

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	13
1.2 Использование семафоров	19
1.3 Критические секции	30
1.4 Объекты-события в качестве средства синхронизации	35
1.5 Условные переменные	40
1.6 Задача читателя-писателя (для потоков одного процесса)	47
1.7 Задача читателя-писателя (для потоков разных процессов)	55
2 Модификация задачи читателя-писателя без доступа к памяти	70
3 Рациональное решение задачи читателя-писателя	82
4 Клиент-серверное приложение для полной задачи читателя-писателя	91
5 Сетевая версия задачи читателя-писателя	108
6 Задача производителя-потребителя	116
7 Задача обедающие философы	128
Заключение	136
Список литературы	138

Постановка задачи

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

Потоки разделяют целочисленный массив, в который заносятся производимые и извлекаются потребляемые данные. Для наглядности и контроля за происходящим в буфер помещается наращиваемое значение, однозначно идентифицирующее производителя и номер его очередной посылки[1].

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.

Полные исходные коды сделать доступными по адресу https://github.com/SemenMartynov/SPbPU_SystemProgramming.

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.
4. Клиент-серверное приложение для полной задачи читатели-писатели.
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

В Windows реализована вытесняющая многозадачность - это значит, что в любой момент система может прервать выполнение одной нити и передать управление другой. Все нити, принадлежащие одному процессу, разделяют некоторые общие ресурсы - такие, как адресное пространство оперативной памяти или открытые файлы. Эти ресурсы принадлежат всему процессу, а значит, и каждой его нити. Следовательно, каждая нить может работать с этими ресурсами без каких-либо ограничений. Отсутствие ограничений приводит к известным проблемам, таким как гонка, тупик или голодание.

Именно поэтому необходим механизм, позволяющий потокам согласовывать свою работу с общими ресурсами. Этот механизм получил название механизма синхронизации нитей (thread synchronization).

Этот механизм представляет собой набор объектов операционной системы, которые создаются и управляются программно, являются общими для всех нитей в системе (некоторые - для нитей, принадлежащих одному процессу) и используются для координирования доступа к ресурсам. В качестве ресурсов может выступать все, что может быть общим для двух и более нитей - начиная с совместно используемого байта в оперативной памяти, и заканчивая чем-то совсем высокоуровневым, вроде записи в базе данных.

Объектов синхронизации существует несколько, основные это взаимное исключение (mutex), критическая секция (critical section), событие (event) и семафор (semaphore). Каждый из этих объектов реализует свой способ синхронизации. Любой объект синхронизации может находиться в так называемом сигнальном состоянии. Для каждого типа объектов это состояние имеет различный смысл. Нити могут проверять текущее состояние объекта и/или ждать изменения этого состояния и таким образом согласовывать свои действия. При этом гарантируется, что когда нить работает с объектами синхронизации (создаёт их, изменяет состояние) система не прервёт ее выполнения, пока она не завершит это действие.

В данной работе рассмотрены основные механизмы на примере конкретных популярных задач. Часть кода и лог-файлов приведена в листингах по ходу работы, однако более полная версия находится в папках `src/SynchronizationPrimitives` и `logs/SynchronizationPrimitives`.

1 Прimitives синхронизации

Код задач в данном разделе разбит на файлы. Некоторые файлы (такие как система логирования) в разных проектах содержат одинаковый код.

В листинге 1 представлена реализация класса, который занимается логированием событий программы. Логирование может быть тихим (когда информация попадает в файл) или громким (когда информация записывается в файл и выводится на экран). При создании класса в качестве параметра передаётся номер процесса, это позволяет отличать логи разных потоков. Каждое событие сопровождается меткой времени.

Листинг 1: Реализация класса логера (src/SynchronizationPrimitives/Mutex/Logger.cpp)

```
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 |     fprintf(logfile, _T("\n"));
79 |     _tprintf(_T("\n"));
80 | }

```

Использование класса позволяет не беспокоиться о корректном захвате ресурса (дескрипторе файла) и его освобождении. Класс создаётся на стеке, следовательно его деструктор (и освобождение дескриптора файла) будет вызван при раскрутке стека. Класс содержит следующие методы:

- `Logger` – конструктор; в качестве параметров принимает имя исполняемого файла, для построения имени лог-файла на его основе, однако в отличие от предыдущих программ у нас может быть несколько процессов/потоков с одинаковым именем. Для решения этой проблемы используется второй параметр, содержащий `ip` потока. Это позволяет использовать разные лог-файлы разными потоками (в противном случае пришлось бы вводить дополнительный примитив синхронизации для доступа к логу).
- `~Logger` – деструктор; освобождает захваченные ресурсы.
- `quietlog` – тихий лог; информация заносится только в лог-файл.
- `loudlog` – громкий лог; информация заносится в лог-файл и выводится на экран.

Записи, попадающие в лог-файл сопровождаются меткой времени для упрощения анализа параллельных потоков/процессов.

Сервисные функции представлены в листинге 2. Они используются для создания потоков и чтения конфигурационного файла. Если конфигурационный файл прочитать не удалось, используются значения по умолчанию. Сам конфигурационный файл может выглядеть следующим образом:

```

NumOfReaders= 10
ReadersDelay= 100
NumOfWriters= 10
WritersDelay= 200
SizeOfQueue= 10
ttl= 3

```

Листинг 2: Сервисные функции (`src/SynchronizationPrimitives/Mutex/utls.cpp`)

```

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"));
45     for (int i = 0; i != config->numOfWriters; ++i, ++count) {
46         log->loudlog(_T("count = %d"), count);
47         //создаем потоки-писатели, которые пока не стартуют
48         if ((allhandlers[count] = CreateThread(NULL, 0, ThreadWriterHandler, (
49             LPVOID)i, CREATE_SUSPENDED, NULL)) == NULL) {
50             log->loudlog(_T("Impossible to create thread-writer, GLE = %d"),
51                 GetLastError());
52             exit(8000);
53         }
54     }
55 }

```

```

        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 поток TimeManager, который 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 ttl;
76     _TCHAR trash[30];
77
78     if (_wfopen_s(&confsource, filename, _T("r"))) {
79         _wprintf(_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); //чу
        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); //чу

```

```

88     сло потоков-писателей
fscanf_s(confsource, "%s %d", trash, _countof(trash), &writersDelay); //за
89     держки потоков-писателей
fscanf_s(confsource, "%s %d", trash, _countof(trash), &sizeOfQueue); //раз
90     мер очереди
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\n\t"),
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     //Буд 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\n"),
139 config->numOfReaders, config->readersDelay, config->numOfWriters, config->writersDelay, config->sizeOfQueue, config->ttl);
140 }

```

1.1 Использование мьютексов

Объекты ядра «мьютексы» гарантируют потокам взаимоисключающий доступ к единственному ресурсу. Это отражено в названии этих объектов (mutual exclusion, mutex). Они содержат счётчик числа пользователей, счетчик рекурсии и переменную, в которой запоминается идентификатор потока. Мьютексы ведут себя точно так же, как и критические секции. Однако, если последние являются объектами пользовательского режима, то мьютексы — объектами ядра. Кроме того, единственный объект мьютекса позволяет синхронизировать доступ к ресурсу нескольких потоков из разных процессов; при этом можно задать максимальное время ожидания доступа к ресурсу[2].

Идентификатор потока определяет, какой поток захватил мьютекс, а счетчик рекурсий — количество. У мьютексов много применений, и это наиболее часто используемые объекты ядра. Как правило, с их помощью защищают блок памяти, к которому обращается множество потоков. Если бы потоки одновременно использовали какой-то блок памяти, данные в нем были бы повреждены. Мьютексы гарантируют, что любой поток получает монопольный доступ к блоку памяти, и тем самым обеспечивают целостность данных.

Листинг 3 демонстрирует инициализацию и закрытие мьютекса, который активно используется читателями (листинг 4) и писателями (листинг 5). Важно отметить, что этот мьютекс имеет пустое имя, следовательно не может использоваться сторонним процессом и служить средством IPC[1].

Листинг 3: Основной файл, демонстрирующий работу с мьютексами (src/SynchronizationPrimitives/Mutex/main.cpp)

```
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 tasks are done!"));
64 _getch();
65 return 0;
66 }

```

Листинг 4: Потоки читатели, синхронизация через мьютекс
(src/SynchronizationPrimitives/Mutex/threadReader.cpp)

```

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 }

```

Листинг 5: Потоки писатели, синхронизация через мьютекс
(src/SynchronizationPrimitives/Mutex/threadWriter.cpp)

```

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            //заноcим 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 }

```

На рисунке 1 показана работа потоков с мьютексом.

```

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.

Файл с логом работы программы содержит почти аналогичный результат (различия в метках времени при переходе из состояния ожидания в состояние владения и обратно). Рисунок 2 показывает множество потоков, каждый из которых пишет свой файл с логом.

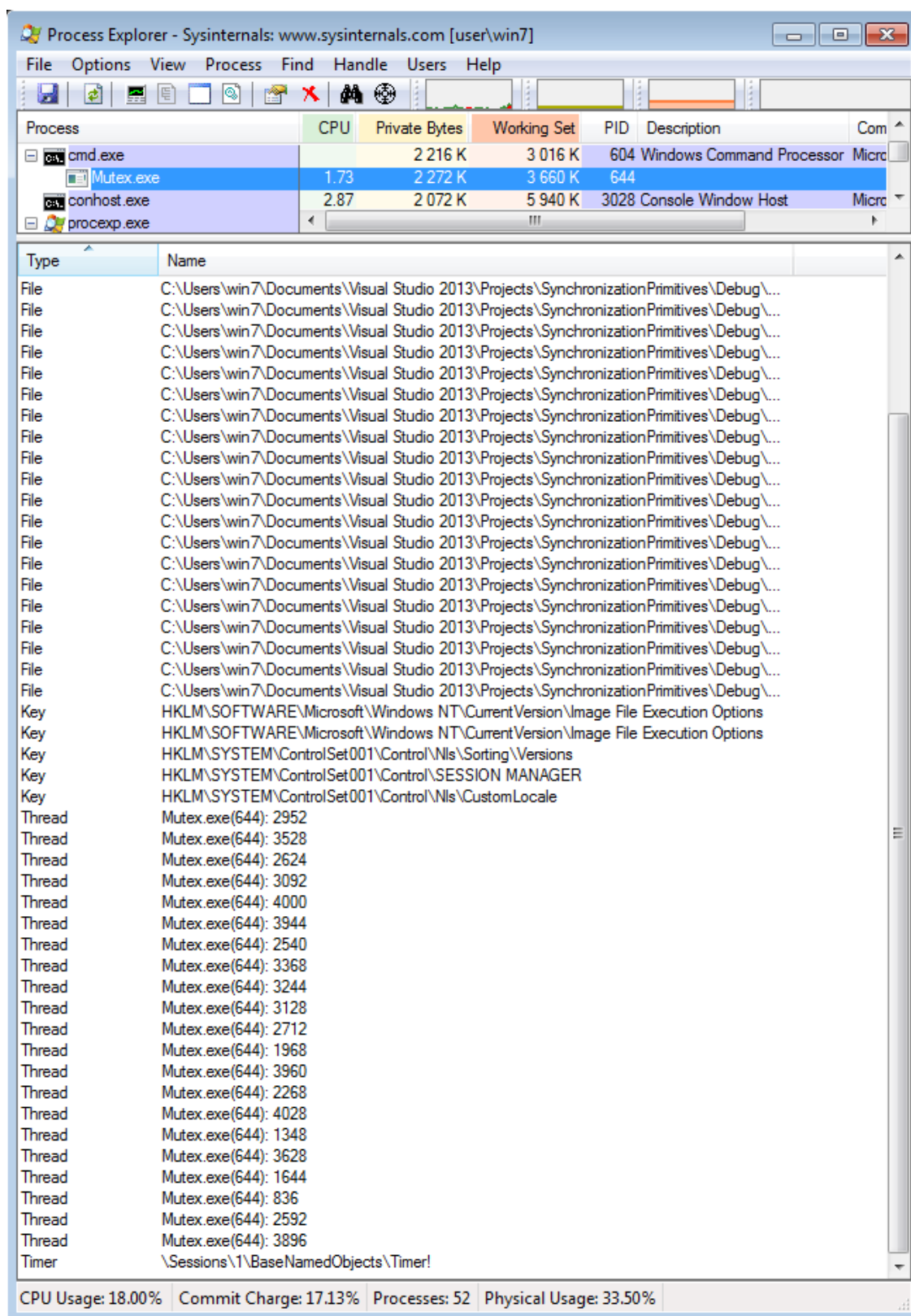


Рис. 2: Потоки и файлы с логами.

1.2 Использование семафоров

Объекты ядра «семафор» используются для учета ресурсов. Он содержит счетчик числа пользователей, но, кроме того, поддерживает два 32 битных значения со знаком: одно определяет максимальное число ресурсов (контролируемое семафором), другое используется как счетчик текущего числа ресурсов. Таким образом, семафор используется для учёта ресурсов и служат для ограничения одновременного доступа к ресурсу нескольких потоков. Используя семафор, можно организовать работу программы таким образом, что к ресурсу одновременно смогут получить доступ несколько потоков, однако количество этих потоков будет ограничено. Создавая семафор, указывается максимальное количество потоков, которые одновременно смогут работать с ресурсом. Каждый раз, когда программа обращается к семафору, значение счётчика ресурсов семафора уменьшается на единицу. Когда значение счётчика ресурсов становится равным нулю, семафор недоступен[2].

Как и в прошлом случае, Листинг 6 содержит основной файл, в котором семафор инициализируется, а используется он писателем и читателем в листингах 7 и 8.

Листинг 6: Основной файл, инициализация семафора со счётчиком 1 (src/SynchronizationPrimitives/Semaphore/main.cpp)

```
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 используем calloc
61     delete[] queue.data;
62
63     // Завершение работы
64     log.loudlog(_T("All tasks are done!"));
65     _getch();
66     return 0;
67 }

```

Листинг 7: Потоки писатели, синхронизация через семафор
(src/SynchronizationPrimitives/Semaphore/threadWriter.cpp)

```
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            //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 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: Потоки читатели, синхронизация через семафор
(src/SynchronizationPrimitives/Semaphore/threadReader.cpp)

```

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 }

```

Результат работы на рисунке 3.

```

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. 3: Использование семафоров.

Результат работы можно изучать и по логам программы. Листинг 9 содержит общий лог работы программы, листинг 10 показывает работу одного из писателей, а листинг 11 - одного из читателей.

Листинг 9: Общий протокол работы программы

```
1 [25/3/2015 18:41:56] Semaphore is starting.
2 [25/3/2015 18:41:56] Using default config
3 [25/3/2015 18:41:56] Config:
4   NumOfReaders = 10
5   ReadersDelay = 100
6   NumOfWriters = 10
7   WritersDelay = 200
8   SizeOfQueue = 10
9   ttl = 3
10 [25/3/2015 18:41:56] Total num of threads is 21
11 [25/3/2015 18:41:56] Create readers
12 [25/3/2015 18:41:56] Count = 0
13 [25/3/2015 18:41:56] Count = 1
14 [25/3/2015 18:41:56] Count = 2
15 [25/3/2015 18:41:56] Count = 3
16 [25/3/2015 18:41:56] Count = 4
17 [25/3/2015 18:41:56] Count = 5
18 [25/3/2015 18:41:56] Count = 6
19 [25/3/2015 18:41:56] Count = 7
20 [25/3/2015 18:41:56] Count = 8
21 [25/3/2015 18:41:56] Count = 9
22 [25/3/2015 18:41:56] Create writers
23 [25/3/2015 18:41:56] count = 10
24 [25/3/2015 18:41:56] count = 11
25 [25/3/2015 18:41:56] count = 12
26 [25/3/2015 18:41:56] count = 13
27 [25/3/2015 18:41:56] count = 14
28 [25/3/2015 18:41:56] count = 15
29 [25/3/2015 18:41:56] count = 16
30 [25/3/2015 18:41:56] count = 17
31 [25/3/2015 18:41:56] count = 18
32 [25/3/2015 18:41:56] count = 19
33 [25/3/2015 18:41:56] Create TimeManager
34 [25/3/2015 18:41:56] Count = 20
35 [25/3/2015 18:41:56] Successfully created threads!
36 [25/3/2015 18:41:59] All tasks are done!
37 [25/3/2015 18:42:27] Shutting down.
```


Листинг 10: Протокол работы писателя

```

1 [25/3/2015 18:41:56] Semaphore.ThreadWriter is starting.
2 [25/3/2015 18:41:56] Waining for Semaphore
3 [25/3/2015 18:41:56] Get Semaphore
4 [25/3/2015 18:41:56] Writer 1 put data: "writer_id = 1 numMsg= 0" in
    position 0
5 [25/3/2015 18:41:56] Release Semaphore
6 [25/3/2015 18:41:56] Waining for Semaphore
7 [25/3/2015 18:41:56] Get Semaphore
8 [25/3/2015 18:41:56] Writer 1 put data: "writer_id = 1 numMsg= 1" in
    position 0
9 [25/3/2015 18:41:56] Release Semaphore
10 [25/3/2015 18:41:56] Waining for Semaphore
11 [25/3/2015 18:41:56] Get Semaphore
12 [25/3/2015 18:41:56] Writer 1 put data: "writer_id = 1 numMsg= 2" in
    position 0
13 [25/3/2015 18:41:56] Release Semaphore
14 [25/3/2015 18:41:57] Waining for Semaphore
15 [25/3/2015 18:41:57] Get Semaphore
16 [25/3/2015 18:41:57] Writer 1 put data: "writer_id = 1 numMsg= 3" in
    position 0
17 [25/3/2015 18:41:57] Release Semaphore
18 [25/3/2015 18:41:57] Waining for Semaphore
19 [25/3/2015 18:41:57] Get Semaphore
20 [25/3/2015 18:41:57] Writer 1 put data: "writer_id = 1 numMsg= 4" in
    position 1
21 [25/3/2015 18:41:57] Release Semaphore
22 [25/3/2015 18:41:57] Waining for Semaphore
23 [25/3/2015 18:41:57] Get Semaphore
24 [25/3/2015 18:41:57] Writer 1 put data: "writer_id = 1 numMsg= 5" in
    position 1
25 [25/3/2015 18:41:57] Release Semaphore
26 [25/3/2015 18:41:57] Waining for Semaphore
27 [25/3/2015 18:41:57] Get Semaphore
28 [25/3/2015 18:41:57] Writer 1 put data: "writer_id = 1 numMsg= 6" in
    position 1
29 [25/3/2015 18:41:57] Release Semaphore
30 [25/3/2015 18:41:57] Waining for Semaphore
31 [25/3/2015 18:41:57] Get Semaphore
32 [25/3/2015 18:41:57] Writer 1 put data: "writer_id = 1 numMsg= 7" in
    position 1
33 [25/3/2015 18:41:57] Release Semaphore
34 [25/3/2015 18:41:58] Waining for Semaphore
35 [25/3/2015 18:41:58] Get Semaphore
36 [25/3/2015 18:41:58] Writer 1 put data: "writer_id = 1 numMsg= 8" in

```

```

    position 1
37 [25/3/2015 18:41:58] Release Semaphore
38 [25/3/2015 18:41:58] Waining for Semaphore
39 [25/3/2015 18:41:58] Get Semaphore
40 [25/3/2015 18:41:58] Writer 1 put data: "writer_id = 1 numMsg= 9" in
    position 0
41 [25/3/2015 18:41:58] Release Semaphore
42 [25/3/2015 18:41:58] Waining for Semaphore
43 [25/3/2015 18:41:58] Get Semaphore
44 [25/3/2015 18:41:58] Writer 1 put data: "writer_id = 1 numMsg= 10" in
    position 1
45 [25/3/2015 18:41:58] Release Semaphore
46 [25/3/2015 18:41:58] Waining for Semaphore
47 [25/3/2015 18:41:58] Get Semaphore
48 [25/3/2015 18:41:58] Writer 1 put data: "writer_id = 1 numMsg= 11" in
    position 0
49 [25/3/2015 18:41:58] Release Semaphore
50 [25/3/2015 18:41:58] Waining for Semaphore
51 [25/3/2015 18:41:58] Get Semaphore
52 [25/3/2015 18:41:58] Writer 1 put data: "writer_id = 1 numMsg= 12" in
    position 0
53 [25/3/2015 18:41:58] Release Semaphore
54 [25/3/2015 18:41:58] Waining for Semaphore
55 [25/3/2015 18:41:58] Get Semaphore
56 [25/3/2015 18:41:58] Writer 1 put data: "writer_id = 1 numMsg= 13" in
    position 1
57 [25/3/2015 18:41:58] Release Semaphore
58 [25/3/2015 18:41:59] Waining for Semaphore
59 [25/3/2015 18:41:59] Get Semaphore
60 [25/3/2015 18:41:59] Writer 1 put data: "writer_id = 1 numMsg= 14" in
    position 1
61 [25/3/2015 18:41:59] Release Semaphore
62 [25/3/2015 18:41:59] Writer 1 finishing work
63 [25/3/2015 18:41:59] Shutting down.

```

Листинг 11: Протокол работы читателя

```

1 [25/3/2015 18:41:56] Semaphore.ThreadReader is starting.
2 [25/3/2015 18:41:56] Waining for Semaphore
3 [25/3/2015 18:41:56] Get Semaphore
4 [25/3/2015 18:41:56] Reader 1 get data: "writer_id = 8 numMsg= 0" from
    position 5
5 [25/3/2015 18:41:56] Release Semaphore
6 [25/3/2015 18:41:56] Waining for Semaphore
7 [25/3/2015 18:41:56] Get Semaphore

```

```

8 [25/3/2015 18:41:56] Release Semaphore
9 [25/3/2015 18:41:56] Waining for Semaphore
10 [25/3/2015 18:41:56] Get Semaphore
11 [25/3/2015 18:41:56] Reader 1 get data: "writer_id = 4 numMsg= 1" from
    position 6
12 [25/3/2015 18:41:56] Release Semaphore
13 [25/3/2015 18:41:56] Waining for Semaphore
14 [25/3/2015 18:41:56] Get Semaphore
15 [25/3/2015 18:41:56] Release Semaphore
16 [25/3/2015 18:41:56] Waining for Semaphore
17 [25/3/2015 18:41:56] Get Semaphore
18 [25/3/2015 18:41:56] Reader 1 get data: "writer_id = 7 numMsg= 2" from
    position 4
19 [25/3/2015 18:41:56] Release Semaphore
20 [25/3/2015 18:41:57] Waining for Semaphore
21 [25/3/2015 18:41:57] Get Semaphore
22 [25/3/2015 18:41:57] Release Semaphore
23 [25/3/2015 18:41:57] Waining for Semaphore
24 [25/3/2015 18:41:57] Get Semaphore
25 [25/3/2015 18:41:57] Reader 1 get data: "writer_id = 8 numMsg= 3" from
    position 5
26 [25/3/2015 18:41:57] Release Semaphore
27 [25/3/2015 18:41:57] Waining for Semaphore
28 [25/3/2015 18:41:57] Get Semaphore
29 [25/3/2015 18:41:57] Release Semaphore
30 [25/3/2015 18:41:57] Waining for Semaphore
31 [25/3/2015 18:41:57] Get Semaphore
32 [25/3/2015 18:41:57] Reader 1 get data: "writer_id = 2 numMsg= 4" from
    position 6
33 [25/3/2015 18:41:57] Release Semaphore
34 [25/3/2015 18:41:57] Waining for Semaphore
35 [25/3/2015 18:41:57] Get Semaphore
36 [25/3/2015 18:41:57] Release Semaphore
37 [25/3/2015 18:41:57] Waining for Semaphore
38 [25/3/2015 18:41:57] Get Semaphore
39 [25/3/2015 18:41:57] Reader 1 get data: "writer_id = 5 numMsg= 5" from
    position 9
40 [25/3/2015 18:41:57] Release Semaphore
41 [25/3/2015 18:41:57] Waining for Semaphore
42 [25/3/2015 18:41:57] Get Semaphore
43 [25/3/2015 18:41:57] Release Semaphore
44 [25/3/2015 18:41:57] Waining for Semaphore
45 [25/3/2015 18:41:57] Get Semaphore
46 [25/3/2015 18:41:57] Release Semaphore
47 [25/3/2015 18:41:57] Waining for Semaphore

```

```

48 [25/3/2015 18:41:57] Get Semaphore
49 [25/3/2015 18:41:57] Reader 1 get data: "writer_id = 3 numMsg= 7" from
    position 3
50 [25/3/2015 18:41:57] Release Semaphore
51 [25/3/2015 18:41:58] Waining for Semaphore
52 [25/3/2015 18:41:58] Get Semaphore
53 [25/3/2015 18:41:58] Release Semaphore
54 [25/3/2015 18:41:58] Waining for Semaphore
55 [25/3/2015 18:41:58] Get Semaphore
56 [25/3/2015 18:41:58] Reader 1 get data: "writer_id = 7 numMsg= 8" from
    position 4
57 [25/3/2015 18:41:58] Release Semaphore
58 [25/3/2015 18:41:58] Waining for Semaphore
59 [25/3/2015 18:41:58] Get Semaphore
60 [25/3/2015 18:41:58] Release Semaphore
61 [25/3/2015 18:41:58] Waining for Semaphore
62 [25/3/2015 18:41:58] Get Semaphore
63 [25/3/2015 18:41:58] Reader 1 get data: "writer_id = 2 numMsg= 9" from
    position 7
64 [25/3/2015 18:41:58] Release Semaphore
65 [25/3/2015 18:41:58] Waining for Semaphore
66 [25/3/2015 18:41:58] Get Semaphore
67 [25/3/2015 18:41:58] Release Semaphore
68 [25/3/2015 18:41:58] Waining for Semaphore
69 [25/3/2015 18:41:58] Get Semaphore
70 [25/3/2015 18:41:58] Reader 1 get data: "writer_id = 9 numMsg= 10" from
    position 6
71 [25/3/2015 18:41:58] Release Semaphore
72 [25/3/2015 18:41:58] Waining for Semaphore
73 [25/3/2015 18:41:58] Get Semaphore
74 [25/3/2015 18:41:58] Release Semaphore
75 [25/3/2015 18:41:58] Waining for Semaphore
76 [25/3/2015 18:41:58] Get Semaphore
77 [25/3/2015 18:41:58] Reader 1 get data: "writer_id = 4 numMsg= 11" from
    position 9
78 [25/3/2015 18:41:58] Release Semaphore
79 [25/3/2015 18:41:58] Waining for Semaphore
80 [25/3/2015 18:41:58] Get Semaphore
81 [25/3/2015 18:41:58] Reader 1 get data: "writer_id = 0 numMsg= 12" from
    position 1
82 [25/3/2015 18:41:58] Release Semaphore
83 [25/3/2015 18:41:58] Waining for Semaphore
84 [25/3/2015 18:41:58] Get Semaphore
85 [25/3/2015 18:41:58] Reader 1 get data: "writer_id = 9 numMsg= 12" from
    position 7

```

```
86 [25/3/2015 18:41:58] Release Semaphore
87 [25/3/2015 18:41:59] Waining for Semaphore
88 [25/3/2015 18:41:59] Get Semaphore
89 [25/3/2015 18:41:59] Reader 1 get data: "writer_id = 6 numMsg= 13" from
    position 3
90 [25/3/2015 18:41:59] Release Semaphore
91 [25/3/2015 18:41:59] Waining for Semaphore
92 [25/3/2015 18:41:59] Get Semaphore
93 [25/3/2015 18:41:59] Release Semaphore
94 [25/3/2015 18:41:59] Waining for Semaphore
95 [25/3/2015 18:41:59] Get Semaphore
96 [25/3/2015 18:41:59] Reader 1 get data: "writer_id = 3 numMsg= 14" from
    position 3
97 [25/3/2015 18:41:59] Release Semaphore
98 [25/3/2015 18:41:59] Waining for Semaphore
99 [25/3/2015 18:41:59] Get Semaphore
100 [25/3/2015 18:41:59] Release Semaphore
101 [25/3/2015 18:41:59] Reader 1 finishing work
102 [25/3/2015 18:41:59] Shutting down.
```

1.3 Критические секции

Критическая секция (critical section) – это небольшой участок кода, который должен использоваться только одним потоком одновременно. Если в одно время несколько потоков попытаются получить доступ к критическому участку, то контроль над ним будет предоставлен только одному из потоков, а все остальные будут переведены в состояние ожидания до тех пор, пока участок не освободится[3].

Критический раздел анализирует значение специальной переменной процесса, которая используется как флаг, предотвращающий исполнение участка кода несколькими потоками одновременно.

Среди синхронизирующих объектов критические разделы наиболее просты. Инициализация критической секции показана в листинге 12 (при этом имя секции ни где не указывается), а захват в листинге 13 и листинге 14 (для захвата нет ожидания события, секция оказывается захваченной когда это возможно).

Листинг 12: Основной файл, инициализирующий критическую секцию (src/SynchronizationPrimitives/CriticalSection/main.cpp)

```
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 tasks are done!"));
61     _getch();
62     return 0;
63 }

```

Листинг 13: Потоки писатели, синхронизация через семафор критическую секцию (src/SynchronizationPrimitives/CriticalSection/threadWriter.cpp)

```

1 #include <windows.h>
2 #include <stdio.h>
3 #include <tchar.h>

```

```

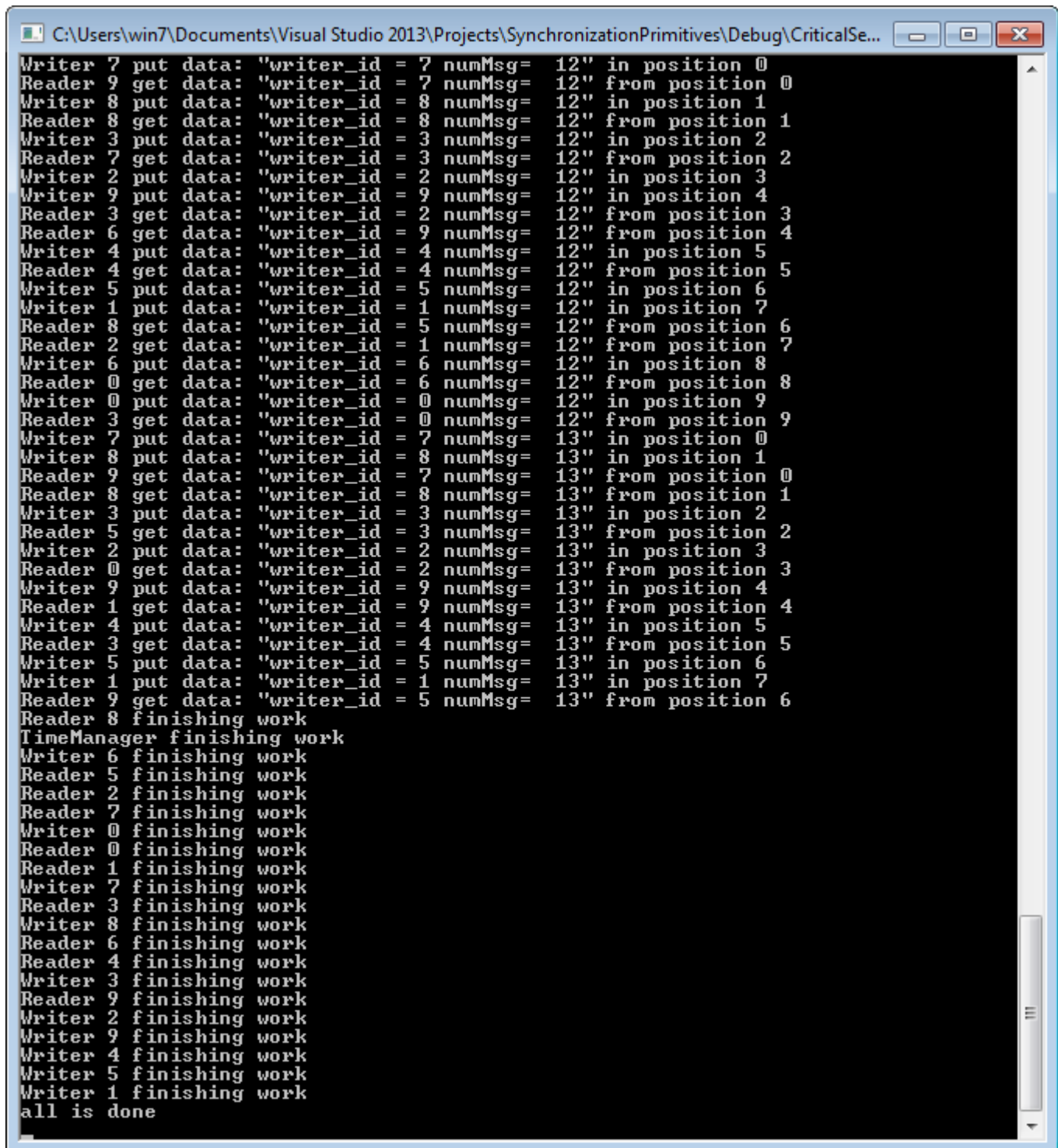
4
5 #include "utils.h"
6
7 DWORD WINAPI ThreadWriterHandler(LPVOID prm) {
8     int myid = (int)prm;
9
10    Logger log(_T("CriticalSection.ThreadWriter"), myid);
11    extern bool isDone;
12    extern struct FIFOQueue queue;
13    extern struct Configuration config;
14    extern CRITICAL_SECTION crs;
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 14: Поток читателя, синхронизация через семафор критическую секцию (src/SynchronizationPrimitives/CriticalSection/threadReader.cpp)

```
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 }
```

Демонстрация работы синхронизации через критическую секцию показана на рисунке 4.



```
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. 4: работа с критической секцией

1.4 Объекты-события в качестве средства синхронизации

События обычно просто оповещают об окончании какой-либо операции, они также являются объектами ядра. Можно не просто явным образом освободить, но так же есть операция установки события. События могут быть ручным (manual) и единичными (single).

Единичное событие (single event) – это общий флаг. Событие находится в сигнальном состоянии, если его установил какой-нибудь поток. Если для работы программы требуется, чтобы в случае возникновения события на него реагировал только один из потоков, в то время как все остальные потоки продолжали ждать, то используют единичное событие[2].

Ручное событие (manual event) — это не просто общий флаг для нескольких потоков. Оно выполняет несколько более сложные функции. Любой поток может установить это событие или сбросить (очистить) его. Если событие установлено, оно останется в этом состоянии сколько угодно долгое время, вне зависимости от того, сколько потоков ожидают установки этого события. Когда все потоки, ожидающие этого события, получают сообщение о том, что событие произошло, оно автоматически сбросится.

В листинге 15 происходит инициализация и закрытие объектов-событий, а в листинге 16 и 17 происходит ожидание и захват.

Листинг 15: Основной файл инициализации объектов-событий (src/SynchronizationPrimitives/Event/main.cpp)

```
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 автосброа, 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 используем calloc
62  delete[] queue.data;
63
64  // Завершение работы
65  log.loudlog(_T("All tasks are done!"));

```

```

66  _getch();
67  return 0;
68 }

```

Листинг 16: Потоки писатели, синхронизация через объекты-события (src/SynchronizationPrimitives/Event/threadWriter.cpp)

```

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             //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("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 17: Потоки читатели, синхронизация через объекты-события (src/SynchronizationPrimitives/Event/threadReader.cpp)

```

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 }
```

Рисунок 5 содержит результат работы, дублирующий лог программы.

```
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. 5: Объекты-события в качестве средства синхронизации.

1.5 Условные переменные

Условные переменные – это объекты, поддерживающие операции ожидания и уведомления. Ожидание возможно только внутри какой-либо критической секции, то есть данная операция связана не только с какой-либо условной переменной, но еще и с соответствующим мьютексом. При этом возможны ситуации, когда, например, разные условные переменные должны быть связаны с одним и тем же мьютексом, в случае, если они относятся к одному и

тому же ресурсу, но означают разные условия. Операция ожидания атомарно освобождает мьютекс и блокирует процесс до момента уведомления о том, что условие выполнено[4].

В листинге 18 происходит инициализация условных переменных, но для организации полноценной работы их не достаточно. Поэтому в качестве вспомогательного объекта синхронизации используется критическая секция.

Листинг 18: Основной файл инициализации примитивов синхронизации (src/SynchronizationPrimitives/ConditionVariable/main.cpp)

```
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 используем calloc
66 delete[] queue.data;
67
68 // Завершение работы
69 log.loudlog(_T("All tasks are done!"));
70 _getch();
71 return 0;
72 }

```

Читатели (листинг 19) и писатели (листинг 20) входят в критическую секцию, но для работы ожидают условную переменную.

Листинг 19: Потоки читатели, синхронизация через условные переменные и критические секции (src/SynchronizationPrimitives/ConditionVariable/threadReader.cpp)

```

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("Waiting for Critical Section"));
20        EnterCriticalSection(&crs);
21        log.quietlog(_T("Get Critical Section"));
22
23        log.quietlog(_T("Waiting for empty space in the queue"));
24        while (!(queue.readindex != queue.writeindex || queue.full == 1))
25            //спим 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 }

```

Листинг 20: Поток писателя, синхронизация через условные переменные и критические секции (src/SynchronizationPrimitives/ConditionVariable/threadWriter.cpp)

```

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("Waiting for Critical Section"));
23         EnterCriticalSection(&crs);
24         log.quietlog(_T("Get Critical Section"));
25
26         log.quietlog(_T("Waiting for empty space in the queue"));
27         while (!(queue.readindex != queue.writeindex || !queue.full == 1))
28             //спим пока в очереди не освободится место
29             SleepConditionVariableCS(&condwrite, &crs, INFINITE);
30         log.quietlog(_T("Get space in the queue"));
31
32         //записываем 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);

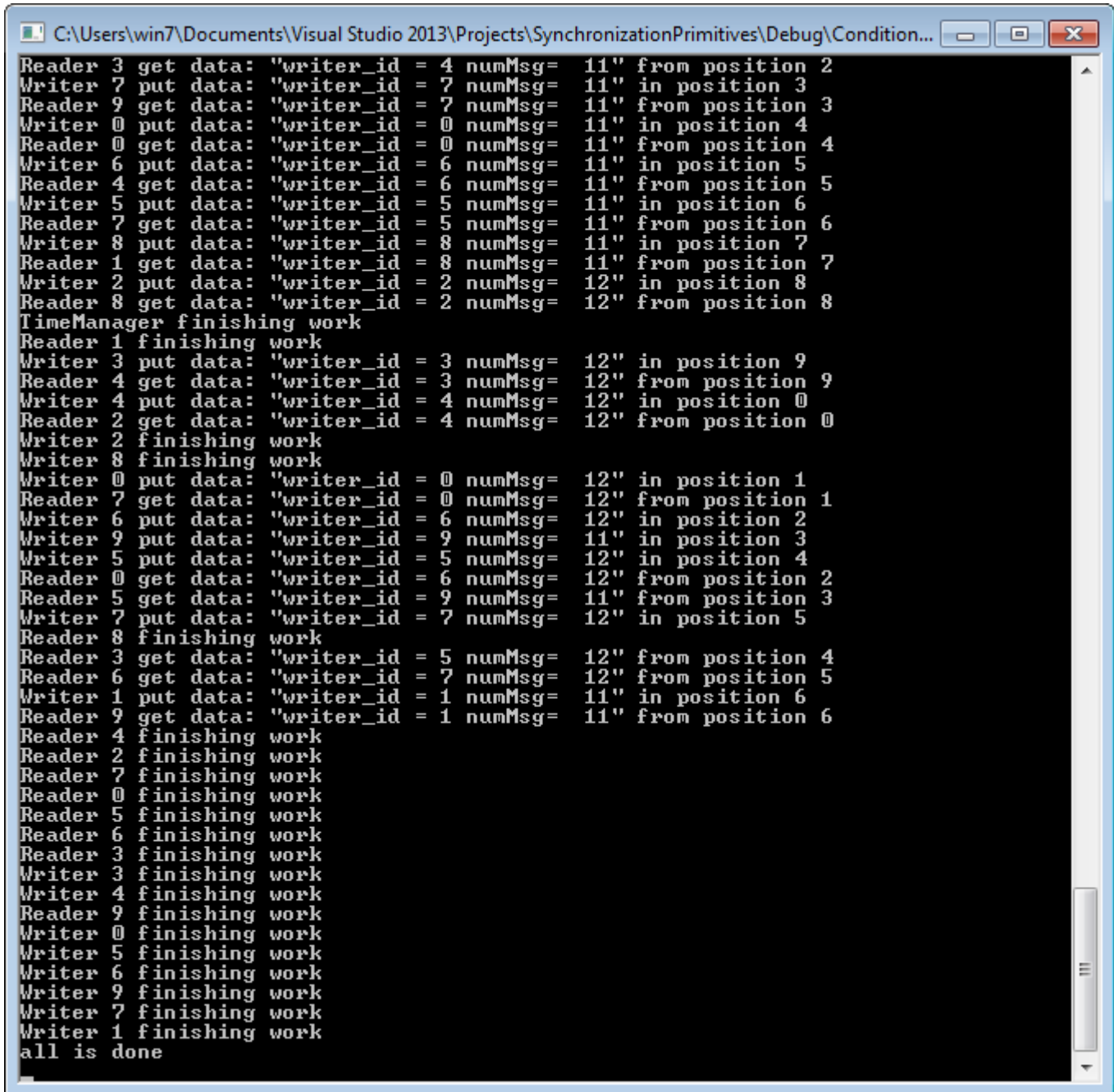
```

```

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 }

```

Результат работы сразу с двумя примитивами показан на рисунке 6. Можно заметить, что читатели не по порядку обрабатывают сообщения, но все сообщения оказываются обработанными.



```
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. 6: Условные переменные и критические секции.

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 ждёт, пока все читатели не прочитают данное сообщение.

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

Для этой задачи были внесены некоторые изменения в файл с сервисными функциями, которые позволили запустить только один поток-писатель. Прimitives синхронизации (события) и общая память инициализируются в листинге 21, а используются писателем и читателями в листинге 22 и 23[1].

Листинг 21: Основной файл (src/SynchronizationPrimitives/ThreadsReaderWriter/main.cpp)

```
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 //                                     (ожидающих сигнала от писателя)
29
30 //имя разделяемой памяти
31 wchar_t shareFileName[] = L"$MyVerySpecialShareFileName$";
32
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
55     // INVALID_HANDLE_VALUE)
56     if ((hFileMapping = CreateFileMapping(INVALID_HANDLE_VALUE, NULL,
57     PAGE_READWRITE, 0, 1500, shareFileName)) == NULL) {
58         // INVALID_HANDLE_VALUE - дескриптор открытого файла
59         // (INVALID_HANDLE_VALUE - файл подкачки)
60         // NULL - атрибуты защиты объекта-отображения
61         // PAGE_READWRITE - возможности доступа к представлению файла при
62         // отображении (PAGE_READWRITE - чтение/запись)
63         // 0, 1500 - старшая и младшая части значения максимального
64         // размера объекта отображения файла
65         // shareFileName - имя объекта-отображения.
66         log.loudlog(_T("Impossible to create shareFile, GLE = %d"),
67             GetLastError());
68         ExitProcess(10000);
69     }
70     //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 tasks are done!"));
121 _getch();
122 return 0;
123 }

```

Листинг 22: Единственный поток-писатель

(src/SynchronizationPrimitives/ThreadsReaderWriter/threadWriter.cpp)

```

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("Waiting 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
50             ятое
51         log.quietlog(_T("Set Event canReadEvent"));
52         SetEvent(canReadEvent);
53         break;
54     default:
55         log.loudlog(_T("Error with func WaitForMultipleObjects in writerHandle
56             , GLE = %d"), GetLastError());
57         ExitProcess(1000);
58     }
59 }
60 log.loudlog(_T("Writer %d finishing work"), myid);
61 return 0;
62 }

```

Листинг 23: Потоки читатели

(src/SynchronizationPrimitives/ThreadsReaderWriter/threadReader.cpp)

```

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("ThreadsReaderWriter.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"));

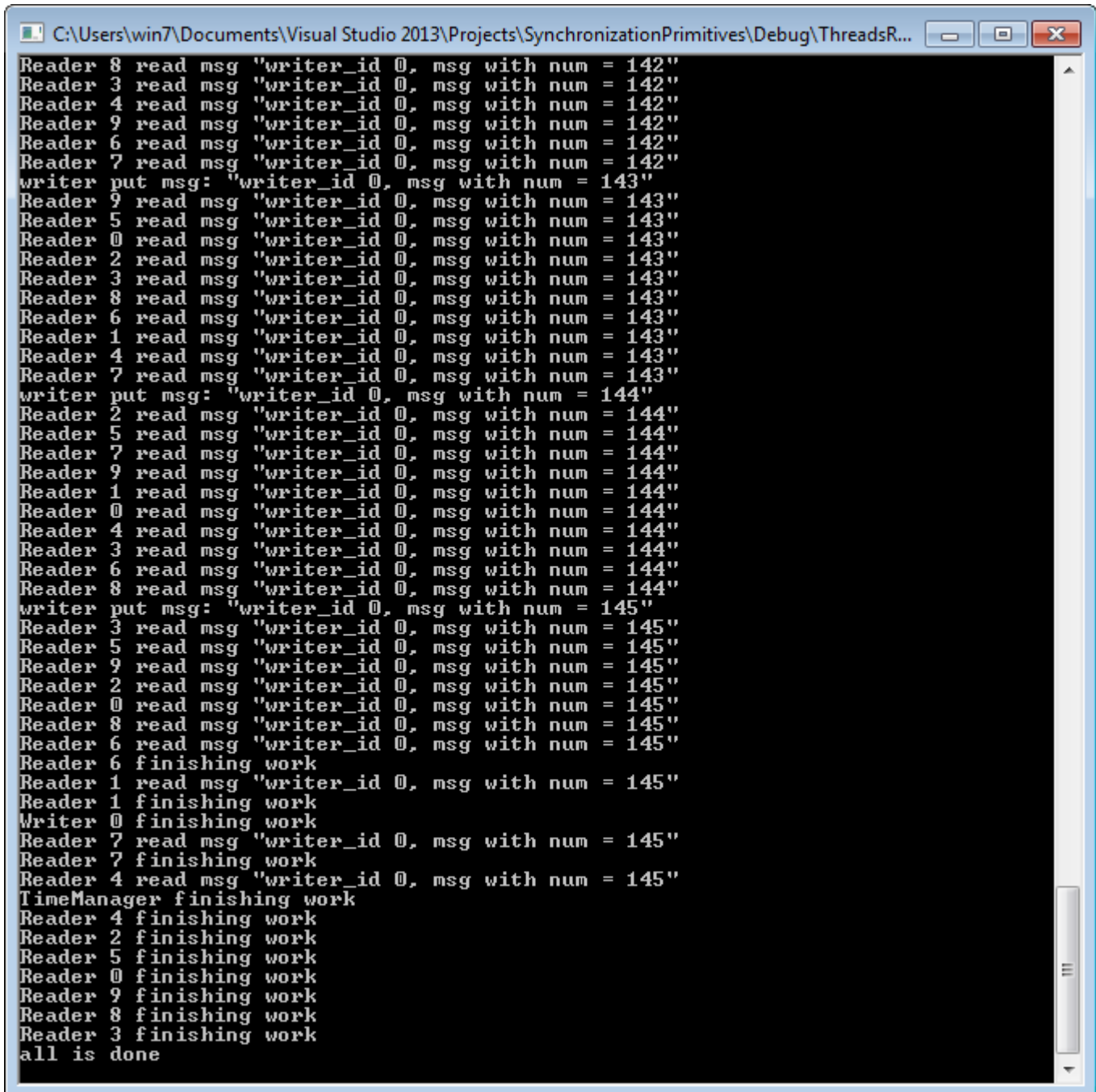
```

```

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
68         еще не успели
69         log.quietlog(_T("Waining for changeCountEvent"));
70         WaitForSingleObject(changeCountEvent, INFINITE);
71         countread--;
72
73         // если мы последние читали, то запрещаем читать и открываем границу
74         if (countread == 0) {
75             log.quietlog(_T("Reset Event canReadEvent"));
76             ResetEvent(canReadEvent);
77             log.quietlog(_T("Set Event allReadEvent"));
78             SetEvent(allReadEvent);
79         }
80
81         //разрешаем изменять счетчик
82         log.quietlog(_T("Set Event changeCountEvent"));
83         SetEvent(changeCountEvent);
84         break;
85     default:
86         log.loudlog(_T("Error with func WaitForMultipleObjects in readerHandle
87             , GLE = %d"), GetLastError());
88         ExitProcess(1001);
89     }
90 }
91 log.loudlog(_T("Reader %d finishing work"), myid);
92 return 0;
93 }

```

Каждый читатель, в соответствии с условиями задачи, по одному разу прочитал сообщение писателя (рисунок 7).



```
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. 7: Задача читатели и писатели.

1.7 Задача читателя-писателя (для потоков разных процессов)

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

Это более сложный случай, т.к. требуется синхронизация разных процессов, а не потоков! Для этого инициализируются объекты-события (листинг 24) с указанием имени (по которому смогут обратиться читатели), потом стартует поток писатель (листинг 25) и запускаются (листинг 26) процессы-читатели (листинг 27)[1].

Листинг 24: Основной файл (src/SynchronizationPrimitives/ProcessWriter/main.cpp)

```
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     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
55     INVALID_HANDLE_VALUE)
56     if ((hFileMapping = CreateFileMapping(INVALID_HANDLE_VALUE, NULL,
57     PAGE_READWRITE, 0, 1500, shareFileName)) == NULL) {
58         // INVALID_HANDLE_VALUE - дескриптор открытого файла
59         // (INVALID_HANDLE_VALUE - файл подкачки)
60         // NULL - атрибуты защиты объекта-отображения
61         // PAGE_READWRITE - возможности доступа к представлению файла при
62         // отображении (PAGE_READWRITE - чтение/запись)
63         // 0, 1500 - старшая и младшая части значения максимального
64         // размера объекта отображения файла
65         // shareFileName - имя объекта-отображения.
66         log.loudlog(_T("Impossible to create shareFile, GLE = %d"),
67             GetLastError());
68         ExitProcess(10000);
69     }
70
71     //отображаем файл на адресное пространство нашего процесса для потока-писа
72     теля
73     lpFileMapForWriters = MapViewOfFile(hFileMapping, FILE_MAP_WRITE, 0, 0, 0)
74
75     ;
76
77     // hFileMapping - дескриптор объекта-отображения файла
78     // FILE_MAP_WRITE - доступа к файлу
79     // 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 tasks are done!"));
116 _getch();
117 return 0;
118 }

```

Листинг 25: Потоки писатели (src/SynchronizationPrimitives/ProcessWriter/threadWriter.cpp)

```

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
62         log.quietlog(_T("Set Event canReadEvent"));
63         SetEvent(canReadEvent);
64
65         break;
66     default:
67         log.loudlog(_T("Error with func WaitForMultipleObjects in writerHandle
68             , GLE = %d"), GetLastError());
69         ExitProcess(1000);
70     }
71 }

```

Листинг 26: Запуск клиентских процессов

(src/SynchronizationPrimitives/ProcessWriter/Utils.cpp)

```

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,
26             CREATE_NEW_CONSOLE |
27             CREATE_SUSPENDED, NULL, NULL, &si, &pi)) {
28             log->loudlog(_T("Impossible to create Process-reader, GLE = %d"),
29                 GetLastError());
30             exit(8000);
31         }
32         allhandlers[count] = pi.hThread;
33     }
34
35     //создаем потоки-писатели
36     log->loudlog(_T("Create writers"));
37     for (int i = 0; i != config->numOfWriters; i++, count++) {
38         log->loudlog(_T("count = %d"), count);
39         //создаем потоки-читатели, которые пока не стартуют
40         if ((allhandlers[count] = CreateThread(NULL, 0, ThreadWriterHandler,
41             (LPVOID)i, CREATE_SUSPENDED, NULL)) == NULL) {
42             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 поток TimeManager, который пока не стартуют
49 if ((allhandlers[count] = CreateThread(NULL, 0, ThreadTimeManagerHandler,
50     (LPVOID)config->t1, 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 27: Поток читателя (src/SynchronizationPrimitives/ProcessReader/main.cpp)

```

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 //Общий ресурс (атрибуты защиты, наследование описателя, имя):
40 HANDLE hFileMapping = OpenFileMapping(FILE_MAP_ALL_ACCESS, false,
41     L"$$$MyVerySpecialShareFileName$$");
42
43 //если объекты не созданы, то не сможем работать
44 if (canReadEvent == NULL || canWriteEvent == NULL || allReadEvent == NULL
45     || changeCountEvent == NULL || exitEvent == NULL
46     || hFileMapping == NULL) {
47     log.loudlog(_T("Impossible to open objects, run server first\n
48         getlasterror=%d"),
49         GetLastError());
50     _getch();
51     return 1001;
52 }
53 //отображаем файл на адресное пространство нашего процесса для потоков-чит
54     ателей
55 LPVOID lpFileMapForReaders = MapViewOfFile(hFileMapping,
56     FILE_MAP_ALL_ACCESS, 0, 0, 0);
57 // hFileMapping - дескриптор объекта-отображения файла
58 // FILE_MAP_ALL_ACCESS - доступа к файлу
59 // 0, 0 - старшая и младшая части смещения начала отображаемого участка в
60 файле
61 // (0 - начало отображаемого участка совпадает с началом файла)
62 // 0 - размер отображаемого участка файла в байтах (0 - весь файл)
63
64 HANDLE readerhandlers[2];
65 readerhandlers[0] = exitEvent;
66 readerhandlers[1] = canReadEvent;

```

```

66 while (1) { //основной цикл
67     //ждем, пока все прочитают
68     log.quietlog(_T("Waining for allReadEvent"));
69     WaitForSingleObject(allReadEvent, INFINITE);
70     //узнаем, сколько потоков-читателей прошло данную границу
71     log.quietlog(_T("Waining 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("Waining 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 tasks are done!"));
142     _getch();
143     return 0;
144 }

```

Результат работы показан на рисунке 8, но тут интереснее взглянуть на process explorer (рисунок 9), который показывает наличие множества процессов-читателей, а среди ресурсов имена событий и общую память.

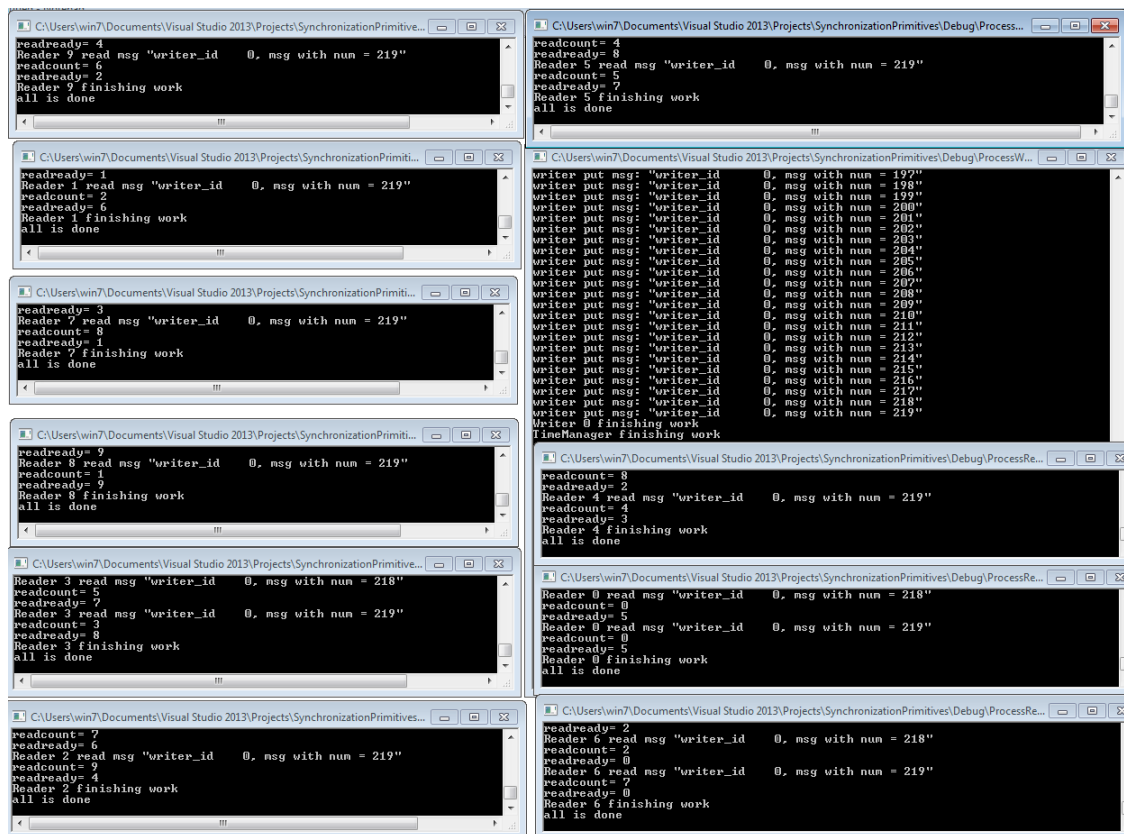


Рис. 8: Решение задачи читатели-писатели для процессов.

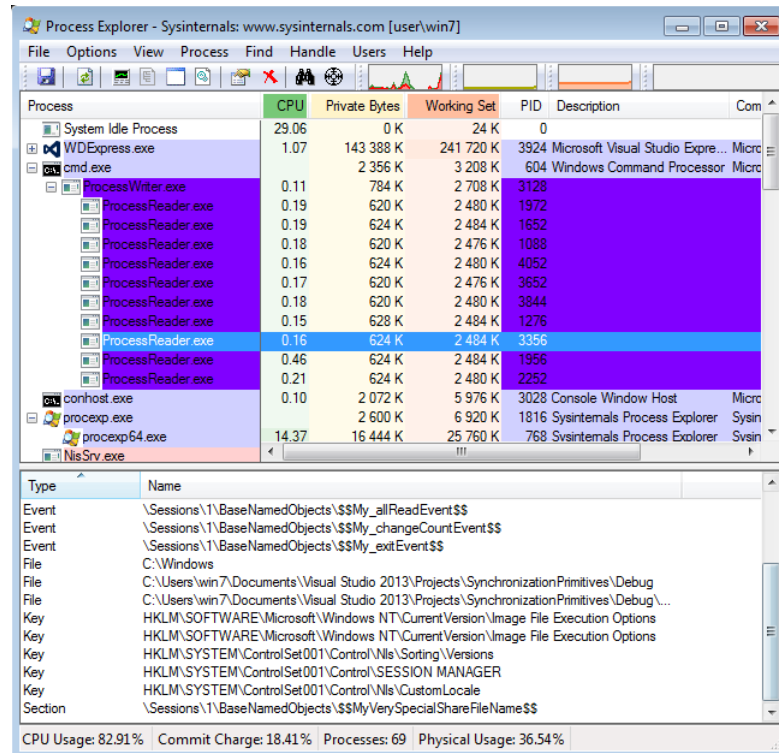


Рис. 9: Вывод process expl orer.

Листинг 28 содержит протокол работы единственного потока писателя, а листинг 29 - одного из читателей.

Листинг 28: Единственный писатель

```
1 [25/3/2015 18:59:7] ProcessWriter.ThreadWriter is starting.
2 [25/3/2015 18:59:7] Waining for multiple objects
3 [25/3/2015 18:59:9] Get canWriteEvent
4 [25/3/2015 18:59:9] Writer put msg: "Writer_id 0, msg with num = 1"
5 [25/3/2015 18:59:9] Waining for changeCountEvent
6 [25/3/2015 18:59:9] Set Event changeCountEvent
7 [25/3/2015 18:59:9] Set Event canReadEvent
8 [25/3/2015 18:59:9] Waining for multiple objects
9 [25/3/2015 18:59:9] Get canWriteEvent
10 [25/3/2015 18:59:9] Writer put msg: "Writer_id 0, msg with num = 2"
11 [25/3/2015 18:59:9] Waining for changeCountEvent
12 [25/3/2015 18:59:9] Set Event changeCountEvent
13 [25/3/2015 18:59:9] Set Event canReadEvent
14 [25/3/2015 18:59:9] Waining for multiple objects
15 [25/3/2015 18:59:9] Get canWriteEvent
16 [25/3/2015 18:59:9] Writer put msg: "Writer_id 0, msg with num = 3"
17 [25/3/2015 18:59:9] Waining for changeCountEvent
18 [25/3/2015 18:59:9] Set Event changeCountEvent
19 [25/3/2015 18:59:9] Set Event canReadEvent
20 [25/3/2015 18:59:9] Waining for multiple objects
21 [25/3/2015 18:59:9] Get canWriteEvent
22 [25/3/2015 18:59:9] Writer put msg: "Writer_id 0, msg with num = 4"
23 [25/3/2015 18:59:9] Waining for changeCountEvent
24 [25/3/2015 18:59:9] Set Event changeCountEvent
25 [25/3/2015 18:59:9] Set Event canReadEvent
26 [25/3/2015 18:59:9] Waining for multiple objects
27 [25/3/2015 18:59:9] Get canWriteEvent
28 [25/3/2015 18:59:9] Writer put msg: "Writer_id 0, msg with num = 5"
29 [25/3/2015 18:59:9] Waining for changeCountEvent
30 [25/3/2015 18:59:9] Set Event changeCountEvent
31 [25/3/2015 18:59:9] Set Event canReadEvent
32 [25/3/2015 18:59:9] Waining for multiple objects
33 [25/3/2015 18:59:9] Get canWriteEvent
34 [25/3/2015 18:59:9] Writer put msg: "Writer_id 0, msg with num = 6"
35 [25/3/2015 18:59:9] Waining for changeCountEvent
36 [25/3/2015 18:59:9] Set Event changeCountEvent
37 [25/3/2015 18:59:9] Set Event canReadEvent
38 [25/3/2015 18:59:9] Waining for multiple objects
39 [25/3/2015 18:59:9] Get canWriteEvent
40 [25/3/2015 18:59:9] Writer put msg: "Writer_id 0, msg with num = 7"
41 [25/3/2015 18:59:9] Waining for changeCountEvent
```

```

42 [25/3/2015 18:59:9] Set Event changeCountEvent
43 [25/3/2015 18:59:9] Set Event canReadEvent
44 [25/3/2015 18:59:9] Waining for multiple objects
45 [25/3/2015 18:59:10] Get canWriteEvent
46 [25/3/2015 18:59:10] Writer put msg: "Writer_id 0, msg with num = 8"
47 [25/3/2015 18:59:10] Waining for changeCountEvent
48 [25/3/2015 18:59:10] Set Event changeCountEvent
49 [25/3/2015 18:59:10] Set Event canReadEvent
50 [25/3/2015 18:59:10] Waining for multiple objects
51 [25/3/2015 18:59:10] Get canWriteEvent
52 [25/3/2015 18:59:10] Writer put msg: "Writer_id 0, msg with num = 9"
53 [25/3/2015 18:59:10] Waining for changeCountEvent
54 [25/3/2015 18:59:10] Set Event changeCountEvent
55 [25/3/2015 18:59:10] Set Event canReadEvent
56 [25/3/2015 18:59:10] Waining for multiple objects
57 [25/3/2015 18:59:10] Get exitEvent
58 [25/3/2015 18:59:10] Writer 0 finishing work
59 [25/3/2015 18:59:10] Shutting down.

```

Листинг 29: Один из читателей

```

1 [25/3/2015 18:59:9] ProcessReader is starting.
2 [25/3/2015 18:59:9] Reader with id= 1 is started
3 [25/3/2015 18:59:9] Waining for allReadEvent
4 [25/3/2015 18:59:9] Waining for changeCountEvent
5 [25/3/2015 18:59:9] Readready= 4
6
7 [25/3/2015 18:59:9] Set Event changeCountEvent
8 [25/3/2015 18:59:9] Waining for multiple objects
9 [25/3/2015 18:59:9] Get canReadEvent
10 [25/3/2015 18:59:9] Reader 1 read msg "Writer_id 0, msg with num = 1"
11 [25/3/2015 18:59:9] Waining for changeCountEvent
12 [25/3/2015 18:59:9] Readcount= 6
13 [25/3/2015 18:59:9] Set Event changeCountEvent
14 [25/3/2015 18:59:9] Waining for allReadEvent
15 [25/3/2015 18:59:9] Waining for changeCountEvent
16 [25/3/2015 18:59:9] Readready= 9
17
18 [25/3/2015 18:59:9] Set Event changeCountEvent
19 [25/3/2015 18:59:9] Waining for multiple objects
20 [25/3/2015 18:59:9] Get canReadEvent
21 [25/3/2015 18:59:9] Reader 1 read msg "Writer_id 0, msg with num = 2"
22 [25/3/2015 18:59:9] Waining for changeCountEvent
23 [25/3/2015 18:59:9] Readcount= 5
24 [25/3/2015 18:59:9] Set Event changeCountEvent

```

```

25 [25/3/2015 18:59:9] Waining for allReadEvent
26 [25/3/2015 18:59:9] Waining for changeCountEvent
27 [25/3/2015 18:59:9] Readready= 6
28
29 [25/3/2015 18:59:9] Set Event changeCountEvent
30 [25/3/2015 18:59:9] Waining for multiple objects
31 [25/3/2015 18:59:9] Get canReadEvent
32 [25/3/2015 18:59:9] Reader 1 read msg "Writer_id 0, msg with num = 3"
33 [25/3/2015 18:59:9] Waining for changeCountEvent
34 [25/3/2015 18:59:9] Readcount= 6
35 [25/3/2015 18:59:9] Set Event changeCountEvent
36 [25/3/2015 18:59:9] Waining for allReadEvent
37 [25/3/2015 18:59:9] Waining for changeCountEvent
38 [25/3/2015 18:59:9] Readready= 4
39
40 [25/3/2015 18:59:9] Set Event changeCountEvent
41 [25/3/2015 18:59:9] Waining for multiple objects
42 [25/3/2015 18:59:9] Get canReadEvent
43 [25/3/2015 18:59:9] Reader 1 read msg "Writer_id 0, msg with num = 4"
44 [25/3/2015 18:59:9] Waining for changeCountEvent
45 [25/3/2015 18:59:9] Readcount= 3
46 [25/3/2015 18:59:9] Set Event changeCountEvent
47 [25/3/2015 18:59:9] Waining for allReadEvent
48 [25/3/2015 18:59:9] Waining for changeCountEvent
49 [25/3/2015 18:59:9] Readready= 6
50
51 [25/3/2015 18:59:9] Set Event changeCountEvent
52 [25/3/2015 18:59:9] Waining for multiple objects
53 [25/3/2015 18:59:9] Get canReadEvent
54 [25/3/2015 18:59:9] Reader 1 read msg "Writer_id 0, msg with num = 5"
55 [25/3/2015 18:59:9] Waining for changeCountEvent
56 [25/3/2015 18:59:9] Readcount= 7
57 [25/3/2015 18:59:9] Set Event changeCountEvent
58 [25/3/2015 18:59:9] Waining for allReadEvent
59 [25/3/2015 18:59:9] Waining for changeCountEvent
60 [25/3/2015 18:59:9] Readready= 7
61
62 [25/3/2015 18:59:9] Set Event changeCountEvent
63 [25/3/2015 18:59:9] Waining for multiple objects
64 [25/3/2015 18:59:9] Get canReadEvent
65 [25/3/2015 18:59:9] Reader 1 read msg "Writer_id 0, msg with num = 6"
66 [25/3/2015 18:59:9] Waining for changeCountEvent
67 [25/3/2015 18:59:9] Readcount= 3
68 [25/3/2015 18:59:9] Set Event changeCountEvent
69 [25/3/2015 18:59:9] Waining for allReadEvent

```

```

70 [25/3/2015 18:59:9] Waining for changeCountEvent
71 [25/3/2015 18:59:9] Readready= 9
72
73 [25/3/2015 18:59:9] Set Event changeCountEvent
74 [25/3/2015 18:59:9] Waining for multiple objects
75 [25/3/2015 18:59:9] Get canReadEvent
76 [25/3/2015 18:59:9] Reader 1 read msg "Writer_id 0, msg with num = 7"
77 [25/3/2015 18:59:9] Waining for changeCountEvent
78 [25/3/2015 18:59:9] Readcount= 8
79 [25/3/2015 18:59:9] Set Event changeCountEvent
80 [25/3/2015 18:59:9] Waining for allReadEvent
81 [25/3/2015 18:59:9] Waining for changeCountEvent
82 [25/3/2015 18:59:10] Readready= 6
83
84 [25/3/2015 18:59:10] Set Event changeCountEvent
85 [25/3/2015 18:59:10] Waining for multiple objects
86 [25/3/2015 18:59:10] Get canReadEvent
87 [25/3/2015 18:59:10] Reader 1 read msg "Writer_id 0, msg with num = 8"
88 [25/3/2015 18:59:10] Waining for changeCountEvent
89 [25/3/2015 18:59:10] Readcount= 5
90 [25/3/2015 18:59:10] Set Event changeCountEvent
91 [25/3/2015 18:59:10] Waining for allReadEvent
92 [25/3/2015 18:59:10] Waining for changeCountEvent
93 [25/3/2015 18:59:10] Readready= 4
94
95 [25/3/2015 18:59:10] Set Event changeCountEvent
96 [25/3/2015 18:59:10] Waining for multiple objects
97 [25/3/2015 18:59:10] Get canReadEvent
98 [25/3/2015 18:59:10] Reader 1 read msg "Writer_id 0, msg with num = 9"
99 [25/3/2015 18:59:10] Waining for changeCountEvent
100 [25/3/2015 18:59:10] Readcount= 3
101 [25/3/2015 18:59:10] Set Event changeCountEvent
102 [25/3/2015 18:59:10] Waining for allReadEvent
103 [25/3/2015 18:59:10] Waining for changeCountEvent
104 [25/3/2015 18:59:10] Readready= 4
105
106 [25/3/2015 18:59:10] Set Event changeCountEvent
107 [25/3/2015 18:59:10] Waining for multiple objects
108 [25/3/2015 18:59:10] Get exitEvent
109 [25/3/2015 18:59:10] Reader 1 finishing work
110 [25/3/2015 18:59:10] All tasks are done!
111 [25/3/2015 18:59:18] Shutting down.

```

2 Модификация задачи читатели-писатели без доступа к памяти

Требуется решить задачу читатели-писатели таким образом, чтобы читатели не имели доступа к памяти по записи. Задача сводится к тому, чтобы счётчик был под управлением какого-то одного потока (в моём случае это писатель), а остальные "отчитывались"бы ему о своей работе. Задача решена на механизме событие[1].

Как и в предыдущем случае, у нас есть основной файл программы (листинг 30), поток писатель (листинг 31), сервисные функции (листинг 32) и потоки читатели (листинг 33), которые не имеют доступа на запись к общей памяти.

Листинг 30: Основной файл (src/SynchronizationPrimitives/NoMemProcessWriter/main.cpp)

```
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
65         // отображении (PAGE_READWRITE - чтение/запись)
66         // 0, 1500 - старшая и младшая части значения максимального
67         // размера объекта отображения файла
68         // shareFileName - имя объекта-отображения.
69         log.loudlog(_T("Impossible to create shareFile, GLE = %d"),
70             GetLastError());
71         ExitProcess(10000);
72     }
73
74     //отображаем файл на адресное пространство нашего процесса для потока-писа
75     //теля
76
77     lpFileMapForWriters = MapViewOfFile(hFileMapping, FILE_MAP_WRITE, 0, 0, 0)
78     ;
79
80     // 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 tasks are done!"));
112 _getch();
113 return 0;
114 }

```

Листинг 31: Потоки писатели (src/SynchronizationPrimitives/NoMemProcessWriter/threadWriter.cpp)

```

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 canChangeEvent;
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
69             // Теперь ожидаем отчётов о прочтении
70             readyState = true;
71             // Больше ни кто не готовится
72             log.quietlog(_T("Reset Event readerGetReadyEvent"));
73             ResetEvent(readerGetReadyEvent);
74             // Можно читать
75             log.quietlog(_T("Set Event readerCanReadEvent"));
76             SetEvent(readerCanReadEvent);
77         }
78     }
79     // Ждём следующего отчёта
80     log.quietlog(_T("Set Event canChangeCountEvent"));
81     SetEvent(canChangeCountEvent);

```

```

81
82     break;
83     default:
84         log.loudlog(_T("Error with func WaitForMultipleObjects in writerHandle
            , GLE = %d"), GetLastError());
85         ExitProcess(1000);
86     }
87 }
88 log.loudlog(_T("Writer %d finishing work"), myid);
89 return 0;
90 }

```

Листинг 32: Запуск клиентских процессов (src/SynchronizationPrimitives/NoMemProcessWriter/utls.c)

```

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"),
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 поток TimeManager, который пока не стартуют
50 if ((allhandlers[count] = CreateThread(NULL, 0, ThreadTimeManagerHandler,
51     (LPVOID)config->t1, 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 33: Поток читателя (src/SynchronizationPrimitives/NoMemProcessReader/main.cpp)

```

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("NoMemProcessReader"));
12        log.loudlog(_T("Error with start reader process. Need 2 arguments, but %
13            d presented."), argc);
14        _getch();
15        ExitProcess(1000);
16    }
17    //получаем из командной строки наш номер
18    int myid = _wtoi(argv[1]);
19    int pause = _wtoi(argv[2]);
20
21    Logger log(_T("NoMemProcessReader"), myid);
22    log.loudlog(_T("Reader with id= %d is started"), myid);
23
24    // Состояние готовности:
25    // true - ждём сообщение для чтения
26    // false - текущее сообщение уже прочитано,
27    // ждём сигнала перехода в режим готовности
28    bool readyState = false;
29
30    //Инициализируем средства синхронизации
31    // (атрибуты защиты, наследование описателя, имя):
32    //писатель записал сообщение (ручной сброс);
33    HANDLE readerCanReadEvent = OpenEvent(EVENT_ALL_ACCESS, false,
34        L"$$$My_readerCanReadEvent$$");
35    //все читатели готовы к приему следующего (автосброс);
36    HANDLE readerGetReadyEvent = OpenEvent(EVENT_ALL_ACCESS, false,
37        L"$$$My_readerGetReadyEvent$$");
38    //разрешение работы со счетчиком (автосброс);
39    HANDLE canChangeCountEvent = OpenEvent(EVENT_ALL_ACCESS, false,
40        L"$$$My_canChangeCountEvent$$");
41    //
42    HANDLE changeCountEvent = OpenEvent(EVENT_ALL_ACCESS, false,
43        L"$$$My_changeCountEvent$$");
44    //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("Waiting 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 tasks are done!"));
162     _getch();
163     return 0;
164 }

```


Результат работы на рисунке 10.

The image displays 12 individual screenshots of a Visual Studio 2013 debugger window, each showing the output of a different thread in a reader-writer synchronization program. The threads are identified by their IDs (e.g., Reader 7, Reader 8, Reader 4, Reader 5, Reader 1, Reader 2, Reader 6, Reader 9, Reader 3, Reader 0). The output for each thread follows a similar pattern: it shows messages read by the reader (e.g., "Reader 7 read msg: 'writer_id 0, msg with num = 114'") and messages put by the writer (e.g., "writer put msg: 'writer_id 0, msg with num = 91'"). The threads conclude their execution with "Reader X finishing work" and "all is done". The sequence of screenshots illustrates the interleaved execution of multiple readers and writers, demonstrating the synchronization logic.

Рис. 10: Модификация задачи читателя-писателя.

3 Рациональное решение задачи читатели-писатели

В задаче предлагается найти более рациональное решение задачи читатели-писатели, но при имеющихся ограничениях (каждый процесс производит чтение ровно один раз) ничего существенно нового привнести невозможно, т.к. фактически всё сведётся к вырождению одних примитивов синхронизации в другие. Но предыдущую задачу, когда взаимодействие происходило в рамках одного процесса, можно рассмотреть Slim Reader/Writer (SRW) Lock, который появился в Windows Vista и позволяет накладывать различные ограничения в зависимости от задачи. Решение с использованием Slim Reader/Writer (SRW) Lock приведено в листинге 34 (инициализация), 35 (писатели) и 36 (читатели)[5]. Сравнивать скорость этой реализации с реализациями, работающими с процессами не корректно, так что это можно рассматривать скорее как изучение ещё одного инструмента для синхронизации.

Листинг 34: Основной файл (src/SynchronizationPrimitives/OptimalReaderWriter/main.cpp)

```
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 // инструмент синхронизации:
17 SRWLOCK lock;
18 //условная переменная для потоков-читателей
19 CONDITION_VARIABLE condread;
20
21 HANDLE exitEvent; //завершение программы (ручной сброс);
22
23 //имя разделяемой памяти
24 wchar_t shareFileName[] = L"$MyVerySpecialShareFileName$";
25
26 HANDLE hFileMapping; //объект-отображение файла
27 // указатели на отображаемую память
28 LPVOID lpFileMapForWriters;
29 LPVOID lpFileMapForReaders;
```

```

30
31 int _tmain(int argc, _TCHAR* argv[]) {
32     Logger log(_T("OptimalReaderWriter"));
33
34     if (argc < 2)
35         // Используем конфигурацию по-умолчанию
36         SetDefaultConfig(&config, &log);
37     else
38         // Загрузка конфига из файла
39         SetConfig(argv[1], &config, &log);
40
41     //создаем необходимые потоки без их запуска
42     CreateAllThreads(&config, &log);
43
44     //Инициализируем ресурс (share memory): создаем объект "отображаемый файл"
45     // будет использован системный файл подкачки (на диске файл создаваться
46     // не будет), т.к. в качестве дескриптора файла использовано значение
47     // равное 0xFFFFFFFF (его эквивалент - символическая константа
48     // INVALID_HANDLE_VALUE)
49     if ((hFileMapping = CreateFileMapping(INVALID_HANDLE_VALUE, NULL,
50     PAGE_READWRITE, 0, 1500, shareFileName)) == NULL) {
51         // INVALID_HANDLE_VALUE - дескриптор открытого файла
52         // (INVALID_HANDLE_VALUE - файл подкачки)
53         // NULL - атрибуты защиты объекта-отображения
54         // PAGE_READWRITE - возможности доступа к представлению файла при
55         // отображении (PAGE_READWRITE - чтение/запись)
56         // 0, 1500 - старшая и младшая части значения максимального
57         // размера объекта отображения файла
58         // shareFileName - имя объекта-отображения.
59         log.loudlog(_T("Impossible to create shareFile, GLE = %d"),
60             GetLastError());
61         ExitProcess(10000);
62     }
63     //отображаем файл на адресное пространство нашего процесса для потока-писа
64     // теля
65     lpFileMapForWriters = MapViewOfFile(hFileMapping, FILE_MAP_WRITE, 0, 0, 0)
66     ;
67     // hFileMapping - дескриптор объекта-отображения файла
68     // FILE_MAP_WRITE - доступа к файлу
69     // 0, 0 - старшая и младшая части смещения начала отображаемого участка в
70     // файле
71     // (0 - начало отображаемого участка совпадает с началом файла)
72     // 0 - размер отображаемого участка файла в байтах (0 - весь файл)
73
74     //отображаем файл на адресное пространство нашего процесса для потоков-чит

```

ателей

```
71 lpFileMapForReaders = MapViewOfFile(hFileMapping, FILE_MAP_READ, 0, 0, 0);
72
73 //инициализируем средства синхронизации
74 //событие "завершение работы программы", ручной сброс, изначально занято
75 exitEvent = CreateEvent(NULL, true, false, L "");
76 InitializeSRWLock(&lock);
77 InitializeConditionVariable(&condread);
78
79 //запускаем потоки-писатели и поток-планировщик на исполнение
80 for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
81     ResumeThread(allhandlers[i]);
82
83 //ожидаем завершения всех потоков
84 WaitForMultipleObjects(config.numOfReaders + config.numOfWriters + 1,
85     allhandlers, TRUE, INFINITE);
86
87 //закрываем handle потоков
88 for (int i = 0; i < config.numOfReaders + config.numOfWriters + 1; i++)
89     CloseHandle(allhandlers[i]);
90
91 //закрываем описатели объектов синхронизации
92 CloseHandle(exitEvent);
93
94 //закрываем handle общего ресурса
95 UnmapViewOfFile(lpFileMapForReaders);
96 UnmapViewOfFile(lpFileMapForWriters);
97
98 //закрываем объект "отображаемый файл"
99 CloseHandle(hFileMapping);
100
101 // Завершение работы
102 log.loudlog(_T("All tasks are done!"));
103 _getch();
104 return 0;
105 }
```

Листинг 35: Поток писателя (src/SynchronizationPrimitives/OptimalReaderWriter/threadWriter.cpp)

```

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("OptimalReaderWriter.ThreadWriter"), myid);
11    extern bool isDone;
12    extern struct Configuration config;
13    extern SRWLOCK lock;
14    extern CONDITION_VARIABLE condread;
15    extern LPVOID lpFileMapForWriters;
16
17    int msgnum = 0;
18    while (isDone != true) {
19        // Захват объекта синхронизации (монопольный доступ!)
20        log.quietlog(_T("Waiting for Slim Reader/Writer (SRW) Lock"));
21        AcquireSRWLockExclusive(&lock);
22        log.quietlog(_T("Get SRW Lock"));
23
24        // Запись сообщения
25        swprintf_s((_TCHAR *)lpFileMapForWriters, 1500,
26            _T("writer_id %d, msg with num = %d"), myid, msgnum++);
27        log.loudlog(_T("writer put msg: \"%s\""), lpFileMapForWriters);
28
29        //освобождение объекта синхронизации
30        log.quietlog(_T("Release SRW Lock"));
31        WakeAllConditionVariable(&condread);
32        ReleaseSRWLockExclusive(&lock);
33
34        //задержка
35        Sleep(config.writersDelay);
36    }
37    log.loudlog(_T("Writer %d finishing work"), myid);
38    return 0;
39 }

```

Листинг 36: Поток читателя (src/SynchronizationPrimitives/OptimalReaderWriter/threadReader.cpp)

```

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("OptimalReaderWriter.ThreadReader"), myid);
11    extern bool isDone;
12    extern struct Configuration config;
13    extern SRWLOCK lock;
14    extern CONDITION_VARIABLE condread;
15    extern LPVOID lpFileMapForReaders;
16
17    while (isDone != true) {
18        // Захват объекта синхронизации (совместный доступ!)
19        log.quietlog(_T("Waiting for Slim Reader/Writer (SRW) Lock"));
20        AcquireSRWLockShared(&lock);
21        SleepConditionVariableSRW(&condread, &lock, INFINITE,
22            CONDITION_VARIABLE_LOCKMODE_SHARED);
23        log.quietlog(_T("Get SRW Lock"));
24
25        //читаем сообщение
26        log.loudlog(_T("Reader %d read msg \"%s\""), myid,
27            (_TCHAR *)lpFileMapForReaders);
28
29        //освобождение объекта синхронизации
30        log.quietlog(_T("Release SRW Lock"));
31        ReleaseSRWLockShared(&lock);
32
33        //задержка
34        Sleep(config.readersDelay);
35    }
36    log.loudlog(_T("Reader %d finishing work"), myid);
37    return 0;
38 }

```

Требование по обязательному прочтению каждого сообщения каждым читателем выполняется, как видно на рисунке 11.

```

C:\Windows\system32\cmd.exe
writer put msg: "writer_id 0, msg with num = 12"
Reader 3 read msg "writer_id 0, msg with num = 12"
Reader 1 read msg "writer_id 0, msg with num = 12"
Reader 7 read msg "writer_id 0, msg with num = 12"
Reader 6 read msg "writer_id 0, msg with num = 12"
Reader 0 read msg "writer_id 0, msg with num = 12"
Reader 4 read msg "writer_id 0, msg with num = 12"
Reader 5 read msg "writer_id 0, msg with num = 12"
Reader 8 read msg "writer_id 0, msg with num = 12"
Reader 2 read msg "writer_id 0, msg with num = 12"
Reader 9 read msg "writer_id 0, msg with num = 12"
writer put msg: "writer_id 0, msg with num = 13"
Reader 4 read msg "writer_id 0, msg with num = 13"
Reader 5 read msg "writer_id 0, msg with num = 13"
Reader 2 read msg "writer_id 0, msg with num = 13"
Reader 8 read msg "writer_id 0, msg with num = 13"
Reader 0 read msg "writer_id 0, msg with num = 13"
Reader 9 read msg "writer_id 0, msg with num = 13"
Reader 7 read msg "writer_id 0, msg with num = 13"
Reader 6 read msg "writer_id 0, msg with num = 13"
Reader 3 read msg "writer_id 0, msg with num = 13"
Reader 1 read msg "writer_id 0, msg with num = 13"
writer put msg: "writer_id 0, msg with num = 14"
Reader 1 read msg "writer_id 0, msg with num = 14"
Reader 7 read msg "writer_id 0, msg with num = 14"
Reader 6 read msg "writer_id 0, msg with num = 14"
Reader 4 read msg "writer_id 0, msg with num = 14"
Reader 8 read msg "writer_id 0, msg with num = 14"
Reader 3 read msg "writer_id 0, msg with num = 14"
Reader 0 read msg "writer_id 0, msg with num = 14"
Reader 2 read msg "writer_id 0, msg with num = 14"
Reader 5 read msg "writer_id 0, msg with num = 14"
Reader 9 read msg "writer_id 0, msg with num = 14"
TimeManager finishing work
Writer 0 finishing work

```

Рис. 11: Модификация задачи читатели-писатели.

Листинг 37 показывает лог работы писателя, а листинг 38 - одного из читателей.

Листинг 37: Протокол работы писателя

1	[25/3/2015 19:9:5] OptimalReaderWriter.ThreadWriter is starting.
2	[25/3/2015 19:9:5] Waining for Slim Reader/Writer (SRW) Lock
3	[25/3/2015 19:9:5] Get SRW Lock
4	[25/3/2015 19:9:5] writer put msg: "writer_id 0, msg with num = 0"
5	[25/3/2015 19:9:5] Release SRW Lock
6	[25/3/2015 19:9:5] Waining for Slim Reader/Writer (SRW) Lock
7	[25/3/2015 19:9:5] Get SRW Lock
8	[25/3/2015 19:9:5] writer put msg: "writer_id 0, msg with num = 1"
9	[25/3/2015 19:9:5] Release SRW Lock

10 [25/3/2015 19:9:5] Waining for Slim Reader/Writer (SRW) Lock
11 [25/3/2015 19:9:5] Get SRW Lock
12 [25/3/2015 19:9:5] writer put msg: "writer_id 0, msg with num = 2"
13 [25/3/2015 19:9:5] Release SRW Lock
14 [25/3/2015 19:9:5] Waining for Slim Reader/Writer (SRW) Lock
15 [25/3/2015 19:9:5] Get SRW Lock
16 [25/3/2015 19:9:5] writer put msg: "writer_id 0, msg with num = 3"
17 [25/3/2015 19:9:5] Release SRW Lock
18 [25/3/2015 19:9:5] Waining for Slim Reader/Writer (SRW) Lock
19 [25/3/2015 19:9:5] Get SRW Lock
20 [25/3/2015 19:9:5] writer put msg: "writer_id 0, msg with num = 4"
21 [25/3/2015 19:9:5] Release SRW Lock
22 [25/3/2015 19:9:6] Waining for Slim Reader/Writer (SRW) Lock
23 [25/3/2015 19:9:6] Get SRW Lock
24 [25/3/2015 19:9:6] writer put msg: "writer_id 0, msg with num = 5"
25 [25/3/2015 19:9:6] Release SRW Lock
26 [25/3/2015 19:9:6] Waining for Slim Reader/Writer (SRW) Lock
27 [25/3/2015 19:9:6] Get SRW Lock
28 [25/3/2015 19:9:6] writer put msg: "writer_id 0, msg with num = 6"
29 [25/3/2015 19:9:6] Release SRW Lock
30 [25/3/2015 19:9:6] Waining for Slim Reader/Writer (SRW) Lock
31 [25/3/2015 19:9:6] Get SRW Lock
32 [25/3/2015 19:9:6] writer put msg: "writer_id 0, msg with num = 7"
33 [25/3/2015 19:9:6] Release SRW Lock
34 [25/3/2015 19:9:6] Waining for Slim Reader/Writer (SRW) Lock
35 [25/3/2015 19:9:6] Get SRW Lock
36 [25/3/2015 19:9:6] writer put msg: "writer_id 0, msg with num = 8"
37 [25/3/2015 19:9:6] Release SRW Lock
38 [25/3/2015 19:9:6] Waining for Slim Reader/Writer (SRW) Lock
39 [25/3/2015 19:9:6] Get SRW Lock
40 [25/3/2015 19:9:6] writer put msg: "writer_id 0, msg with num = 9"
41 [25/3/2015 19:9:6] Release SRW Lock
42 [25/3/2015 19:9:7] Waining for Slim Reader/Writer (SRW) Lock
43 [25/3/2015 19:9:7] Get SRW Lock
44 [25/3/2015 19:9:7] writer put msg: "writer_id 0, msg with num = 10"
45 [25/3/2015 19:9:7] Release SRW Lock
46 [25/3/2015 19:9:7] Waining for Slim Reader/Writer (SRW) Lock
47 [25/3/2015 19:9:7] Get SRW Lock
48 [25/3/2015 19:9:7] writer put msg: "writer_id 0, msg with num = 11"
49 [25/3/2015 19:9:7] Release SRW Lock
50 [25/3/2015 19:9:7] Waining for Slim Reader/Writer (SRW) Lock
51 [25/3/2015 19:9:7] Get SRW Lock
52 [25/3/2015 19:9:7] writer put msg: "writer_id 0, msg with num = 12"
53 [25/3/2015 19:9:7] Release SRW Lock
54 [25/3/2015 19:9:7] Waining for Slim Reader/Writer (SRW) Lock


```

55 [25/3/2015 19:9:7] Get SRW Lock
56 [25/3/2015 19:9:7] writer put msg: "writer_id 0, msg with num = 13"
57 [25/3/2015 19:9:7] Release SRW Lock
58 [25/3/2015 19:9:7] Waining for Slim Reader/Writer (SRW) Lock
59 [25/3/2015 19:9:7] Get SRW Lock
60 [25/3/2015 19:9:7] writer put msg: "writer_id 0, msg with num = 14"
61 [25/3/2015 19:9:7] Release SRW Lock
62 [25/3/2015 19:9:8] Writer 0 finishing work
63 [25/3/2015 19:9:8] Shutting down.

```

Листинг 38: Протокол работы читателя

```

1 [25/3/2015 19:9:5] OptimalReaderWriter.ThreadReader is starting.
2 [25/3/2015 19:9:5] Waining for Slim Reader/Writer (SRW) Lock
3 [25/3/2015 19:9:5] Get SRW Lock
4 [25/3/2015 19:9:5] Reader 1 read msg "writer_id 0, msg with num = 0"
5 [25/3/2015 19:9:5] Release SRW Lock
6 [25/3/2015 19:9:5] Waining for Slim Reader/Writer (SRW) Lock
7 [25/3/2015 19:9:5] Get SRW Lock
8 [25/3/2015 19:9:5] Reader 1 read msg "writer_id 0, msg with num = 1"
9 [25/3/2015 19:9:5] Release SRW Lock
10 [25/3/2015 19:9:5] Waining for Slim Reader/Writer (SRW) Lock
11 [25/3/2015 19:9:5] Get SRW Lock
12 [25/3/2015 19:9:5] Reader 1 read msg "writer_id 0, msg with num = 2"
13 [25/3/2015 19:9:5] Release SRW Lock
14 [25/3/2015 19:9:5] Waining for Slim Reader/Writer (SRW) Lock
15 [25/3/2015 19:9:5] Get SRW Lock
16 [25/3/2015 19:9:5] Reader 1 read msg "writer_id 0, msg with num = 3"
17 [25/3/2015 19:9:5] Release SRW Lock
18 [25/3/2015 19:9:5] Waining for Slim Reader/Writer (SRW) Lock
19 [25/3/2015 19:9:5] Get SRW Lock
20 [25/3/2015 19:9:5] Reader 1 read msg "writer_id 0, msg with num = 4"
21 [25/3/2015 19:9:5] Release SRW Lock
22 [25/3/2015 19:9:5] Waining for Slim Reader/Writer (SRW) Lock
23 [25/3/2015 19:9:6] Get SRW Lock
24 [25/3/2015 19:9:6] Reader 1 read msg "writer_id 0, msg with num = 5"
25 [25/3/2015 19:9:6] Release SRW Lock
26 [25/3/2015 19:9:6] Waining for Slim Reader/Writer (SRW) Lock
27 [25/3/2015 19:9:6] Get SRW Lock
28 [25/3/2015 19:9:6] Reader 1 read msg "writer_id 0, msg with num = 6"
29 [25/3/2015 19:9:6] Release SRW Lock
30 [25/3/2015 19:9:6] Waining for Slim Reader/Writer (SRW) Lock
31 [25/3/2015 19:9:6] Get SRW Lock
32 [25/3/2015 19:9:6] Reader 1 read msg "writer_id 0, msg with num = 7"
33 [25/3/2015 19:9:6] Release SRW Lock

```

```
34 [25/3/2015 19:9:6] Waining for Slim Reader/Writer (SRW) Lock
35 [25/3/2015 19:9:6] Get SRW Lock
36 [25/3/2015 19:9:6] Reader 1 read msg "writer_id 0, msg with num = 8"
37 [25/3/2015 19:9:6] Release SRW Lock
38 [25/3/2015 19:9:6] Waining for Slim Reader/Writer (SRW) Lock
39 [25/3/2015 19:9:6] Get SRW Lock
40 [25/3/2015 19:9:6] Reader 1 read msg "writer_id 0, msg with num = 9"
41 [25/3/2015 19:9:6] Release SRW Lock
42 [25/3/2015 19:9:6] Waining for Slim Reader/Writer (SRW) Lock
43 [25/3/2015 19:9:7] Get SRW Lock
44 [25/3/2015 19:9:7] Reader 1 read msg "writer_id 0, msg with num = 10"
45 [25/3/2015 19:9:7] Release SRW Lock
46 [25/3/2015 19:9:7] Waining for Slim Reader/Writer (SRW) Lock
47 [25/3/2015 19:9:7] Get SRW Lock
48 [25/3/2015 19:9:7] Reader 1 read msg "writer_id 0, msg with num = 11"
49 [25/3/2015 19:9:7] Release SRW Lock
50 [25/3/2015 19:9:7] Waining for Slim Reader/Writer (SRW) Lock
51 [25/3/2015 19:9:7] Get SRW Lock
52 [25/3/2015 19:9:7] Reader 1 read msg "writer_id 0, msg with num = 12"
53 [25/3/2015 19:9:7] Release SRW Lock
54 [25/3/2015 19:9:7] Waining for Slim Reader/Writer (SRW) Lock
55 [25/3/2015 19:9:7] Get SRW Lock
56 [25/3/2015 19:9:7] Reader 1 read msg "writer_id 0, msg with num = 13"
57 [25/3/2015 19:9:7] Release SRW Lock
58 [25/3/2015 19:9:7] Waining for Slim Reader/Writer (SRW) Lock
59 [25/3/2015 19:9:7] Get SRW Lock
60 [25/3/2015 19:9:7] Reader 1 read msg "writer_id 0, msg with num = 14"
61 [25/3/2015 19:9:7] Release SRW Lock
62 [25/3/2015 19:9:7] Waining for Slim Reader/Writer (SRW) Lock
```

4 Клиент-серверное приложение для полной задачи читателя-писателя

В данной задаче появляется уже несколько писателей, а модель работы становится клиент-серверной[1]. В качестве IPC был выбран именованный канал, а для синхронизации использованы сразу несколько инструментов: читатели ожидают на условной переменной, писатели между собой делят время при помощи критической секции, разделяемая память защищена при помощи SRW-замка.

В данной реализации (листинг 39) писатель сосредоточен в серверном модуле (всего два писателя), а читатель (листинг 40) разнесён между серверной и клиентской частью: на сервере есть клиентский поток, который читает информацию, и отправляет её в именованный канал, где она может быть получена клиентской половиной читателя (число читателей зависит от количества запущенных клиентов и не известно заранее).

Листинг 39: Сервер-писатель (src/SynchronizationPrimitives/ReaderWriterServer/main.cpp)

```
1 #include <windows.h>
2 #include <stdio.h>
3 #include <conio.h>
4 #include <tchar.h>
5 #include <strsafe.h>
6 #include "Logger.h"
7
8 #define BUFSIZE 512
9
10 DWORD WINAPI InstanceThread(LPVOID);
11 HANDLE CreateAndStartWaitableTimer(int);
12 DWORD WINAPI ThreadTimeManagerHandler(LPVOID);
13 DWORD WINAPI ThreadWriter(LPVOID);
14
15 //Init log
16 Logger log(_T("ReaderWriterServer"), -1);
17 // инструмент синхронизации:
18 SRWLOCK lock;
19 CONDITION_VARIABLE condread;
20 CRITICAL_SECTION crs; // Объявление критической секции
21 int message; // сообщение
22 bool isDone = false; //флаг завершения
23
24 int _tmain(int argc, _TCHAR* argv[]) {
25     log.loudlog(_T("Server is started.\n\n"));
26 }
```

```

27  BOOL    fConnected = FALSE; // Флаг наличия подключенных клиентов
28  DWORD   dwThreadId = 0; // Номер обслуживающего потока
29  HANDLE  hPipe = INVALID_HANDLE_VALUE; // Идентификатор канала
30  HANDLE  hThread = NULL; // Идентификатор обслуживающего потока
31  HANDLE  service[3]; // Идентификатор потока писателя и таймера
32  LPTSTR  lpszPipename = _T("\\\\.\\pipe\\$$MyPipe$$"); // Имя создаваемого к
    анала
33
34  // начальное сообщение
35  message = 0;
36  InitializeSRWLock(&lock);
37  InitializeConditionVariable(&condread);
38
39  // Создание потока-таймера
40  log.loudlog(_T("Time Manager creation!"));
41  service[0] = CreateThread(
42      NULL,          // дескриптор защиты
43      0,             // начальный размер стека
44      ThreadTimeManagerHandler, // функция потока
45      (LPVOID)5,      // параметр потока (5 секунд)
46      NULL,          // опции создания
47      NULL);         // номер потока
48
49  // Создание потоков-писателей
50  log.loudlog(_T("Writers creation!"));
51  service[1] = CreateThread(
52      NULL,          // дескриптор защиты
53      0,             // начальный размер стека
54      ThreadWriter,   // функция потока
55      NULL,          // параметр потока
56      NULL,          // опции создания
57      NULL);         // номер потока
58
59  service[2] = CreateThread(
60      NULL,          // дескриптор защиты
61      0,             // начальный размер стека
62      ThreadWriter,   // функция потока
63      NULL,          // параметр потока
64      NULL,          // опции создания
65      NULL);         // номер потока
66
67  //инициализируем средство синхронизации
68  InitializeCriticalSection(&crs);
69
70  // Ожидаем соединения со стороны клиента

```

```

71 log.loudlog(_T("Waiting for connect..."));
72 // Цикл ожидает клиентов и создаёт для них потоки обработки
73 while (isDone != true) {
74     // Создаем канал:
75     log.loudlog(_T("Try to create named pipe on %s"), lpzPipeName);
76     if ((hPipe = CreateNamedPipe(
77         lpzPipeName,    // имя канала,
78         PIPE_ACCESS_DUPLEX, // режим открытия канала - двунаправленный,
79         PIPE_TYPE_MESSAGE | // данные записываются в канал в виде потока сообщ
            ений,
80         PIPE_WAIT,      // функции передачи и приема блокируются до их оконча
            ния,
81         PIPE_UNLIMITED_INSTANCES, // максимальное число экземпляров каналов не
            ограничено,
82         BUFSIZE * sizeof(_TCHAR), //размеры выходного и входного буферов канала
            ,
83         BUFSIZE * sizeof(_TCHAR),
84         5000,          // 5 секунд - длительность для функции WaitNamedPipe,
85         NULL))        // дескриптор безопасности по умолчанию.
86         == INVALID_HANDLE_VALUE) {
87         log.loudlog(_T("CreateNamedPipe failed, GLE=%d."), GetLastError());
88         exit(1);
89     }
90     log.loudlog(_T("Named pipe created successfully!"));
91
92     // Если произошло соединение
93     if (ConnectNamedPipe(hPipe, NULL)) {
94         log.loudlog(_T("Client connected!"));
95
96         // Создаём поток для обслуживания клиента
97         hThread = CreateThread(
98             NULL,          // дескриптор защиты
99             0,             // начальный размер стека
100            InstanceThread, // функция потока
101            (LPVOID)hPipe,  // параметр потока
102            0,             // опции создания
103            &dwThreadId);   // номер потока
104
105         // Если поток создать не удалось - сообщаем об ошибке
106         if (hThread == NULL) {
107             double errorcode = GetLastError();
108             log.loudlog(_T("CreateThread failed, GLE=%d."), errorcode);
109             exit(1);
110         }
111         else CloseHandle(hThread);

```

```

112     }
113     else {
114         // Если клиенту не удалось подключиться, закрываем канал
115         CloseHandle(hPipe);
116         log.loudlog(_T("There are not connecrtion requests.));
117     }
118 }
119
120 //ожидаем завершения всех потоков
121 WaitForMultipleObjects(3, service, TRUE, INFINITE);
122
123 //удаляем объект синхронизации
124 DeleteCriticalSection(&crs);
125
126 //закрываем handle потоков
127 for (int i = 0; i != 3; ++i)
128     CloseHandle(service[i]);
129
130 // Завершение работы
131 log.loudlog(_T("All tasks are done!));
132 _getch();
133 exit(0);
134 }
135
136 DWORD WINAPI InstanceThread(LPVOID lpvParam) {
137     log.loudlog(_T("Thread %d started!"), GetCurrentThreadId());
138     HANDLE hPipe = (HANDLE)lpvParam; // Идентификатор канала
139     HANDLE hHeap = GetProcessHeap(); // локальная куча
140     // Буфер для хранения передаваемого сообщения
141     _TCHAR* chBuf = (_TCHAR*)HeapAlloc(hHeap, 0, BUFSIZE * sizeof(TCHAR));
142     DWORD writebytes; // Число байт прочитанных и переданных
143
144     while (isDone != true) {
145         // Захват объекта синхронизации (совместный доступ!)
146         log.quietlog(_T("Waining for Slim Reader/Writer (SRW) Lock));
147         AcquireSRWLockShared(&lock);
148         SleepConditionVariableSRW(&condread, &lock, INFINITE,
149             CONDITION_VARIABLE_LOCKMODE_SHARED);
150         log.quietlog(_T("Get SRW Lock));
151
152         // Пошлaем эту команду клиентскому приложению
153         swprintf_s(chBuf, BUFSIZE, L"%i", message);
154         if (WriteFile(hPipe, chBuf, (lstrlen(chBuf) + 1)*sizeof(_TCHAR), &
155             writebytes, NULL)) {
156             // Выводим сообщение на консоль

```

```

155     log.quietlog(_T("Client %d: get msg: %s"), GetCurrentThreadId(), chBuf
156         );
157 }
158 else {
159     log.loudlog(_T("Thread %d: WriteFile: Error %ld"), GetCurrentThreadId
160         (), GetLastError());
161     break;
162 }
163
164 //освобождение объекта синхронизации
165 log.quietlog(_T("Release SRW Lock"));
166 ReleaseSRWLockShared(&lock);
167
168 //задержка
169 Sleep(100);
170 }
171
172 // завершаем работу приложения
173 StringCchCopy(chBuf, BUFSIZE, L"exit");
174 WriteFile(hPipe, chBuf, (lstrlen(chBuf) + 1)*sizeof(_TCHAR), &writebytes,
175     NULL);
176
177 // Освобождение ресурсов
178 FlushFileBuffers(hPipe);
179 DisconnectNamedPipe(hPipe);
180 CloseHandle(hPipe);
181
182 HeapFree(hHeap, 0, chBuf);
183
184 log.quietlog(_T("Thread %d: InstanceThread exiting."), GetCurrentThreadId
185     ());
186 return 0;
187 }
188
189 DWORD WINAPI ThreadWriter(LPVOID lpvParam) {
190     log.loudlog(_T("Writer thread %d started!"), GetCurrentThreadId());
191
192     while (isDone != true) {
193         EnterCriticalSection(&crs);
194         // Захват объекта синхронизации (монопольный доступ!)
195         log.quietlog(_T("Writer: Waining for Slim Reader/Writer (SRW) Lock"));
196         AcquireSRWLockExclusive(&lock);
197         log.quietlog(_T("Writer: Get SRW Lock"));
198
199         // меняем значение сообщения

```

```

196     ++message;
197     log.loudlog(_T("Server %d: send msg: %d"), GetCurrentThreadId(), message
        );
198
199     //освобождение объекта синхронизации
200     log.quietlog(_T("Writer: Release SRW Lock"));
201     WakeAllConditionVariable(&condread);
202     ReleaseSRWLockExclusive(&lock);
203
204     //задержка
205     Sleep(200);
206     LeaveCriticalSection(&crs);
207 }
208
209 log.quietlog(_T("Thread %d: Writer Thread exiting."), GetCurrentThreadId
    ());
210 return 0;
211 }
212
213 //создание, установка и запуск таймера
214 HANDLE CreateAndStartWaitableTimer(int sec) {
215     __int64 end_time;
216     LARGE_INTEGER end_time2;
217     HANDLE tm = CreateWaitableTimer(NULL, false, _T("Timer!"));
218     end_time = -1 * sec * 100000000;
219     end_time2.LowPart = (DWORD)(end_time & 0xFFFFFFFF);
220     end_time2.HighPart = (LONG)(end_time >> 32);
221     SetWaitableTimer(tm, &end_time2, 0, NULL, NULL, false);
222     return tm;
223 }
224
225 DWORD WINAPI ThreadTimeManagerHandler(LPVOID prm) {
226     int ttl = (int)prm;
227     if (ttl < 0) {
228         //завершение по команде оператора
229         _TCHAR buf[100];
230         while (1) {
231             fgetws(buf, sizeof(buf), stdin);
232             if (buf[0] == _T('s')) {
233                 log.quietlog(_T("'s' signal received, set Event exitEvent"));
234                 isDone = true;
235                 break;
236             }
237         }
238     }

```



```

239     else {
240         //завершение по таймеру
241         HANDLE h = CreateAndStartWaitableTimer(ttl);
242         WaitForSingleObject(h, INFINITE);
243         log.quietlog(_T("Timer signal received, set Event exitEvent"));
244         isDone = true;
245         CloseHandle(h);
246     }
247     log.loudlog(_T("TimeManager finishing work"));
248     return 0;
249 }

```

Листинг 40: Клиент-читатель (src/SynchronizationPrimitives/ReaderWriterClient/main.cpp)

```

1  #include <windows.h>
2  #include <stdio.h>
3  #include <conio.h>
4  #include <tchar.h>
5  #include <strsafe.h>
6  #include "Logger.h"
7
8  #define BUFSIZE 512
9  //Init log
10 Logger log(_T("ReaderWriterClient"), GetCurrentProcessId());
11
12 int _tmain(int argc, _TCHAR* argv[]) {
13     log.loudlog(_T("Client is started!\n\n"));
14
15     HANDLE hPipe = INVALID_HANDLE_VALUE; // Идентификатор канала
16     LPTSTR lpszPipename = _T("\\\\.\\pipe\\$$MyPipe$$"); // Имя создаваемого к
        анала Pipe
17     _TCHAR chBuf[BUFSIZE]; // Буфер для передачи данных через канал
18     DWORD readbytes; // Число байт прочитанных и переданных
19
20     log.loudlog(_T("Try to use WaitNamedPipe..."));
21
22     // Пытаемся открыть именованный канал, если надо - ожидаем его освобождени
        я
23     while (1) {
24         // Создаем канал с процессом-клиентом:
25         hPipe = CreateFile(
26             lpszPipename,    // имя канала,
27             GENERIC_READ,    // текущий клиент имеет доступ на чтение,
28             0,               // тип доступа,
29             NULL,            // атрибуты защиты,

```

```

30     OPEN_EXISTING,      // открывается существующий файл,
31     0,                  // атрибуты и флаги для файла,
32     NULL);              // доступа к файлу шаблона.
33
34     // Продолжаем работу, если канал создать удалось
35     if (hPipe != INVALID_HANDLE_VALUE)
36         break;
37
38     // Выход, если ошибка связана не с занятым каналом.
39     double errorcode = GetLastError();
40     if (errorcode != ERROR_PIPE_BUSY) {
41         log.loudlog(_T("Could not open pipe. GLE=%d\n"), errorcode);
42         exit(1);
43     }
44
45     // Если все каналы заняты, ждём 20 секунд
46     if (!WaitNamedPipe(lpszPipename, 20000)) {
47         log.loudlog(_T("Could not open pipe: 20 second wait timed out.));
48         exit(2);
49     }
50 }
51
52 // Выводим сообщение о создании канала
53 log.loudlog(_T("Successfully connected!"));
54 // Цикл обмена данными с серверным процессом
55 while (1) {
56     // Получаем команду от сервера
57     if (ReadFile(hPipe, chBuf, BUFSIZE * sizeof(TCHAR), &readbytes, NULL)) {
58         log.loudlog(_T("Received from server: %s"), chBuf);
59     } else {
60         // Если произошла ошибка, выводим ее код и завершаем работу приложения
61         double errorcode = GetLastError();
62         log.loudlog(_T("ReadFile: Error %ld\n"), errorcode);
63         _getch();
64         break;
65     }
66     // В ответ на команду "exit" завершаем цикл обмена данными с серверным п
67     // роцессом
68     if (!_tcsncmp(chBuf, L"exit", 4)) {
69         log.loudlog(_T("Processing exit code"));
70         break;
71     }
72 }
73 // Закрываем идентификатор канала
74 CloseHandle(hPipe);

```

```

74
75 // Завершение работы
76 log.loudlog(_T("All tasks are done!"));
77 _getch();
78 exit(0);
79 return 0;
80 }

```

Сервер создаёт именованный канал, и начинает писать; клиент подцепляется к каналу, и начинает читать (рисунок 12).

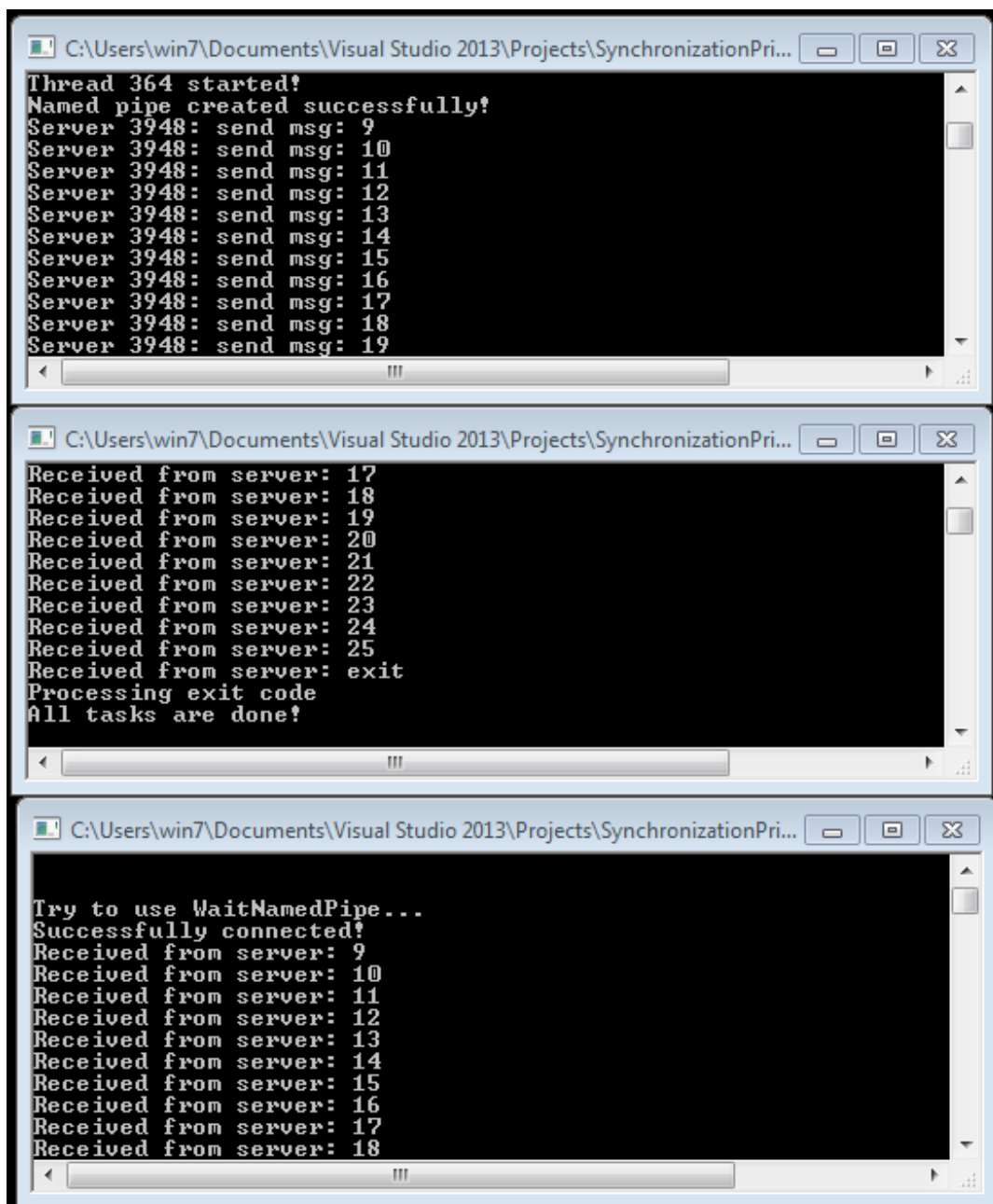


Рис. 12: Клиент-серверное приложение

Листинг 41 показывает лог работы сервера, а листинг 42 - одного из читателей. Стоит заметить, что оба потока-писателя (3968 и 3948) пишут в свой лог в 1 файл, из-за этого в некоторых местах возникает путаница в записях. В таких случаях стоит использовать разные файлы либо использовать примитивы синхронизации для ограничения доступа к логу. Читатель тоже получает не всю информацию. Её объём зависит от времени подключения, т.е. доступ читателей к информации разграничен по времени.

Листинг 41: Протокол работы сервера с двумя писателями

```
1 [25/3/2015 19:49:32] ReaderWriterServer is starting.
2 [25/3/2015 19:49:32] Server is started.
3 [25/3/2015 19:49:32] Time Manager creation!
4 [25/3/2015 19:49:32] Writers creation!
5 [25/3/2015 19:49:32] Waiting for connect...
6 [25/3/2015 19:49:32] Try to create named pipe on \\.\pipe\$$$MyPipe$$
   [25/3/2015 19:49:32] Writer thread 3968 started![25/3/2015 19:49:32]
   Writer thread 3948 started!
7 [25/3/2015 19:49:32] Named pipe created successfully!
8 [25/3/2015 19:49:32] Writer: Waining for Slim Reader/Writer (SRW) Lock
9 [25/3/2015 19:49:32] Writer: Get SRW Lock
10 [25/3/2015 19:49:32] Server 3948: send msg: 1
11 [25/3/2015 19:49:32] Writer: Release SRW Lock
12 [25/3/2015 19:49:32] Writer: Waining for Slim Reader/Writer (SRW) Lock
13 [25/3/2015 19:49:32] Writer: Get SRW Lock
14 [25/3/2015 19:49:32] Server 3948: send msg: 2
15 [25/3/2015 19:49:32] Writer: Release SRW Lock
16 [25/3/2015 19:49:33] Writer: Waining for Slim Reader/Writer (SRW) Lock
17 [25/3/2015 19:49:33] Writer: Get SRW Lock
18 [25/3/2015 19:49:33] Server 3948: send msg: 3
19 [25/3/2015 19:49:33] Writer: Release SRW Lock
20 [25/3/2015 19:49:33] Writer: Waining for Slim Reader/Writer (SRW) Lock
21 [25/3/2015 19:49:33] Writer: Get SRW Lock
22 [25/3/2015 19:49:33] Server 3948: send msg: 4
23 [25/3/2015 19:49:33] Writer: Release SRW Lock
24 [25/3/2015 19:49:33] Client connected!
25 [25/3/2015 19:49:33] Try to create named pipe on \\.\pipe\$$$MyPipe$$
   [25/3/2015 19:49:33] Thread 1568 started!
26 [25/3/2015 19:49:33] Named pipe created successfully!
27 [25/3/2015 19:49:33] Waining for Slim Reader/Writer (SRW) Lock
28 [25/3/2015 19:49:33] Writer: Waining for Slim Reader/Writer (SRW) Lock
29 [25/3/2015 19:49:33] Writer: Get SRW Lock
30 [25/3/2015 19:49:33] Server 3948: send msg: 5
31 [25/3/2015 19:49:33] Writer: Release SRW Lock
32 [25/3/2015 19:49:33] Get SRW Lock
33 [25/3/2015 19:49:33] Client 1568: get msg: 5
```

```

34 [25/3/2015 19:49:33] Release SRW Lock
35 [25/3/2015 19:49:33] Waining for Slim Reader/Writer (SRW) Lock
36 [25/3/2015 19:49:33] Writer: Waining for Slim Reader/Writer (SRW) Lock
37 [25/3/2015 19:49:33] Writer: Get SRW Lock
38 [25/3/2015 19:49:33] Server 3948: send msg: 6
39 [25/3/2015 19:49:33] Writer: Release SRW Lock
40 [25/3/2015 19:49:33] Get SRW Lock
41 [25/3/2015 19:49:33] Client 1568: get msg: 6
42 [25/3/2015 19:49:33] Release SRW Lock
43 [25/3/2015 19:49:33] Waining for Slim Reader/Writer (SRW) Lock
44 [25/3/2015 19:49:33] Writer: Waining for Slim Reader/Writer (SRW) Lock
45 [25/3/2015 19:49:33] Writer: Get SRW Lock
46 [25/3/2015 19:49:33] Server 3948: send msg: 7
47 [25/3/2015 19:49:33] Writer: Release SRW Lock
48 [25/3/2015 19:49:33] Get SRW Lock
49 [25/3/2015 19:49:33] Client 1568: get msg: 7
50 [25/3/2015 19:49:33] Release SRW Lock
51 [25/3/2015 19:49:33] Waining for Slim Reader/Writer (SRW) Lock
52 [25/3/2015 19:49:34] Writer: Waining for Slim Reader/Writer (SRW) Lock
53 [25/3/2015 19:49:34] Writer: Get SRW Lock
54 [25/3/2015 19:49:34] Server 3948: send msg: 8
55 [25/3/2015 19:49:34] Writer: Release SRW Lock
56 [25/3/2015 19:49:34] Get SRW Lock
57 [25/3/2015 19:49:34] Client 1568: get msg: 8
58 [25/3/2015 19:49:34] Release SRW Lock
59 [25/3/2015 19:49:34] Client connected!
60 [25/3/2015 19:49:34] Try to create named pipe on \\.\pipe\$$$MyPipe$$
    [25/3/2015 19:49:34] Thread 364 started!
61 [25/3/2015 19:49:34] Named pipe created successfully!
62 [25/3/2015 19:49:34] Waining for Slim Reader/Writer (SRW) Lock
63 [25/3/2015 19:49:34] Waining for Slim Reader/Writer (SRW) Lock
64 [25/3/2015 19:49:34] Writer: Waining for Slim Reader/Writer (SRW) Lock
65 [25/3/2015 19:49:34] Writer: Get SRW Lock
66 [25/3/2015 19:49:34] Server 3948: send msg: 9
67 [25/3/2015 19:49:34] Writer: Release SRW Lock
68 [25/3/2015 19:49:34] Get SRW Lock
69 [25/3/2015 19:49:34] Get SRW Lock
70 [25/3/2015 19:49:34] Client 364: get msg: 9
71 [25/3/2015 19:49:34] Release SRW Lock
72 [25/3/2015 19:49:34] Client 1568: get msg: 9
73 [25/3/2015 19:49:34] Release SRW Lock
74 [25/3/2015 19:49:34] Waining for Slim Reader/Writer (SRW) Lock
75 [25/3/2015 19:49:34] Waining for Slim Reader/Writer (SRW) Lock
76 [25/3/2015 19:49:34] Writer: Waining for Slim Reader/Writer (SRW) Lock
77 [25/3/2015 19:49:34] Writer: Get SRW Lock

```

```

78 [25/3/2015 19:49:34] Server 3948: send msg: 10
79 [25/3/2015 19:49:34] Writer: Release SRW Lock
80 [25/3/2015 19:49:34] [25/3/2015 19:49:34] Get SRW LockGet SRW Lock
81 [25/3/2015 19:49:34] Client 1568: get msg: 10
82 [25/3/2015 19:49:34] Release SRW Lock
83 [25/3/2015 19:49:34] Client 364: get msg: 10
84 [25/3/2015 19:49:34] Release SRW Lock
85 [25/3/2015 19:49:34] Waining for Slim Reader/Writer (SRW) Lock
86 [25/3/2015 19:49:34] Waining for Slim Reader/Writer (SRW) Lock
87 [25/3/2015 19:49:34] Writer: Waining for Slim Reader/Writer (SRW) Lock
88 [25/3/2015 19:49:34] Writer: Get SRW Lock
89 [25/3/2015 19:49:34] Server 3948: send msg: 11
90 [25/3/2015 19:49:34] Writer: Release SRW Lock
91 [25/3/2015 19:49:34] Get SRW Lock
92 [25/3/2015 19:49:34] Get SRW Lock
93 [25/3/2015 19:49:34] Client 364: get msg: 11
94 [25/3/2015 19:49:34] Release SRW Lock
95 [25/3/2015 19:49:34] Client 1568: get msg: 11
96 [25/3/2015 19:49:34] Release SRW Lock
97 [25/3/2015 19:49:34] Waining for Slim Reader/Writer (SRW) Lock
98 [25/3/2015 19:49:34] Waining for Slim Reader/Writer (SRW) Lock
99 [25/3/2015 19:49:34] Writer: Waining for Slim Reader/Writer (SRW) Lock
100 [25/3/2015 19:49:34] Writer: Get SRW Lock
101 [25/3/2015 19:49:34] Server 3948: send msg: 12
102 [25/3/2015 19:49:34] Writer: Release SRW Lock
103 [25/3/2015 19:49:34] Get SRW Lock[25/3/2015 19:49:34]
104 Get SRW Lock
105 [25/3/2015 19:49:34] Client 364: get msg: 12
106 [25/3/2015 19:49:34] Release SRW Lock[25/3/2015 19:49:34]
107 Client 1568: get msg: 12
108 [25/3/2015 19:49:34] Release SRW Lock
109 [25/3/2015 19:49:34] Waining for Slim Reader/Writer (SRW) Lock
110 [25/3/2015 19:49:34] Waining for Slim Reader/Writer (SRW) Lock
111 [25/3/2015 19:49:35] Writer: Waining for Slim Reader/Writer (SRW) Lock
112 [25/3/2015 19:49:35] Writer: Get SRW Lock
113 [25/3/2015 19:49:35] Server 3948: send msg: 13
114 [25/3/2015 19:49:35] Writer: Release SRW Lock
115 [25/3/2015 19:49:35] Get SRW Lock
116 [25/3/2015 19:49:35] Get SRW Lock
117 [25/3/2015 19:49:35] Client 1568: get msg: 13
118 [25/3/2015 19:49:35] Release SRW Lock
119 [25/3/2015 19:49:35] Client 364: get msg: 13
120 [25/3/2015 19:49:35] Release SRW Lock
121 [25/3/2015 19:49:35] Waining for Slim Reader/Writer (SRW) Lock
122 [25/3/2015 19:49:35] Waining for Slim Reader/Writer (SRW) Lock

```

123 [25/3/2015 19:49:35] Writer: Waining for Slim Reader/Writer (SRW) Lock
124 [25/3/2015 19:49:35] Writer: Get SRW Lock
125 [25/3/2015 19:49:35] Server 3948: send msg: 14
126 [25/3/2015 19:49:35] Writer: Release SRW Lock
127 [25/3/2015 19:49:35] Get SRW Lock
128 [25/3/2015 19:49:35] Client 364: get msg: 14
129 [25/3/2015 19:49:35] Release SRW Lock
130 [25/3/2015 19:49:35] Get SRW Lock
131 [25/3/2015 19:49:35] Client 1568: get msg: 14
132 [25/3/2015 19:49:35] Release SRW Lock
133 [25/3/2015 19:49:35] Waining for Slim Reader/Writer (SRW) Lock
134 [25/3/2015 19:49:35] Waining for Slim Reader/Writer (SRW) Lock
135 [25/3/2015 19:49:35] Writer: Waining for Slim Reader/Writer (SRW) Lock
136 [25/3/2015 19:49:35] Writer: Get SRW Lock
137 [25/3/2015 19:49:35] Server 3948: send msg: 15
138 [25/3/2015 19:49:35] Writer: Release SRW Lock
139 [25/3/2015 19:49:35] Get SRW Lock
140 [25/3/2015 19:49:35] Client 1568: get msg: 15
141 [25/3/2015 19:49:35] Release SRW Lock
142 [25/3/2015 19:49:35] Get SRW Lock
143 [25/3/2015 19:49:35] Client 364: get msg: 15
144 [25/3/2015 19:49:35] Release SRW Lock
145 [25/3/2015 19:49:35] Waining for Slim Reader/Writer (SRW) Lock
146 [25/3/2015 19:49:35] Waining for Slim Reader/Writer (SRW) Lock
147 [25/3/2015 19:49:35] Writer: Waining for Slim Reader/Writer (SRW) Lock
148 [25/3/2015 19:49:35] Writer: Get SRW Lock
149 [25/3/2015 19:49:35] Server 3948: send msg: 16
150 [25/3/2015 19:49:35] Writer: Release SRW Lock
151 [25/3/2015 19:49:35] Get SRW Lock
152 [25/3/2015 19:49:35] Client 364: get msg: 16
153 [25/3/2015 19:49:35] Release SRW Lock
154 [25/3/2015 19:49:35] Get SRW Lock
155 [25/3/2015 19:49:35] Client 1568: get msg: 16
156 [25/3/2015 19:49:35] Release SRW Lock
157 [25/3/2015 19:49:35] Waining for Slim Reader/Writer (SRW) Lock
158 [25/3/2015 19:49:35] Waining for Slim Reader/Writer (SRW) Lock
159 [25/3/2015 19:49:35] Writer: Waining for Slim Reader/Writer (SRW) Lock
160 [25/3/2015 19:49:35] Writer: Get SRW Lock
161 [25/3/2015 19:49:35] Server 3948: send msg: 17
162 [25/3/2015 19:49:35] Writer: Release SRW Lock
163 [25/3/2015 19:49:35] Get SRW Lock
164 [25/3/2015 19:49:35] Get SRW Lock
165 [25/3/2015 19:49:35] Client 1568: get msg: 17
166 [25/3/2015 19:49:35] Release SRW Lock
167 [25/3/2015 19:49:35] Client 364: get msg: 17

168 [25/3/2015 19:49:35] Release SRW Lock
 169 [25/3/2015 19:49:36] Waining for Slim Reader/Writer (SRW) Lock [25/3/2015
 19:49:36]
 170 Waining for Slim Reader/Writer (SRW) Lock
 171 [25/3/2015 19:49:36] Writer: Waining for Slim Reader/Writer (SRW) Lock
 172 [25/3/2015 19:49:36] Writer: Get SRW Lock
 173 [25/3/2015 19:49:36] Server 3948: send msg: 18
 174 [25/3/2015 19:49:36] Writer: Release SRW Lock
 175 [25/3/2015 19:49:36] Get SRW Lock
 176 [25/3/2015 19:49:36] [25/3/2015 19:49:36] Get SRW Lock Client 1568: get msg:
 18
 177 [25/3/2015 19:49:36] Release SRW Lock
 178 [25/3/2015 19:49:36] Client 364: get msg: 18
 179 [25/3/2015 19:49:36] Release SRW Lock
 180 [25/3/2015 19:49:36] Waining for Slim Reader/Writer (SRW) Lock
 181 [25/3/2015 19:49:36] Waining for Slim Reader/Writer (SRW) Lock
 182 [25/3/2015 19:49:36] Writer: Waining for Slim Reader/Writer (SRW) Lock
 183 [25/3/2015 19:49:36] Writer: Get SRW Lock
 184 [25/3/2015 19:49:36] Server 3948: send msg: 19
 185 [25/3/2015 19:49:36] Writer: Release SRW Lock
 186 [25/3/2015 19:49:36] Get SRW Lock
 187 [25/3/2015 19:49:36] Client 1568: get msg: 19
 188 [25/3/2015 19:49:36] Release SRW Lock
 189 [25/3/2015 19:49:36] Get SRW Lock
 190 [25/3/2015 19:49:36] Client 364: get msg: 19
 191 [25/3/2015 19:49:36] Release SRW Lock
 192 [25/3/2015 19:49:36] [25/3/2015 19:49:36] Waining for Slim Reader/Writer (SRW) Lock Waining for Slim Reader/Writer (SRW) Lock
 193 [25/3/2015 19:49:36] Writer: Waining for Slim Reader/Writer (SRW) Lock
 194 [25/3/2015 19:49:36] Writer: Get SRW Lock
 195 [25/3/2015 19:49:36] Server 3948: send msg: 20
 196 [25/3/2015 19:49:36] Writer: Release SRW Lock
 197 [25/3/2015 19:49:36] Get SRW Lock
 198 [25/3/2015 19:49:36] Get SRW Lock [25/3/2015 19:49:36]
 199 Client 1568: get msg: 20
 200 [25/3/2015 19:49:36] Release SRW Lock
 201 [25/3/2015 19:49:36] Client 364: get msg: 20
 202 [25/3/2015 19:49:36] Release SRW Lock
 203 [25/3/2015 19:49:36] Waining for Slim Reader/Writer (SRW) Lock
 204 [25/3/2015 19:49:36] Waining for Slim Reader/Writer (SRW) Lock
 205 [25/3/2015 19:49:36] Writer: Waining for Slim Reader/Writer (SRW) Lock
 206 [25/3/2015 19:49:36] Writer: Get SRW Lock
 207 [25/3/2015 19:49:36] Server 3948: send msg: 21
 208 [25/3/2015 19:49:36] Writer: Release SRW Lock
 209 [25/3/2015 19:49:36] Get SRW Lock

210 [25/3/2015 19:49:36] Client 364: get msg: 21
 211 [25/3/2015 19:49:36] Release SRW Lock
 212 [25/3/2015 19:49:36] Get SRW Lock
 213 [25/3/2015 19:49:36] Client 1568: get msg: 21
 214 [25/3/2015 19:49:36] Release SRW Lock
 215 [25/3/2015 19:49:36] Waining for Slim Reader/Writer (SRW) Lock[25/3/2015
 19:49:36]
 216 Waining for Slim Reader/Writer (SRW) Lock
 217 [25/3/2015 19:49:37] Writer: Waining for Slim Reader/Writer (SRW) Lock
 218 [25/3/2015 19:49:37] Writer: Get SRW Lock
 219 [25/3/2015 19:49:37] Server 3948: send msg: 22
 220 [25/3/2015 19:49:37] Writer: Release SRW Lock
 221 [25/3/2015 19:49:37] Get SRW Lock
 222 [25/3/2015 19:49:37] Client 1568: get msg: 22
 223 [25/3/2015 19:49:37] Release SRW Lock
 224 [25/3/2015 19:49:37] Get SRW Lock
 225 [25/3/2015 19:49:37] Client 364: get msg: 22
 226 [25/3/2015 19:49:37] Release SRW Lock
 227 [25/3/2015 19:49:37] Waining for Slim Reader/Writer (SRW) Lock
 228 [25/3/2015 19:49:37] Waining for Slim Reader/Writer (SRW) Lock
 229 [25/3/2015 19:49:37] Writer: Waining for Slim Reader/Writer (SRW) Lock
 230 [25/3/2015 19:49:37] Writer: Get SRW Lock
 231 [25/3/2015 19:49:37] Server 3968: send msg: 23
 232 [25/3/2015 19:49:37] Writer: Release SRW Lock
 233 [25/3/2015 19:49:37] Get SRW Lock
 234 [25/3/2015 19:49:37] [25/3/2015 19:49:37] Get SRW LockClient 1568: get msg:
 23
 235 [25/3/2015 19:49:37] Release SRW Lock
 236 [25/3/2015 19:49:37] Client 364: get msg: 23
 237 [25/3/2015 19:49:37] Release SRW Lock
 238 [25/3/2015 19:49:37] Waining for Slim Reader/Writer (SRW) Lock
 239 [25/3/2015 19:49:37] Waining for Slim Reader/Writer (SRW) Lock
 240 [25/3/2015 19:49:37] Writer: Waining for Slim Reader/Writer (SRW) Lock
 241 [25/3/2015 19:49:37] Writer: Get SRW Lock
 242 [25/3/2015 19:49:37] Server 3968: send msg: 24
 243 [25/3/2015 19:49:37] Writer: Release SRW Lock
 244 [25/3/2015 19:49:37] Get SRW Lock
 245 [25/3/2015 19:49:37] Get SRW Lock
 246 [25/3/2015 19:49:37] Client 364: get msg: 24
 247 [25/3/2015 19:49:37] Release SRW Lock
 248 [25/3/2015 19:49:37] Client 1568: get msg: 24
 249 [25/3/2015 19:49:37] Release SRW Lock
 250 [25/3/2015 19:49:37] Waining for Slim Reader/Writer (SRW) Lock
 251 [25/3/2015 19:49:37] Waining for Slim Reader/Writer (SRW) Lock
 252 [25/3/2015 19:49:37] Writer: Waining for Slim Reader/Writer (SRW) Lock

```

253 [25/3/2015 19:49:37] Writer: Get SRW Lock
254 [25/3/2015 19:49:37] Server 3968: send msg: 25
255 [25/3/2015 19:49:37] Writer: Release SRW Lock
256 [25/3/2015 19:49:37] Get SRW Lock
257 [25/3/2015 19:49:37] [25/3/2015 19:49:37] Get SRW LockClient 1568: get msg:
    25
258 [25/3/2015 19:49:37] Release SRW Lock
259 [25/3/2015 19:49:37] Client 364: get msg: 25
260 [25/3/2015 19:49:37] Release SRW Lock
261 [25/3/2015 19:49:37] Timer signal received, set Event exitEvent
262 [25/3/2015 19:49:37] TimeManager finishing work
263 [25/3/2015 19:49:37] Thread 364: InstanceThread exiting.
264 [25/3/2015 19:49:37] Thread 1568: InstanceThread exiting.
265 [25/3/2015 19:49:37] Thread 3968: Writer Thread exiting.
266 [25/3/2015 19:49:37] Writer: Waining for Slim Reader/Writer (SRW) Lock
267 [25/3/2015 19:49:37] Writer: Get SRW Lock
268 [25/3/2015 19:49:37] Server 3948: send msg: 26
269 [25/3/2015 19:49:37] Writer: Release SRW Lock
270 [25/3/2015 19:49:38] Thread 3948: Writer Thread exiting.
271 [25/3/2015 19:56:7] Shutting down.

```

Листинг 42: Протокол работы клиента-читателя

```

1 [25/3/2015 19:49:34] ReaderWriterClient is starting.
2 [25/3/2015 19:49:34] Client is started!
3 [25/3/2015 19:49:34] Try to use WaitNamedPipe...
4 [25/3/2015 19:49:34] Successfully connected!
5 [25/3/2015 19:49:34] Received from server: 9
6 [25/3/2015 19:49:34] Received from server: 10
7 [25/3/2015 19:49:34] Received from server: 11
8 [25/3/2015 19:49:34] Received from server: 12
9 [25/3/2015 19:49:35] Received from server: 13
10 [25/3/2015 19:49:35] Received from server: 14
11 [25/3/2015 19:49:35] Received from server: 15
12 [25/3/2015 19:49:35] Received from server: 16
13 [25/3/2015 19:49:35] Received from server: 17
14 [25/3/2015 19:49:36] Received from server: 18
15 [25/3/2015 19:49:36] Received from server: 19
16 [25/3/2015 19:49:36] Received from server: 20
17 [25/3/2015 19:49:36] Received from server: 21
18 [25/3/2015 19:49:37] Received from server: 22
19 [25/3/2015 19:49:37] Received from server: 23
20 [25/3/2015 19:49:37] Received from server: 24
21 [25/3/2015 19:49:37] Received from server: 25
22 [25/3/2015 19:49:37] Received from server: exit

```

23	[25/3/2015 19:49:37] Processing exit code
24	[25/3/2015 19:49:37] All tasks are done!
25	[25/3/2015 19:56:4] Shutting down.

5 Сетевая версия задачи читатели-писатели

Задача похожа на предыдущую, но взаимодействие читателей и писателей нужно осуществлять по сети[1]. По сути, можно было взять предыдущую задачу, и использовать её в сетевом варианте (именованные каналы это позволяют), но ниже приведена реализация на сокетах. Эта реализация оказалась на много проще и гораздо лучше масштабируется.

Реализация сервера (с двумя процессами-писателями) показана в листинге 43, реализация клиента - листинг 44.

Листинг 43: Сервер (src/SynchronizationPrimitives/NetReaderWriterServer/main.cpp)

```
1 #define _WINSOCK_DEPRECATED_NO_WARNINGS
2 #include <winsock2.h>
3 #include <list>
4
5 #include "Logger.h"
6
7 #include <time.h>
8 #include <stdio.h>
9 #include <strsafe.h>
10
11 #pragma comment(lib, "Ws2_32.lib")
12
13 struct CLIENT_INFO
14 {
15     SOCKET hClientSocket;          // Сокет
16     struct sockaddr_in clientAddr;
17 };
18
19 _TCHAR szServerIPAddr[] = _T("127.0.0.1"); // server IP
20 int nServerPort = 5050;                  // server port
21
22 std::list<CLIENT_INFO *> clients;       // Все клиенты
23 CRITICAL_SECTION csClients;             // Защита списка
24
25 bool InitWinSock2_0();
26 BOOL WINAPI ClientThread(LPVOID lpData);
27 void sendToAll(_TCHAR *pBuffer);
28
29 //Init log
30 Logger mylog(_T("NetReaderWriterServer"));
31
32 int _tmain(int argc, _TCHAR* argv[]) {
33     if (!InitWinSock2_0()) {
```

```

34     double errorcode = WSAGetLastError();
35     mylog.loudlog(_T("Unable to Initialize Windows Socket environment, GLE=%
        d"), errorcode);
36     exit(1);
37 }
38 mylog.loudlog(_T("Windows Socket environment ready"));
39
40 SOCKET hServerSocket;
41 hServerSocket = socket(
42     AF_INET,          // The address family. AF_INET specifies TCP/IP
43     SOCK_STREAM,      // Protocol type. SOCK_STREAM specified TCP
44     0                 // Protocol Name. Should be 0 for AF_INET address family
45 );
46
47 if (hServerSocket == INVALID_SOCKET) {
48     mylog.loudlog(_T("Unable to create Server socket"));
49     // Cleanup the environment initialized by WSASStartup()
50     WSACleanup();
51     exit(2);
52 }
53 mylog.loudlog(_T("Server socket created"));
54
55 // Create the structure describing various Server parameters
56 struct sockaddr_in serverAddr;
57
58 serverAddr.sin_family = AF_INET;      // The address family. MUST be
        AF_INET
59 size_t    convtd;
60 char *pMBBuffer = new char[20];
61 wcstombs_s(&convtd, pMBBuffer, 20, szServerIPAddr, 20);
62 //serverAddr.sin_addr.s_addr = inet_addr(pMBBuffer);
63 serverAddr.sin_addr.s_addr = INADDR_ANY;
64 delete[] pMBBuffer;
65 serverAddr.sin_port = htons(nServerPort);
66
67 // Bind the Server socket to the address & port
68 if (bind(hServerSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr
        )) == SOCKET_ERROR) {
69     mylog.loudlog(_T("Unable to bind to %s on port %d"), szServerIPAddr,
        nServerPort);
70     // Free the socket and cleanup the environment initialized by WSASStartup
        ()
71     closesocket(hServerSocket);
72     WSACleanup();
73     exit(3);

```

```

74 }
75 mylog.loudlog(_T("Bind completed"));
76
77 // Put the Server socket in listen state so that it can wait for client
   connections
78 if (listen(hServerSocket, SOMAXCONN) == SOCKET_ERROR) {
79     mylog.loudlog(_T("Unable to put server in listen state"));
80     // Free the socket and cleanup the environment initialized by WSStartup
       ()
81     closesocket(hServerSocket);
82     WSACleanup();
83     exit(4);
84 }
85 mylog.loudlog(_T("Ready for connection on %s:%d"), szServerIPAddr,
       nServerPort);
86
87 HANDLE hClientThread[2];
88 DWORD dwThreadId[2];
89 for (int i = 0; i != 2; ++i) {
90     // Start the client thread
91     hClientThread[i] = CreateThread(NULL, 0,
92         (LPTHREAD_START_ROUTINE)ClientThread,
93         0, 0, &dwThreadId[i]);
94     if (hClientThread[i] == NULL) {
95         mylog.loudlog(_T("Unable to create client thread"));
96     }
97     else {
98         CloseHandle(hClientThread);
99     }
100 }
101
102
103 //инициализируем средство синхронизации
104 InitializeCriticalSection(&csClients);
105
106 // Start the infinite loop
107 while (true) {
108     // As the socket is in listen mode there is a connection request pending
       .
109     // Calling accept( ) will succeed and return the socket for the request.
110     CLIENT_INFO* pClientInfo = new CLIENT_INFO;
111
112     int nSize = sizeof(pClientInfo->clientAddr);
113     pClientInfo->hClientSocket = accept(hServerSocket, (struct sockaddr *) &
       pClientInfo->clientAddr, &nSize);

```

```

114     if (pClientInfo->hClientSocket == INVALID_SOCKET) {
115         mylog.loudlog(_T("accept() failed"));
116     }
117     else {
118         wchar_t* sin_addr = new wchar_t[20];
119         size_t   convtd;
120         mbstowcs_s(&convtd, sin_addr, 20, inet_ntoa(pClientInfo->clientAddr.
            sin_addr), 20);
121         mylog.loudlog(_T("Client connected from %s:%d"), sin_addr, pClientInfo
            ->clientAddr.sin_port);
122         delete[] sin_addr;
123
124         EnterCriticalSection(&csClients);
125         clients.push_front(pClientInfo); // Добавить нового клиента в список
126         LeaveCriticalSection(&csClients);
127     }
128 }
129
130 //удаляем объект синхронизации
131 DeleteCriticalSection(&csClients);
132
133 closesocket(hServerSocket);
134 WSACleanup();
135 exit(0);
136 }
137
138 bool InitWinSock2_0() {
139     WSADATA wsaData;
140     WORD wVersion = MAKEWORD(2, 0);
141
142     if (!WSAStartup(wVersion, &wsaData))
143         return true;
144
145     return false;
146 }
147
148 BOOL WINAPI ClientThread(LPVOID lpData) {
149     // Chat loop:
150     while (1) {
151         _TCHAR szBuffer[124];
152         swprintf_s(szBuffer, _T("%d"), GetCurrentThreadId() % rand());
153         mylog.loudlog(_T("Server %d: %s"), GetCurrentThreadId(), szBuffer);
154         sendToAll(szBuffer);
155         Sleep(100);
156     }

```

```

157
158     return 0;
159 }
160
161 void sendToAll(_TCHAR *pBuffer) {
162     // Пока мы обрабатываем список, его ни кто не должен менять!
163     EnterCriticalSection(&csClients);
164     std::list<CLIENT_INFO *>::iterator client;
165     for (client = clients.begin(); client != clients.end(); ++client) {
166         int nLength = (lstrlen(pBuffer) + 1) * sizeof(_TCHAR);
167         int nCntSend = 0;
168
169         while ((nCntSend = send((*client)->hClientSocket, (char *)pBuffer,
170             nLength, 0) != nLength)) {
171             if (nCntSend == -1) {
172                 mylog.loudlog(_T("Error sending the client"));
173                 break;
174             }
175             if (nCntSend == nLength)
176                 break;
177
178             pBuffer += nCntSend;
179             nLength -= nCntSend;
180         }
181     }
182     LeaveCriticalSection(&csClients);
183 }

```

Листинг 44: Клиент (src/SynchronizationPrimitives/NetReaderWriterClient/main.cpp)

```

1 #define _WINSOCK_DEPRECATED_NO_WARNINGS
2 #include <winsock2.h>
3 #include "Logger.h"
4
5 #pragma comment(lib, "Ws2_32.lib")
6
7 _TCHAR szServerIPAddr[20];           // server IP
8 int nServerPort;                     // server port
9
10 bool InitWinSock2_0();
11
12 //Init log
13 Logger mylog(_T("NetReaderWriterClient"), GetCurrentProcessId());
14
15 int _tmain(int argc, _TCHAR* argv[]) {

```



```

16  _tprintf(_T("Enter the server IP Address: "));
17  wscanf_s(_T("%19s"), szServerIPAddr, _countof(szServerIPAddr));
18  _tprintf(_T("Enter the server port number: "));
19  wscanf_s(_T("%i"), &nServerPort);
20
21  if (!InitWinSock2_0()) {
22      double errorcode = WSAGetLastError();
23      mylog.loudlog(_T("Unable to Initialize Windows Socket environment, GLE=%d"), errorcode);
24      exit(1);
25  }
26  mylog.quietlog(_T("Windows Socket environment ready"));
27
28  SOCKET hClientSocket;
29  hClientSocket = socket(
30      AF_INET,          // The address family. AF_INET specifies TCP/IP
31      SOCK_STREAM,      // Protocol type. SOCK_STREAM specified TCP
32      0);               // Protocol Name. Should be 0 for AF_INET address family
33
34  if (hClientSocket == INVALID_SOCKET) {
35      mylog.loudlog(_T("Unable to create socket"));
36      // Cleanup the environment initialized by WSASStartup()
37      WSACleanup();
38      exit(2);
39  }
40  mylog.quietlog(_T("Client socket created"));
41
42  // Create the structure describing various Server parameters
43  struct sockaddr_in serverAddr;
44
45  serverAddr.sin_family = AF_INET;      // The address family. MUST be
    AF_INET
46  size_t  convtd;
47  char *pMBBuffer = new char[20];
48  wcstombs_s(&convtd, pMBBuffer, 20, szServerIPAddr, 20);
49  serverAddr.sin_addr.s_addr = inet_addr(pMBBuffer);
50  delete[] pMBBuffer;
51  serverAddr.sin_port = htons(nServerPort);
52
53  // Connect to the server
54  if (connect(hClientSocket, (struct sockaddr *) &serverAddr, sizeof(
    serverAddr)) < 0) {
55      mylog.loudlog(_T("Unable to connect to %s on port %d"), szServerIPAddr,
        nServerPort);
56      closesocket(hClientSocket);

```

```

57     WSACleanup();
58     exit(3);
59 }
60 mylog.quietlog(_T("Connect"));
61
62 // Main loop:
63 while (1) {
64     _TCHAR szBuffer[1024];
65     int nLength = recv(hClientSocket, (char *)szBuffer, sizeof(szBuffer), 0)
        ;
66
67     if (nLength > 0) {
68         szBuffer[nLength] = '\0';
69         mylog.loudlog(_T("%s"), szBuffer);
70         _tprintf(_T(">> "));
71     }
72 }
73
74 closesocket(hClientSocket);
75 WSACleanup();
76 exit(0);
77 }
78
79 bool InitWinSock2_0() {
80     WSADATA wsaData;
81     WORD wVersion = MAKEWORD(2, 0);
82
83     if (!WSAStartup(wVersion, &wsaData))
84         return true;
85
86     return false;
87 }

```

На рисунке 13 изображена работа приложения с двумя клиентами. На рисунке 14 показано, что работа происходит именно через сокет.

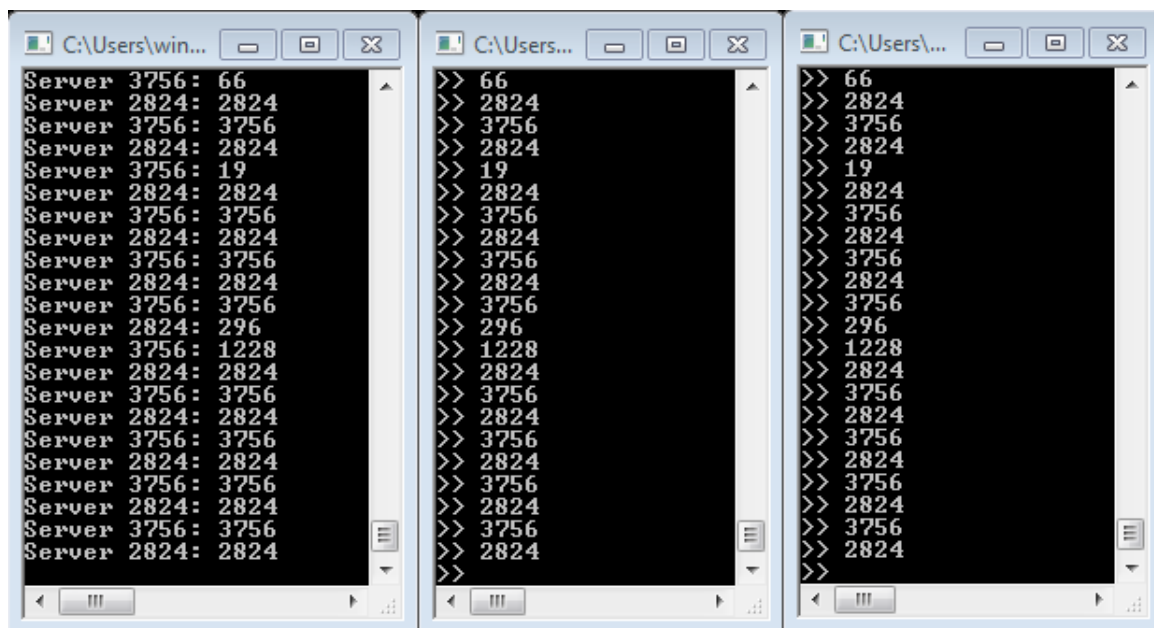


Рис. 13: Сетевая версия задачи читатели-писатели

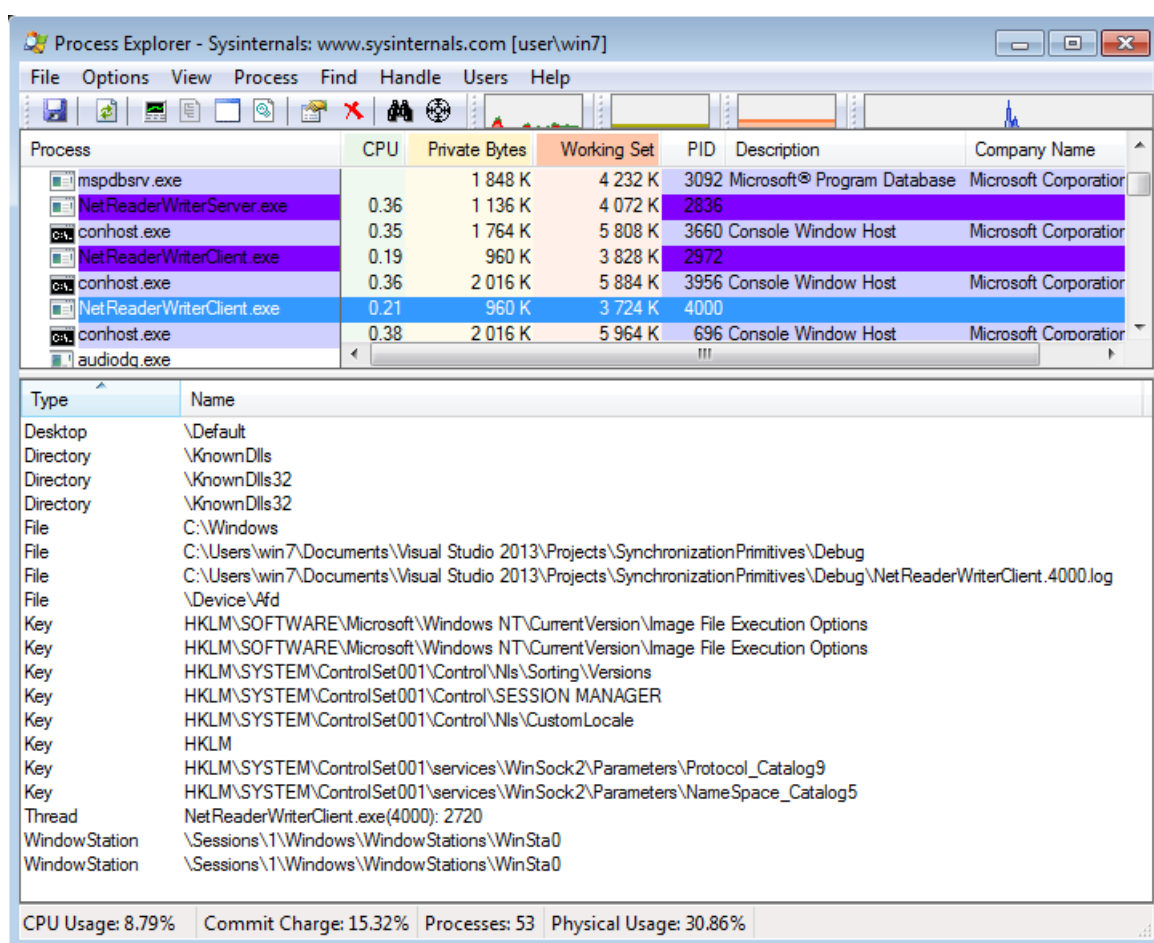


Рис. 14: Запуск process explorer для сетевой версии задачи читатели-писатели

6 Задача производителя-потребителя

Отличие этой задачи в том, что теперь каждый читатель может быть писателем. Фактически, речь идёт о сетевом чате[1]. Это самая интересная задача: каждый процесс тут является и производителем и потребителем, а взаимодействие происходит по сети. Обмен сообщениями устроен через список сокетов - на не больших числах (до 50 подключений) это не сказывается на работе, но при росте числа клиентов производительность начинает снижаться. Для решения этого вопроса можно добавить асинхронную рассылку уведомлений и сбалансировать нагрузку на центральном узле.

Эта идея демонстрируется на рисунке 15: все клиенты попадают в некоторый список, и как только кто-либо из них генерирует событие (отправляет сообщение), сервер проходит по этому списку и пересылает это событие остальным.



Рис. 15: Схема взаимодействия клиентом в сервером.

Реализация сервера представлена в листинге 45, реализация клиента - в листинге 46. При этом сложно выделить производителя и потребителя, т.к. каждый клиент является и производителем и потребителем, который должен получить каждое сообщение.

С точки зрения сервера, каждый клиент описывается структурой `CLIENT_INFO` (стр. 13)содержащей три поля:

- `hClientSocket` – сокет, отвечающий за взаимодействие с конкретным клиентом (стр. 15);
- `clientAddr` – используется Windows Sockets, чтобы указать локальный или удаленный адрес конечной точки, к которому подключение сокет; по большому счёту эту инфор-

мацию всегда можно получить из сокета, но традиционно этого не делают, и хранят все записи отдельной структурой (стр. 16);

- username – имя пользователя, которое, по протоколу, каждый участник беседы сообщает при подключении (стр 16).

Все структуры типа CLIENT_INFO хранятся в списке clients (стр 23). Для защиты этого списка от доступа из разных потоков используется критическая секция csClients (стр 23).

После запуска сервера, он проходит стандартные процедуры:

- Инициализация механизмов работы с Win сокетами версии 2 (стр. 34);
- Создание "слушающего"сокета hServerSocket, в качестве семейства протоколов выбран AF_INET, а в качестве типа протокола SOCK_STREAM, таким образом связь будет осуществляться по надёжному соединению с подтверждением доставки (стр. 42);
- Далее идёт определение адреса (стр. 62) и порта (66) для ожидания соединения, их значения были определены выше (стр 20, 21);
- Сокет переводится в слушающий режим (стр. 69) и ожидает подключения в бесконечном цикле (стр 92);
- Как только происходит подключения, для работы с новым клиентом создаётся отдельный сокет (стр 98) а его обработка передаётся в отдельный поток (стр. 113), таким образом "слушающий"сокет оказывается свободным для подключения новых клиентов.

Когда запускается отдельный поток ClientThread (стр 142)), обрабатывающий клиента, в первую очередь заполняется имя пользователя (стр 156), и информация о подключении рассылается всем пользователям (стр. 166). Далее эта процедура повторяется (стр. 169): сервер получает сообщение от клиента (стр. 170), и, если это не команда завершения работы (стр. 177), снабжает его меткой времени (стр. 199) , рассылает всем клиентам (стр. 205). Если сообщение содержало команду завершение сеанса, то информация об отключении клиента также рассылается всем клиентам (стр. 180), а сам клиент удаляется из списка (183). Последняя операция защищена критической секцией (стр 182, 183).

Рассылка сообщений описана в функции sendToAll (стр. 215). В ней блокируется доступ к списку сокетов (стр. 217), после этого происходит итерация по всему списку (стр. 218) с рассылкой сообщения (стр. 223), после чего блокировка снимается 235. Стоит заметить, что не всегда удаётся сообщение отправить за один пакет. Для этого сообщение бьётся на несколько частей и отправка происходит в несколько этапов (стр 223) уменьшая счётчик оставшихся частей (стр. 231).

С точки зрения клиента, происходит похожая процедура инициализации (стр. 22, листинг 46) и перевода сокета в состояние соединения (стр. 55, листинг 46), после чего пользователь вводит свой имя (стр. 67, листинг 46), а в отдельном потоке происходит активное ожидание сообщений от сервера (стр. 88, листинг 46). Это обеспечивает асинхронность приёма относительно отправки, в противном случае клиент бы не смог ничего получить пока сам что-то не отправит. При этом основной поток ожидает (стр. 97, листинг 46) ввода сообщения от пользователя (стр. 99, листинг 46) и занимается отправкой этого сообщения серверу (стр. 108, листинг 46) с разбивкой на пакеты, как это рассматривалось ранее.

Листинг 45: Сервер (src/SynchronizationPrimitives/FullReaderWriterServer/main.cpp)

```
1 #define _WINSOCK_DEPRECATED_NO_WARNINGS
2 #include <winsock2.h>
3 #include <list>
4
5 #include "Logger.h"
6
7 #include <time.h>
8 #include <stdio.h>
9 #include <strsafe.h>
10
11 #pragma comment(lib, "Ws2_32.lib")
12
13 struct CLIENT_INFO
14 {
15     SOCKET hClientSocket;           // Сокет
16     struct sockaddr_in clientAddr;
17     _TCHAR username[128];           // Имя пользователя
18 };
19
20 _TCHAR szServerIPAddr[] = _T("127.0.0.1"); // server IP
21 int nServerPort = 5050;                 // server port
22
23 std::list<CLIENT_INFO*> clients;         // Все клиенты
24 CRITICAL_SECTION csClients;              // Защита списка
25
26 bool InitWinSock2_0();
27 BOOL WINAPI ClientThread(LPVOID lpData);
28 void sendToAll(_TCHAR *pBuffer);
29
30 //Init log
31 Logger mylog(_T("FullReaderWriterServer"));
32
33 int _tmain(int argc, _TCHAR* argv[]) {
34     if (!InitWinSock2_0()) {
```

```

35     double errorcode = WSAGetLastError();
36     mylog.loudlog(_T("Unable to Initialize Windows Socket environment, GLE=%
        d"), errorcode);
37     exit(1);
38 }
39 mylog.loudlog(_T("Windows Socket environment ready"));
40
41 SOCKET hServerSocket;
42 hServerSocket = socket(
43     AF_INET,          // The address family. AF_INET specifies TCP/IP
44     SOCK_STREAM,      // Protocol type. SOCK_STREAM specified TCP
45     0                 // Protocol Name. Should be 0 for AF_INET address family
46 );
47
48 if (hServerSocket == INVALID_SOCKET) {
49     mylog.loudlog(_T("Unable to create Server socket"));
50     // Cleanup the environment initialized by WSASStartup()
51     WSACleanup();
52     exit(2);
53 }
54 mylog.loudlog(_T("Server socket created"));
55
56 // Create the structure describing various Server parameters
57 struct sockaddr_in serverAddr;
58
59 serverAddr.sin_family = AF_INET;      // The address family. MUST be
        AF_INET
60 size_t    convtd;
61 char *pMBBuffer = new char[20];
62 wcstombs_s(&convtd, pMBBuffer, 20, szServerIPAddr, 20);
63 //serverAddr.sin_addr.s_addr = inet_addr(pMBBuffer);
64 serverAddr.sin_addr.s_addr = INADDR_ANY;
65 delete[] pMBBuffer;
66 serverAddr.sin_port = htons(nServerPort);
67
68 // Bind the Server socket to the address & port
69 if (bind(hServerSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr
        )) == SOCKET_ERROR) {
70     mylog.loudlog(_T("Unable to bind to %s on port %d"), szServerIPAddr,
        nServerPort);
71     // Free the socket and cleanup the environment initialized by WSASStartup
        ()
72     closesocket(hServerSocket);
73     WSACleanup();
74     exit(3);

```

```

75 }
76 mylog.loudlog(_T("Bind completed"));
77
78 // Put the Server socket in listen state so that it can wait for client
   connections
79 if (listen(hServerSocket, SOMAXCONN) == SOCKET_ERROR) {
80     mylog.loudlog(_T("Unable to put server in listen state"));
81     // Free the socket and cleanup the environment initialized by WSASStartup
       ()
82     closesocket(hServerSocket);
83     WSACleanup();
84     exit(4);
85 }
86 mylog.loudlog(_T("Ready for connection on %s:%d"), szServerIPAddr,
       nServerPort);
87
88 //инициализируем средство синхронизации
89 InitializeCriticalSection(&csClients);
90
91 // Start the infinite loop
92 while (true) {
93     // As the socket is in listen mode there is a connection request pending
       .
94     // Calling accept( ) will succeed and return the socket for the request.
95     CLIENT_INFO* pClientInfo = new CLIENT_INFO;
96
97     int nSize = sizeof(pClientInfo->clientAddr);
98     pClientInfo->hClientSocket = accept(hServerSocket, (struct sockaddr *) &
       pClientInfo->clientAddr, &nSize);
99     if (pClientInfo->hClientSocket == INVALID_SOCKET) {
100         mylog.loudlog(_T("accept() failed"));
101     }
102     else {
103         HANDLE hClientThread;
104         DWORD dwThreadId;
105
106         wchar_t* sin_addr = new wchar_t[20];
107         size_t convtd;
108         mbstowcs_s(&convtd, sin_addr, 20, inet_ntoa(pClientInfo->clientAddr.
           sin_addr), 20);
109         mylog.loudlog(_T("Client connected from %s:%d"), sin_addr, pClientInfo
           ->clientAddr.sin_port);
110         delete[] sin_addr;
111
112         // Start the client thread

```



```

113     hClientThread = CreateThread(NULL, 0,
114         (LPTHREAD_START_ROUTINE)ClientThread,
115         (LPVOID)pClientInfo, 0, &dwThreadId);
116     if (hClientThread == NULL) {
117         mylog.loudlog(_T("Unable to create client thread"));
118     }
119     else {
120         CloseHandle(hClientThread);
121     }
122 }
123 }
124
125 //удаляем объект синхронизации
126 DeleteCriticalSection(&csClients);
127 closesocket(hServerSocket);
128 WSACleanup();
129 exit(0);
130 }
131
132 bool InitWinSock2_0() {
133     WSADATA wsaData;
134     WORD wVersion = MAKEWORD(2, 0);
135
136     if (!WSAStartup(wVersion, &wsaData))
137         return true;
138
139     return false;
140 }
141
142 BOOL WINAPI ClientThread(LPVOID lpData) {
143     CLIENT_INFO *pClientInfo = (CLIENT_INFO *)lpData;
144
145     EnterCriticalSection(&csClients);
146     clients.push_front(pClientInfo); // Добавить нового клиента в список
147     LeaveCriticalSection(&csClients);
148
149     _TCHAR szBuffer[1024];
150     _TCHAR szMessage[1024 + 255 + 128];
151
152     int nCntRecv = 0;
153     int nCntSend = 0;
154
155     // Set username
156     nCntRecv = recv(pClientInfo->hClientSocket, (char *)szBuffer, sizeof(
        szBuffer), 0);

```

```

157     if (nCntRecv <= 0) {
158         mylog.loudlog(_T("Error reading username from client"));
159         return 1;
160     }
161
162     szBuffer[nCntRecv] = '\0';
163     StringCchCopy(pClientInfo->username, sizeof(pClientInfo->username),
164         szBuffer);
165     swprintf_s(szMessage, _T("System: %s has joined this chat"), pClientInfo->
166         username);
167     mylog.loudlog(_T("%s"), szMessage);
168     sendToAll(szMessage);
169
170     // Chat loop:
171     while (1) {
172         nCntRecv = recv(pClientInfo->hClientSocket, (char *)szBuffer, sizeof(
173             szBuffer), 0);
174         if (nCntRecv > 0) {
175             // Process message
176             szBuffer[nCntRecv] = '\0';
177
178             // Check, if its not QUIT
179             _wcsdup(szBuffer);
180             if (wcscmp(szBuffer, _T("QUIT")) == 0) {
181                 swprintf_s(szMessage, _T("System: %s has left this chat"),
182                     pClientInfo->username);
183                 mylog.loudlog(_T("%s"), szMessage);
184                 sendToAll(szMessage);
185
186                 EnterCriticalSection(&csClients);
187                 clients.remove(pClientInfo); // Удалить клиента из списка
188                 LeaveCriticalSection(&csClients);
189
190                 closesocket(pClientInfo->hClientSocket);
191                 delete pClientInfo;
192                 return 0;
193             }
194
195             // Time
196             struct tm newtime;
197             __time64_t long_time;
198             // Get time as 64-bit integer.
199             _time64(&long_time);
200             // Convert to local time.
201             _localtime64_s(&newtime, &long_time);

```

```

198     // Create message.
199     swprintf_s(szMessage, _T("[%02d/%02d/%04d %02d:%02d:%02d] %s: %s"),
200         newtime.tm_mday,
201         newtime.tm_mon + 1, newtime.tm_year + 1900, newtime.tm_hour,
202         newtime.tm_min, newtime.tm_sec, pClientInfo->username, szBuffer);
203     mylog.loudlog(_T("%s"), szMessage);
204
205     sendToAll(szMessage);
206 }
207 else {
208     mylog.loudlog(_T("Error reading the data from %s"), pClientInfo->
209         username);
210 }
211
212 return 0;
213 }
214
215 void sendToAll(_TCHAR *pBuffer) {
216     // Пока мы обрабатываем список, его ни кто не должен менять!
217     EnterCriticalSection(&csClients);
218     std::list<CLIENT_INFO *>::iterator client;
219     for (client = clients.begin(); client != clients.end(); ++client) {
220         int nLength = (lstrlen(pBuffer) + 1) * sizeof(_TCHAR);
221         int nCntSend = 0;
222
223         while ((nCntSend = send((*client)->hClientSocket, (char *)pBuffer,
224             nLength, 0) != nLength)) {
225             if (nCntSend == -1) {
226                 mylog.loudlog(_T("Error sending the data to %s"), (*client)->
227                     username);
228                 break;
229             }
230             if (nCntSend == nLength)
231                 break;
232
233             pBuffer += nCntSend;
234             nLength -= nCntSend;
235         }
236     }
237     LeaveCriticalSection(&csClients);
238 }

```

Листинг 46: Клиент (src/SynchronizationPrimitives/FullReaderWriterClient/main.cpp)

```

1 #define _WINSOCK_DEPRECATED_NO_WARNINGS
2 #include <winsock2.h>
3 #include "Logger.h"
4
5 #pragma comment(lib, "Ws2_32.lib")
6
7 _TCHAR szServerIPAddr[20];      // server IP
8 int nServerPort;               // server port
9
10 bool InitWinSock2_0();
11 BOOL WINAPI aReader(LPVOID lpData); // Чтение
12
13 //Init log
14 Logger mylog(_T("FullReaderWriterClient"), GetCurrentProcessId());
15
16 int _tmain(int argc, _TCHAR* argv[]) {
17     _tprintf(_T("Enter the server IP Address: "));
18     wscanf_s(_T("%19s"), szServerIPAddr, _countof(szServerIPAddr));
19     _tprintf(_T("Enter the server port number: "));
20     wscanf_s(_T("%i"), &nServerPort);
21
22     if (!InitWinSock2_0()) {
23         double errorcode = WSAGetLastError();
24         mylog.loudlog(_T("Unable to Initialize Windows Socket environment, GLE=%d"), errorcode);
25         exit(1);
26     }
27     mylog.quietlog(_T("Windows Socket environment ready"));
28
29     SOCKET hClientSocket;
30     hClientSocket = socket(
31         AF_INET,      // The address family. AF_INET specifies TCP/IP
32         SOCK_STREAM,   // Protocol type. SOCK_STREAM specified TCP
33         0);           // Protocol Name. Should be 0 for AF_INET address family
34
35     if (hClientSocket == INVALID_SOCKET) {
36         mylog.loudlog(_T("Unable to create socket"));
37         // Cleanup the environment initialized by WSASStartup()
38         WSACleanup();
39         exit(2);
40     }
41     mylog.quietlog(_T("Client socket created"));
42
43     // Create the structure describing various Server parameters

```

```

44 struct sockaddr_in serverAddr;
45
46 serverAddr.sin_family = AF_INET;      // The address family. MUST be
    AF_INET
47 size_t    convtd;
48 char *pMBBuffer = new char[20];
49 wcstombs_s(&convtd, pMBBuffer, 20, szServerIPAddr, 20);
50 serverAddr.sin_addr.s_addr = inet_addr(pMBBuffer);
51 delete[] pMBBuffer;
52 serverAddr.sin_port = htons(nServerPort);
53
54 // Connect to the server
55 if (connect(hClientSocket, (struct sockaddr *) &serverAddr, sizeof(
    serverAddr)) < 0) {
56     mylog.loudlog(_T("Unable to connect to %s on port %d"), szServerIPAddr,
        nServerPort);
57     closesocket(hClientSocket);
58     WSACleanup();
59     exit(3);
60 }
61 mylog.quietlog(_T("Connect"));
62
63 _TCHAR szBuffer[1024] = _T("");
64
65 // Choose username
66 _tprintf(_T("Enter your username: "));
67 wscanf_s(_T("%1023s"), szBuffer, _countof(szBuffer));
68 int nLength = (wcslen(szBuffer) + 1) * sizeof(_TCHAR);
69
70 // send( ) may not be able to send the complete data in one go.
71 // So try sending the data in multiple requests
72 int nCntSend = 0;
73 _TCHAR *pBuffer = szBuffer;
74
75 while ((nCntSend = send(hClientSocket, (char *)pBuffer, nLength, 0) !=
    nLength)) {
76     if (nCntSend == -1) {
77         mylog.loudlog(_T("Error sending the data to server"));
78         break;
79     }
80     if (nCntSend == nLength)
81         break;
82
83     pBuffer += nCntSend;
84     nLength -= nCntSend;

```

```

85 }
86
87 // Занульс чумающего мреда
88 HANDLE haReader = CreateThread(NULL, 0,
89     (LPTHREAD_START_ROUTINE)aReader,
90     (LPVOID)&hClientSocket, 0, 0);
91 if (haReader == NULL) {
92     mylog.loudlog(_T("Unable to create Reader thread"));
93 }
94
95 // Chat loop:
96 _tprintf(_T("Enter your messages or QUIT for exit.\n"));
97 while (wcscmp(szBuffer, _T("QUIT")) != 0) {
98     _tprintf(_T(">> "));
99     wscanf_s(_T("%1023s"), szBuffer, _countof(szBuffer));
100
101     nLength = (wcslen(szBuffer) + 1) * sizeof(_TCHAR);
102
103     // send( ) may not be able to send the complete data in one go.
104     // So try sending the data in multiple requests
105     nCntSend = 0;
106     pBuffer = szBuffer;
107
108     while ((nCntSend = send(hClientSocket, (char *)pBuffer, nLength, 0) !=
109         nLength)) {
110         if (nCntSend == -1) {
111             mylog.loudlog(_T("Error sending the data to server"));
112             break;
113         }
114         if (nCntSend == nLength)
115             break;
116
117         pBuffer += nCntSend;
118         nLength -= nCntSend;
119     }
120
121     _wcsdup(szBuffer);
122     if (wcscmp(szBuffer, _T("QUIT")) == 0) {
123         TerminateThread(haReader, 0);
124         break;
125     }
126 }
127 closesocket(hClientSocket);
128 WSACleanup();

```

```

129     exit(0);
130 }
131
132 bool InitWinSock2_0() {
133     WSADATA wsaData;
134     WORD wVersion = MAKEWORD(2, 0);
135
136     if (!WSAStartup(wVersion, &wsaData))
137         return true;
138
139     return false;
140 }
141
142 BOOL WINAPI aReader(LPVOID lpData) {
143     SOCKET *hClientSocket = (SOCKET *)lpData;
144     _TCHAR szBuffer[1024];
145     int nLength = 0;
146
147     while (1) {
148         nLength = recv(*hClientSocket, (char *)szBuffer, sizeof(szBuffer), 0);
149
150         if (nLength > 0) {
151             szBuffer[nLength] = '\\0';
152             mylog.loudlog(_T("%s"), szBuffer);
153             _tprintf(_T(">> "));
154         }
155     }
156
157     return 0;
158 }

```

Рисунок 16 показывает сервер (левый верхний угол) и трёх клиентов, которые общаются между собой. Особенность реализованного протокола в том, что каждый клиент должен ввести своё имя, перед началом общения.

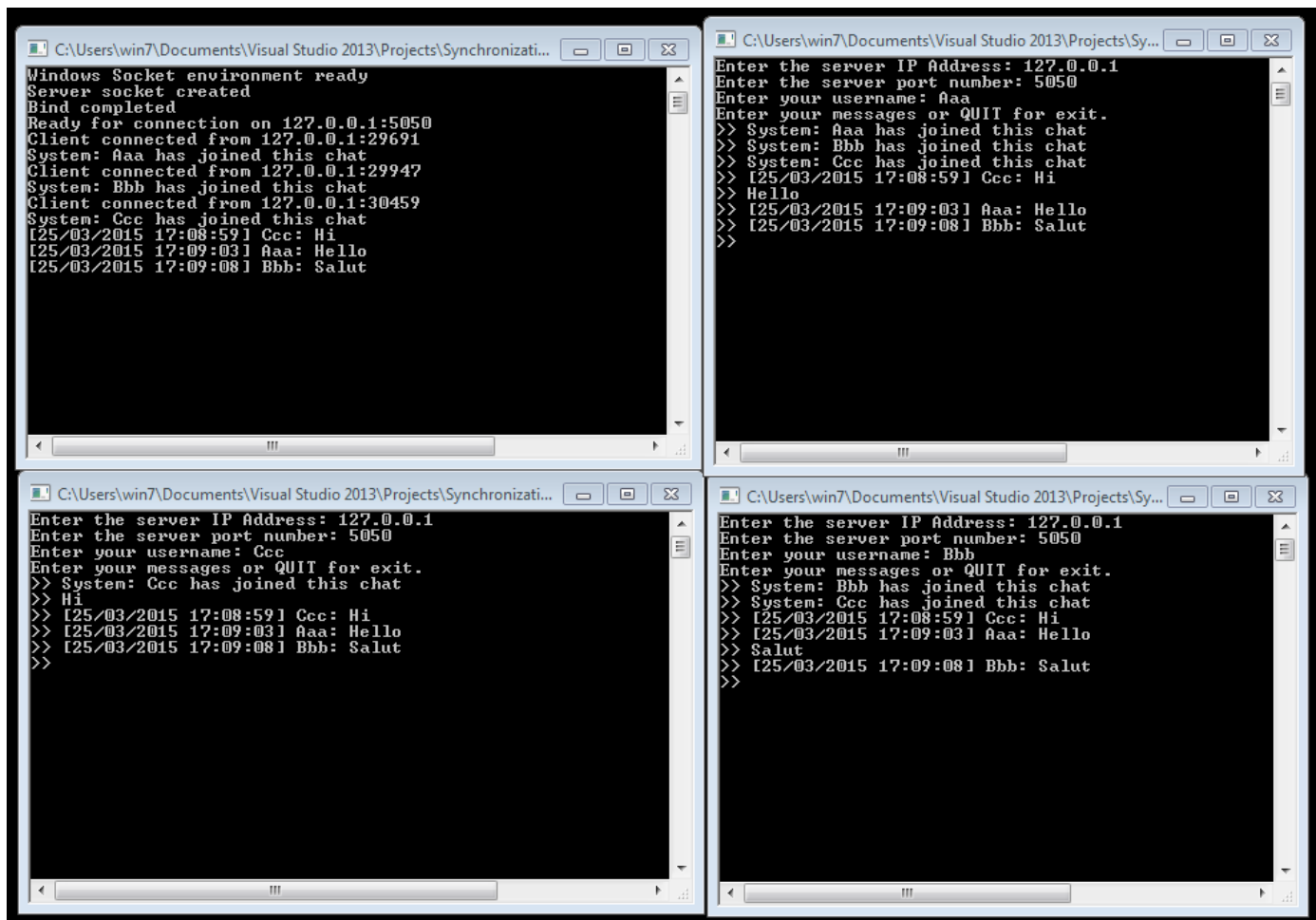


Рис. 16: Полноценный чат на Win-сокетах.

7 Задача обедающие философы

Классическая задача про обедающих философов[6]. Она имеет несколько классических решений. В данном решении (листинг 47) упор сделан на то, что вовсе не обязательно вешать объект-синхронизацию на вилку, т.к. её состояние можно вывести из состояния другого философа (если он обедает, то, очевидно, вилка занята). В качестве механизма синхронизации была выбрана критическая сессия, как наиболее простая и легковесная.

Листинг 47: Клиент (src/SynchronizationPrimitives/DiningPhilosophersProblem/main.cpp)

```

1 #include <windows.h>
2 #include <conio.h>
3 #include <tchar.h>
4
5 #include "Logger.h"
6
7 #define N          5          // всего философов
8 #define LEFT      (id+N-1)%N // левый сосед

```



```

9 #define RIGHT      (id+1)%N    // правый сосед
10
11 #define THINKING  0          // состояние размышления
12 #define HUNGRY    1          // состояние голода
13 #define EATING    2          // философ ест
14
15 int philosopher_state[N]; // состояние философов
16 CRITICAL_SECTION crs[N];  // Объявление критических секций
17
18 DWORD WINAPI philosopherThread(LPVOID prm);
19 void take_forks(int id);
20 void put_forks(int id);
21 void think();
22 void eat();
23 void wait();
24
25 //Init log
26 Logger mylog(_T("DiningPhilosophersProblem"));
27
28 // Размышления
29 void think() {
30     Sleep(100 + rand() % 500);
31 }
32
33 // Еда
34 void eat() {
35     Sleep(50 + rand() % 450);
36 }
37
38 // Голодное ожидание
39 void wait() {
40     Sleep(50 + rand() % 150);
41 }
42
43 // симуляция жизни философа
44 DWORD WINAPI philosopherThread(LPVOID prm) {
45     int phil_id = (int)prm;
46     while (true) {
47         think();
48         // Либо обе вилки, либо блокировка
49         take_forks(phil_id);
50         eat();
51         // Вернуть вилки на стол
52         put_forks(phil_id);
53     }

```

```

54 }
55
56 void take_forks(int id) {
57     bool done = false;
58     while (!done){
59         if (rand() % 2) { // left hand first
60             mylog.quietlog(_T("Waining for Critical Section"));
61             EnterCriticalSection(&crs[id]);
62             mylog.quietlog(_T("Get Critical Section"));
63
64             philosopher_state[id] = HUNGRY;
65             mylog.loudlog(_T("Philosopher %d status HUNGRY"), id);
66
67             if (TryEnterCriticalSection(&crs[LEFT])) {
68                 if (philosopher_state[LEFT] != EATING) {
69                     if (TryEnterCriticalSection(&crs[RIGHT])) {
70                         if (philosopher_state[RIGHT] != EATING) {
71                             philosopher_state[id] = EATING;
72                             mylog.loudlog(_T("Philosopher %d status EATING"), id);
73                             done = true;
74                         }
75                         LeaveCriticalSection(&crs[RIGHT]);
76                     }
77                 }
78                 LeaveCriticalSection(&crs[LEFT]);
79             }
80             mylog.quietlog(_T("Leave Critical Section"));
81             LeaveCriticalSection(&crs[id]);
82         }
83         else { // right hand first
84             mylog.quietlog(_T("Waining for Critical Section"));
85             EnterCriticalSection(&crs[id]);
86             mylog.quietlog(_T("Get Critical Section"));
87
88             philosopher_state[id] = HUNGRY;
89             mylog.loudlog(_T("Philosopher %d status HUNGRY"), id);
90
91             if (TryEnterCriticalSection(&crs[RIGHT])) {
92                 if (philosopher_state[RIGHT] != EATING) {
93                     if (TryEnterCriticalSection(&crs[LEFT])) {
94                         if (philosopher_state[LEFT] != EATING) {
95                             philosopher_state[id] = EATING;
96                             mylog.loudlog(_T("Philosopher %d status EATING"), id);
97                             done = true;
98                         }

```

```

99         LeaveCriticalSection(&crs[LEFT]);
100     }
101 }
102     LeaveCriticalSection(&crs[RIGHT]);
103 }
104     mylog.quietlog(_T("Leave Critical Section"));
105     LeaveCriticalSection(&crs[id]);
106 }
107
108     if (!done)
109         wait();
110 }
111 }
112
113 void put_forks(int id) {
114     mylog.quietlog(_T("Waiting for Critical Section"));
115     EnterCriticalSection(&crs[id]);
116     mylog.quietlog(_T("Get Critical Section"));
117
118     philosopher_state[id] = THINKING;
119     mylog.loudlog(_T("Philosopher %d status THINKING"), id);
120
121     mylog.quietlog(_T("Leave Critical Section"));
122     LeaveCriticalSection(&crs[id]);
123 }
124
125 int _tmain(int argc, _TCHAR* argv[]) {
126     // Массовые потоковые
127     HANDLE allhandlers[N];
128
129     //создаем потоки-читатели
130     mylog.loudlog(_T("Create threads"));
131     for (int i = 0; i != N; ++i) {
132         mylog.loudlog(_T("Count = %d"), i);
133         //создаем потоки-читатели, которые пока не стартуют
134         if ((allhandlers[i] = CreateThread(NULL, 0, philosopherThread, (LPVOID)i
            , CREATE_SUSPENDED, NULL)) == NULL) {
135             mylog.loudlog(_T("Impossible to create thread-reader, GLE = %d"),
                GetLastError());
136             exit(8000);
137         }
138     }
139
140     //инициализируем средство синхронизации
141     for (int i = 0; i != N; ++i) {

```

```

142     InitializeCriticalSection(&crs[i]);
143 }
144
145 //запускаем потоки на исполнение
146 for (int i = 0; i < N; ++i)
147     ResumeThread(allhandlers[i]);
148
149 //ожидаем завершения всех потоков
150 WaitForMultipleObjects(N, allhandlers, TRUE, INFINITE);
151 //закрываем handle потоков
152 for (int i = 0; i < N; ++i)
153     CloseHandle(allhandlers[i]);
154
155 //удаляем объект синхронизации
156 for (int i = 0; i != N; ++i) {
157     DeleteCriticalSection(&crs[i]);
158 }
159
160 // Завершение работы
161 mylog.loudlog(_T("All tasks are done!"));
162 _getch();
163 return 0;
164 }

```

Результаты работы программы показаны на рисунке 17 и в листинге 48 (в этом отрывке опять наблюдается наложение записей, т.к. несколько потоков пишут в 1 файл).

Критическая секция не является объектом ядра, так что process explorer показывает только потоки (рисунок 18).

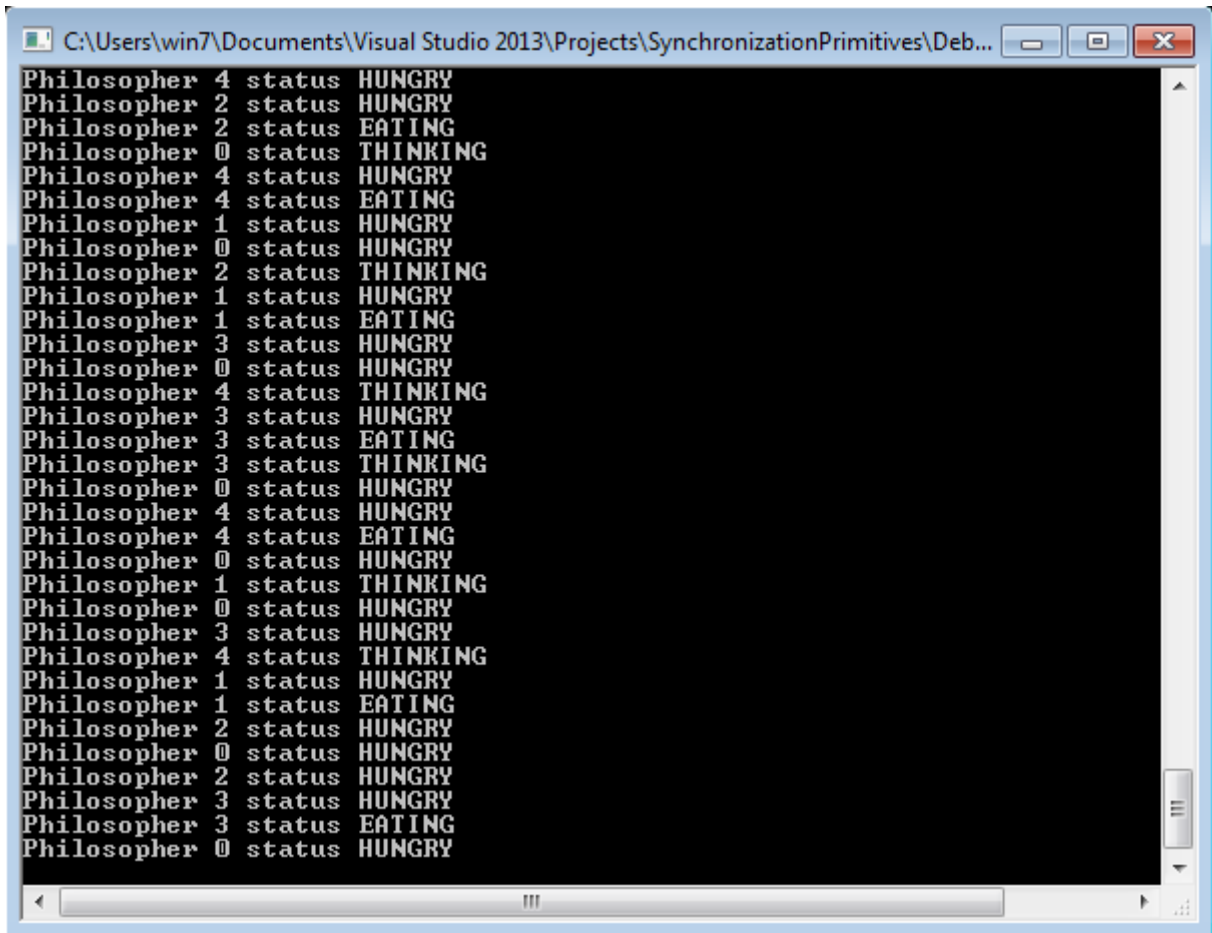


Рис. 17: Эмулятор жизни философа.

Листинг 48: Отрывок протокол работы эмулятора обедающих философов

```

1 [25/3/2015 20:27:26] DiningPhilosophersProblem is starting.
2 [25/3/2015 20:27:26] Create threads
3 [25/3/2015 20:27:26] Count = 0
4 [25/3/2015 20:27:26] Count = 1
5 [25/3/2015 20:27:26] Count = 2
6 [25/3/2015 20:27:26] Count = 3
7 [25/3/2015 20:27:26] Count = 4
8 [25/3/2015 20:27:26] Waining for Critical Section
9 [25/3/2015 20:27:26] Get Critical Section
10 [25/3/2015 20:27:26] Philosopher 1 status HUNGRY[25/3/2015 20:27:26] Waining
    for Critical Section
11 [25/3/2015 20:27:26] Get Critical Section
12 [25/3/2015 20:27:26] Philosopher 0 status HUNGRY[25/3/2015 20:27:26] Waining
    for Critical Section
13 [25/3/2015 20:27:26] Get Critical Section
14 [25/3/2015 20:27:26] Philosopher 3 status HUNGRY[25/3/2015 20:27:26] Waining
    for Critical Section
15 [25/3/2015 20:27:26] Get Critical Section

```

16 [25/3/2015 20:27:26] Philosopher 2 status HUNGRY [25/3/2015 20:27:26] Waining
 for Critical Section
 17 [25/3/2015 20:27:26] Get Critical Section
 18 [25/3/2015 20:27:26] Philosopher 4 status HUNGRY
 19 [25/3/2015 20:27:26] Leave Critical Section
 20
 21 [25/3/2015 20:27:26] Leave Critical Section
 22
 23 [25/3/2015 20:27:26] Leave Critical Section
 24
 25 [25/3/2015 20:27:26] Philosopher 2 status EATING
 26 [25/3/2015 20:27:26] Leave Critical Section
 27
 28 [25/3/2015 20:27:26] Philosopher 0 status EATING
 29 [25/3/2015 20:27:26] Leave Critical Section
 30 [25/3/2015 20:27:26] Waining for Critical Section
 31 [25/3/2015 20:27:26] [25/3/2015 20:27:26] Get Critical Section [25/3/2015
 20:27:26]
 32 Waining for Critical Section [25/3/2015 20:27:26]
 33 Philosopher 1 status HUNGRY [25/3/2015 20:27:26] Get Critical Section
 34 [25/3/2015 20:27:26] Philosopher 3 status HUNGRY Waining for Critical Section
 35 [25/3/2015 20:27:26] Get Critical Section
 36 [25/3/2015 20:27:26] Philosopher 4 status HUNGRY
 37 [25/3/2015 20:27:26] Leave Critical Section
 38
 39 [25/3/2015 20:27:26] Leave Critical Section
 40
 41 [25/3/2015 20:27:26] Leave Critical Section
 42 [25/3/2015 20:27:26] Waining for Critical Section
 43 [25/3/2015 20:27:26] Get Critical Section
 44 [25/3/2015 20:27:26] Philosopher 0 status THINKING [25/3/2015 20:27:26]
 Waining for Critical Section
 45 [25/3/2015 20:27:26] Get Critical Section
 46 [25/3/2015 20:27:26] Philosopher 2 status THINKING
 47 [25/3/2015 20:27:26] Leave Critical Section
 48
 49 [25/3/2015 20:27:26] Leave Critical Section
 50 [25/3/2015 20:27:26] Waining for Critical Section
 51 [25/3/2015 20:27:26] Get Critical Section
 52 [25/3/2015 20:27:26] Philosopher 2 status HUNGRY [25/3/2015 20:27:26] Waining
 for Critical Section
 53 [25/3/2015 20:27:26] Get Critical Section
 54 [25/3/2015 20:27:26] Philosopher 0 status HUNGRY
 55 [25/3/2015 20:27:26] Philosopher 2 status EATING
 56 [25/3/2015 20:27:26] Leave Critical Section

```

57
58 [25/3/2015 20:27:26] Philosopher 0 status EATING
59 [25/3/2015 20:27:26] Leave Critical Section
60 [25/3/2015 20:27:26] Waining for Critical Section
61 [25/3/2015 20:27:26] Get Critical Section
62 [25/3/2015 20:27:26] Philosopher 1 status HUNGRY [25/3/2015 20:27:26] Waining
    for Critical Section
63 [25/3/2015 20:27:26] Get Critical Section
64 [25/3/2015 20:27:26] Philosopher 3 status HUNGRY [25/3/2015 20:27:26] Waining
    for Critical Section
65 [25/3/2015 20:27:26] Get Critical Section
66 [25/3/2015 20:27:26] Philosopher 4 status HUNGRY
67 [25/3/2015 20:27:26] Leave Critical Section
68
69 [25/3/2015 20:27:26] Leave Critical Section
70
71 [25/3/2015 20:27:30] Shutting down.

```

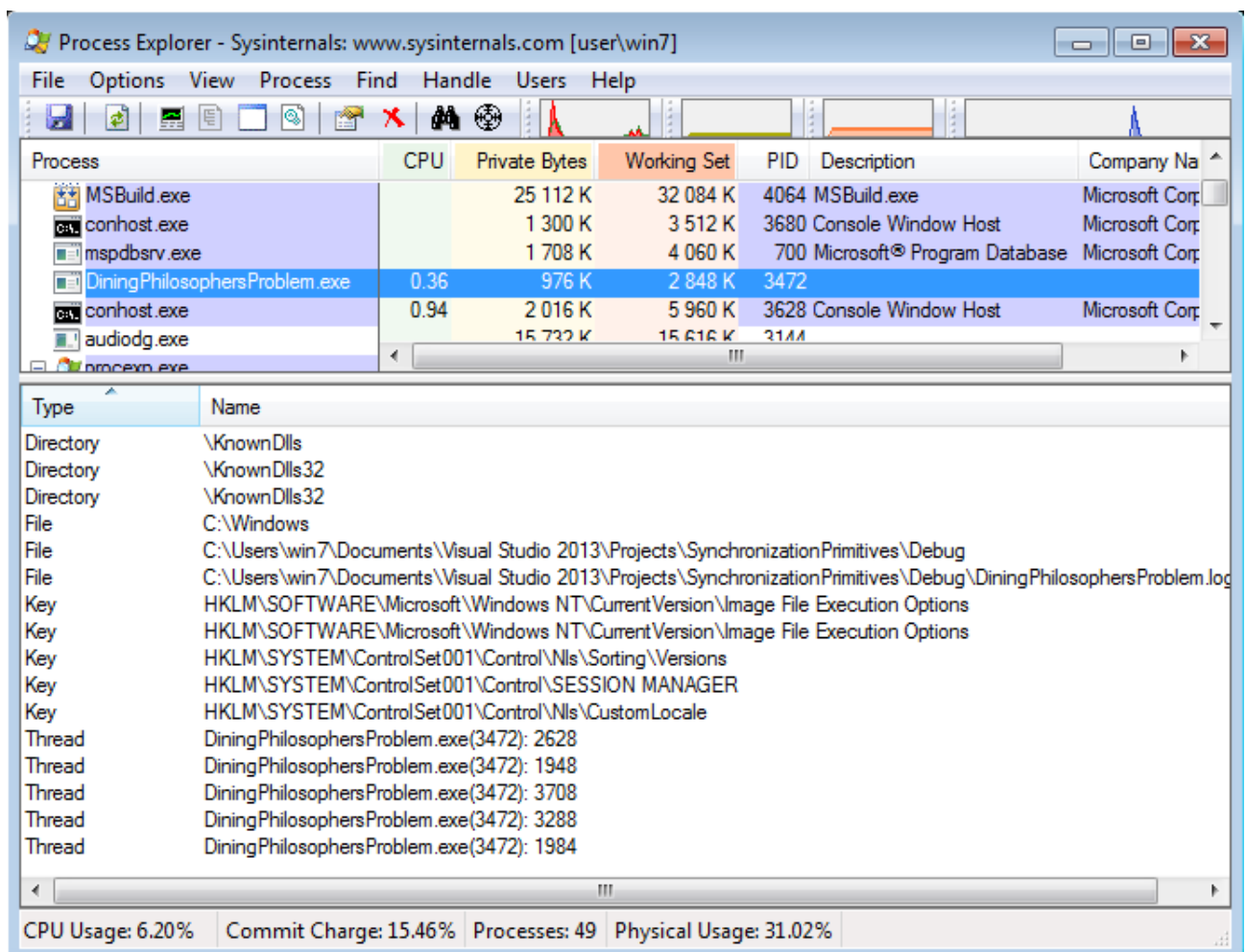


Рис. 18: вывод process explorer для задачи философов.

Заключение

В данной работе были рассмотрены все примитивы синхронизаций, начиная с мьютексов и семафоров, и заканчивая не особо популярными - Slim Reader/Writer (SRW) Lock.

- Мьютекс – является объектом ядра, у него есть имя, а значит с его помощью можно синхронизировать доступ к общим данным со стороны нескольких процессов, точнее, со стороны потоков разных процессов. Ни один другой поток не может завладеть мьютексом, который уже принадлежит одному из потоков. Если мьютекс защищает какие-то совместно используемые данные, он сможет выполнить свою функцию только в случае, если перед обращением к этим данным каждый из потоков будет проверять состояние этого мьютекса. Windows расценивает мьютекс как объект общего доступа, который можно перевести в сигнальное состояние или сбросить. Сигнальное состояние мьютекса говорит о том, что он занят. Потоки должны самостоятельно анализировать текущее состояние мьютексов.
- Семафор – используются для учёта ресурсов и служат для ограничения одновременного доступа к ресурсу нескольких потоков. Используя семафор, можно организовать работу программы таким образом, что к ресурсу одновременно смогут получить доступ несколько потоков, однако количество этих потоков будет ограничено. При создании семафора указывается максимальное количество потоков, которые одновременно смогут работать с ресурсом. Каждый раз, когда программа обращается к семафору, значение счетчика ресурсов семафора уменьшается на единицу. Когда значение счетчика ресурсов становится равным нулю, семафор недоступен.
- Критические секции – это объект, который принадлежит процессу, а не ядру. А значит, не может синхронизировать потоки из разных процессов. Среди синхронизирующих объектов критические разделы наиболее просты. Ещё одним его плюсом является скорость работы.
- Объекты события – оповещают об окончании какой-либо операции, они также являются объектами ядра. Можно не просто явным образом освободить, но так же есть операция установки события. События могут быть мануальными (manual) и единичными (single).
- Условные переменные – не являются самостоятельным механизмом синхронизации и используется совместно с другим механизмом (чаще всего с мьютексом). В отличие от семафора, не обладают памятью после сброса.
- Slim Reader/Writer (SRW) Lock – более "продвинутая" версия критической секции, где разделены понятия захвата на чтение и на запись.

Наиболее интересной задачей была задача по полноценного производителя-потребителя. Сложность была в передаче данных от одного клиента к другому. В текущей реализации порядок доставки сообщений может быть нарушен, но это не критично, т.к. каждое сообщение имеет метку времени.

Наиболее простым механизмом является критическая секция, и именно она была использована для решения задачи философов, но, к сожалению, критическая секция может быть использована только в рамках одного процесса.

Список литературы

1. Душутина Е.В. Межпроцессные взаимодействия в операционных системах. Учебное пособие – СПб.: 2014 – 135 стр.
2. Огинский А.А., Набатчиков А.М., Бурлак Е.А. Организация межпотокowego взаимодействия с использованием объектов ядра операционной системы – С. 52-56. Вестник компьютерных и информационных технологий. №8. Август 2012.
3. Щупак Ю. А. Win32 API. Разработка приложений для Windows – СПб.: Питер, 2008 – 592 стр.
4. MSDN: Обзор асинхронной модели, основанной на событиях – <https://msdn.microsoft.com/ru-ru/wewwczdw.aspx>
5. MSDN: Slim Reader/Writer (SRW) Locks – <https://msdn.microsoft.com/aa904937.aspx>
6. Э. Таненбаум. Современные операционные системы – СПб.:Питер, 2010. – 1120 стр.