, false, L"$$My_exitEvent$$");
92

```



```

93 //запускаем потоки-писатели и поток-планировщик на исполнение
94 for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
95     ResumeThread(allhandlers[i]);
96
97 //ожидаем завершения всех потоков
98 WaitForMultipleObjects(config.numOfReaders + config.numOfWriters + 1,
99     allhandlers, TRUE, INFINITE);
100
101 //закрываем handle потоков
102 for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
103     CloseHandle(allhandlers[i]);
104
105 //закрываем описатели объектов синхронизации
106 CloseHandle(canReadEvent);
107 CloseHandle(canWriteEvent);
108 CloseHandle(allReadEvent);
109 CloseHandle(changeCountEvent);
110 CloseHandle(exitEvent);
111
112 UnmapViewOfFile(lpFileMapForWriters); //закрываем handle общего ресурса
113 CloseHandle(hFileMapping); //закрываем объект "отображаемый файл"
114
115 log.loudlog(_T("All is done!"));
116 _getch();
117 return 0;
118 }

```

Листинг 22: Потоки писатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadWriterHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("ProcessWriter.ThreadWriter"), myid);
11    extern bool isDone;
12    extern struct Configuration config;
13
14    extern HANDLE canReadEvent;
15    extern HANDLE canWriteEvent;
16    extern HANDLE changeCountEvent;

```

```

17 extern HANDLE exitEvent;
18
19 extern int countread;
20 extern LPVOID lpFileMapForWriters;
21
22 int msgnum = 0;
23 HANDLE writerhandlers[2];
24 writerhandlers[0] = exitEvent;
25 writerhandlers[1] = canWriteEvent;
26
27 while (isDone != true) {
28     log.quietlog(_T("Waiting for multiple objects"));
29     DWORD dwEvent = WaitForMultipleObjects(2, writerhandlers, false,
30     INFINITE);
31     // 2 - следим за 2-я параметрами
32     // writerhandlers - из массива writerhandlers
33     // false - ждём, когда освободится хотя бы один
34     // INFINITE - ждать бесконечно
35     switch (dwEvent) {
36     case WAIT_OBJECT_0: //сработало событие exit
37         log.quietlog(_T("Get exitEvent"));
38         log.loudlog(_T("Writer %d finishing work"), myid);
39         return 0;
40     case WAIT_OBJECT_0 + 1: // сработало событие на возможность записи
41         log.quietlog(_T("Get canWriteEvent"));
42         //увеличиваем номер сообщения
43         msgnum++;
44
45         // Запись сообщения
46         swprintf_s((_TCHAR *)lpFileMapForWriters + sizeof(int) * 2, 1500 -
47             sizeof(int) * 2,
48             _T("Writer_id %d, msg with num = %d"), myid, msgnum);
49         log.loudlog(_T("Writer put msg: \"%s\""), (_TCHAR *)
50             lpFileMapForWriters + sizeof(int) * 2);
51
52         //число потоков которые должны прочитать сообщение
53         log.quietlog(_T("Waiting for changeCountEvent"));
54         WaitForSingleObject(changeCountEvent, INFINITE);
55         *(((int *)lpFileMapForWriters) += config.numOfReaders;
56         *(((int *)lpFileMapForWriters) + 1) += config.numOfReaders;
57         log.quietlog(_T("Set Event changeCountEvent"));
58         SetEvent(changeCountEvent);
59
60         //разрешаем потокам-читателям прочитать сообщение и опять ставим событ
61         ие в состояние "занято"

```

```

59     log.quietlog(_T("Set Event canReadEvent"));
60     SetEvent(canReadEvent);
61
62     break;
63 default:
64     log.loudlog(_T("Error with func WaitForMultipleObjects in writerHandle
        , GLE = %d"), GetLastError());
65     ExitProcess(1000);
66 }
67 }
68 log.loudlog(_T("Writer %d finishing work"), myid);
69 return 0;
70 }

```

Листинг 23: Запуск клиентских процессов

```

1
2 //создание всех потоков
3 void CreateAllThreads(struct Configuration* config, Logger* log) {
4     extern HANDLE *allhandlers;
5
6     int total = config->numOfReaders + config->numOfWriters + 1;
7     log->quietlog(_T("Total num of threads is %d"), total);
8     allhandlers = new HANDLE[total];
9     int count = 0;
10
11     //создаем потоки-читатели
12     log->loudlog(_T("Create readers"));
13
14     STARTUPINFO si;
15     PROCESS_INFORMATION pi;
16
17     ZeroMemory(&si, sizeof(si));
18     si.cb = sizeof(si);
19     ZeroMemory(&pi, sizeof(pi));
20     TCHAR szCommandLine[100];
21
22     for (int i = 0; i != config->numOfReaders; i++, count++) {
23         _stprintf_s(szCommandLine, _T("ProcessReader.exe %d"), i);
24         log->loudlog(_T("Count = %d"), count);
25         if (!CreateProcess(NULL, szCommandLine, NULL, NULL, FALSE,
            CREATE_NEW_CONSOLE |
26             CREATE_SUSPENDED, NULL, NULL, &si, &pi)) {
27             log->loudlog(_T("Impossible to create Process-reader, GLE = %d"),
                GetLastError());

```

```

28     exit(8000);
29 }
30 allhandlers[count] = pi.hThread;
31 }
32
33 //создаем потоки-писатели
34 log->loudlog(_T("Create writers"));
35 for (int i = 0; i != config->numOfWriters; i++, count++) {
36     log->loudlog(_T("count = %d"), count);
37     //создаем потоки-читатели, которые пока не стартуют
38     if ((allhandlers[count] = CreateThread(NULL, 0, ThreadWriterHandler,
39         (LPVOID)i, CREATE_SUSPENDED, NULL)) == NULL) {
40         log->loudlog(_T("Impossible to create thread-writer, GLE = %d"),
41             GetLastError());
42         exit(8001);
43     }
44 }
45
46 //создаем поток TimeManager
47 log->loudlog(_T("Create TimeManager"));
48 log->loudlog(_T("Count = %d"), count);
49 //создаем потоки-читатели, которые пока не стартуют
50 if ((allhandlers[count] = CreateThread(NULL, 0, ThreadTimeManagerHandler,
51     (LPVOID)config->ttml, CREATE_SUSPENDED, NULL)) == NULL) {
52     log->loudlog(_T("impossible to create thread-reader, GLE = %d"),
53         GetLastError());
54     exit(8002);
55 }
56 log->loudlog(_T("Successfully created threads!"));
57 return;
58 }

```

Листинг 24: Потоки читатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4 #include <conio.h>
5
6 #include "Logger.h"
7
8 int _tmain(int argc, _TCHAR* argv[]) {
9     //проверяем число аргументов
10    if (argc != 2) {
11        Logger log(_T("ProcessReader"));

```

```

12     log.loudlog(_T("Error with start reader process. Need 2 arguments."));
13     _getch();
14     ExitProcess(1000);
15 }
16 //получаем из командной строки наш номер
17 int myid = _wtoi(argv[1]);
18
19 Logger log(_T("ProcessReader"), myid);
20 log.loudlog(_T("Reader with id= %d is started"), myid);
21
22 //Инициализируем средства синхронизации
23 // (атрибуты защиты, наследование описателя, имя):
24 //писатель записал сообщение (ручной сброс);
25 HANDLE canReadEvent = OpenEvent(EVENT_ALL_ACCESS, false,
26     L"$$My_canReadEvent$$");
27 //все читатели готовы к приему следующего (автосброс);
28 HANDLE canWriteEvent = OpenEvent(EVENT_ALL_ACCESS, false,
29     L"$$My_canWriteEvent$$");
30 //все читатели прочитали сообщение (ручной сброс);
31 HANDLE allReadEvent = OpenEvent(EVENT_ALL_ACCESS, false,
32     L"$$My_allReadEvent$$");
33 //разрешение работы со счетчиком (автосброс);
34 HANDLE changeCountEvent = OpenEvent(EVENT_ALL_ACCESS, false,
35     L"$$My_changeCountEvent$$");
36 //завершение программы (ручной сброс);
37 HANDLE exitEvent = OpenEvent(EVENT_ALL_ACCESS, false, L"$$My_exitEvent$$")
38     ;
39
40 //Общий ресурс (атрибуты защиты, наследование описателя, имя):
41 HANDLE hFileMapping = OpenFileMapping(FILE_MAP_ALL_ACCESS, false,
42     L"$$MyVerySpecialShareFileName$$");
43
44 //если объекты не созданы, то не сможем работать
45 if (canReadEvent == NULL || canWriteEvent == NULL || allReadEvent == NULL
46     || changeCountEvent == NULL || exitEvent == NULL
47     || hFileMapping == NULL) {
48     log.loudlog(_T("Impossible to open objects, run server first\n
49         getlasterror=%d"),
50         GetLastError());
51     _getch();
52     return 1001;
53 }
54
55 //отображаем файл на адресное пространство нашего процесса для потоков-чит
56     ателей

```

```

54 LPVOID lpFileMapForReaders = MapViewOfFile(hFileMapping,
55     FILE_MAP_ALL_ACCESS, 0, 0, 0);
56 // hFileMapping - дескриптор объекта-отображения файла
57 // FILE_MAP_ALL_ACCESS - доступа к файлу
58 // 0, 0 - старшая и младшая части смещения начала отображаемого участка в
    файле
59 // (0 - начало отображаемого участка совпадает с началом файла)
60 // 0 - размер отображаемого участка файла в байтах (0 - весь файл)
61
62 HANDLE readerhandlers[2];
63 readerhandlers[0] = exitEvent;
64 readerhandlers[1] = canReadEvent;
65
66 while (1) { //основной цикл
67     //ждем, пока все прочитают
68     log.quietlog(_T("Waiting for allReadEvent"));
69     WaitForSingleObject(allReadEvent, INFINITE);
70     //узнаем, сколько потоков-читателей прошло данную границу
71     log.quietlog(_T("Waiting for changeCountEvent"));
72     WaitForSingleObject(changeCountEvent, INFINITE);
73     (*((int *)lpFileMapForReaders) + 1)--;
74     log.loudlog(_T("Readready= %d\n"), (*((int *)lpFileMapForReaders) + 1))
        );
75     //если все прошли, то "закрываем за собой дверь" и разрешаем писать
76     if (*((int *)lpFileMapForReaders) + 1) == 0) {
77         log.quietlog(_T("Reset Event allReadEvent"));
78         ResetEvent(allReadEvent);
79         log.quietlog(_T("Set Event canWriteEvent"));
80         SetEvent(canWriteEvent);
81     }
82
83     //разрешаем изменять счетчик
84     log.quietlog(_T("Set Event changeCountEvent"));
85     SetEvent(changeCountEvent);
86
87     log.quietlog(_T("Waiting for multiple objects"));
88     DWORD dwEvent = WaitForMultipleObjects(2, readerhandlers, false,
89         INFINITE);
90     // 2 - следим за 2-я параметрами
91     // readerhandlers - из массива readerhandlers
92     // false - ждём, когда освободится хотя бы один
93     // INFINITE - ждать бесконечно
94     switch (dwEvent) {
95     case WAIT_OBJECT_0: //сработало событие exit
96         log.quietlog(_T("Get exitEvent"));

```

```

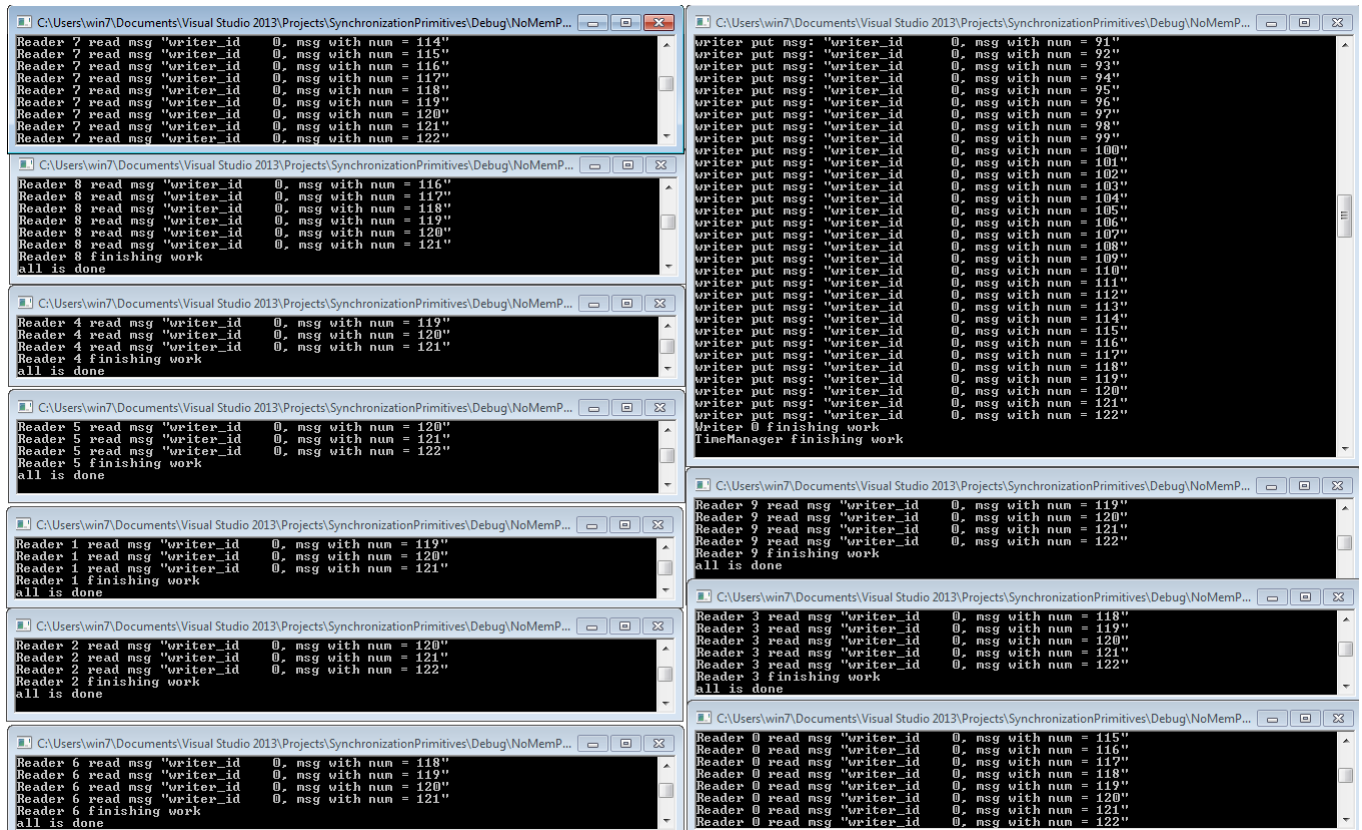
97     log.loudlog(_T("Reader %d finishing work"), myid);
98     goto exit;
99     case WAIT_OBJECT_0 + 1: // сработало событие на возможность чтения
100         log.quietlog(_T("Get canReadEvent"));
101         //читаем сообщение
102         log.loudlog(_T("Reader %d read msg \"%s\""), myid,
103             ((_TCHAR *)lpFileMapForReaders) + sizeof(int) * 2);
104
105         //необходимо уменьшить счетчик количества читателей, которые прочитать
            еще не успели
106         log.quietlog(_T("Waining for changeCountEvent"));
107         WaitForSingleObject(changeCountEvent, INFINITE);
108         (*((int *)lpFileMapForReaders)--);
109         log.loudlog(_T("Readcount= %d"), (*((int *)lpFileMapForReaders)));
110
111         // если мы последние читали, то запрещаем читать и открываем границу
112         if ((*((int *)lpFileMapForReaders)) == 0) {
113             log.quietlog(_T("Reset Event canReadEvent"));
114             ResetEvent(canReadEvent);
115             log.quietlog(_T("Set Event allReadEvent"));
116             SetEvent(allReadEvent);
117         }
118
119         //разрешаем изменять счетчик
120         log.quietlog(_T("Set Event changeCountEvent"));
121         SetEvent(changeCountEvent);
122         break;
123     default:
124         log.loudlog(_T("Error with func WaitForMultipleObjects in readerHandle
            , GLE = %d"), GetLastError());
125         getchar();
126         ExitProcess(1001);
127         break;
128     }
129 }
130 exit:
131     //закрываем HANDLE объектов синхронизации
132     CloseHandle(canReadEvent);
133     CloseHandle(canWriteEvent);
134     CloseHandle(allReadEvent);
135     CloseHandle(changeCountEvent);
136     CloseHandle(exitEvent);
137
138     UnmapViewOfFile(lpFileMapForReaders); //закрываем общий ресурс
139     CloseHandle(hFileMapping); //закрываем объект "отображаемый файл"

```

```
140  
141     log.loudlog(_T("All is done"));  
142     _getch();  
143     return 0;  
144 }
```


2 Модификация задачи читатели-писатели

Читатели не имеют доступа к памяти по записи.



The image displays 12 separate console windows, each representing the output of a different thread in a reader-writer program. The windows are arranged in two columns of six. Each window has a title bar indicating the file path: 'C:\Users\win7\Documents\Visual Studio 2013\Projects\SynchronizationPrimitives\Debug\NoMemP...'. The logs show the following patterns:

- Readers (7-12):** Each reader thread (7-12) logs multiple 'read msg' events with increasing message numbers (e.g., 114-122 for reader 7, 116-121 for reader 8, etc.). Each thread concludes with 'Reader X finishing work' and 'all is done'.
- Writers (0-6):** Each writer thread (0-6) logs multiple 'put msg' events with increasing message numbers (e.g., 91-122 for writer 0, 101-122 for writer 1, etc.). Each thread concludes with 'Writer X finishing work'.
- TimeManager:** One window shows 'TimeManager finishing work'.

Рис. 8: Модификация задачи читатели-писатели.

Листинг 25: Основной файл

```
1 #include <windows.h>
2 #include <string.h>
3 #include <stdio.h>
4 #include <conio.h>
5 #include <tchar.h>
6
7 #include "thread.h"
8 #include "utils.h"
9 #include "Logger.h"
10
11 //глобальные переменные
12 struct Configuration config; //конфигурация программы
13 bool isDone = false; //флаг завершения
14 HANDLE *allhandlers; //массив всех создаваемых потоков
15
16 //события для синхронизации:
17 // писатель записал сообщение, читатель может его прочитать
```

```

18 HANDLE readerCanReadEvent;
19 // все читатели должны перейти в режим готовности
20 HANDLE readerGetReadyEvent;
21 // отчёт может быть отправлен
22 HANDLE canChangeCountEvent;
23 // отчёт
24 HANDLE changeCountEvent;
25 //завершение программы (ручной сброс);
26 HANDLE exitEvent;
27
28 //переменные для синхронизации работы потоков:
29 int reportCounter = 0; // Счётчиков отчётов
30
31 //имя разделяемой памяти
32 wchar_t shareFileName[] = L"$$MyVerySpecialShareFileName$$";
33
34 HANDLE hFileMapping; //объект-отображение файла
35 LPVOID lpFileMapForWriters; // указатели на отображаемую память
36
37 int _tmain(int argc, _TCHAR* argv[]) {
38     Logger log(_T("NoMemProcessWriter"));
39
40     if (argc < 2)
41         // Используем конфигурацию по-умолчанию
42         SetDefaultConfig(&config, &log);
43     else
44         // Загрузка конфига из файла
45         SetConfig(argv[1], &config, &log);
46
47     //создаем необходимые потоки без их запуска
48     //потоки-читатели запускаются сразу (чтобы они успели дойти до функции ожи
49     //дания)
50
51     CreateAllThreads(&config, &log);
52
53     //Инициализируем ресурс (share memory): создаем объект "отображаемый файл"
54     // будет использован системный файл подкачки (на диске файл создаваться
55     // не будет), т.к. в качестве дескриптора файла использовано значение
56     // равное 0xFFFFFFFF (его эквивалент - символическая константа
57     // INVALID_HANDLE_VALUE)
58
59     if ((hFileMapping = CreateFileMapping(INVALID_HANDLE_VALUE, NULL,
60     PAGE_READWRITE, 0, 1500, shareFileName)) == NULL) {
61         // INVALID_HANDLE_VALUE - дескриптор открытого файла
62         // (INVALID_HANDLE_VALUE - файл подкачки)
63         // NULL - атрибуты защиты объекта-отображения
64         // PAGE_READWRITE - возможности доступа к представлению файла при

```

```

61 //          отображению (PAGE_READWRITE - чтение/запись)
62 // 0, 1500 - старшая и младшая части значения максимального
63 //          размера объекта отображения файла
64 // shareFileName - имя объекта-отображения.
65 log.loudlog(_T("Impossible to create shareFile, GLE = %d"),
66             GetLastError());
67 ExitProcess(10000);
68 }
69 //отображаем файл на адресное пространство нашего процесса для потока-писа
    теля
70 lpFileMapForWriters = MapViewOfFile(hFileMapping, FILE_MAP_WRITE, 0, 0, 0)
    ;
71 // hFileMapping - дескриптор объекта-отображения файла
72 // FILE_MAP_WRITE - доступа к файлу
73 // 0, 0 - старшая и младшая части смещения начала отображаемого участка в
    файле
74 //          (0 - начало отображаемого участка совпадает с началом файла)
75 // 0 - размер отображаемого участка файла в байтах (0 - весь файл)
76
77 //инициализируем средства синхронизации
78 // (атрибуты защиты, ручной сброс, начальное состояние, имя):
79 //событие "окончание записи" (можно читать), ручной сброс, изначально заня
    то
80 readerCanReadEvent = CreateEvent(NULL, true, false, L"
    $$My_readerCanReadEvent$$");
81 //событие - "можно писать", автосброс(разрешаем писать только одному), изна
    чально свободно
82 readerGetReadyEvent = CreateEvent(NULL, true, true, L"
    $$My_readerGetReadyEvent$$");
83 //событие для изменения счетчика (сколько клиентов еще не прочитало сообще
    ние)
84 canChangeCountEvent = CreateEvent(NULL, false, true, L"
    $$My_canChangeCountEvent$$");
85 changeCountEvent = CreateEvent(NULL, false, false, L"
    $$My_changeCountEvent$$");
86 //событие "завершение работы программы", ручной сброс, изначально занято
87 exitEvent = CreateEvent(NULL, true, false, L"$$My_exitEvent$$");
88
89 //запускаем потоки-писатели и поток-планировщик на исполнение
90 for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
91     ResumeThread(allhandlers[i]);
92
93 //ожидаем завершения всех потоков
94 WaitForMultipleObjects(config.numOfReaders + config.numOfWriters + 1,
95     allhandlers, TRUE, INFINITE);

```

```

96
97 //закрываем handle потоков
98 for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
99     CloseHandle(allhandlers[i]);
100
101 //закрываем описатели объектов синхронизации
102 CloseHandle(readerCanReadEvent);
103 CloseHandle(readerGetReadyEvent);
104 CloseHandle(canChangeCountEvent);
105 CloseHandle(changeCountEvent);
106 CloseHandle(exitEvent);
107
108 UnmapViewOfFile(lpFileMapForWriters); //закрываем handle общего ресурса
109 CloseHandle(hFileMapping); //закрываем объект "отображаемый файл"
110
111 log.loudlog(_T("All is done!"));
112 _getch();
113 return 0;
114 }

```

Листинг 26: Потоки писатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadWriterHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("NoMemProcessWriter.ThreadWriter"), myid);
11    extern bool isDone;
12    extern struct Configuration config;
13
14    extern HANDLE readerCanReadEvent;
15    extern HANDLE readerGetReadyEvent;
16    extern HANDLE canChangeCountEvent;
17    extern HANDLE changeCountEvent;
18    extern HANDLE exitEvent;
19
20    extern int reportCounter; // Счётчиков отчётов
21    extern LPVOID lpFileMapForWriters;
22
23    int msgnum = 0;

```

```

24 HANDLE writerHandlers[2];
25 writerHandlers[0] = exitEvent;
26 writerHandlers[1] = changeCountEvent;
27
28 // Состояние готовности:
29 // true - сообщение записано, ждём отчётов о прочтении
30 // false - переводим всех читателей в состояние готовности
31 bool readyState = false;
32
33 while (isDone != true) {
34     log.quietlog(_T("Waiting for multiple objects"));
35     DWORD dwEvent = WaitForMultipleObjects(2, writerHandlers, false,
36     INFINITE);
37     // 2 - следим за 2-я параметрами
38     // writerHandlers - из массива writerHandlers
39     // false - ждём, когда освободится хотя бы один
40     // INFINITE - ждать бесконечно
41     switch (dwEvent) {
42     case WAIT_OBJECT_0: //сработало событие exit
43         log.quietlog(_T("Get exitEvent"));
44         log.loudlog(_T("Writer %d finishing work"), myid);
45         return 0;
46     case WAIT_OBJECT_0 + 1: // Пришёл отчёт о выполнении
47         log.quietlog(_T("Get changeCountEvent"));
48         // Если отчитались все читатели
49         if (++reportCounter == config.numOfReaders) {
50             // Обнуление счётчика
51             reportCounter = 0;
52             if (readyState) { // все всё прочитали
53                 // Теперь ожидаем отчётов о готовности
54                 readyState = false;
55                 // Больше ни кто не читает
56                 log.quietlog(_T("Reset Event readerCanReadEvent"));
57                 ResetEvent(readerCanReadEvent);
58                 // Можно готовиться
59                 log.quietlog(_T("Set Event readerGetReadyEvent"));
60                 SetEvent(readerGetReadyEvent);
61             }
62             else { // все готовы читать
63                 // Запись сообщения
64                 swprintf_s((_TCHAR *)lpFileMapForWriters, 1500,
65                 _T("Writer_id %d, msg with num = %d"), myid, ++msgnum);
66                 log.loudlog(_T("Writer put msg: \"%s\""), (_TCHAR *)
67                     lpFileMapForWriters);

```

```

68         // Теперь ожидаем отчётов о прочтении
69         readyState = true;
70         // Больше ни кто не готовится
71         log.quietlog(_T("Reset Event readerGetReadyEvent"));
72         ResetEvent(readerGetReadyEvent);
73         // Можно читать
74         log.quietlog(_T("Set Event readerCanReadEvent"));
75         SetEvent(readerCanReadEvent);
76     }
77 }
78 // Ждём следующего отчёта
79 log.quietlog(_T("Set Event canChangeCountEvent"));
80 SetEvent(canChangeCountEvent);
81
82     break;
83 default:
84     log.loudlog(_T("Error with func WaitForMultipleObjects in writerHandle
85         , GLE = %d"), GetLastError());
86     ExitProcess(1000);
87 }
88 log.loudlog(_T("Writer %d finishing work"), myid);
89 return 0;
90 }

```

Листинг 27: Запуск клиентских процессов

```

1
2 //создание всех потоков
3 void CreateAllThreads(struct Configuration* config, Logger* log) {
4     extern HANDLE *allhandlers;
5
6     int total = config->numOfReaders + config->numOfWriters + 1;
7     log->quietlog(_T("Total num of threads is %d"), total);
8     allhandlers = new HANDLE[total];
9     int count = 0;
10
11     //создаем потоки-читатели
12     log->loudlog(_T("Create readers"));
13
14     STARTUPINFO si;
15     PROCESS_INFORMATION pi;
16
17     ZeroMemory(&si, sizeof(si));
18     si.cb = sizeof(si);

```

```

19 ZeroMemory(&pi, sizeof(pi));
20 TCHAR szCommandLine[100];
21
22 for (int i = 0; i != config->numOfReaders; i++, count++) {
23     _stprintf_s(szCommandLine, _T("NoMemProcessReader.exe %d %d"), i, config
        ->readersDelay);
24     log->loudlog(_T("Count = %d"), count);
25     if (!CreateProcess(NULL, szCommandLine, NULL, NULL, FALSE,
        CREATE_NEW_CONSOLE |
26         CREATE_SUSPENDED, NULL, NULL, &si, &pi)) {
27         log->loudlog(_T("Impossible to create Process-reader, GLE = %d"),
            GetLastError());
28         exit(8000);
29     }
30     allhandlers[count] = pi.hThread;
31 }
32
33 //создаем потоки-писатели
34 log->loudlog(_T("Create writers"));
35 for (int i = 0; i != config->numOfWriters; i++, count++) {
36     log->loudlog(_T("count = %d"), count);
37     //создаем потоки-читатели, которые пока не стартуют
38     if ((allhandlers[count] = CreateThread(NULL, 0, ThreadWriterHandler,
39         (LPVOID)i, CREATE_SUSPENDED, NULL)) == NULL) {
40         log->loudlog(_T("Impossible to create thread-writer, GLE = %d"),
            GetLastError());
41         exit(8001);
42     }
43 }
44
45 //создаем поток TimeManager
46 log->loudlog(_T("Create TimeManager"));
47 log->loudlog(_T("Count = %d"), count);
48 //создаем потоки-читатели, которые пока не стартуют
49 if ((allhandlers[count] = CreateThread(NULL, 0, ThreadTimeManagerHandler,
50     (LPVOID)config->ttml, CREATE_SUSPENDED, NULL)) == NULL) {
51     log->loudlog(_T("impossible to create thread-reader, GLE = %d"),
        GetLastError());
52     exit(8002);
53 }
54 log->loudlog(_T("Successfully created threads!"));
55 return;
56 }

```

Листинг 28: Потоки читатели

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>
4 #include <conio.h>
5
6 #include "Logger.h"
7
8 int _tmain(int argc, _TCHAR* argv[]) {
9     //проверяем число аргументов
10    if (argc != 3) {
11        Logger log(_T("ProcessReader"));
12        log.loudlog(_T("Error with start reader process. Need 2 arguments, but %
            d presented."), argc);
13        _getch();
14        ExitProcess(1000);
15    }
16    //получаем из командной строки наш номер
17    int myid = _wtoi(argv[1]);
18    int pause = _wtoi(argv[2]);
19
20    Logger log(_T("ProcessReader"), myid);
21    log.loudlog(_T("Reader with id= %d is started"), myid);
22
23    // Состояние готовности:
24    // true - ждём сообщение для чтения
25    // false - текущее сообщение уже прочитано,
26    // ждём сигнала перехода в режим готовности
27    bool readyState = false;
28
29    //Инициализируем средства синхронизации
30    // (атрибуты защиты, наследование описателя, имя):
31    //писатель записал сообщение (ручной сброс);
32    HANDLE readerCanReadEvent = OpenEvent(EVENT_ALL_ACCESS, false,
33        L"$$$My_readerCanReadEvent$$");
34    //все читатели готовы к приему следующего (автосброс);
35    HANDLE readerGetReadyEvent = OpenEvent(EVENT_ALL_ACCESS, false,
36        L"$$$My_readerGetReadyEvent$$");
37    //разрешение работы со счетчиком (автосброс);
38    HANDLE canChangeCountEvent = OpenEvent(EVENT_ALL_ACCESS, false,
39        L"$$$My_canChangeCountEvent$$");
40    //
41    HANDLE changeCountEvent = OpenEvent(EVENT_ALL_ACCESS, false,
42        L"$$$My_changeCountEvent$$");
43    //завершение программы (ручной сброс);

```



```

44 HANDLE exitEvent = OpenEvent(EVENT_ALL_ACCESS, false, L"$$My_exitEvent$$")
    ;
45
46 //Общий ресурс (атрибуты защиты, наследование описателя, имя):
47 HANDLE hFileMapping = OpenFileMapping(FILE_MAP_READ, false,
48     L"$$MyVerySpecialShareFileName$$");
49
50 //если объекты не созданы, то не сможем работать
51 if (readerCanReadEvent == NULL || readerGetReadyEvent == NULL ||
    canChangeCountEvent == NULL
52     || changeCountEvent == NULL || exitEvent == NULL
53     || hFileMapping == NULL) {
54     log.loudlog(_T("Impossible to open objects, run server first\n
        getlasterror=%d"),
55         GetLastError());
56     _getch();
57     return 1001;
58 }
59
60 //отображаем файл на адресное пространство нашего процесса для потоков-чит
    ателей
61 LPVOID lpFileMapForReaders = MapViewOfFile(hFileMapping,
62     FILE_MAP_READ, 0, 0, 0);
63 // hFileMapping - дескриптор объекта-отображения файла
64 // FILE_MAP_ALL_ACCESS - доступа к файлу
65 // 0, 0 - старшая и младшая части смещения начала отображаемого участка в
    файле
66 // (0 - начало отображаемого участка совпадает с началом файла)
67 // 0 - размер отображаемого участка файла в байтах (0 - весь файл)
68
69 // События чтения
70 HANDLE readerHandlers[2];
71 readerHandlers[0] = exitEvent;
72 readerHandlers[1] = readerCanReadEvent;
73
74 // События готовности
75 HANDLE readyHandlers[2];
76 readyHandlers[0] = exitEvent;
77 readyHandlers[1] = readerGetReadyEvent;
78
79 while (1) { //основной цикл
80     // Ожидаем набор событий в зависимости от состояния
81     if (readyState) {
82         log.quietlog(_T("Waining for multiple objects"));
83         DWORD dwEvent = WaitForMultipleObjects(2, readerHandlers, false,

```

```

84     INFINITE);
85     // 2 - следим за 2-я параметрами
86     // readerHandlers - из массива readerHandlers
87     // false - ждём, когда освободится хотя бы один
88     // INFINITE - ждать бесконечно
89     switch (dwEvent) {
90     case WAIT_OBJECT_0: //сработало событие exit
91         log.quietlog(_T("Get exitEvent"));
92         log.loudlog(_T("Reader %d finishing work"), myid);
93         goto exit;
94
95     case WAIT_OBJECT_0 + 1: // сработало событие на возможность чтения
96         log.quietlog(_T("Get readerCanReadEvent"));
97         //читаем сообщение
98         log.loudlog(_T("Reader %d read msg \"%s\""), myid, (_TCHAR *)
99             lpFileMapForReaders);
100
101         // Отправляем отчёт
102         log.quietlog(_T("Waiting for canChangeCountEvent"));
103         WaitForSingleObject(canChangeCountEvent, INFINITE);
104         log.quietlog(_T("Set Event changeCountEvent"));
105         SetEvent(changeCountEvent);
106
107         // Завершаем работу
108         readyState = false;
109         break;
110     default:
111         log.loudlog(_T("Error with func WaitForMultipleObjects in
112             readerHandle, GLE = %d"), GetLastError());
113         getchar();
114         ExitProcess(1001);
115         break;
116     }
117 }
118 else {
119     log.quietlog(_T("Waiting for multiple objects"));
120     DWORD dwEvent = WaitForMultipleObjects(2, readyHandlers, false,
121         INFINITE);
122     // 2 - следим за 2-я параметрами
123     // readyHandlers - из массива readyHandlers
124     // false - ждём, когда освободится хотя бы один
125     // INFINITE - ждать бесконечно
126     switch (dwEvent) {
127     case WAIT_OBJECT_0: //сработало событие exit
128         log.quietlog(_T("Get exitEvent"));

```

```

127     log.loudlog(_T("Reader %d finishing work"), myid);
128     goto exit;
129
130     case WAIT_OBJECT_0 + 1: // сработало событие перехода в режим готовнос
        ти
131         log.quietlog(_T("Get readerGetReadyEvent"));
132         // Отправляем отчёт
133         log.quietlog(_T("Waiting for canChangeCountEvent"));
134         WaitForSingleObject(canChangeCountEvent, INFINITE);
135         log.quietlog(_T("Set Event changeCountEvent"));
136         SetEvent(changeCountEvent);
137
138         // Завершаем работу
139         readyState = true;
140         break;
141     default:
142         log.loudlog(_T("Error with func WaitForMultipleObjects in
            readerHandle, GLE = %d"), GetLastError());
143         getchar();
144         ExitProcess(1001);
145         break;
146     }
147 }
148 Sleep(pause);
149 }
150 exit:
151     //закрываем HANDLE объектов синхронизации
152     CloseHandle(readerCanReadEvent);
153     CloseHandle(readerGetReadyEvent);
154     CloseHandle(canChangeCountEvent);
155     CloseHandle(changeCountEvent);
156     CloseHandle(exitEvent);
157
158     UnmapViewOfFile(lpFileMapForReaders); //закрываем общий ресурс
159     CloseHandle(hFileMapping); //закрываем объект "отображаемый файл"
160
161     log.loudlog(_T("All is done"));
162     _getch();
163     return 0;
164 }

```

Заключение