### 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](#_TOC_250010)

[Введение 5](#_TOC_250009)

1. [Примитивы синхронизации 6](#_TOC_250008)
   1. Использование мьютексов 13
   2. Использование семафоров 19
   3. Критические секции 30
   4. Объекты-события в качестве средства синхронизации 35
   5. Условные переменные 40
   6. Задача читатели-писатели (для потоков одного процесса) 47
   7. Задача читатели-писатели (для потоков разных процессов) 55
2. [Модификация задачи читатели-писатели без доступа к памяти 70](#_TOC_250007)
3. [Рациональное решение задачи читатели-писатели 82](#_TOC_250006)
4. [Клиент-серверное приложение для полной задачи читатели-писатели 91](#_TOC_250005)
5. [Сетевая версия задачи читатели-писатели 108](#_TOC_250004)
6. [Задача производители-потребители 116](#_TOC_250003)
7. [Задача обедающие философы 128](#_TOC_250002)

[Заключение 136](#_TOC_250001)

[Список литературы 138](#_TOC_250000)

# Постановка задачи

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

Потоки разделяют целочисленный массив, в который заносятся производимые и извле- каются 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.

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 воз- можности 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 созда- ются 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 взаимоисключение (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 Примитивы 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

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

# include

" Logger . h"

# include

# include

# include

# include

# include

# include

< stdio . h>

< stdlib . h>

< tchar . h>

< stdarg . h>

< time . h>

< Windows . h>

Logger :: Logger ( const \_ TCHAR \* prog , int tid */\* = -1 \*/* ) {

\_ TCHAR logname [ 255 ];

if ( tid > 0)

swprintf\_ s ( logname , \_T("% s.% d. log "), prog , tid ); else

swprintf\_ s ( logname , \_T("% s. log "), prog );

*// Try to open log file for append*

if ( \_ wfopen\_ s (& logfile , logname , \_T(" a+"))) {

\_ wperror (\_T(" The following error occurred "));

\_ tprintf (\_T(" Can ’t open log file % s\ n"), logname ); exit ( -1);

}

quietlog (\_T("% s is starting ."), prog );

}

Logger ::~ Logger () {

quietlog (\_T(" Shutting down .\ n")); fclose ( logfile );

}

void Logger :: quietlog ( \_ TCHAR \* format , ...) {

1. \_ TCHAR buf [ 255 ];
2. va\_ list ap;
3. struct tm newtime ;
4. \_\_ time 64 \_ t long\_ time ;
5. *// Get time as 64 - bit integer .*
6. \_ time 64 (& long\_ time );
7. *// Convert to local time.*
8. \_ localtime 64 \_ s (& newtime , & long\_ time );
9. *// Convert to normal representation .*
10. swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday ,
11. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
12. newtime . tm\_min , newtime . tm\_ sec );
13. *// Write date and time*
14. fwprintf ( logfile , \_T("% s"), buf );
15. *// Write all params*
16. va\_ start ( ap , format );
17. \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap);
18. fwprintf ( logfile , \_T("% s"), buf );
19. va\_ end ( ap);
20. *// New sting*
21. fwprintf ( logfile , \_T("\ n"));

54 }

#### 55

1. void Logger :: loudlog ( \_ TCHAR \* format , ...) {
2. \_ TCHAR buf [ 255 ];
3. va\_ list ap;
4. struct tm newtime ;
5. \_\_ time 64 \_ t long\_ time ;
6. *// Get time as 64 - bit integer .*
7. \_ time 64 (& long\_ time );
8. *// Convert to local time.*
9. \_ localtime 64 \_ s (& newtime , & long\_ time );
10. *// Convert to normal representation .*
11. swprintf\_ s ( buf , \_T("[% d/% d/% d % d:% d:% d] "), newtime . tm\_mday ,
12. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
13. newtime . tm\_min , newtime . tm\_ sec );
14. *// Write date and time*
15. fwprintf ( logfile , \_T("% s"), buf );
16. *// Write all params*
17. va\_ start ( ap , format );
18. \_ vsnwprintf\_ s ( buf , sizeof ( buf ) - 1 , format , ap);
19. fwprintf ( logfile , \_T("% s"), buf );
20. \_ tprintf (\_T("% s"), buf );
21. va\_ end ( ap);
22. *// New sting*

78 fwprintf ( logfile , \_T("\ n"));

79 \_ tprintf (\_T("\ n"));

80

}

### Использование класса позволяет не беспокоиться о корректном захвате ресурса (дескрип- торе 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 отличии 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/utils.cpp)

1

2

3

4

5

# include

# include

# include

< windows . h>

< stdio . h>

< tchar . h>

# include

" thread . h"

1. # include " utils . h"
2. # include " Logger . h"

#### 8

1. *// создание, установка и запуск таймера*
2. HANDLE Create And Start Waitable Timer ( int sec ) {
3. \_\_ int 64 end\_ time ;
4. LARGE\_ INTEGER end\_ time 2 ;
5. HANDLE tm = Create Waitable Timer ( NULL , false , \_T(" Timer !"));

14 end\_ time = -1 \* sec \* 10000000 ;

1. end\_ time 2 . Low Part = ( DWORD )( end\_ time & 0 x FFFFFFFF );
2. end\_ time 2 . High Part = ( LONG )( end\_ time >> 32) ;
3. Set Waitable Timer ( tm , & end\_time2 , 0 , NULL , NULL , false );
4. return tm;

19 }

#### 20

1. *// создание всех потоков*
2. void Create All Threads ( struct Configuration \* config , Logger \* log ) {
3. extern HANDLE \* allhandlers ;

#### 24

1. int total = config -> num Of Readers + config -> num Of Writers + 1;
2. log -> quietlog (\_T(" Total num of threads is % d"), total );
3. allhandlers = new HANDLE [ total ];
4. int count = 0;

#### 29

1. *// создаем потоки- читатели*
2. log -> loudlog (\_T(" Create readers "));
3. for ( int i = 0; i != config -> num Of Readers ; ++ i, ++ count ) {
4. log -> loudlog (\_T(" Count = % d"), count );
5. *// создаем потоки- читатели, которые пока не стартуют*
6. if (( allhandlers [ count ] = Create Thread ( NULL , 0 , Thread Reader Handler , ( LPVOID ) i, CREATE\_ SUSPENDED , NULL )) == NULL ) {
7. log -> loudlog (\_T(" Impossible to create thread - reader , GLE = % d"),

Get Last Error ());

37 exit ( 8000 ) ;

38 }

39 }

#### 40

1. *// создаем потоки- писатели*
2. log -> loudlog (\_T(" Create writers "));
3. for ( int i = 0; i != config -> num Of Writers ; ++ i, ++ count ) {
4. log -> loudlog (\_T(" count = % d"), count );
5. *// создаем потоки- писателии, которые пока не стартуют*
6. if (( allhandlers [ count ] = Create Thread ( NULL , 0 , Thread Writer Handler , ( LPVOID ) i, CREATE\_ SUSPENDED , NULL )) == NULL ) {
7. log -> loudlog (\_T(" Impossible to create thread - writer , GLE = % d"),

Get Last Error ());

48 exit ( 8001 ) ;

49 }

50 }

#### 51

1. *// создаем поток Time Manager*
2. log -> loudlog (\_T(" Create Time Manager "));
3. log -> loudlog (\_T(" Count = % d"), count );
4. *// создаем поток Time Manager , который пока не стартуют*
5. if (( allhandlers [ count ] = Create Thread ( NULL , 0 , Thread Time Manager Handler , ( LPVOID ) config -> ttl , CREATE\_ SUSPENDED , NULL )) == NULL ) {
6. log -> loudlog (\_T(" impossible to create thread - reader , GLE = % d"),

Get Last Error ());

58 exit ( 8002 ) ;

59 }

60 log -> loudlog (\_T(" Successfully created threads !"));

61 }

#### 62

1. *// функция установки конфигурации*
2. void Set Config ( \_ TCHAR \* path , struct Configuration \* config , Logger \* log ) {
3. \_ TCHAR filename [ 255 ];
4. wcscpy\_ s ( filename , path );
5. log -> quietlog (\_T(" Using config from % s"), filename );

#### 68

1. FILE \* confsource ;
2. int num Of Readers ;
3. int num Of Writers ;
4. int readers Delay ;
5. int writers Delay ;
6. int size Of Queue ;
7. int ttl ;
8. \_ TCHAR trash [30];

#### 77

1. if ( \_ wfopen\_ s (& confsource , filename , \_T(" r"))) {
2. \_ wperror (\_T(" The following error occurred "));
3. log -> loudlog (\_T(" impossible open config file % s\ n"), filename );

81 exit ( 1000 ) ;

82 }

#### 83

1. *// начинаем читать конфигурацию*
2. fscanf\_ s ( confsource , "% s % d", trash , \_ countof ( trash ), & num Of Readers ); *// чи сло потоков-читателей*
3. fscanf\_ s ( confsource , "% s % d", trash , \_ countof ( trash ), & readers Delay ); *// за держки потоков-читателей*
4. fscanf\_ s ( confsource , "% s % d", trash , \_ countof ( trash ), & num Of Writers ); *// чи*

*сло потоков-писателей*

1. fscanf\_ s ( confsource , "% s % d", trash , \_ countof ( trash ), & writers Delay ); *// за держки потоков-писателей*
2. fscanf\_ s ( confsource , "% s % d", trash , \_ countof ( trash ), & size Of Queue ); *// раз мер очереди*
3. fscanf\_ s ( confsource , "% s % d", trash , \_ countof ( trash ), & ttl ); *// время жизни*

#### 91

1. if ( num Of Readers <= 0 || num Of Writers <= 0) {
2. log -> loudlog (\_T(" Incorrect num of Readers or writers "));

94 exit ( 500) ;

95 }

1. else if ( readers Delay <= 0 || writers Delay <= 0) {
2. log -> loudlog (\_T(" Incorrect delay of Readers or writers "));

98 exit ( 501) ;

99 }

1. else if ( size Of Queue <= 0) {
2. log -> loudlog (\_T(" Incorrect size of queue "));

102 exit ( 502) ;

#### 103 }

1. else if ( ttl == 0) {
2. log -> loudlog (\_T(" Incorrect ttl "));

106 exit ( 503) ;

#### 107 }

108 fclose ( confsource );

#### 109

1. config -> num Of Readers = num Of Readers ;
2. config -> readers Delay = readers Delay ;
3. config -> num Of Writers = num Of Writers ;
4. config -> writers Delay = writers Delay ;
5. config -> size Of Queue = size Of Queue ;
6. config -> ttl = ttl ;

#### 116

1. log -> quietlog (\_T(" Config :\ n\ t Num Of Readers = % d\ n\ t Readers Delay = % d\ n\ t Num Of Writers = % d\ n\ t Writers Delay = % d\ n\ t Size Of Queue = % d\ n\ tttl = % d "),
2. config -> num Of Readers , config -> readers Delay , config -> num Of Writers , config

-> writers Delay , config -> size Of Queue , config -> ttl );

#### 119 }

120

1. void Set Default Config ( struct Configuration \* config , Logger \* log ) {
2. log -> quietlog (\_T(" Using default config "));
3. *// Вид конфигурационного файла:*

|  |  |  |  |
| --- | --- | --- | --- |
| 124 | *//* | *Num Of Readers =* | *10* |
| 125 | *//* | *Readers Delay =* | *100* |
| 126 | *//* | *Num Of Writers =* | *10* |

127

128

129

130

131

132

133

134

135

136

137

138

*//*

*//*

*//*

*Writers Delay = 200*

*Size Of Queue = 10*

*ttl= 3*

config -> num Of Readers

config -> readers Delay config -> num Of Writers config -> writers Delay

=

=

=

=

10;

100;

10;

200;

config -> size Of Queue = 10;

config -> ttl = 3;

139

log -> quietlog (\_T(" Config :\ n\ t Num Of Readers = % d\ n\ t Readers Delay = % d\ n\

t Num Of Writers = % d\ n\ t Writers Delay = % d\ n\ t Size Of Queue = % d\ n\ tttl = % d "),

config -> num Of Readers , config -> readers Delay , config -> num Of Writers , config

-> writers Delay , config -> size Of Queue , config -> ttl );

140 }

## Использование мьютексов

### Объекты ядра «мьютексы» гарантируют потокам взаимоисключающий доступ к един- ственному ресурсу. Это отражено в название 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 крити- ческие секции. Однако, если последние являются объектами пользовательского режима, то мьютексы — объектами ядра. Кроме того, единственный объект мью текс 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 объек- ты 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 использу- ется 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

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

# include

# include

# include

# include

# include

< windows . h>

< string . h>

< stdio . h>

< conio . h>

< tchar . h>

# include

# include

# include

* thread . h"
* utils . h" " Logger . h"

*// глобальные переменные:*

struct FIFOQueue queue ; *// структура очереди*

struct Configuration bool is Done = false ; HANDLE \* allhandlers ;

config ; *// конфигурация программы*

*// Признак завершения*

*// массив всех создаваемых потоков*

HANDLE mutex ; *// описатель мьютекса*

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. Logger log (\_T(" Mutex "));

#### 20

1. if ( argc < 2)
2. *// Используем конфигурацию по- умолчанию*
3. Set Default Config (& config , & log );
4. else
5. *// Загрузка конфига из файла*
6. Set Config ( argv [1] , & config , & log );

#### 27

1. *// создаем необходимые потоки без их запуска*
2. Create All Threads (& config , & log );

#### 30

1. *// Инициализируем очередь*
2. queue . full = 0;
3. queue . readindex = 0;
4. queue . writeindex = 0;
5. queue . size = config . size Of Queue ;
6. queue . data = new \_ TCHAR \*[ config . size Of Queue ];
7. *// инициализируем средство синхронизации*
8. mutex = Create Mutex ( NULL , FALSE , L"");
9. *// NULL - параметры безопасности*
10. *// FALSE - создаваемый мьютекс никому изначально не принадлежит*
11. *// "" - имя мьютекса*

#### 42

1. *// запускаем потоки на исполнение*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Resume Thread ( allhandlers [ i]);

#### 46

1. *// ожидаем завершения всех потоков*
2. Wait For Multiple Objects ( config . num Of Readers + config . num Of Writers + 1 ,
3. allhandlers , TRUE , INFINITE );
4. *// закрываем handle потоков*
5. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
6. Close Handle ( allhandlers [ i]);
7. *// удаляем объект синхронизации*
8. Close Handle ( mutex );

#### 55

1. *// Очистка памяти*
2. for ( size\_ t i = 0; i != config . size Of Queue ; ++ i)
3. if ( queue . data [ i])
4. free ( queue . data [ i]); *// \_ wcsdup использует calloc*
5. delete [] queue . data ;

#### 61

62 *// Завершение работы*

1. log . loudlog (\_T(" All tasks are done !"));
2. \_ getch ();
3. 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

1. DWORD WINAPI Thread Reader Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Mutex . Thread Reader "), myid );
2. extern bool is Done ;
3. extern struct FIFOQueue queue ;
4. extern struct Configuration config ;
5. extern HANDLE mutex ;

#### 15

1. while ( is Done != true ) {
2. *// Захват объекта синхронизации*
3. log . quietlog (\_T(" Waining for Mutex "));
4. Wait For Single Object ( mutex , INFINITE );
5. log . quietlog (\_T(" Get Mutex "));

#### 21

1. *// если в очереди есть данные*
2. if ( queue . readindex != queue . writeindex || queue . full == 1) {
3. *// взяли данные, значит очередь не пуста*
4. queue . full = 0;
5. *// печатаем принятые данные*
6. log . loudlog (\_T(" Reader % d get data : \"% s\" from position % d"), myid ,
7. queue . data [ queue . readindex ], queue . readindex );
8. free ( queue . data [ queue . readindex ]); *// очищаем очередь от данных*
9. queue . data [ queue . readindex ] = NULL ;
10. queue . readindex = ( queue . readindex + 1) % queue . size ;

32 }

1. *// Освобождение объекта синхронизации*
2. log . quietlog (\_T(" Release Mutex "));
3. Release Mutex ( mutex );

#### 36

37 *// задержка*

38 Sleep ( config . readers Delay );

39 }

1. log . loudlog (\_T(" Reader % d finishing work "), myid );
2. 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

1. DWORD WINAPI Thread Writer Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Threads Reader Writer . Thread Writer "), myid );
2. extern bool is Done ;
3. extern struct FIFOQueue queue ;
4. extern struct Configuration config ;
5. extern HANDLE mutex ;

#### 15

1. \_ TCHAR tmp [50];
2. int msgnum = 0; *// номер передаваемого сообщения*
3. while ( is Done != true ) {
4. *// Захват синхронизирующего объекта*
5. log . quietlog (\_T(" Waining for Mutex "));
6. Wait For Single Object ( mutex , INFINITE );
7. log . quietlog (\_T(" Get Mutex "));

#### 23

1. *// если в очереди есть место*
2. if ( queue . readindex != queue . writeindex || ! queue . full == 1) {
3. *// заносим в очередь данные*
4. swprintf\_ s ( tmp , \_T(" writer\_ id = % d num Msg = %3 d"), myid , msgnum );
5. queue . data [ queue . writeindex ] = \_ wcsdup ( tmp );
6. msgnum ++;

#### 30

1. *// печатаем принятые данные*
2. log . loudlog (\_T(" Writer % d put data : \"% s\" in position % d"), myid ,
3. queue . data [ queue . writeindex ], queue . writeindex );
4. queue . writeindex = ( queue . writeindex + 1) % queue . size ;
5. *// если очередь заполнилась*
6. queue . full = queue . writeindex == queue . readindex ? 1 : 0;

37

38

39

40

41

42

43

44

45

46

47

}

*// освобождение объекта синхронизации* log . quietlog (\_T(" Release Mutex ")); Release Mutex ( mutex );

*// задержка*

Sleep ( config . writers Delay );

}

log . loudlog (\_T(" Writer % d finishing work "), myid ); return 0;

}

### На рисунке 1 показана работа потоков с мьютексом.

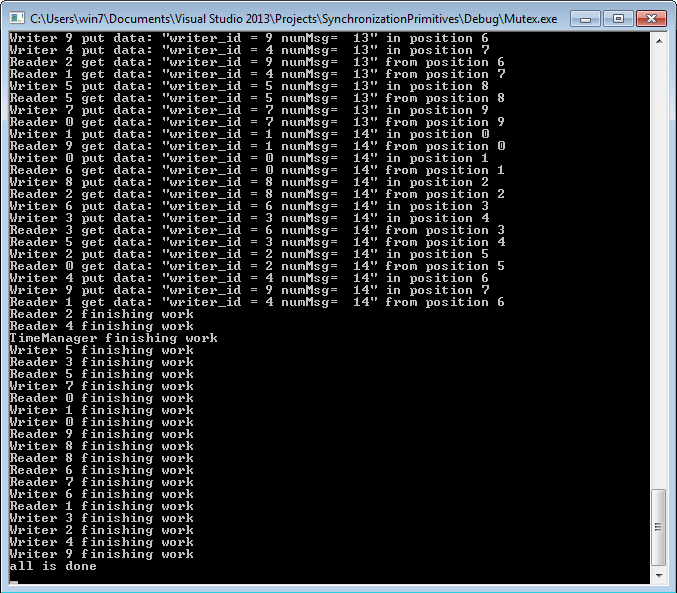


Рис. 1: Использование мьютексов.

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.

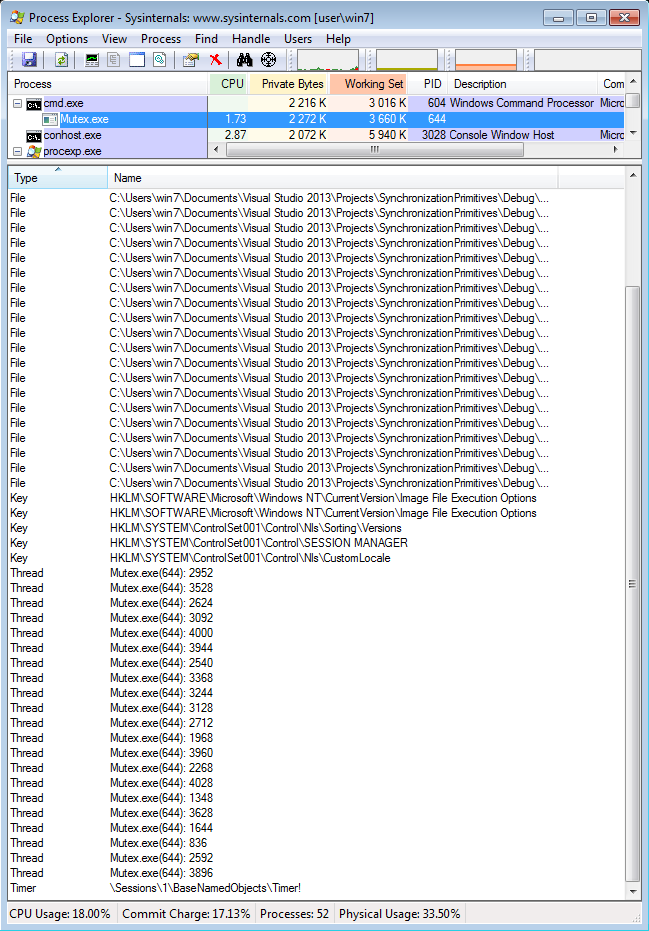


Рис. 2: Потоки и файлы с логами.

* 1. **Использование семафоров**

Объекты ядра «семафор» используются для учета ресурсов. Он содержат счетчик числа пользователей, но, кроме того, поддерживает два 32 битных значения со знаком: одно определяет максимальное число ресурсов (контролируемое семафором), другое использу- ется как счетчик текущего числа ресурсов. Таким образом, 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. 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 инициа- лизируется, 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

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

# include

# include

# include

# include

# include

< windows . h>

< string . h>

< stdio . h>

< conio . h>

< tchar . h>

# include

# include

# include

* thread . h"
* utils . h" " Logger . h"

*// глобальные переменные:*

struct FIFOQueue queue ; *// структура очереди*

struct Configuration bool is Done = false ; HANDLE \* allhandlers ;

config ; *// конфигурация программы*

*// Признак завершения*

*// массив всех создаваемых потоков*

HANDLE sem ; *// описатель семафора*

int \_ tmain ( int argc , \_ TCHAR \* argv []) { Logger log (\_T(" Semaphore "));

if ( argc < 2)

*// Используем конфигурацию по- умолчанию*

Set Default Config (& config , & log ); else

1. *// Загрузка конфига из файла*
2. Set Config ( argv [1] , & config , & log );

#### 27

1. *// создаем необходимые потоки без их запуска*
2. Create All Threads (& config , & log );

#### 30

1. *// Инициализируем очередь*
2. queue . full = 0;
3. queue . readindex = 0;
4. queue . writeindex = 0;
5. queue . size = config . size Of Queue ;
6. queue . data = new \_ TCHAR \*[ config . size Of Queue ];
7. *// инициализируем средство синхронизации*
8. sem = Create Semaphore ( NULL , 1 , 1 , L""); *// изначально семафор свободен*
9. *// NULL - аттрибуты безопасности*
10. *// 1 - Сколько свободно ресурсов в начале*
11. *// 1 - Сколько ресурсов всего*

42 *// "" - Имя*

#### 43

1. *// запускаем потоки на исполнение*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Resume Thread ( allhandlers [ i]);

#### 47

1. *// ожидаем завершения всех потоков*
2. Wait For Multiple Objects ( config . num Of Readers + config . num Of Writers + 1 ,
3. allhandlers , TRUE , INFINITE );
4. *// закрываем handle потоков*
5. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
6. Close Handle ( allhandlers [ i]);
7. *// удаляем объект синхронизации*
8. Close Handle ( sem );

#### 56

1. *// Очистка памяти*
2. for ( size\_ t i = 0; i != config . size Of Queue ; ++ i)
3. if ( queue . data [ i])
4. free ( queue . data [ i]); *// \_ wcsdup использует calloc*
5. delete [] queue . data ;

#### 62

1. *// Завершение работы*
2. log . loudlog (\_T(" All tasks are done !"));
3. \_ getch ();
4. 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

1. DWORD WINAPI Thread Writer Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Semaphore . Thread Writer "), myid );
2. extern bool is Done ;
3. extern struct FIFOQueue queue ;
4. extern struct Configuration config ;
5. extern HANDLE sem ;

#### 15

1. \_ TCHAR tmp [50];
2. int msgnum = 0; *// номер передаваемого сообщения*
3. while ( is Done != true ) {
4. *// Захват синхронизирующего объекта*
5. log . quietlog (\_T(" Waining for Semaphore "));
6. Wait For Single Object ( sem , INFINITE );
7. log . quietlog (\_T(" Get Semaphore "));

#### 23

1. *// если в очереди есть место*
2. if ( queue . readindex != queue . writeindex || ! queue . full == 1) {
3. *// заносим в очередь данные*
4. swprintf\_ s ( tmp , \_T(" writer\_ id = % d num Msg = %3 d"), myid , msgnum );
5. queue . data [ queue . writeindex ] = \_ wcsdup ( tmp );
6. msgnum ++;

#### 30

1. *// печатаем принятые данные*
2. log . loudlog (\_T(" Writer % d put data : \"% s\" in position % d"), myid ,
3. queue . data [ queue . writeindex ], queue . writeindex );
4. queue . writeindex = ( queue . writeindex + 1) % queue . size ;
5. *// если очередь заполнилась*
6. queue . full = queue . writeindex == queue . readindex ? 1 : 0;

37 }

1. *// освобождение объекта синхронизации*
2. log . quietlog (\_T(" Release Semaphore "));
3. Release Semaphore ( sem , 1 , NULL );

#### 41

42 *// задержка*

43 Sleep ( config . writers Delay );

44 }

1. log . loudlog (\_T(" Writer % d finishing work "), myid );
2. 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

1. DWORD WINAPI Thread Reader Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Semaphore . Thread Reader "), myid );
2. extern bool is Done ;
3. extern struct FIFOQueue queue ;
4. extern struct Configuration config ;
5. extern HANDLE sem ;

#### 15

1. while ( is Done != true ) {
2. *// Захват объекта синхронизации*
3. log . quietlog (\_T(" Waining for Semaphore "));
4. Wait For Single Object ( sem , INFINITE );
5. log . quietlog (\_T(" Get Semaphore "));

#### 21

1. *// если в очереди есть данные*
2. if ( queue . readindex != queue . writeindex || queue . full == 1) {
3. *// взяли данные, значит очередь не пуста*
4. queue . full = 0;
5. *// печатаем принятые данные*
6. log . loudlog (\_T(" Reader % d get data : \"% s\" from position % d"), myid ,
7. queue . data [ queue . readindex ], queue . readindex );
8. free ( queue . data [ queue . readindex ]); *// очищаем очередь от данных*
9. queue . data [ queue . readindex ] = NULL ;
10. queue . readindex = ( queue . readindex + 1) % queue . size ;

32 }

1. *// Освобождение объекта синхронизации*
2. log . quietlog (\_T(" Release Semaphore "));
3. Release Semaphore ( sem , 1 , NULL );

#### 36

1. *// задержка*
2. Sleep ( config . readers Delay );

39 }

1. log . loudlog (\_T(" Reader % d finishing work "), myid );
2. return 0;

42

}

Результат работы на рисунке 3.

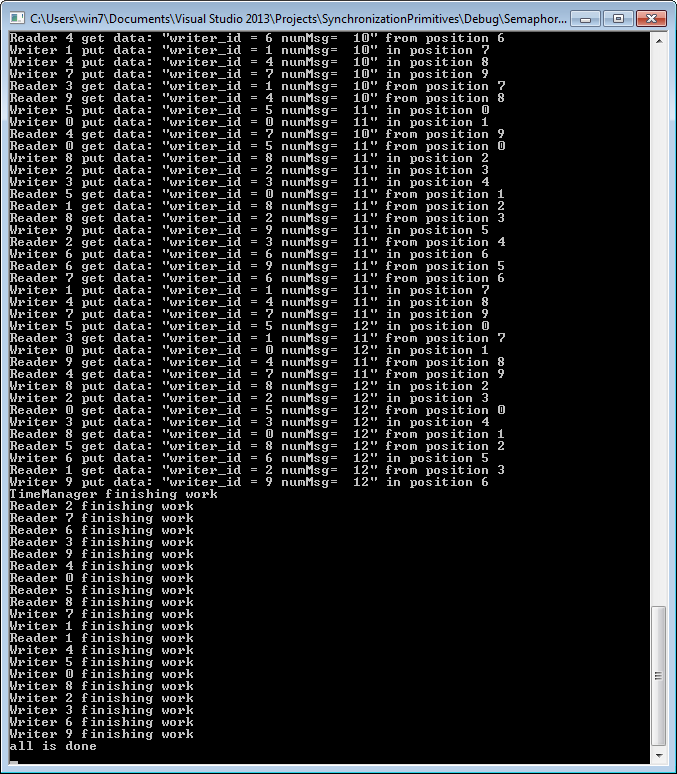


Рис. 3: Использование семафоров.

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 :

* 1. Num Of Readers = 10
  2. Readers Delay = 100
  3. Num Of Writers = 10
  4. Writers Delay = 200
  5. Size Of Queue = 10
  6. 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 Time Manager

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 . Thread Writer 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 . Thread Reader 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 num Msg =

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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 num Msg = 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 .

## Критические секции

### Критическая секция (critical section) – это небольшой участок кода, который должен использоваться только одним потоком одновременно. Если в одно время несколько пото- ков 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

1. # include " thread . h"
2. # include " utils . h"
3. # include " Logger . h"

#### 10

1. *// глобальные переменные:*
2. struct FIFOQueue queue ; *// структура очереди*
3. struct Configuration config ; *// конфигурация программы*
4. bool is Done = false ; *// Признак завершения*
5. HANDLE \* allhandlers ; *// массив всех создаваемых потоков*
6. CRITICAL\_ SECTION crs ; *// Объявление критической секции*

#### 17

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. Logger log (\_T(" Critical Section "));

#### 20

1. if ( argc < 2)
2. *// Используем конфигурацию по- умолчанию*
3. Set Default Config (& config , & log );
4. else
5. *// Загрузка конфига из файла*
6. Set Config ( argv [1] , & config , & log );

#### 27

1. *// создаем необходимые потоки без их запуска*
2. Create All Threads (& config , & log );

#### 30

1. *// Инициализируем очередь*
2. queue . full = 0;
3. queue . readindex = 0;
4. queue . writeindex = 0;
5. queue . size = config . size Of Queue ;
6. queue . data = new \_ TCHAR \*[ config . size Of Queue ];
7. *// инициализируем средство синхронизации*
8. Initialize Critical Section (& crs );

#### 39

1. *// запускаем потоки на исполнение*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Resume Thread ( allhandlers [ i]);

#### 43

1. *// ожидаем завершения всех потоков*
2. Wait For Multiple Objects ( config . num Of Readers + config . num Of Writers + 1 ,
3. allhandlers , TRUE , INFINITE );
4. *// закрываем handle потоков*
5. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
6. Close Handle ( allhandlers [ i]);
7. *// удаляем объект синхронизации*
8. Delete Critical Section (& crs );

#### 52

1. *// Очистка памяти*
2. for ( size\_ t i = 0; i != config . size Of Queue ; ++ i)
3. if ( queue . data [ i])
4. free ( queue . data [ i]); *// \_ wcsdup использует calloc*
5. delete [] queue . data ;

#### 58

1. *// Завершение работы*
2. log . loudlog (\_T(" All tasks are done !"));
3. \_ getch ();
4. 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

1. DWORD WINAPI Thread Writer Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Critical Section . Thread Writer "), myid );
2. extern bool is Done ;
3. extern struct FIFOQueue queue ;
4. extern struct Configuration config ;
5. extern CRITICAL\_ SECTION crs ;

#### 15

1. \_ TCHAR tmp [50];
2. int msgnum = 0; *// номер передаваемого сообщения*
3. while ( is Done != true ) {
4. *// Захват синхронизирующего объекта*
5. log . quietlog (\_T(" Waining for Critical Section "));
6. Enter Critical Section (& crs );
7. log . quietlog (\_T(" Get Critical Section "));

#### 23

1. *// если в очереди есть место*
2. if ( queue . readindex != queue . writeindex || ! queue . full == 1) {
3. *// заносим в очередь данные*
4. swprintf\_ s ( tmp , \_T(" writer\_ id = % d num Msg = %3 d"), myid , msgnum );
5. queue . data [ queue . writeindex ] = \_ wcsdup ( tmp );
6. msgnum ++;

#### 30

1. *// печатаем принятые данные*
2. log . loudlog (\_T(" Writer % d put data : \"% s\" in position % d"), myid ,
3. queue . data [ queue . writeindex ], queue . writeindex );
4. queue . writeindex = ( queue . writeindex + 1) % queue . size ;
5. *// если очередь заполнилась*
6. queue . full = queue . writeindex == queue . readindex ? 1 : 0;

37 }

1. *// освобождение объекта синхронизации*
2. log . quietlog (\_T(" Leave Critical Section "));
3. Leave Critical Section (& crs );

#### 41

1. *// задержка*
2. Sleep ( config . writers Delay );

44 }

1. log . loudlog (\_T(" Writer % d finishing work "), myid );
2. return 0;

47 }

### Листинг 14: Потоки читатели, 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

1. DWORD WINAPI Thread Reader Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Critical Section . Thread Reader "), myid );
2. extern bool is Done ;
3. extern struct FIFOQueue queue ;
4. extern struct Configuration config ;
5. extern CRITICAL\_ SECTION crs ;

#### 15

1. while ( is Done != true ) {
2. *// Захват объекта синхронизации*
3. log . quietlog (\_T(" Waining for Critical Section "));
4. Enter Critical Section (& crs );
5. log . quietlog (\_T(" Get Critical Section "));

#### 21

1. *// если в очереди есть данные*
2. if ( queue . readindex != queue . writeindex || queue . full == 1) {
3. *// взяли данные, значит очередь не пуста*
4. queue . full = 0;
5. *// печатаем принятые данные*
6. log . loudlog (\_T(" Reader % d get data : \"% s\" from position % d"), myid ,
7. queue . data [ queue . readindex ], queue . readindex );
8. free ( queue . data [ queue . readindex ]); *// очищаем очередь от данных*
9. queue . data [ queue . readindex ] = NULL ;
10. queue . readindex = ( queue . readindex + 1) % queue . size ;

32 }

1. *// Освобождение объекта синхронизации*
2. log . quietlog (\_T(" Leave Critical Section "));
3. Leave Critical Section (& crs );

#### 36

1. *// задержка*
2. Sleep ( config . readers Delay );

39 }

1. log . loudlog (\_T(" Reader % d finishing work "), myid );
2. return 0;

42 }

### Демонстрация работы синхронизации через критическую секцию показана на рисунке 4.

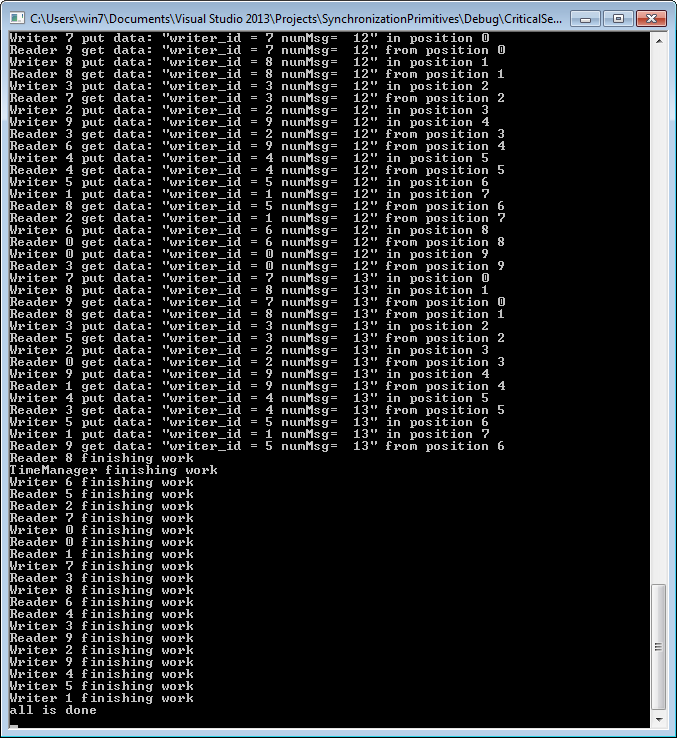


Рис. 4: работа с критической секцией

* 1. **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 сколь угодно долгое время, вне зависимости от того, сколько потоков ожидают установки этого события. Когда все потоки, ожидающие этого события, получат 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

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

# include

# include

# include

# include

# include

< windows . h>

< string . h>

< stdio . h>

< conio . h>

< tchar . h>

# include

# include

# include

* thread . h"
* utils . h" " Logger . h"

*// глобальные переменные:*

struct FIFOQueue queue ; *// структура очереди*

struct Configuration bool is Done = false ; HANDLE \* allhandlers ;

config ; *// конфигурация программы*

*// Признак завершения*

*// массив всех создаваемых потоков*

HANDLE event ; *// объект-событие*

int \_ tmain ( int argc , \_ TCHAR \* argv []) {

Logger log (\_T(" Event "));

1. if ( argc < 2)
2. *// Используем конфигурацию по- умолчанию*
3. Set Default Config (& config , & log );
4. else
5. *// Загрузка конфига из файла*
6. Set Config ( argv [1] , & config , & log );

#### 27

1. *// создаем необходимые потоки без их запуска*
2. Create All Threads (& config , & log );

#### 30

1. *// Инициализируем очередь*
2. queue . full = 0;
3. queue . readindex = 0;
4. queue . writeindex = 0;
5. queue . size = config . size Of Queue ;
6. queue . data = new \_ TCHAR \*[ config . size Of Queue ];
7. *// инициализируем средство синхронизации*
8. event = Create Event ( NULL , false , true , L"");
9. *// NULL - атрибуты защиты*
10. *// false - режим переключения ( без автосброса, после захвата*
11. *// потоком события, оно его нужно сделать занятым)*
12. *// true - начальное состояние события ( свободное)*

43 *// "" - имя*

#### 44

1. *// запускаем потоки на исполнение*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Resume Thread ( allhandlers [ i]);

#### 48

1. *// ожидаем завершения всех потоков*
2. Wait For Multiple Objects ( config . num Of Readers + config . num Of Writers + 1 ,
3. allhandlers , TRUE , INFINITE );
4. *// закрываем handle потоков*
5. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
6. Close Handle ( allhandlers [ i]);
7. *// удаляем объект синхронизации*
8. Close Handle ( event );

#### 57

1. *// Очистка памяти*
2. for ( size\_ t i = 0; i != config . size Of Queue ; ++ i)
3. if ( queue . data [ i])
4. free ( queue . data [ i]); *// \_ wcsdup использует calloc*
5. delete [] queue . data ;

#### 63

1. *// Завершение работы*
2. log . loudlog (\_T(" All tasks are done !"));
3. \_ getch ();
4. 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

1. DWORD WINAPI Thread Writer Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Event . Thread Writer "), myid );
2. extern bool is Done ;
3. extern struct FIFOQueue queue ;
4. extern struct Configuration config ;
5. extern HANDLE event ;

#### 15

1. \_ TCHAR tmp [50];
2. int msgnum = 0; *// номер передаваемого сообщения*
3. while ( is Done != true ) {
4. *// Захват синхронизирующего объекта*
5. log . quietlog (\_T(" Waining for Event "));
6. Wait For Single Object ( event , INFINITE );
7. log . quietlog (\_T(" Get Event "));

#### 23

1. *// если в очереди есть место*
2. if ( queue . readindex != queue . writeindex || ! queue . full == 1) {
3. *// заносим в очередь данные*
4. swprintf\_ s ( tmp , \_T(" writer\_ id = % d num Msg = %3 d"), myid , msgnum );
5. queue . data [ queue . writeindex ] = \_ wcsdup ( tmp );
6. msgnum ++;

#### 30

1. *// печатаем принятые данные*
2. log . loudlog (\_T(" Writer % d put data : \"% s\" in position % d"), myid ,
3. queue . data [ queue . writeindex ], queue . writeindex );
4. queue . writeindex = ( queue . writeindex + 1) % queue . size ;
5. *// если очередь заполнилась*
6. queue . full = queue . writeindex == queue . readindex ? 1 : 0;

37 }

38 *// освобождение объекта синхронизации*

39

40

41

42

43

44

45

46

47

log . quietlog (\_T(" Set Event "));

Set Event ( event );

*// задержка*

Sleep ( config . writers Delay );

}

log . loudlog (\_T(" Writer % d finishing work "), myid ); return 0;

}

### 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

1. DWORD WINAPI Thread Reader Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Event . Thread Reader "), myid );
2. extern bool is Done ;
3. extern struct FIFOQueue queue ;
4. extern struct Configuration config ;
5. extern HANDLE event ;

#### 15

1. while ( is Done != true ) {
2. *// Захват объекта синхронизации*
3. log . quietlog (\_T(" Waining for Event "));
4. Wait For Single Object ( event , INFINITE );
5. log . quietlog (\_T(" Get Event "));

#### 21

1. *// если в очереди есть данные*
2. if ( queue . readindex != queue . writeindex || queue . full == 1) {
3. *// взяли данные, значит очередь не пуста*
4. queue . full = 0;
5. *// печатаем принятые данные*
6. log . loudlog (\_T(" Reader % d get data : \"% s\" from position % d"), myid ,
7. queue . data [ queue . readindex ], queue . readindex );
8. free ( queue . data [ queue . readindex ]); *// очищаем очередь от данных*
9. queue . data [ queue . readindex ] = NULL ;
10. queue . readindex = ( queue . readindex + 1) % queue . size ;

32 }

33

34

35

36

37

38

39

40

41

42

*// Освобождение объекта синхронизации*

log . quietlog (\_T(" Release Event ")); Set Event ( event );

*// задержка*

Sleep ( config . readers Delay );

}

log . loudlog (\_T(" Reader % d finishing work "), myid ); return 0;

}

### Рисунок 5 содержит результат работы, дублирующий лог программы.

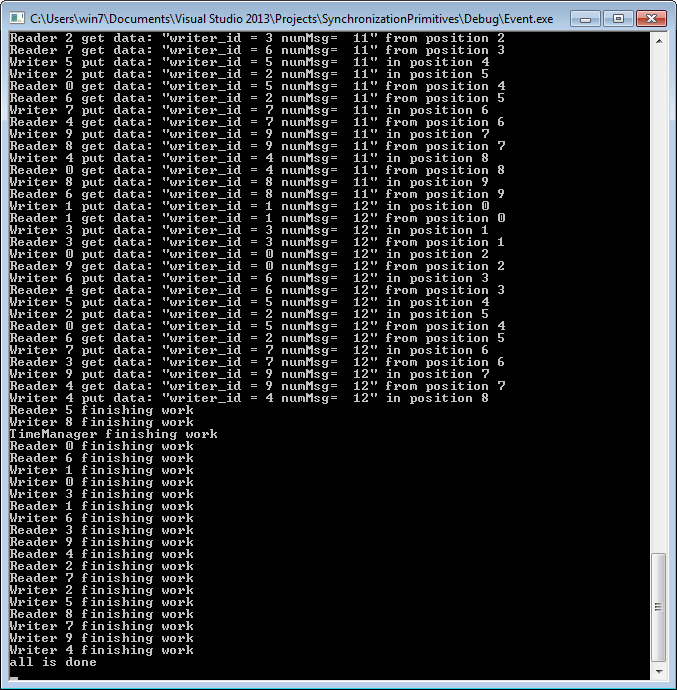


Рис. 5: Объекты-события в качестве средства синхронизации.

* 1. **Условные переменные**

Условные переменные – это объекты, поддерживающие операции ожидания и уведомления. Ожидание возможно только внутри какой-либо критической секции, то есть данная опера- ция 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

1. # include " thread . h"
2. # include " utils . h"
3. # include " Logger . h"

#### 10

1. *// глобальные переменные:*
2. struct FIFOQueue queue ; *// структура очереди*
3. struct Configuration config ; *// конфигурация программы*
4. bool is Done = false ; *// Признак завершения*
5. HANDLE \* allhandlers ; *// массив всех создаваемых потоков*

#### 16

1. *// критическая секция общая и для писателей и для читателей*
2. CRITICAL\_ SECTION crs ;
3. *// условная переменная для потоков- писателей*
4. CONDITION\_ VARIABLE condread ;
5. *// условная переменная для потоков- читателей*
6. CONDITION\_ VARIABLE condwrite ;

#### 23

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. Logger log (\_T(" Condition Variable "));

#### 26

1. if ( argc < 2)
2. *// Используем конфигурацию по- умолчанию*
3. Set Default Config (& config , & log );
4. else
5. *// Загрузка конфига из файла*
6. Set Config ( argv [1] , & config , & log );

#### 33

1. *// создаем необходимые потоки без их запуска*
2. Create All Threads (& config , & log );

#### 36

1. *// Инициализируем очередь*
2. queue . full = 0;
3. queue . readindex = 0;
4. queue . writeindex = 0;
5. queue . size = config . size Of Queue ;
6. queue . data = new \_ TCHAR \*[ config . size Of Queue ];
7. *// инициализируем средство синхронизации*
8. Initialize Critical Section (& crs );
9. Initialize Condition Variable (& condread );
10. Initialize Condition Variable (& condwrite );

#### 47

1. *// запускаем потоки на исполнение*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Resume Thread ( allhandlers [ i]);

#### 51

1. *// ожидаем завершения всех потоков*
2. Wait For Multiple Objects ( config . num Of Readers + config . num Of Writers + 1 ,
3. allhandlers , TRUE , 5000) ;

#### 55

1. *// закрываем handle потоков*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Close Handle ( allhandlers [ i]);
4. *// удаляем объект синхронизации*
5. Delete Critical Section (& crs );

#### 61

1. *// Очистка памяти*
2. for ( size\_ t i = 0; i != config . size Of Queue ; ++ i)
3. if ( queue . data [ i])
4. free ( queue . data [ i]); *// \_ wcsdup использует calloc*
5. delete [] queue . data ;

#### 67

1. *// Завершение работы*
2. log . loudlog (\_T(" All tasks are done !"));
3. \_ getch ();
4. 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

1. DWORD WINAPI Thread Reader Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 8

1. Logger log (\_T(" Condition Variable . Thread Reader "), myid );
2. extern bool is Done ;
3. extern struct FIFOQueue queue ;
4. extern struct Configuration config ;
5. extern CRITICAL\_ SECTION crs ;
6. extern CONDITION\_ VARIABLE condread ;
7. extern CONDITION\_ VARIABLE condwrite ;

#### 16

1. while ( is Done != true ) {
2. *// Захват объекта синхронизации*
3. log . quietlog (\_T(" Waining for Critical Section "));
4. Enter Critical Section (& crs );
5. log . quietlog (\_T(" Get Critical Section "));

#### 22

1. log . quietlog (\_T(" Waining for empty space in the queue "));
2. while (!( queue . readindex != queue . writeindex || queue . full == 1))
3. *// спим пока в очереди не появятся данные*
4. Sleep Condition Variable CS (& condread , & crs , INFINITE );
5. log . quietlog (\_T(" Get space in the queue "));

#### 28

1. *// взяли данные, значит очередь не пуста*
2. queue . full = 0;
3. *// печатаем принятые данные*
4. log . loudlog (\_T(" Reader % d get data : \"% s\" from position % d"), myid ,
5. queue . data [ queue . readindex ], queue . readindex );
6. free ( queue . data [ queue . readindex ]); *// очищаем очередь от данных*
7. queue . data [ queue . readindex ] = NULL ;
8. queue . readindex = ( queue . readindex + 1) % queue . size ;

#### 37

1. *// шлем сигнал потокам-читателям*
2. log . quietlog (\_T(" Wake Condition Variable "));
3. Wake Condition Variable (& condwrite );
4. *// освобождение синхронизируемого объекта*
5. log . quietlog (\_T(" Leave Critical Section "));
6. Leave Critical Section (& crs );

#### 44

1. *// задержка*
2. Sleep ( config . readers Delay );

47 }

48 log . loudlog (\_T(" Reader % d finishing work "), myid );

49 return 0;

50 }

### Листинг 20: Потоки писатели, 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

1. DWORD WINAPI Thread Writer Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Condition Variable . Thread Writer "), myid );
2. extern bool is Done ;
3. extern struct FIFOQueue queue ;
4. extern struct Configuration config ;
5. extern CRITICAL\_ SECTION crs ;
6. extern CONDITION\_ VARIABLE condread ;
7. extern CONDITION\_ VARIABLE condwrite ;

#### 17

1. \_ TCHAR tmp [50];
2. int msgnum = 0; *// номер передаваемого сообщения*
3. while ( is Done != true ) {
4. *// Захват синхронизирующего объекта*
5. log . quietlog (\_T(" Waining for Critical Section "));
6. Enter Critical Section (& crs );
7. log . quietlog (\_T(" Get Critical Section "));

#### 25

1. log . quietlog (\_T(" Waining for empty space in the queue "));
2. while (!( queue . readindex != queue . writeindex || ! queue . full == 1))
3. *// спим пока в очереди не освободится место*
4. Sleep Condition Variable CS (& condwrite , & crs , INFINITE );
5. log . quietlog (\_T(" Get space in the queue "));

#### 31

1. *// заносим в очередь данные*
2. swprintf\_ s ( tmp , \_T(" writer\_ id = % d num Msg = %3 d"), myid , msgnum );
3. queue . data [ queue . writeindex ] = \_ wcsdup ( tmp );
4. msgnum ++;

#### 36

1. *// печатаем принятые данные*
2. log . loudlog (\_T(" Writer % d put data : \"% s\" in position % d"), myid ,
3. queue . data [ queue . writeindex ], queue . writeindex );

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

queue . writeindex = ( queue . writeindex + 1) % queue . size ;

*// если очередь заполнилась*

queue . full = queue . writeindex == queue . readindex ? 1 : 0;

if ( queue . full == 1)

log . loudlog (\_T(" Queue is full "));

*// шлем сигнал потокам-читателям*

log . quietlog (\_T(" Wake Condition Variable ")); Wake Condition Variable (& condread );

*// освобождение синхронизируемого объекта*

log . quietlog (\_T(" Leave Critical Section ")); Leave Critical Section (& crs );

*// задержка*

Sleep ( config . writers Delay );

}

log . loudlog (\_T(" Writer % d finishing work "), myid ); return 0;

}

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

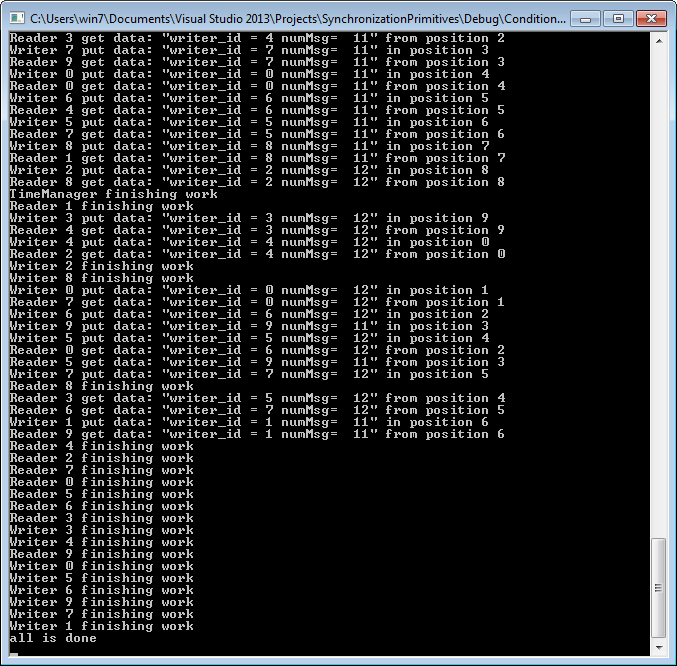


Рис. 6: Условные переменные и критические секции.

* 1. **Задача читатели-писатели (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 – раз- деляемую 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. Примитивы 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

1. # include " thread . h"
2. # include " utils . h"
3. # include " Logger . h"

#### 10

1. *// глобальные переменные:*
2. struct Configuration config ; *// конфигурация программы*
3. bool is Done = false ; *// флаг завершения*
4. HANDLE \* allhandlers ; *// массив всех создаваемых потоков*

#### 15

1. *// события для синхронизации:*
2. HANDLE can Read Event ; *// писатель записал сообщение ( ручной сброс);*
3. HANDLE can Write Event ; *// все читатели готовы к приему следующего ( автосброс);*
4. HANDLE all Read Event ; *// все читатели прочитали сообщение ( ручной сброс);*
5. HANDLE change Count Event ; *// разрешение работы со счетчиком ( автосброс);*
6. HANDLE exit Event ; *// завершение программы ( ручной сброс);*

#### 22

1. *// переменные для синхронизации работы потоков:*
2. int countread = 0; *// число потоков, которое уже прочитали данные*
3. *// ( устанавливается писателем и изменяется*
4. *// читателями после прочтения сообщения)*
5. int countready = 0; *// число потоков, готовых для чтения сообщения*
6. *// ( ожидающих сигнала от писателя)*

#### 29

1. *// имя разделяемой памяти*
2. wchar\_ t share File Name [] = L" $$ My Very Special Share File Name$$ ";

#### 32

1. HANDLE h File Mapping ; *// объект-отображение файла*
2. *// указатели на отображаемую память*
3. LPVOID lp File Map For Writers ;
4. LPVOID lp File Map For Readers ;

#### 37

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. Logger log (\_T(" Threads Reader Writer "));

#### 40

1. if ( argc < 2)
2. *// Используем конфигурацию по- умолчанию*
3. Set Default Config (& config , & log );
4. else
5. *// Загрузка конфига из файла*
6. Set Config ( argv [1] , & config , & log );

#### 47

1. *// создаем необходимые потоки без их запуска*
2. Create All Threads (& config , & log );

#### 50

1. *// Инициализируем ресурс ( share memory): создаем объект " отображаемый файл"*
2. *// будет использован системный файл подкачки ( на диске файл создаваться*
3. *// не будет), т. к. в качестве дескриптора файла использовано значение*
4. *// равное 0 x FFFFFFFF ( его эквивалент - символическая константа INVALID\_ HANDLE\_ VALUE )*
5. if (( h File Mapping = Create File Mapping ( INVALID\_ HANDLE\_ VALUE , NULL ,
6. PAGE\_ READWRITE , 0 , 1500 , share File Name )) == NULL ) {
7. *// INVALID\_ HANDLE\_ VALUE - дескриптор открытого файла*
8. *// ( INVALID\_ HANDLE\_ VALUE - файл подкачки)*
9. *// NULL - атрибуты защиты объекта- отображения*
10. *// PAGE\_ READWRITE - озможности доступа к представлению файла при*
11. *// отображении ( PAGE\_ READWRITE - чтение/ запись)*
12. *// 0 , 1500 - старшая и младшая части значения максимального*
13. *// размера объекта отображения файла*
14. *// share File Name - имя объекта- отображения.*
15. log . loudlog (\_T(" Impossible to create shareFile , GLE = % d"),
16. Get Last Error ());
17. Exit Process ( 10000 ) ;

68 }

69 *// отображаем файл на адресное пространство нашего процесса для потока- писа*

*теля*

1. lp File Map For Writers = Map View Of File ( h File Mapping , FILE\_ MAP\_ WRITE , 0 , 0 , 0)

;

1. *// h File Mapping - дескриптор объекта-отображения файла*
2. *// FILE\_ MAP\_ WRITE - доступа к файлу*
3. *// 0 , 0 - старшая и младшая части смещения начала отображаемого участка в файле*
4. *// (0 - начало отображаемого участка совпадает с началом файла)*
5. *// 0 - размер отображаемого участка файла в байтах (0 - весь файл)*

#### 76

1. *// отображаем файл на адресное пространство нашего процесса для потоков- чит ателей*
2. lp File Map For Readers = Map View Of File ( h File Mapping , FILE\_ MAP\_ READ , 0 , 0 , 0);

#### 79

1. *// инициализируем средства синхронизации*
2. *// ( атрибуты защиты, автосброс, начальное состояние, имя):*
3. *// событие " окончание записи" ( можно читать), ручной сброс, изначально заня то*
4. can Read Event = Create Event ( NULL , true , false , L"");
5. *// событие - " можно писать", автосброс( разрешаем писать только одному), изна чально свободно*
6. can Write Event = Create Event ( NULL , false , false , L"");
7. *// событие " все прочитали"*
8. all Read Event = Create Event ( NULL , true , true , L"");
9. *// событие для изменения счетчика ( сколько клиентов еще не прочитало сообще ние)*
10. change Count Event = Create Event ( NULL , false , true , L"");
11. *// событие " завершение работы программы", ручной сброс, изначально занято*
12. exit Event = Create Event ( NULL , true , false , L"");

#### 92

1. *// запускаем потоки- писатели и поток- планировщик на исполнение*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Resume Thread ( allhandlers [ i]);

#### 96

1. *// ожидаем завершения всех потоков*
2. Wait For Multiple Objects ( config . num Of Readers + config . num Of Writers + 1 ,
3. allhandlers , TRUE , INFINITE );

#### 100

1. *// закрываем handle потоков*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Close Handle ( allhandlers [ i]);

#### 104

1. *// закрываем описатели объектов синхронизации*
2. Close Handle ( can Read Event );
3. Close Handle ( can Write Event );

|  |  |  |
| --- | --- | --- |
| 108 |  | Close Handle ( all Read Event ); |
| 109 |  | Close Handle ( change Count Event ); |
| 110 |  | Close Handle ( exit Event ); |
| 111 |  |  |
| 112 |  | *// закрываем handle общего ресурса* |
| 113 |  | Unmap View Of File ( lp File Map For Readers ); |
| 114 |  | Unmap View Of File ( lp File Map For Writers ); |
| 115 |  |  |
| 116 |  | *// закрываем объект " отображаемый файл"* |
| 117 |  | Close Handle ( h File Mapping ); |
| 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

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

# include

# include

# include

< windows . h>

< stdio . h>

< tchar . h>

# include

" utils . h"

DWORD WINAPI Thread Writer Handler ( LPVOID prm ) {

int myid = ( int ) prm ;

Logger extern extern

log (\_T(" Threads Reader Writer . Thread Writer "),

bool is Done ;

myid );

struct

Configuration config ;

extern HANDLE can Read Event ;

extern HANDLE can Write Event ; extern HANDLE exit Event ;

extern int countread ;

extern LPVOID lp File Map For Writers ;

int msgnum = 0;

HANDLE writerhandlers [2]; writerhandlers [0] = exit Event ; writerhandlers [1] = can Write Event ;

1. while ( is Done != true ) {
2. log . quietlog (\_T(" Waining for multiple objects "));
3. DWORD dw Event = Wait For Multiple Objects (2 , writerhandlers , false ,
4. INFINITE );
5. *// 2 - следим за 2 - я параметрами*
6. *// writerhandlers - из массива writerhandlers*
7. *// false - жд e¨ м, когда освободится хотя бы один*
8. *// INFINITE - ждать бесконечно*
9. switch ( dw Event ) {
10. case WAIT\_ OBJECT\_ 0 : *// сработало событие exit*
11. log . quietlog (\_T(" Get exit Event "));
12. log . loudlog (\_T(" Writer % d finishing work "), myid );
13. return 0;
14. case WAIT\_ OBJECT\_ 0 + 1: *// сработало событие на возможность записи*
15. log . quietlog (\_T(" Get can Write Event "));
16. *// увеличиваем номер сообщения*
17. msgnum ++;
18. *// число потоков которые должны прочитать сообщение*
19. countread = config . num Of Readers ;
20. *// Запись сообщения*
21. swprintf\_ s (( \_ TCHAR \*) lp File Map For Writers , 1500 ,
22. \_T(" writer\_ id %d, msg with num = % d"), myid , msgnum );
23. log . loudlog (\_T(" writer put msg : \"% s\""), lp File Map For Writers );
24. *// разрешаем читателям прочитать сообщение и опять ставим событие в зан ятое*
25. log . quietlog (\_T(" Set Event can Read Event "));
26. Set Event ( can Read Event );
27. break ;
28. default :
29. log . loudlog (\_T(" Error with func Wait For Multiple Objects in writer Handle

, GLE = % d"), Get Last Error ());

1. Exit Process ( 1000 ) ;

56 }

57 }

1. log . loudlog (\_T(" Writer % d finishing work "), myid );
2. return 0;

60 }

### Листинг 23: Потоки читатели (src/SynchronizationPrimitives/ThreadsReaderWriter/threadReader.cpp)

1

2

3

4

# include

# include

# include

< windows . h>

< stdio . h>

< tchar . h>

5 # include " utils . h"

#### 6

1. DWORD WINAPI Thread Reader Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Threads Reader Writer . Thread Reader "), myid );
2. extern bool is Done ;
3. extern struct Configuration config ;

#### 13

1. extern HANDLE can Read Event ;
2. extern HANDLE can Write Event ;
3. extern HANDLE all Read Event ;
4. extern HANDLE change Count Event ;
5. extern HANDLE exit Event ;

#### 19

1. extern int countread ;
2. extern int countready ;
3. extern LPVOID lp File Map For Readers ;

#### 23

1. HANDLE readerhandlers [2];
2. readerhandlers [0] = exit Event ;
3. readerhandlers [1] = can Read Event ;

#### 27

1. while ( is Done != true ) {
2. *// ждем, пока все прочитают*
3. log . quietlog (\_T(" Waining for all Read Event "));
4. Wait For Single Object ( all Read Event , INFINITE );
5. *// узнаем, сколько потоков-читателей прошло данную границу*
6. log . quietlog (\_T(" Waining for change Count Event "));
7. Wait For Single Object ( change Count Event , INFINITE );
8. countready ++;
9. *// если все прошли, то " закрываем за собой дверь" и разрешаем писать*
10. if ( countready == config . num Of Readers ) {
11. countready = 0;
12. log . quietlog (\_T(" Reset Event all Read Event "));
13. Reset Event ( all Read Event );
14. log . quietlog (\_T(" Set Event can Write Event "));
15. Set Event ( can Write Event );

43 }

#### 44

1. *// разрешаем изменять счетчик*
2. log . quietlog (\_T(" Set Event change Count Event "));
3. Set Event ( change Count Event );

#### 48

1. log . quietlog (\_T(" Waining for multiple objects "));
2. DWORD dw Event = Wait For Multiple Objects (2 , readerhandlers , false ,
3. INFINITE );
4. *// 2 - следим за 2 - я параметрами*
5. *// readerhandlers - из массива readerhandlers*
6. *// false - жд e¨ м, когда освободится хотя бы один*
7. *// INFINITE - ждать бесконечно*
8. switch ( dw Event ) {
9. case WAIT\_ OBJECT\_ 0 : *// сработало событие exit*
10. log . quietlog (\_T(" Get exit Event "));
11. log . loudlog (\_T(" Reader % d finishing work "), myid );
12. return 0;
13. case WAIT\_ OBJECT\_ 0 + 1: *// сработало событие на возможность чтения*
14. log . quietlog (\_T(" Get can Read Event "));
15. *// читаем сообщение*
16. log . loudlog (\_T(" Reader % d read msg \"% s\""), myid ,
17. ( \_ TCHAR \*) lp File Map For Readers );

#### 66

1. *// необходимо уменьшить счетчик количества читателей, которые прочитать еще не успели*
2. log . quietlog (\_T(" Waining for change Count Event "));
3. Wait For Single Object ( change Count Event , INFINITE );
4. countread - -;

#### 71

1. *// если мы последние читали, то запрещаем читать и открываем границу*
2. if ( countread == 0) {
3. log . quietlog (\_T(" Reset Event can Read Event "));
4. Reset Event ( can Read Event );
5. log . quietlog (\_T(" Set Event all Read Event "));
6. Set Event ( all Read Event );

78 }

#### 79

1. *// разрешаем изменять счетчик*
2. log . quietlog (\_T(" Set Event change Count Event "));
3. Set Event ( change Count Event );
4. break ;
5. default :
6. log . loudlog (\_T(" Error with func Wait For Multiple Objects in reader Handle

, GLE = % d"), Get Last Error ());

1. Exit Process ( 1001 ) ;

87 }

88 }

1. log . loudlog (\_T(" Reader % d finishing work "), myid );
2. return 0;

91 }

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

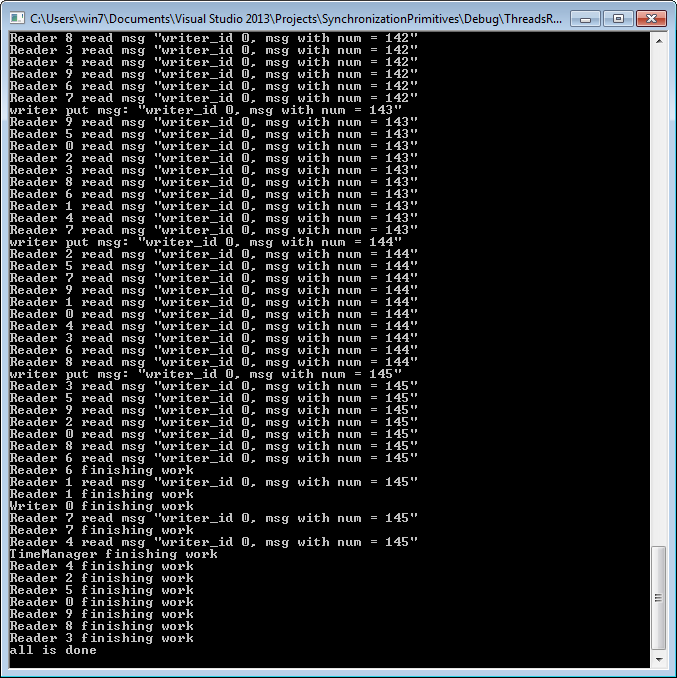


Рис. 7: Задача читатели и писатели.

* 1. **Задача читатели-писатели (для потоков разных процессов)**

В данной программе главный поток и поток-писатель будут принадлежать одному процес- су, 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 пото- ков! 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

1. # include " thread . h"
2. # include " utils . h"
3. # include " Logger . h"

#### 10

1. *// глобальные переменные*
2. struct Configuration config ; *// конфигурация программы*
3. bool is Done = false ; *// флаг завершения*
4. HANDLE \* allhandlers ; *// массив всех создаваемых потоков*

#### 15

1. *// события для синхронизации:*
2. HANDLE can Read Event ; *// писатель записал сообщение ( ручной сброс);*
3. HANDLE can Write Event ; *// все читатели готовы к приему следующего ( автосброс);*
4. HANDLE all Read Event ; *// все читатели прочитали сообщение ( ручной сброс);*
5. HANDLE change Count Event ; *// разрешение работы со счетчиком ( автосброс);*
6. HANDLE exit Event ; *// завершение программы ( ручной сброс);*

#### 22

1. *// переменные для синхронизации работы потоков:*
2. int countread = 0; *// число потоков, которое уже прочитали данные*
3. *// ( устанавливается писателем и изменяется*
4. *// читателями после прочтения сообщения)*
5. int countready = 0; *// число потоков, готовых для чтения сообщения*
6. *// ( ожидающих сигнала от писателя)*

#### 29

1. *// имя разделяемой памяти*
2. wchar\_ t share File Name [] = L" $$ My Very Special Share File Name$$ ";

#### 32

1. HANDLE h File Mapping ; *// объект-отображение файла*
2. LPVOID lp File Map For Writers ; *// указатели на отображаемую память*

#### 35

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. Logger log (\_T(" Process Writer "));

#### 38

1. if ( argc < 2)
2. *// Используем конфигурацию по- умолчанию*
3. Set Default Config (& config , & log );
4. else
5. *// Загрузка конфига из файла*
6. Set Config ( argv [1] , & config , & log );

#### 45

1. *// создаем необходимые потоки без их запуска*
2. *// потоки-читатели запускаются сразу ( чтобы они успели дойти до функции ожи дания)*
3. Create All Threads (& config , & log );

#### 49

1. *// Инициализируем ресурс ( share memory): создаем объект " отображаемый файл"*
2. *// будет использован системный файл подкачки ( на диске файл создаваться*
3. *// не будет), т. к. в качестве дескриптора файла использовано значение*
4. *// равное 0 x FFFFFFFF ( его эквивалент - символическая константа INVALID\_ HANDLE\_ VALUE )*
5. if (( h File Mapping = Create File Mapping ( INVALID\_ HANDLE\_ VALUE , NULL ,
6. PAGE\_ READWRITE , 0 , 1500 , share File Name )) == NULL ) {
7. *// INVALID\_ HANDLE\_ VALUE - дескриптор открытого файла*
8. *// ( INVALID\_ HANDLE\_ VALUE - файл подкачки)*
9. *// NULL - атрибуты защиты объекта- отображения*
10. *// PAGE\_ READWRITE - озможности доступа к представлению файла при*
11. *// отображении ( PAGE\_ READWRITE - чтение/ запись)*
12. *// 0 , 1500 - старшая и младшая части значения максимального*
13. *// размера объекта отображения файла*
14. *// share File Name - имя объекта- отображения.*
15. log . loudlog (\_T(" Impossible to create shareFile , GLE = % d"),
16. Get Last Error ());
17. Exit Process ( 10000 ) ;

67 }

1. *// отображаем файл на адресное пространство нашего процесса для потока- писа теля*
2. lp File Map For Writers = Map View Of File ( h File Mapping , FILE\_ MAP\_ WRITE , 0 , 0 , 0)

;

1. *// h File Mapping - дескриптор объекта-отображения файла*
2. *// FILE\_ MAP\_ WRITE - доступа к файлу*
3. *// 0 , 0 - старшая и младшая части смещения начала отображаемого участка в*

*файле*

1. *// (0 - начало отображаемого участка совпадает с началом файла)*
2. *// 0 - размер отображаемого участка файла в байтах (0 - весь файл)*

#### 75

1. *// инициализируем 2 переменные в общей памяти ( readready и readcount )*
2. \*(( int \*) lp File Map For Writers ) = 0;
3. \*((( int \*) lp File Map For Writers ) + 1) = config . num Of Readers ;

#### 79

1. *// инициализируем средства синхронизации*
2. *// ( атрибуты защиты, автосброс, начальное состояние, имя):*
3. *// событие " окончание записи" ( можно читать), ручной сброс, изначально заня то*
4. can Read Event = Create Event ( NULL , true , false , L" $$ My\_ can Read Event$$ ");
5. *// событие - " можно писать", автосброс( разрешаем писать только одному), изна чально свободно*
6. can Write Event = Create Event ( NULL , false , false , L" $$ My\_ can Write Event$$ ");
7. *// событие " все прочитали"*
8. all Read Event = Create Event ( NULL , true , true , L" $$ My\_ all Read Event$$ ");
9. *// событие для изменения счетчика ( сколько клиентов еще не прочитало сообще ние)*
10. change Count Event = Create Event ( NULL , false , true , L"

$$ My\_ change Count Event$$ ");

1. *// событие " завершение работы программы", ручной сброс, изначально занято*
2. exit Event = Create Event ( NULL , true , false , L" $$ My\_ exit Event$$ ");

#### 92

1. *// запускаем потоки- писатели и поток- планировщик на исполнение*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Resume Thread ( allhandlers [ i]);

#### 96

1. *// ожидаем завершения всех потоков*
2. Wait For Multiple Objects ( config . num Of Readers + config . num Of Writers + 1 ,
3. allhandlers , TRUE , INFINITE );

#### 100

1. *// закрываем handle потоков*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Close Handle ( allhandlers [ i]);

#### 104

1. *// закрываем описатели объектов синхронизации*
2. Close Handle ( can Read Event );
3. Close Handle ( can Write Event );
4. Close Handle ( all Read Event );
5. Close Handle ( change Count Event );
6. Close Handle ( exit Event );

#### 111

112 Unmap View Of File ( lp File Map For Writers ); *// закрываем handle общего ресурса*

|  |  |  |
| --- | --- | --- |
| 113 |  | Close Handle ( h File Mapping ); *// 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

1. DWORD WINAPI Thread Writer Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Process Writer . Thread Writer "), myid );
2. extern bool is Done ;
3. extern struct Configuration config ;

#### 13

1. extern HANDLE can Read Event ;
2. extern HANDLE can Write Event ;
3. extern HANDLE change Count Event ;
4. extern HANDLE exit Event ;

#### 18

1. extern int countread ;
2. extern LPVOID lp File Map For Writers ;

#### 21

1. int msgnum = 0;
2. HANDLE writerhandlers [2];
3. writerhandlers [0] = exit Event ;
4. writerhandlers [1] = can Write Event ;

#### 26

1. while ( is Done != true ) {
2. log . quietlog (\_T(" Waining for multiple objects "));
3. DWORD dw Event = Wait For Multiple Objects (2 , writerhandlers , false ,
4. INFINITE );
5. *// 2 - следим за 2 - я параметрами*
6. *// writerhandlers - из массива writerhandlers*
7. *// false - жд e¨ м, когда освободится хотя бы один*
8. *// INFINITE - ждать бесконечно*
9. switch ( dw Event ) {
10. case WAIT\_ OBJECT\_ 0 : *// сработало событие exit*
11. log . quietlog (\_T(" Get exit Event "));
12. log . loudlog (\_T(" Writer % d finishing work "), myid );
13. return 0;
14. case WAIT\_ OBJECT\_ 0 + 1: *// сработало событие на возможность записи*
15. log . quietlog (\_T(" Get can Write Event "));
16. *// увеличиваем номер сообщения*
17. msgnum ++;

#### 44

1. *// Запись сообщения*
2. swprintf\_ s (( \_ TCHAR \*) lp File Map For Writers + sizeof ( int ) \* 2 , 1500 - sizeof ( int ) \* 2 ,
3. \_T(" Writer\_ id %d, msg with num = % d"), myid , msgnum );
4. log . loudlog (\_T(" Writer put msg : \"% s\""), ( \_ TCHAR \*) lp File Map For Writers + sizeof ( int ) \* 2);

#### 49

1. *// число потоков которые должны прочитать сообщение*
2. log . quietlog (\_T(" Waining for change Count Event "));
3. Wait For Single Object ( change Count Event , INFINITE );
4. \*(( int \*) lp File Map For Writers ) += config . num Of Readers ;
5. \*((( int \*) lp File Map For Writers ) + 1) += config . num Of Readers ;
6. log . quietlog (\_T(" Set Event change Count Event "));
7. Set Event ( change Count Event );

#### 57

1. *// разрешаем потокам- читателям прочитать сообщение и опять ставим событ ие в состояние занято*
2. log . quietlog (\_T(" Set Event can Read Event "));
3. Set Event ( can Read Event );

#### 61

1. break ;
2. default :
3. log . loudlog (\_T(" Error with func Wait For Multiple Objects in writer Handle

, GLE = % d"), Get Last Error ());

1. Exit Process ( 1000 ) ;

66 }

67 }

1. log . loudlog (\_T(" Writer % d finishing work "), myid );
2. return 0;

70 }

### Листинг 26: Запуск клиентских процессов (src/SynchronizationPrimitives/ProcessWriter/utils.cpp)

#### 1

* 1. *// создание всех потоков*
  2. void Create All Threads ( struct Configuration \* config , Logger \* log ) {
  3. extern HANDLE \* allhandlers ;

#### 5

1. int total = config -> num Of Readers + config -> num Of Writers + 1;
2. log -> quietlog (\_T(" Total num of threads is % d"), total );
3. allhandlers = new HANDLE [ total ];
4. int count = 0;

#### 10

1. *// создаем потоки- читатели*
2. log -> loudlog (\_T(" Create readers "));

#### 13

1. STARTUPINFO si;
2. PROCESS\_ INFORMATION pi;

#### 16

1. Zero Memory (& si , sizeof ( si));
2. si. cb = sizeof ( si);
3. Zero Memory (& pi , sizeof ( pi));
4. TCHAR sz Command Line [ 100 ];

#### 21

1. for ( int i = 0; i != config -> num Of Readers ; i++ , count ++) {
2. \_ stprintf\_ s ( sz Command Line , \_T(" Process Reader . exe % d"), i);
3. log -> loudlog (\_T(" Count = % d"), count );
4. if (! Create Process ( NULL , sz Command Line , NULL , NULL , FALSE , CREATE\_ NEW\_ CONSOLE |
5. CREATE\_ SUSPENDED , NULL , NULL , & si , & pi)) {
6. log -> loudlog (\_T(" Impossible to create Process - reader , GLE = % d"), Get Last Error ());

28 exit ( 8000 ) ;

29 }

30 allhandlers [ count ] = pi. h Thread ;

31 }

#### 32

1. *// создаем потоки- писатели*
2. log -> loudlog (\_T(" Create writers "));
3. for ( int i = 0; i != config -> num Of Writers ; i++ , count ++) {
4. log -> loudlog (\_T(" count = % d"), count );
5. *// создаем потоки- читатели, которые пока не стартуют*
6. if (( allhandlers [ count ] = Create Thread ( NULL , 0 , Thread Writer Handler ,
7. ( LPVOID )i, CREATE\_ SUSPENDED , NULL )) == NULL ) {
8. log -> loudlog (\_T(" Impossible to create thread - writer , GLE = % d"),

41

42

43

44

45

46

47

48

49

50

51

Get Last Error ());

exit ( 8001 ) ;

}

}

52

53

54

55

56

*// создаем поток Time Manager*

log -> loudlog (\_T(" Create Time Manager ")); log -> loudlog (\_T(" Count = % d"), count );

*// создаем поток Time Manager , который пока не стартуют*

if (( allhandlers [ count ] = Create Thread ( NULL , 0 , Thread Time Manager Handler , ( LPVOID ) config -> ttl , CREATE\_ SUSPENDED , NULL )) == NULL ) {

log -> loudlog (\_T(" impossible to create thread - reader , GLE = % d"), Get Last Error ());

exit ( 8002 ) ;

}

log -> loudlog (\_T(" Successfully created threads !")); return ;

}

### Листинг 27: Потоки читатели (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

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// проверяем число аргументов*
3. if ( argc != 2) {
4. Logger log (\_T(" Process Reader "));
5. log . loudlog (\_T(" Error with start reader process . Need 2 arguments ."));
6. \_ getch ();
7. Exit Process ( 1000 ) ;

15 }

1. *// получаем из командной строки наш номер*
2. int myid = \_wtoi ( argv [1]) ;

#### 18

1. Logger log (\_T(" Process Reader "), myid );
2. log . loudlog (\_T(" Reader with id= % d is started "), myid );

#### 21

1. *// Инициализируем средства синхронизации*
2. *// ( атрибуты защиты, наследование описателя, имя):*
3. *// писатель записал сообщение ( ручной сброс);*
4. HANDLE can Read Event = Open Event ( EVENT\_ ALL\_ ACCESS , false ,
5. L" $$ My\_ can Read Event$$ ");
6. *// все читатели готовы к приему следующего ( автосброс);*
7. HANDLE can Write Event = Open Event ( EVENT\_ ALL\_ ACCESS , false ,
8. L" $$ My\_ can Write Event$$ ");
9. *// все читатели прочитали сообщение ( ручной сброс);*
10. HANDLE all Read Event = Open Event ( EVENT\_ ALL\_ ACCESS , false ,
11. L" $$ My\_ all Read Event$$ ");
12. *// разрешение работы со счетчиком ( автосброс);*
13. HANDLE change Count Event = Open Event ( EVENT\_ ALL\_ ACCESS , false ,
14. L" $$ My\_ change Count Event$$ ");
15. *// завершение программы ( ручной сброс);*
16. HANDLE exit Event = Open Event ( EVENT\_ ALL\_ ACCESS , false , L" $$ My\_ exit Event$$ ")

;

#### 38

1. *// Общий ресурс ( атрибуты защиты, наследование описателя, имя):*
2. HANDLE h File Mapping = Open File Mapping ( FILE\_ MAP\_ ALL\_ ACCESS , false ,
3. L" $$ My Very Special Share File Name$$ ");

#### 42

1. *// если объекты не созданы, то не сможем работать*
2. if ( can Read Event == NULL || can Write Event == NULL || all Read Event == NULL
3. || change Count Event == NULL || exit Event == NULL
4. || h File Mapping == NULL ) {
5. log . loudlog (\_T(" Impossible to open objects , run server first \ n getlasterror =% d"),
6. Get Last Error ());
7. \_ getch ();
8. return 1001 ;

51 }

#### 52

1. *// отображаем файл на адресное пространство нашего процесса для потоков- чит ателей*
2. LPVOID lp File Map For Readers = Map View Of File ( h File Mapping ,
3. FILE\_ MAP\_ ALL\_ ACCESS , 0 , 0 , 0);
4. *// h File Mapping - дескриптор объекта-отображения файла*
5. *// FILE\_ MAP\_ ALL\_ ACCESS - доступа к файлу*
6. *// 0 , 0 - старшая и младшая части смещения начала отображаемого участка в файле*
7. *// (0 - начало отображаемого участка совпадает с началом файла)*
8. *// 0 - размер отображаемого участка файла в байтах (0 - весь файл)*

#### 61

1. HANDLE readerhandlers [2];
2. readerhandlers [0] = exit Event ;
3. readerhandlers [1] = can Read Event ;

#### 65

* 1. while (1) { *// основной цикл*
  2. *// ждем, пока все прочитают*
  3. log . quietlog (\_T(" Waining for all Read Event "));
  4. Wait For Single Object ( all Read Event , INFINITE );
  5. *// узнаем, сколько потоков-читателей прошло данную границу*
  6. log . quietlog (\_T(" Waining for change Count Event "));
  7. Wait For Single Object ( change Count Event , INFINITE );
  8. (\*((( int \*) lp File Map For Readers ) + 1)) --;
  9. log . loudlog (\_T(" Readready = % d\ n"), (\*((( int \*) lp File Map For Readers ) + 1))

);

* 1. *// если все прошли, то " закрываем за собой дверь" и разрешаем писать*
  2. if ((\*((( int \*) lp File Map For Readers ) + 1)) == 0) {
  3. log . quietlog (\_T(" Reset Event all Read Event "));
  4. Reset Event ( all Read Event );
  5. log . quietlog (\_T(" Set Event can Write Event "));
  6. Set Event ( can Write Event );

81 }

#### 82

1. *// разрешаем изменять счетчик*
2. log . quietlog (\_T(" Set Event change Count Event "));
3. Set Event ( change Count Event );

#### 86

1. log . quietlog (\_T(" Waining for multiple objects "));
2. DWORD dw Event = Wait For Multiple Objects (2 , readerhandlers , false ,
3. INFINITE );
4. *// 2 - следим за 2 - я параметрами*
5. *// readerhandlers - из массива readerhandlers*
6. *// false - жд e¨ м, когда освободится хотя бы один*
7. *// INFINITE - ждать бесконечно*
8. switch ( dw Event ) {
9. case WAIT\_ OBJECT\_ 0 : *// сработало событие exit*
10. log . quietlog (\_T(" Get exit Event "));
11. log . loudlog (\_T(" Reader % d finishing work "), myid );
12. goto exit ;
13. case WAIT\_ OBJECT\_ 0 + 1: *// сработало событие на возможность чтения*
14. log . quietlog (\_T(" Get can Read Event "));
15. *// читаем сообщение*
16. log . loudlog (\_T(" Reader % d read msg \"% s\""), myid ,
17. (( \_ TCHAR \*) lp File Map For Readers ) + sizeof ( int ) \* 2);

#### 104

1. *// необходимо уменьшить счетчик количества читателей, которые прочитать еще не успели*
2. log . quietlog (\_T(" Waining for change Count Event "));
3. Wait For Single Object ( change Count Event , INFINITE );
4. (\*(( int \*) lp File Map For Readers )) --;
5. log . loudlog (\_T(" Readcount = % d"), (\*((( int \*) lp File Map For Readers ))));

#### 110

1. *// если мы последние читали, то запрещаем читать и открываем границу*
2. if ((\*(( int \*) lp File Map For Readers )) == 0) {
3. log . quietlog (\_T(" Reset Event can Read Event "));
4. Reset Event ( can Read Event );
5. log . quietlog (\_T(" Set Event all Read Event "));
6. Set Event ( all Read Event );

#### 117 }

118

1. *// разрешаем изменять счетчик*
2. log . quietlog (\_T(" Set Event change Count Event "));
3. Set Event ( change Count Event );
4. break ;
5. default :
6. log . loudlog (\_T(" Error with func Wait For Multiple Objects in reader Handle

, GLE = % d"), Get Last Error ());

1. getchar ();
2. Exit Process ( 1001 ) ;
3. break ;

#### 128 }

129 }

1. exit :
2. *// закрываем HANDLE объектов синхронизации*
3. Close Handle ( can Read Event );
4. Close Handle ( can Write Event );
5. Close Handle ( all Read Event );
6. Close Handle ( change Count Event );
7. Close Handle ( exit Event );

#### 137

1. Unmap View Of File ( lp File Map For Readers ); *// закрываем общий ресурс*
2. Close Handle ( h File Mapping ); *// закрываем объект " отображаемый файл"*

#### 140

1. log . loudlog (\_T(" All tasks are done !"));
2. \_ getch ();
3. 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.

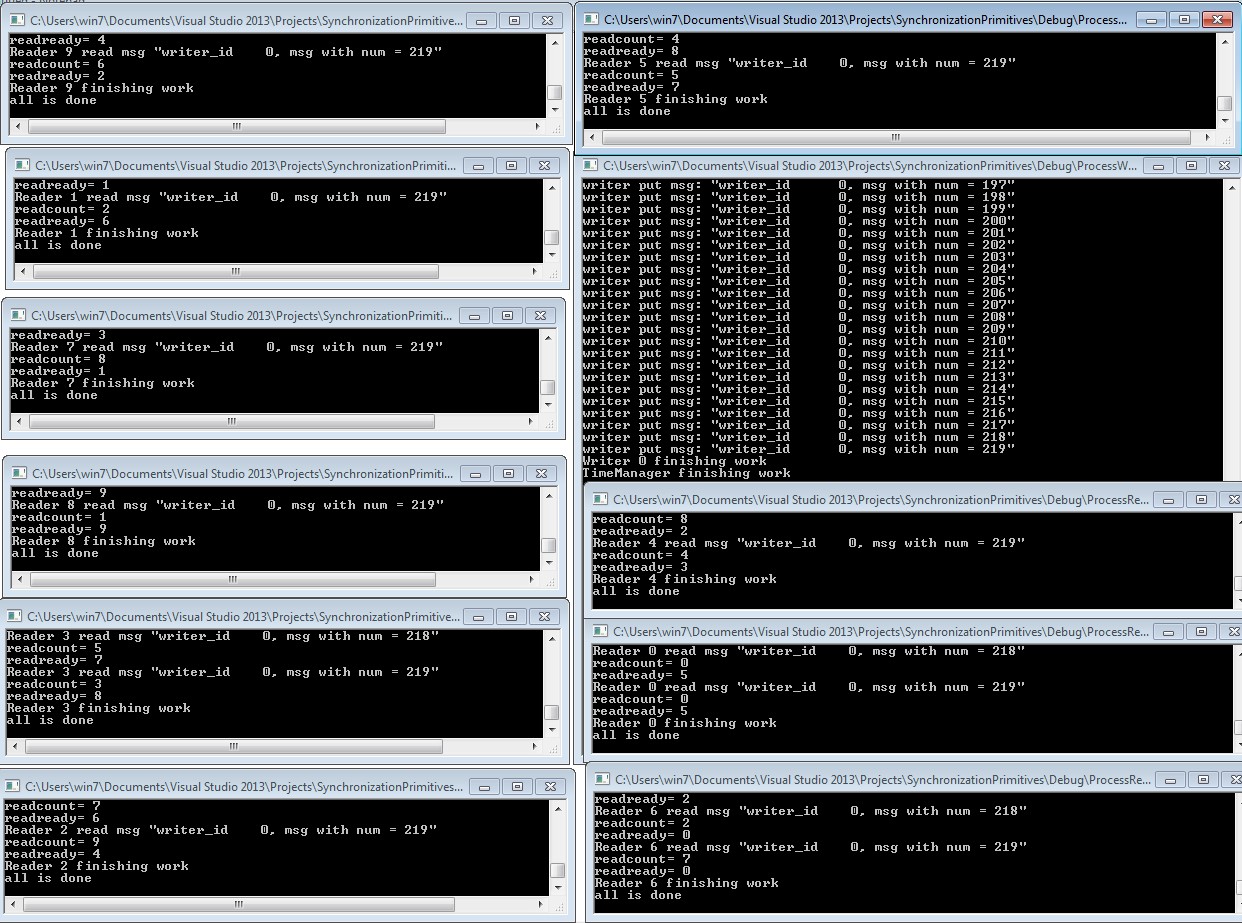


Рис. 8: Решение задачи читатели-писатели для процессов.

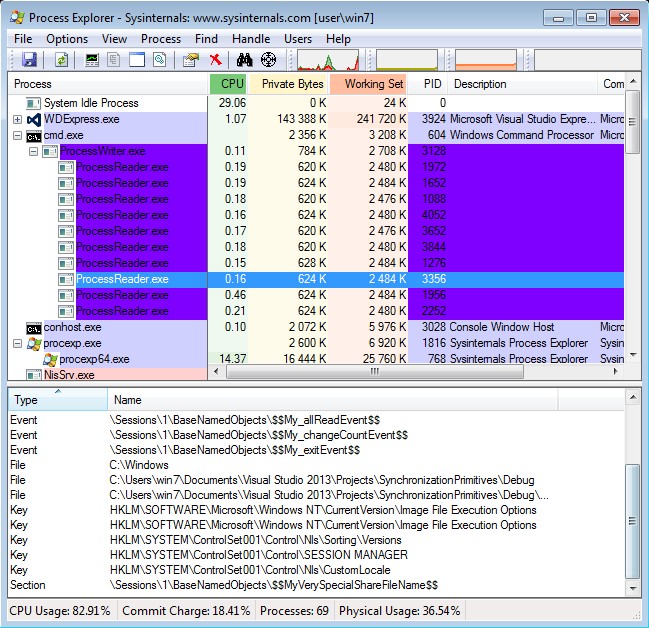


Рис. 9: Вывод process expl orer.

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 ] Process Writer . Thread Writer is starting .
  2. [ 25 / 3 / 2015 18 : 59 : 7 ] Waining for multiple objects

3 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Write Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Writer put msg : " Writer\_ id 0 , msg with num = 1"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
4. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event can Read Event
5. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

9 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Write Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Writer put msg : " Writer\_ id 0 , msg with num = 2"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
4. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event can Read Event
5. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

15 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Write Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Writer put msg : " Writer\_ id 0 , msg with num = 3"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
4. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event can Read Event
5. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

21 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Write Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Writer put msg : " Writer\_ id 0 , msg with num = 4"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
4. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event can Read Event
5. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

27 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Write Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Writer put msg : " Writer\_ id 0 , msg with num = 5"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
4. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event can Read Event
5. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

33 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Write Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Writer put msg : " Writer\_ id 0 , msg with num = 6"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
4. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event can Read Event
5. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

39 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Write Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Writer put msg : " Writer\_ id 0 , msg with num = 7"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
4. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event can Read Event
5. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

45 [ 25 / 3 / 2015 18 : 59 : 10 ] Get can Write Event

1. [ 25 / 3 / 2015 18 : 59 : 10 ] Writer put msg : " Writer\_ id 0 , msg with num = 8"
2. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for change Count Event
3. [ 25 / 3 / 2015 18 : 59 : 10 ] Set Event change Count Event
4. [ 25 / 3 / 2015 18 : 59 : 10 ] Set Event can Read Event
5. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for multiple objects

51 [ 25 / 3 / 2015 18 : 59 : 10 ] Get can Write Event

1. [ 25 / 3 / 2015 18 : 59 : 10 ] Writer put msg : " Writer\_ id 0 , msg with num = 9"
2. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for change Count Event
3. [ 25 / 3 / 2015 18 : 59 : 10 ] Set Event change Count Event
4. [ 25 / 3 / 2015 18 : 59 : 10 ] Set Event can Read Event
5. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for multiple objects

57 [ 25 / 3 / 2015 18 : 59 : 10 ] Get exit Event

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 ] Process Reader is starting .
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Reader with id= 1 is started
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for all Read Event
4. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

5 [ 25 / 3 / 2015 18 : 59 : 9 ] Readready = 4

#### 6

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

9 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Read Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Reader 1 read msg " Writer\_ id 0 , msg with num = 1"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

12 [ 25 / 3 / 2015 18 : 59 : 9 ] Readcount = 6

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for all Read Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

16 [ 25 / 3 / 2015 18 : 59 : 9 ] Readready = 9

#### 17

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

20 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Read Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Reader 1 read msg " Writer\_ id 0 , msg with num = 2"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

23 [ 25 / 3 / 2015 18 : 59 : 9 ] Readcount = 5

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for all Read Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

27 [ 25 / 3 / 2015 18 : 59 : 9 ] Readready = 6

#### 28

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

31 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Read Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Reader 1 read msg " Writer\_ id 0 , msg with num = 3"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

34 [ 25 / 3 / 2015 18 : 59 : 9 ] Readcount = 6

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for all Read Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

38 [ 25 / 3 / 2015 18 : 59 : 9 ] Readready = 4

#### 39

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

42 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Read Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Reader 1 read msg " Writer\_ id 0 , msg with num = 4"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

45 [ 25 / 3 / 2015 18 : 59 : 9 ] Readcount = 3

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for all Read Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

49 [ 25 / 3 / 2015 18 : 59 : 9 ] Readready = 6

#### 50

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

53 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Read Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Reader 1 read msg " Writer\_ id 0 , msg with num = 5"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

56 [ 25 / 3 / 2015 18 : 59 : 9 ] Readcount = 7

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for all Read Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

60 [ 25 / 3 / 2015 18 : 59 : 9 ] Readready = 7

#### 61

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

64 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Read Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Reader 1 read msg " Writer\_ id 0 , msg with num = 6"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

67 [ 25 / 3 / 2015 18 : 59 : 9 ] Readcount = 3

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for all Read Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

71 [ 25 / 3 / 2015 18 : 59 : 9 ] Readready = 9

#### 72

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for multiple objects

75 [ 25 / 3 / 2015 18 : 59 : 9 ] Get can Read Event

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Reader 1 read msg " Writer\_ id 0 , msg with num = 7"
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

78 [ 25 / 3 / 2015 18 : 59 : 9 ] Readcount = 8

1. [ 25 / 3 / 2015 18 : 59 : 9 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for all Read Event
3. [ 25 / 3 / 2015 18 : 59 : 9 ] Waining for change Count Event

82 [ 25 / 3 / 2015 18 : 59 : 10 ] Readready = 6

#### 83

1. [ 25 / 3 / 2015 18 : 59 : 10 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for multiple objects

86 [ 25 / 3 / 2015 18 : 59 : 10 ] Get can Read Event

1. [ 25 / 3 / 2015 18 : 59 : 10 ] Reader 1 read msg " Writer\_ id 0 , msg with num = 8"
2. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for change Count Event

89 [ 25 / 3 / 2015 18 : 59 : 10 ] Readcount = 5

1. [ 25 / 3 / 2015 18 : 59 : 10 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for all Read Event
3. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for change Count Event

93 [ 25 / 3 / 2015 18 : 59 : 10 ] Readready = 4

#### 94

1. [ 25 / 3 / 2015 18 : 59 : 10 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for multiple objects

97 [ 25 / 3 / 2015 18 : 59 : 10 ] Get can Read Event

1. [ 25 / 3 / 2015 18 : 59 : 10 ] Reader 1 read msg " Writer\_ id 0 , msg with num = 9"
2. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for change Count Event

100 [ 25 / 3 / 2015 18 : 59 : 10 ] Readcount = 3

1. [ 25 / 3 / 2015 18 : 59 : 10 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for all Read Event
3. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for change Count Event

104 [ 25 / 3 / 2015 18 : 59 : 10 ] Readready = 4

#### 105

1. [ 25 / 3 / 2015 18 : 59 : 10 ] Set Event change Count Event
2. [ 25 / 3 / 2015 18 : 59 : 10 ] Waining for multiple objects

108 [ 25 / 3 / 2015 18 : 59 : 10 ] Get exit Event

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

1. # include " thread . h"
2. # include " utils . h"
3. # include " Logger . h"

#### 10

1. *// глобальные переменные*
2. struct Configuration config ; *// конфигурация программы*
3. bool is Done = false ; *// флаг завершения*
4. HANDLE \* allhandlers ; *// массив всех создаваемых потоков*

#### 15

1. *// события для синхронизации:*
2. *// писетль записал сообщение, читатель может его прочитать*
3. HANDLE reader Can Read Event ;
4. *// все читатели должны перейти в режим готовности*
5. HANDLE reader Get Ready Event ;
6. *// отч e¨ т может быть отправлен*
7. HANDLE can Change Count Event ;
8. *// отч e¨ т*
9. HANDLE change Count Event ;
10. *// завершение программы ( ручной сброс);*
11. HANDLE exit Event ;

#### 27

1. *// переменные для синхронизации работы потоков:*
2. int report Counter = 0; *// Сч e¨ тчиков отч e¨ тов*

#### 30

1. *// имя разделяемой памяти*
2. wchar\_ t share File Name [] = L" $$ My Very Special Share File Name$$ ";

#### 33

1. HANDLE h File Mapping ; *// объект-отображение файла*
2. LPVOID lp File Map For Writers ; *// указатели на отображаемую память*

#### 36

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. Logger log (\_T(" No Mem Process Writer "));

#### 39

1. if ( argc < 2)
2. *// Используем конфигурацию по- умолчанию*
3. Set Default Config (& config , & log );
4. else
5. *// Загрузка конфига из файла*
6. Set Config ( argv [1] , & config , & log );

#### 46

1. *// создаем необходимые потоки без их запуска*
2. *// потоки-читатели запускаются сразу ( чтобы они успели дойти до функции ожи дания)*
3. Create All Threads (& config , & log );

#### 50

1. *// Инициализируем ресурс ( share memory): создаем объект " отображаемый файл"*
2. *// будет использован системный файл подкачки ( на диске файл создаваться*
3. *// не будет), т. к. в качестве дескриптора файла использовано значение*
4. *// равное 0 x FFFFFFFF ( его эквивалент - символическая константа INVALID\_ HANDLE\_ VALUE )*
5. if (( h File Mapping = Create File Mapping ( INVALID\_ HANDLE\_ VALUE , NULL ,
6. PAGE\_ READWRITE , 0 , 1500 , share File Name )) == NULL ) {
7. *// INVALID\_ HANDLE\_ VALUE - дескриптор открытого файла*
8. *// ( INVALID\_ HANDLE\_ VALUE - файл подкачки)*
9. *// NULL - атрибуты защиты объекта- отображения*
10. *// PAGE\_ READWRITE - озможности доступа к представлению файла при*
11. *// отображении ( PAGE\_ READWRITE - чтение/ запись)*
12. *// 0 , 1500 - старшая и младшая части значения максимального*
13. *// размера объекта отображения файла*
14. *// share File Name - имя объекта- отображения.*
15. log . loudlog (\_T(" Impossible to create shareFile , GLE = % d"),
16. Get Last Error ());
17. Exit Process ( 10000 ) ;

68 }

1. *// отображаем файл на адресное пространство нашего процесса для потока- писа теля*
2. lp File Map For Writers = Map View Of File ( h File Mapping , FILE\_ MAP\_ WRITE , 0 , 0 , 0)

;

1. *// h File Mapping - дескриптор объекта-отображения файла*

|  |  |  |
| --- | --- | --- |
| 72 | *//* | *FILE\_ MAP\_ WRITE - доступа к файлу* |
| 73 | *//* | *0 , 0 - старшая и младшая части смещения начала отображаемого участка в* |
| 74 | *//* | *файле*  *(0 - начало отображаемого участка совпадает с началом файла)* |
| 75 | *//* | *0 - размер отображаемого участка файла в байтах (0 - весь файл)* |
| 76 |  |  |

* 1. *// инициализируем средства синхронизации*
  2. *// ( атрибуты защиты, ручной сброс, начальное состояние, имя):*
  3. *// событие " окончание записи" ( можно читать), ручной сброс, изначально заня то*
  4. reader Can Read Event = Create Event ( NULL , true , false , L"

$$ My\_ reader Can Read Event$$ ");

* 1. *// событие - " можно писать", автосброс( разрешаем писать только одному), изна чально свободно*
  2. reader Get Ready Event = Create Event ( NULL , true , true , L"

$$ My\_ reader Get Ready Event$$ ");

* 1. *// событие для изменения счетчика ( сколько клиентов еще не прочитало сообще ние)*
  2. can Change Count Event = Create Event ( NULL , false , true , L"

$$ My\_ can Change Count Event$$ ");

* 1. change Count Event = Create Event ( NULL , false , false , L"

$$ My\_ change Count Event$$ ");

* 1. *// событие " завершение работы программы", ручной сброс, изначально занято*
  2. exit Event = Create Event ( NULL , true , false , L" $$ My\_ exit Event$$ ");

#### 88

1. *// запускаем потоки- писатели и поток- планировщик на исполнение*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Resume Thread ( allhandlers [ i]);

#### 92

1. *// ожидаем завершения всех потоков*
2. Wait For Multiple Objects ( config . num Of Readers + config . num Of Writers + 1 ,
3. allhandlers , TRUE , INFINITE );

#### 96

1. *// закрываем handle потоков*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Close Handle ( allhandlers [ i]);

#### 100

1. *// закрываем описатели объектов синхронизации*
2. Close Handle ( reader Can Read Event );
3. Close Handle ( reader Get Ready Event );
4. Close Handle ( can Change Count Event );
5. Close Handle ( change Count Event );
6. Close Handle ( exit Event );

#### 107

108 Unmap View Of File ( lp File Map For Writers ); *// закрываем handle общего ресурса*

|  |  |  |
| --- | --- | --- |
| 109 |  | Close Handle ( h File Mapping ); *// 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

1. DWORD WINAPI Thread Writer Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" No Mem Process Writer . Thread Writer "), myid );
2. extern bool is Done ;
3. extern struct Configuration config ;

#### 13

1. extern HANDLE reader Can Read Event ;
2. extern HANDLE reader Get Ready Event ;
3. extern HANDLE can Change Count Event ;
4. extern HANDLE change Count Event ;
5. extern HANDLE exit Event ;

#### 19

1. extern int report Counter ; *// Сч e¨ тчиков отч e¨ тов*
2. extern LPVOID lp File Map For Writers ;

#### 22

1. int msgnum = 0;
2. HANDLE writer Handlers [2];
3. writer Handlers [0] = exit Event ;
4. writer Handlers [1] = change Count Event ;

#### 27

1. *// Состояние готовности:*
2. *// true - сообщение записано, жд e¨ м отч e¨ тов о прочтении*
3. *// false - переводим всех читателей в состояние готовности*
4. bool ready State = false ;

#### 32

1. while ( is Done != true ) {
2. log . quietlog (\_T(" Waining for multiple objects "));
3. DWORD dw Event = Wait For Multiple Objects (2 , writer Handlers , false ,
4. INFINITE );
5. *// 2 - следим за 2 - я параметрами*
6. *// writer Handlers - из массива writer Handlers*
7. *// false - жд e¨ м, когда освободится хотя бы один*
8. *// INFINITE - ждать бесконечно*
9. switch ( dw Event ) {
10. case WAIT\_ OBJECT\_ 0 : *// сработало событие exit*
11. log . quietlog (\_T(" Get exit Event "));
12. log . loudlog (\_T(" Writer % d finishing work "), myid );
13. return 0;
14. case WAIT\_ OBJECT\_ 0 + 1: *// Приш e¨ л отч e¨ т о выполнении*
15. log . quietlog (\_T(" Get change Count Event "));
16. *// Если отчитались все читатели*
17. if (++ report Counter == config . num Of Readers ) {
18. *// Обнуление сч e¨ тчика*
19. report Counter = 0;
20. if ( ready State ) { *// все вс e¨*

*прочитали*

1. *// Теперь ожидаем отч e¨ тов о готовности*
2. ready State = false ;
3. *// Больше ни кто не читает*
4. log . quietlog (\_T(" Reset Event reader Can Read Event "));
5. Reset Event ( reader Can Read Event );
6. *// Можно готовится*
7. log . quietlog (\_T(" Set Event reader Get Ready Event "));
8. Set Event ( reader Get Ready Event );

61 }

1. else { *// все готовы читать*
2. *// Запись сообщения*
3. swprintf\_ s (( \_ TCHAR \*) lp File Map For Writers , 1500 ,
4. \_T(" Writer\_ id %d, msg with num = % d"), myid , ++ msgnum );
5. log . loudlog (\_T(" Writer put msg : \"% s\""), ( \_ TCHAR \*) lp File Map For Writers );

#### 67

1. *// Теперь ожидаем отч e¨ тов о прочтении*
2. ready State = true ;
3. *// Больше ни кто не готовится*
4. log . quietlog (\_T(" Reset Event reader Get Ready Event "));
5. Reset Event ( reader Get Ready Event );
6. *// Можно читать*
7. log . quietlog (\_T(" Set Event reader Can Read Event "));
8. Set Event ( reader Can Read Event );

76 }

77 }

1. *// Жд e¨ м следующего отч e¨ та*
2. log . quietlog (\_T(" Set Event can Change Count Event "));
3. Set Event ( can Change Count Event );

81

1. break ;
2. default :
3. log . loudlog (\_T(" Error with func Wait For Multiple Objects in writer Handle

, GLE = % d"), Get Last Error ());

1. Exit Process ( 1000 ) ;

86 }

87 }

1. log . loudlog (\_T(" Writer % d finishing work "), myid );
2. return 0;

90

}

### Листинг 32: Запуск клиентских процессов (src/SynchronizationPrimitives/NoMemProcessWriter/utils.c

#### 1

* 1. *// создание всех потоков*
  2. void Create All Threads ( struct Configuration \* config , Logger \* log ) {
  3. extern HANDLE \* allhandlers ;

#### 5

1. int total = config -> num Of Readers + config -> num Of Writers + 1;
2. log -> quietlog (\_T(" Total num of threads is % d"), total );
3. allhandlers = new HANDLE [ total ];
4. int count = 0;

#### 10

1. *// создаем потоки- читатели*
2. log -> loudlog (\_T(" Create readers "));

#### 13

1. STARTUPINFO si;
2. PROCESS\_ INFORMATION pi;

#### 16

1. Zero Memory (& si , sizeof ( si));
2. si. cb = sizeof ( si);
3. Zero Memory (& pi , sizeof ( pi));
4. TCHAR sz Command Line [ 100 ];

#### 21

1. for ( int i = 0; i != config -> num Of Readers ; i++ , count ++) {
2. \_ stprintf\_ s ( sz Command Line , \_T(" No Mem Process Reader . exe % d % d"), i, config

-> readers Delay );

1. log -> loudlog (\_T(" Count = % d"), count );
2. if (! Create Process ( NULL , sz Command Line , NULL , NULL , FALSE , CREATE\_ NEW\_ CONSOLE |
3. CREATE\_ SUSPENDED , NULL , NULL , & si , & pi)) {
4. log -> loudlog (\_T(" Impossible to create Process - reader , GLE = % d"), Get Last Error ());

28 exit ( 8000 ) ;

29 }

30 allhandlers [ count ] = pi. h Thread ;

31 }

#### 32

1. *// создаем потоки- писатели*
2. log -> loudlog (\_T(" Create writers "));
3. for ( int i = 0; i != config -> num Of Writers ; i++ , count ++) {
4. log -> loudlog (\_T(" count = % d"), count );
5. *// создаем потоки- читатели, которые пока не стартуют*
6. if (( allhandlers [ count ] = Create Thread ( NULL , 0 , Thread Writer Handler ,
7. ( LPVOID )i, CREATE\_ SUSPENDED , NULL )) == NULL ) {
8. log -> loudlog (\_T(" Impossible to create thread - writer , GLE = % d"), Get Last Error ());

41 exit ( 8001 ) ;

42 }

43 }

#### 44

1. *// создаем поток Time Manager*
2. log -> loudlog (\_T(" Create Time Manager "));
3. log -> loudlog (\_T(" Count = % d"), count );
4. *// создаем поток Time Manager , который пока не стартуют*
5. if (( allhandlers [ count ] = Create Thread ( NULL , 0 , Thread Time Manager Handler ,
6. ( LPVOID ) config -> ttl , CREATE\_ SUSPENDED , NULL )) == NULL ) {
7. log -> loudlog (\_T(" impossible to create thread - reader , GLE = % d"), Get Last Error ());

52 exit ( 8002 ) ;

53 }

1. log -> loudlog (\_T(" Successfully created threads !"));
2. return ;

56 }

### Листинг 33: Потоки читатели (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

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// проверяем число аргументов*
3. if ( argc != 3) {
4. Logger log (\_T(" No Mem Process Reader "));
5. log . loudlog (\_T(" Error with start reader process . Need 2 arguments , but % d presented ."), argc );
6. \_ getch ();
7. Exit Process ( 1000 ) ;

15 }

1. *// получаем из командной строки наш номер*
2. int myid = \_wtoi ( argv [1]) ;
3. int pause = \_wtoi ( argv [2]) ;

#### 19

1. Logger log (\_T(" No Mem Process Reader "), myid );
2. log . loudlog (\_T(" Reader with id= % d is started "), myid );

#### 22

1. *// Состояние готовности:*
2. *// true - жд e¨ м сообщение для чтения*
3. *// false - текущее сообщение уже прочитано,*
4. *// жд e¨ м сигнала перехода в режим готовности*
5. bool ready State = false ;

#### 28

1. *// Инициализируем средства синхронизации*
2. *// ( атрибуты защиты, наследование описателя, имя):*
3. *// писатель записал сообщение ( ручной сброс);*
4. HANDLE reader Can Read Event = Open Event ( EVENT\_ ALL\_ ACCESS , false ,
5. L" $$ My\_ reader Can Read Event$$ ");
6. *// все читатели готовы к приему следующего ( автосброс);*
7. HANDLE reader Get Ready Event = Open Event ( EVENT\_ ALL\_ ACCESS , false ,
8. L" $$ My\_ reader Get Ready Event$$ ");
9. *// разрешение работы со счетчиком ( автосброс);*
10. HANDLE can Change Count Event = Open Event ( EVENT\_ ALL\_ ACCESS , false ,
11. L" $$ My\_ can Change Count Event$$ ");

40 *//*

1. HANDLE change Count Event = Open Event ( EVENT\_ ALL\_ ACCESS , false ,
2. L" $$ My\_ change Count Event$$ ");
3. *// завершение программы ( ручной сброс);*
4. HANDLE exit Event = Open Event ( EVENT\_ ALL\_ ACCESS , false , L" $$ My\_ exit Event$$ ")

;

#### 45

1. *// Общий ресурс ( атрибуты защиты, наследование описателя, имя):*
2. HANDLE h File Mapping = Open File Mapping ( FILE\_ MAP\_ READ , false ,
3. L" $$ My Very Special Share File Name$$ ");

#### 49

1. *// если объекты не созданы, то не сможем работать*
2. if ( reader Can Read Event == NULL || reader Get Ready Event == NULL || can Change Count Event == NULL
3. || change Count Event == NULL || exit Event == NULL
4. || h File Mapping == NULL ) {
5. log . loudlog (\_T(" Impossible to open objects , run server first \ n getlasterror =% d"),
6. Get Last Error ());
7. \_ getch ();
8. return 1001 ;

58 }

#### 59

1. *// отображаем файл на адресное пространство нашего процесса для потоков- чит ателей*
2. LPVOID lp File Map For Readers = Map View Of File ( h File Mapping ,
3. FILE\_ MAP\_ READ , 0 , 0 , 0);
4. *// h File Mapping - дескриптор объекта-отображения файла*
5. *// FILE\_ MAP\_ ALL\_ ACCESS - доступа к файлу*
6. *// 0 , 0 - старшая и младшая части смещения начала отображаемого участка в файле*
7. *// (0 - начало отображаемого участка совпадает с началом файла)*
8. *// 0 - размер отображаемого участка файла в байтах (0 - весь файл)*

#### 68

1. *// События чтиения*
2. HANDLE reader Handlers [2];
3. reader Handlers [0] = exit Event ;
4. reader Handlers [1] = reader Can Read Event ;

#### 73

1. *// События готовности*
2. HANDLE ready Handlers [2];
3. ready Handlers [0] = exit Event ;
4. ready Handlers [1] = reader Get Ready Event ;

#### 78

1. while (1) { *// основной цикл*
2. *// Ожидаем набор событий в зависимости от состояния*
3. if ( ready State ) {
4. log . quietlog (\_T(" Waining for multiple objects "));
5. DWORD dw Event = Wait For Multiple Objects (2 , reader Handlers , false ,
6. INFINITE );
7. *// 2 - следим за 2 - я параметрами*
8. *// reader Handlers - из массива reader Handlers*
9. *// false - жд e¨ м, когда освободится хотя бы один*
10. *// INFINITE - ждать бесконечно*
11. switch ( dw Event ) {
12. case WAIT\_ OBJECT\_ 0 : *// сработало событие exit*
13. log . quietlog (\_T(" Get exit Event "));
14. log . loudlog (\_T(" Reader % d finishing work "), myid );
15. goto exit ;

#### 94

1. case WAIT\_ OBJECT\_ 0 + 1: *// сработало событие на возможность чтения*
2. log . quietlog (\_T(" Get reader Can Read Event "));
3. *// читаем сообщение*
4. log . loudlog (\_T(" Reader % d read msg \"% s\""), myid , ( \_ TCHAR \*) lp File Map For Readers );

#### 99

1. *// Отправляем отч e¨ т*
2. log . quietlog (\_T(" Waining for can Change Count Event "));
3. Wait For Single Object ( can Change Count Event , INFINITE );
4. log . quietlog (\_T(" Set Event change Count Event "));
5. Set Event ( change Count Event );

#### 105

1. *// Завершаем работу*
2. ready State = false ;
3. break ;
4. default :
5. log . loudlog (\_T(" Error with func Wait For Multiple Objects in reader Handle , GLE = % d"), Get Last Error ());
6. getchar ();
7. Exit Process ( 1001 ) ;
8. break ;

#### 114 }

115 }

1. else {
2. log . quietlog (\_T(" Waining for multiple objects "));
3. DWORD dw Event = Wait For Multiple Objects (2 , ready Handlers , false ,
4. INFINITE );
5. *// 2 - следим за 2 - я параметрами*
6. *// ready Handlers - из массива ready Handlers*
7. *// false - жд e¨ м, когда освободится хотя бы один*
8. *// INFINITE - ждать бесконечно*
9. switch ( dw Event ) {
10. case WAIT\_ OBJECT\_ 0 : *// сработало событие exit*
11. log . quietlog (\_T(" Get exit Event "));

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

reader Handle , GLE = % d"), Get Last Error ());

getchar (); Exit Process ( 1001 ) ; break ;

}

}

Sleep ( pause );

}

exit :

*// закрываем HANDLE объектов синхронизации* Close Handle ( reader Can Read Event ); Close Handle ( reader Get Ready Event ); Close Handle ( can Change Count Event ); Close Handle ( change Count Event ); Close Handle ( exit Event );

Unmap View Of File ( lp File Map For Readers ); *// закрываем общий ресурс*

Close Handle ( h File Mapping ); *// закрываем объект " отображаемый файл"*

log . loudlog (\_T(" All tasks are done !"));

\_ getch (); return 0;

}

|  |  |  |  |
| --- | --- | --- | --- |
| 127 | log . loudlog (\_T(" Reader % d finishing work "), myid ); | |  |
| 128 | goto exit ; | |
| 129 |  | |
| 130 | case WAIT\_ OBJECT\_ 0 + 1: *// сработало событие перехода в режим ти* | | *готовнос* |
| 131 | log . quietlog (\_T(" Get reader Get Ready Event ")); |  |  |
| 132 | *// Отправляем отч e¨ т* |  |  |
| 133 | log . quietlog (\_T(" Waining for can Change Count Event ")); |  |  |
| 134 | Wait For Single Object ( can Change Count Event , INFINITE ); |  |  |
| 135 | log . quietlog (\_T(" Set Event change Count Event ")); |  |  |
| 136 | Set Event ( change Count Event ); |  |  |
| 137 |  |  |  |
| 138 | *// Завершаем работу* |  |  |
| 139 | ready State = true ; |  |  |
| 140 | break ; |  |  |
| 141 | default : |  |  |
| 142 | log . loudlog (\_T(" Error with func Wait For Multiple Objects | in |  |

### Результат работы на рисунке 10.

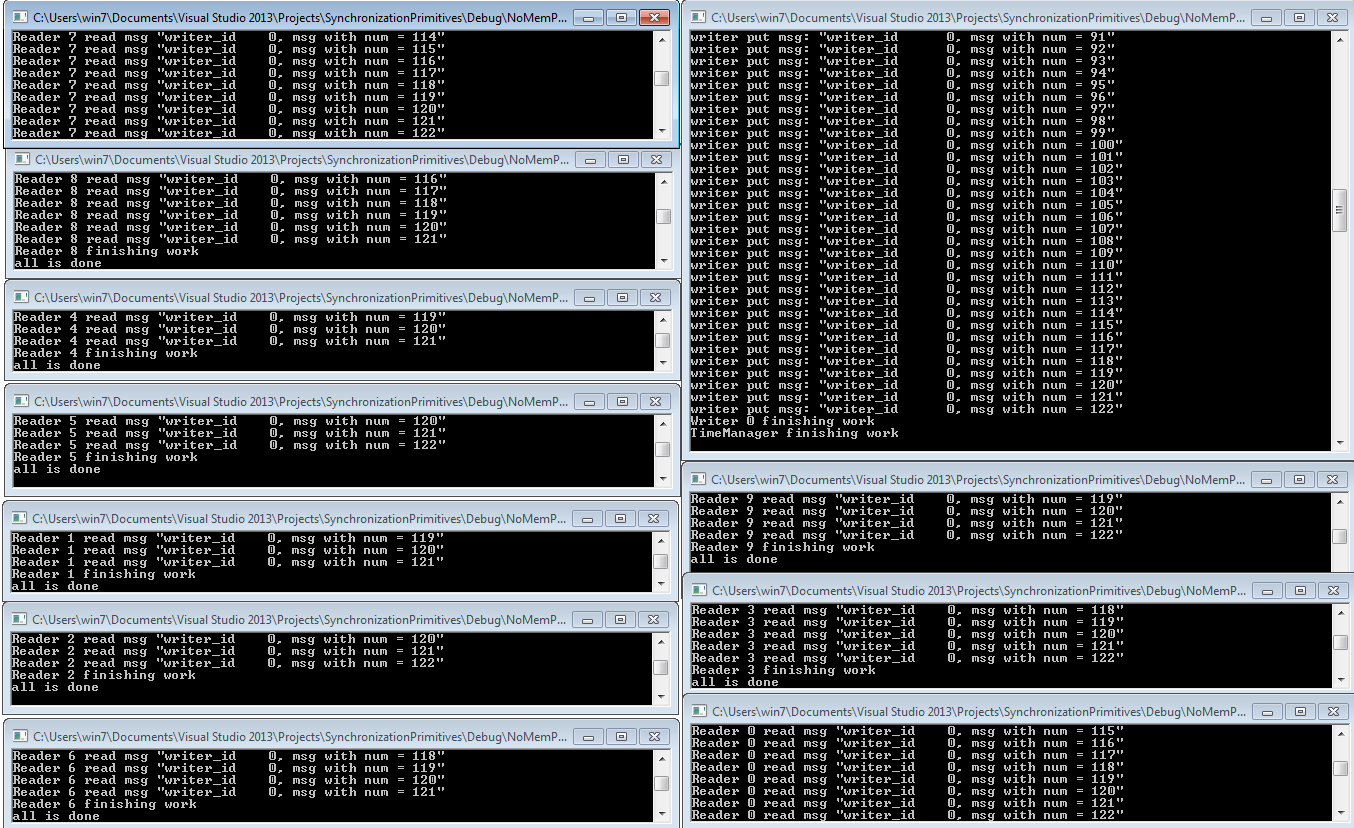


Рис. 10: Модификация задачи читатели-писатели.

# 3 Рациональное решение задачи читатели-писатели

В задаче предлагается найти более рациональное решение задачи читатели-писатели, но при имеющихся ограничениях (каждый процесс производи 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

1. # include " thread . h"
2. # include " utils . h"
3. # include " Logger . h"

#### 10

1. *// глобальные переменные:*
2. struct Configuration config ; *// конфигурация программы*
3. bool is Done = false ; *// флаг завершения*
4. HANDLE \* allhandlers ; *// массив всех создаваемых потоков*

#### 15

1. *// инструмент синхронизации:*
2. SRWLOCK lock ;
3. *// условная переменная для потоков- читателей*
4. CONDITION\_ VARIABLE condread ;

#### 20

21 HANDLE exit Event ; *// завершение программы ( ручной сброс);*

#### 22

1. *// имя разделяемой памяти*
2. wchar\_ t share File Name [] = L" $$ My Very Special Share File Name$$ ";

#### 25

1. HANDLE h File Mapping ; *// объект-отображение файла*
2. *// указатели на отображаемую память*
3. LPVOID lp File Map For Writers ;
4. LPVOID lp File Map For Readers ;

#### 30

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. Logger log (\_T(" Optimal Reader Writer "));

#### 33

1. if ( argc < 2)
2. *// Используем конфигурацию по- умолчанию*
3. Set Default Config (& config , & log );
4. else
5. *// Загрузка конфига из файла*
6. Set Config ( argv [1] , & config , & log );

#### 40

1. *// создаем необходимые потоки без их запуска*
2. Create All Threads (& config , & log );

#### 43

1. *// Инициализируем ресурс ( share memory): создаем объект " отображаемый файл"*
2. *// будет использован системный файл подкачки ( на диске файл создаваться*
3. *// не будет), т. к. в качестве дескриптора файла использовано значение*
4. *// равное 0 x FFFFFFFF ( его эквивалент - символическая константа INVALID\_ HANDLE\_ VALUE )*
5. if (( h File Mapping = Create File Mapping ( INVALID\_ HANDLE\_ VALUE , NULL ,
6. PAGE\_ READWRITE , 0 , 1500 , share File Name )) == NULL ) {
7. *// INVALID\_ HANDLE\_ VALUE - дескриптор открытого файла*
8. *// ( INVALID\_ HANDLE\_ VALUE - файл подкачки)*
9. *// NULL - атрибуты защиты объекта- отображения*
10. *// PAGE\_ READWRITE - озможности доступа к представлению файла при*
11. *// отображении ( PAGE\_ READWRITE - чтение/ запись)*
12. *// 0 , 1500 - старшая и младшая части значения максимального*
13. *// размера объекта отображения файла*
14. *// share File Name - имя объекта- отображения.*
15. log . loudlog (\_T(" Impossible to create shareFile , GLE = % d"),
16. Get Last Error ());
17. Exit Process ( 10000 ) ;

61 }

1. *// отображаем файл на адресное пространство нашего процесса для потока- писа теля*
2. lp File Map For Writers = Map View Of File ( h File Mapping , FILE\_ MAP\_ WRITE , 0 , 0 , 0)

;

1. *// h File Mapping - дескриптор объекта-отображения файла*
2. *// FILE\_ MAP\_ WRITE - доступа к файлу*
3. *// 0 , 0 - старшая и младшая части смещения начала отображаемого участка в файле*
4. *// (0 - начало отображаемого участка совпадает с началом файла)*
5. *// 0 - размер отображаемого участка файла в байтах (0 - весь файл)*

#### 69

70 *// отображаем файл на адресное пространство нашего процесса для потоков- чит*

*ателей*

71 lp File Map For Readers = Map View Of File ( h File Mapping , FILE\_ MAP\_ READ , 0 , 0 , 0);

#### 72

1. *// инициализируем средства синхронизации*
2. *// событие " завершение работы программы", ручной сброс, изначально занято*
3. exit Event = Create Event ( NULL , true , false , L"");
4. Initialize SRWLock (& lock );
5. Initialize Condition Variable (& condread );

#### 78

1. *// запускаем потоки- писатели и поток- планировщик на исполнение*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Resume Thread ( allhandlers [ i]);

#### 82

1. *// ожидаем завершения всех потоков*
2. Wait For Multiple Objects ( config . num Of Readers + config . num Of Writers + 1 ,
3. allhandlers , TRUE , INFINITE );

#### 86

1. *// закрываем handle потоков*
2. for ( int i = 0; i < config . num Of Readers + config . num Of Writers + 1; i ++)
3. Close Handle ( allhandlers [ i]);

#### 90

1. *// закрываем описатели объектов синхронизации*
2. Close Handle ( exit Event );

#### 93

1. *// закрываем handle общего ресурса*
2. Unmap View Of File ( lp File Map For Readers );
3. Unmap View Of File ( lp File Map For Writers );

#### 97

1. *// закрываем объект " отображаемый файл"*
2. Close Handle ( h File Mapping );

#### 100

1. *// Завершение работы*
2. log . loudlog (\_T(" All tasks are done !"));
3. \_ getch ();
4. return 0;

#### 105 }

Листинг 35: Потоки писатели (src/SynchronizationPrimitives/OptimalReaderWriter/threadWriter.cpp)

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

#### 4

5 # include " utils . h"

#### 6

1. DWORD WINAPI Thread Writer Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Optimal Reader Writer . Thread Writer "), myid );
2. extern bool is Done ;
3. extern struct Configuration config ;
4. extern SRWLOCK lock ;
5. extern CONDITION\_ VARIABLE condread ;
6. extern LPVOID lp File Map For Writers ;

#### 16

1. int msgnum = 0;
2. while ( is Done != true ) {
3. *// Захват объекта синхронизации ( монопольный доступ!)*
4. log . quietlog (\_T(" Waining for Slim Reader / Writer ( SRW ) Lock "));
5. Acquire SRWLock Exclusive (& lock );
6. log . quietlog (\_T(" Get SRW Lock "));

#### 23

1. *// Запись сообщения*
2. swprintf\_ s (( \_ TCHAR \*) lp File Map For Writers , 1500 ,
3. \_T(" writer\_ id %d, msg with num = % d"), myid , msgnum ++) ;
4. log . loudlog (\_T(" writer put msg : \"% s\""), lp File Map For Writers );

#### 28

1. *// освобождение объекта синхронизации*
2. log . quietlog (\_T(" Release SRW Lock "));
3. Wake All Condition Variable (& condread );
4. Release SRWLock Exclusive (& lock );

#### 33

1. *// задержка*
2. Sleep ( config . writers Delay );

36 }

1. log . loudlog (\_T(" Writer % d finishing work "), myid );
2. return 0;

39 }

### Листинг 36: Потоки читатели (src/SynchronizationPrimitives/OptimalReaderWriter/threadReader.cpp)

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

#### 4

5 # include " utils . h"

#### 6

1. DWORD WINAPI Thread Reader Handler ( LPVOID prm ) {
2. int myid = ( int ) prm ;

#### 9

1. Logger log (\_T(" Optimal Reader Writer . Thread Reader "), myid );
2. extern bool is Done ;
3. extern struct Configuration config ;
4. extern SRWLOCK lock ;
5. extern CONDITION\_ VARIABLE condread ;
6. extern LPVOID lp File Map For Readers ;

#### 16

1. while ( is Done != true ) {
2. *// Захват объекта синхронизации ( совместный доступ!)*
3. log . quietlog (\_T(" Waining for Slim Reader / Writer ( SRW ) Lock "));
4. Acquire SRWLock Shared (& lock );
5. Sleep Condition Variable SRW (& condread , & lock , INFINITE , CONDITION\_ VARIABLE\_ LOCKMODE\_ SHARED );
6. log . quietlog (\_T(" Get SRW Lock "));

#### 23

1. *// читаем сообщение*
2. log . loudlog (\_T(" Reader % d read msg \"% s\""), myid ,
3. ( \_ TCHAR \*) lp File Map For Readers );

#### 27

1. *// освобождение объекта синхронизации*
2. log . quietlog (\_T(" Release SRW Lock "));
3. Release SRWLock Shared (& lock );

#### 31

1. *// задержка*
2. Sleep ( config . readers Delay );

34 }

1. log . loudlog (\_T(" Reader % d finishing work "), myid );
2. return 0;

37 }

### Требование по обязательному прочтению каждого сообщения каждым читателем выполня- ется, как видно на рисунке 11.

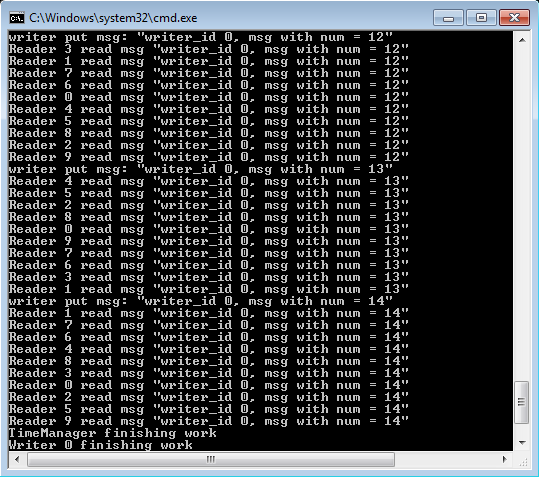


Рис. 11: Модификация задачи читатели-писатели.

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

Листинг 37: Протокол работы писателя

1 [ 25 / 3 / 2015

2 [ 25 / 3 / 2015

3 [ 25 / 3 / 2015

4 [ 25 / 3 / 2015

5 [ 25 / 3 / 2015

6 [ 25 / 3 / 2015

7 [ 25 / 3 / 2015

8 [ 25 / 3 / 2015

9 [ 25 / 3 / 2015

19 : 9 : 5 ]

19 : 9 : 5 ]

19 : 9 : 5 ]

19 : 9 : 5 ]

19 : 9 : 5 ]

19 : 9 : 5 ]

19 : 9 : 5 ]

19 : 9 : 5 ]

19 : 9 : 5 ]

Optimal Reader Writer . Thread Writer is starting .

Waining for Slim Reader / Writer ( SRW ) Lock Get SRW Lock

writer put msg : " writer\_ id 0 , msg with num = 0"

Release

Waining Get SRW

SRW Lock

for Slim Reader / Writer ( SRW ) Lock Lock

writer put msg : " writer\_ id 0 , msg with num = 1"

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

56 [ 25 / 3 / 2015

57 [ 25 / 3 / 2015

58 [ 25 / 3 / 2015

59 [ 25 / 3 / 2015

60 [ 25 / 3 / 2015

61 [ 25 / 3 / 2015

62 [ 25 / 3 / 2015

63 [ 25 / 3 / 2015

19 : 9 : 7 ]

19 : 9 : 7 ]

19 : 9 : 7 ]

19 : 9 : 7 ]

19 : 9 : 7 ]

19 : 9 : 7 ]

19 : 9 : 7 ]

19 : 9 : 8 ]

19 : 9 : 8 ]

Get SRW Lock

writer put msg : " writer\_ id 0 , msg with num = 13"

Release

Waining Get SRW

SRW Lock

for Slim Reader / Writer ( SRW ) Lock Lock

writer put msg : " writer\_ id 0 , msg with num = 14"

Release SRW Lock

Writer 0 finishing work Shutting down .

### Листинг 38: Протокол работы читателя

1. [ 25 / 3 / 2015 19 : 9 : 5 ] Optimal Reader Writer . Thread Reader 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 читатели писатели

### В данной задаче появляется уже несколько писателей, а модель работы становится клиент- серверной[1]. В качестве IPC был выбран именованный канал, а для синхронизации ис- пользованы 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 сер- вере 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

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

# include

# include

# include

# include

# include

# include

< windows . h>

< stdio . h>

< conio . h>

< tchar . h>

< strsafe . h> " Logger . h"

# define BUFSIZE 512

DWORD WINAPI Instance Thread ( LPVOID ); HANDLE Create And Start Waitable Timer ( int );

DWORD WINAPI Thread Time Manager Handler ( LPVOID ); DWORD WINAPI Thread Writer ( LPVOID );

*// Init log*

Logger log (\_T(" Reader Writer Server "), -1);

*// инструмент синхронизации:* SRWLOCK lock ; CONDITION\_ VARIABLE condread ;

CRITICAL\_ SECTION crs ; *// Объявление критической*

int message ; *// сообщение*

bool is Done = false ; *// флаг завершения*

*секции*

int \_ tmain ( int argc , \_ TCHAR \* argv []) {

log . loudlog (\_T(" Server is started .\ n\ n"));

|  |  |  |
| --- | --- | --- |
| 27 | BOOL | f Connected = FALSE ; *// Флаг наличия подключенных клиентов* |
| 28 | DWORD | dw Thread Id = 0; *// Номер обслуживающего потока* |
| 29 | HANDLE | h Pipe = INVALID\_ HANDLE\_ VALUE ; *// Идентификатор канала* |
| 30 | HANDLE | h Thread = NULL ; *// Идентификатор обслуживающего потока* |
| 31 | HANDLE | service [3]; *// Идентификатор потока писателя и таймера* |

32 LPTSTR lpsz Pipename = \_T(" \\\\.\\ pipe \\ $$ My Pipe$$ "); *// Имя создаваемого к анала*

#### 33

1. *// начальное сообщение*
2. message = 0;
3. Initialize SRWLock (& lock );
4. Initialize Condition Variable (& condread );

#### 38

1. *// Создание потока-таймера*
2. log . loudlog (\_T(" Time Manager creation !"));
3. service [0] = Create Thread (
4. NULL , *// дескриптор защиты*
5. 0 , *// начальный размер стека*
6. Thread Time Manager Handler , *// функция потока*
7. ( LPVOID ) 5 , *// параметр потока (5 секунд)*
8. NULL , *// опции создания*
9. NULL ); *// номер потока*

#### 48

1. *// Создание потоков-писателей*
2. log . loudlog (\_T(" Writers creation !"));
3. service [1] = Create Thread (
4. NULL , *// дескриптор защиты*
5. 0 , *// начальный размер стека*
6. Thread Writer , *// функция потока*
7. NULL , *// параметр потока*
8. NULL , *// опции создания*
9. NULL ); *// номер потока*

#### 58

1. service [2] = Create Thread (
2. NULL , *// дескриптор защиты*
3. 0 , *// начальный размер стека*
4. Thread Writer , *// функция потока*
5. NULL , *// параметр потока*
6. NULL , *// опции создания*
7. NULL ); *// номер потока*

#### 66

1. *// инициализируем средство синхронизации*
2. Initialize Critical Section (& crs );

#### 69

70 *// Ожидаем соединения со стороны клиента*

1. log . loudlog (\_T(" Waiting for connect ... "));
2. *// Цикл ожидает клиентов и созда e¨ т для них потоки обработки*
3. while ( is Done != true ) {
4. *// Создаем канал:*
5. log . loudlog (\_T(" Try to create named pipe on % s"), lpsz Pipename );
6. if (( h Pipe = Create Named Pipe (
7. lpsz Pipename , *// имя канала,*
8. PIPE\_ ACCESS\_ DUPLEX , *// режим отрытия канала - двунаправленный,*
9. PIPE\_ TYPE\_ MESSAGE | *// данные записываются в канал в виде потока сообщ ений,*
10. PIPE\_WAIT , *// функции передачи и приема блокируются до их окончан ия,*
11. PIPE\_ UNLIMITED\_ INSTANCES , *// максимальное число экземпляров каналов не ограничено,*
12. BUFSIZE \* sizeof ( \_ TCHAR ), *// размеры выходного и входного буферов канала*

*,*

1. BUFSIZE \* sizeof ( \_ TCHAR ),
2. 5000 , *// 5 секунд - длительность для функции Wait Named Pipe ,*
3. NULL )) *// дескриптор безопасности по умолчанию.*

86 == INVALID\_ HANDLE\_ VALUE ) {

1. log . loudlog (\_T(" Create Named Pipe failed , GLE =% d."), Get Last Error ());
2. exit (1) ;

89 }

90 log . loudlog (\_T(" Named pipe created successfully !"));

#### 91

1. *// Если произошло соединение*
2. if ( Connect Named Pipe ( hPipe , NULL )) {
3. log . loudlog (\_T(" Client connected !"));

#### 95

1. *// Созда e¨ м поток для обслуживания клиента*
2. h Thread = Create Thread (
3. NULL , *// дескриптор защиты*
4. 0 , *// начальный размер стека*
5. Instance Thread , *// функция потока*
6. ( LPVOID ) hPipe , *// параметр потока*
7. 0 , *// опции создания*
8. & dw Thread Id ); *// номер потока*

#### 104

1. *// Если поток создать не удалось - сообщаем об ошибке*
2. if ( h Thread == NULL ) {
3. double errorcode = Get Last Error ();
4. log . loudlog (\_T(" Create Thread failed , GLE =% d."), errorcode );
5. exit (1) ;

#### 110 }

111 else Close Handle ( h Thread );

#### 112 }

1. else {
2. *// Если клиенту не удалось подключиться, закрываем канал*
3. Close Handle ( hPipe );
4. log . loudlog (\_T(" There are not connecrtion reqests ."));

#### 117 }

118 }

119

1. *// ожидаем завершения всех потоков*
2. Wait For Multiple Objects (3 , service , TRUE , INFINITE );

#### 122

1. *// удаляем объект синхронизации*
2. Delete Critical Section (& crs );

#### 125

126 *// закрываем handle потоков*

127 for ( int i = 0; i != 3; ++ i)

128 Close Handle ( service [ i]);

#### 129

1. *// Завершение работы*
2. log . loudlog (\_T(" All tasks are done !"));
3. \_ getch ();
4. exit (0) ;

#### 134 }

135

1. DWORD WINAPI Instance Thread ( LPVOID lpv Param ) {
2. log . loudlog (\_T(" Thread % d started !"), Get Current Thread Id ());
3. HANDLE h Pipe = ( HANDLE ) lpv Param ; *// Идентификатор канала*
4. HANDLE h Heap = Get Process Heap (); *// локальная куча*
5. *// Буфер для хранения передаваемого сообщения*
6. \_ TCHAR \* ch Buf = ( \_ TCHAR \*) Heap Alloc ( hHeap , 0 , BUFSIZE \* sizeof ( TCHAR ));
7. DWORD writebytes ; *// Число байт прочитанных и переданных*

#### 143

1. while ( is Done != true ) {
2. *// Захват объекта синхронизации ( совместный доступ!)*
3. log . quietlog (\_T(" Waining for Slim Reader / Writer ( SRW ) Lock "));
4. Acquire SRWLock Shared (& lock );
5. Sleep Condition Variable SRW (& condread , & lock , INFINITE , CONDITION\_ VARIABLE\_ LOCKMODE\_ SHARED );
6. log . quietlog (\_T(" Get SRW Lock "));

#### 150

1. *// Посылаем эту команду клиентскому приложению*
2. swprintf\_ s ( chBuf , BUFSIZE , L"% i", message );
3. if ( Write File ( hPipe , chBuf , ( lstrlen ( chBuf ) + 1)\* sizeof ( \_ TCHAR ), & writebytes , NULL )) {
4. *// Выводим сообщение на консоль*
5. log . quietlog (\_T(" Client % d: get msg : % s"), Get Current Thread Id () , chBuf

);

|  |  |  |
| --- | --- | --- |
| 156 | } |  |
| 157 | else { |  |
| 158 | log . loudlog (\_T(" Thread % d: Write File : | Error % ld"), Get Current Thread Id |

() , Get Last Error ());

159 break ;

#### 160 }

161

1. *// освобождение объекта синхронизации*
2. log . quietlog (\_T(" Release SRW Lock "));
3. Release SRWLock Shared (& lock );

#### 165

166 *// задержка*

167 Sleep (100 ) ;

#### 168 }

169

1. *// завершаем работу приложения*
2. String Cch Copy ( chBuf , BUFSIZE , L" exit ");
3. Write File ( hPipe , chBuf , ( lstrlen ( chBuf ) + 1)\* sizeof ( \_ TCHAR ), & writebytes , NULL );

#### 173

1. *// Освобождение ресурсов*
2. Flush File Buffers ( hPipe );
3. Disconnect Named Pipe ( hPipe );
4. Close Handle ( hPipe );

#### 178

179 Heap Free ( hHeap , 0 , chBuf );

#### 180

181 log . quietlog (\_T(" Thread % d: Instance Thread exitting ."), Get Current Thread Id ());

|  |  |  |
| --- | --- | --- |
| 182 |  | return 0; |
| 183 | } |  |
| 184 |  |  |
| 185 | DW | ORD WINAPI Thread Writer ( LPVOID lpv Param ) { |
| 186 |  | log . loudlog (\_T(" Writer thread % d started !"), Get Current Thread Id ()); |
| 187 |  |  |
| 188 |  | while ( is Done != true ) { |
| 189 |  | Enter Critical Section (& crs ); |
| 190 |  | *// Захват объекта синхронизации ( монопольный доступ!)* |
| 191 |  | log . quietlog (\_T(" Writer : Waining for Slim Reader / Writer ( SRW ) Lock ")); |
| 192 |  | Acquire SRWLock Exclusive (& lock ); |
| 193 |  | log . quietlog (\_T(" Writer : Get SRW Lock ")); |
| 194 |  |  |
| 195 |  | *// меняем значение сообщения* |

|  |  |  |
| --- | --- | --- |
| 196 |  | ++ message ; |
| 197 |  | log . loudlog (\_T(" Server % d: send msg : % d"), Get Current Thread Id () , message |
| 198 |  | ); |
| 199 |  | *// освобождение объекта синхронизации* |
| 200 |  | log . quietlog (\_T(" Writer : Release SRW Lock ")); |
| 201 |  | Wake All Condition Variable (& condread ); |
| 202 |  | Release SRWLock Exclusive (& lock ); |
| 203 |  |  |
| 204 |  | *// задержка* |
| 205 |  | Sleep (200 ) ; |
| 206 |  | Leave Critical Section (& crs ); |
| 207 | } |  |
| 208 |  |  |
| 209 | lo | g . quietlog (\_T(" Thread % d: Writer Thread exitting ."), Get Current Thread Id |

());

210 return 0;

#### 211 }

212

1. *// создание, установка и запуск таймера*
2. HANDLE Create And Start Waitable Timer ( int sec ) {
3. \_\_ int 64 end\_ time ;
4. LARGE\_ INTEGER end\_ time 2 ;
5. HANDLE tm = Create Waitable Timer ( NULL , false , \_T(" Timer !"));

218 end\_ time = -1 \* sec \* 10000000 ;

1. end\_ time 2 . Low Part = ( DWORD )( end\_ time & 0 x FFFFFFFF );
2. end\_ time 2 . High Part = ( LONG )( end\_ time >> 32) ;
3. Set Waitable Timer ( tm , & end\_time2 , 0 , NULL , NULL , false );
4. return tm;

#### 223 }

224

1. DWORD WINAPI Thread Time Manager Handler ( LPVOID prm ) {
2. int ttl = ( int ) prm ;
3. if ( ttl < 0) {
4. *// завершение по команде оператора*
5. \_ TCHAR buf [ 100 ];
6. while (1) {
7. fgetws ( buf , sizeof ( buf ), stdin );

232 if ( buf [0] == \_T(’s’)) {

1. log . quietlog (\_T("’s’ signal received , set Event exit Event "));
2. is Done = true ;
3. break ;

#### 236 }

237 }

238 }

|  |  |  |
| --- | --- | --- |
| 239 |  | else { |
| 240 |  | *// завершение по таймеру* |
| 241 |  | HANDLE h = Create And Start Waitable Timer ( ttl ); |
| 242 |  | Wait For Single Object (h, INFINITE ); |
| 243 |  | log . quietlog (\_T(" Timer signal received , set Event exit Event ")); |
| 244 |  | is Done = true ; |
| 245 |  | Close Handle ( h); |
| 246 |  | } |
| 247 |  | log . loudlog (\_T(" Time Manager 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

1. # define BUFSIZE 512
2. *// Init log*
3. Logger log (\_T(" Reader Writer Client "), Get Current Process Id ());

#### 11

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. log . loudlog (\_T(" Client is started !\ n\ n"));

#### 14

1. HANDLE h Pipe = INVALID\_ HANDLE\_ VALUE ; *// Идентификатор канала*
2. LPTSTR lpsz Pipename = \_T(" \\\\.\\ pipe \\ $$ My Pipe$$ "); *// Имя создаваемого к анала Pipe*
3. \_ TCHAR chBuf [ BUFSIZE ]; *// Буфер для передачи данных через канал*
4. DWORD readbytes ; *// Число байт прочитанных и переданных*

#### 19

20 log . loudlog (\_T(" Try to use Wait Named Pipe ... "));

#### 21

1. *// Пытаемся открыть именованный канал, если надо - ожидаем его освобождени я*
2. while (1) {
3. *// Создаем канал с процессом- клиентом:*
4. hPipe = Create File (
5. lpsz Pipename , *// имя канала,*
6. GENERIC\_ READ , *// текущий клиент имеет доступ на чтение,*
7. 0 , *// тип доступа,*
8. NULL , *// атрибуты защиты,*
9. OPEN\_ EXISTING , *// открывается существующий файл,*
10. 0 , *// атрибуты и флаги для файла,*
11. NULL ); *// доступа к файлу шаблона.*

#### 33

1. *// Продолжаем работу, если канал создать удалось*
2. if ( h Pipe != INVALID\_ HANDLE\_ VALUE )
3. break ;

#### 37

1. *// Выход, если ошибка связана не с занятым каналом.*
2. double errorcode = Get Last Error ();
3. if ( errorcode != ERROR\_ PIPE\_ BUSY ) {
4. log . loudlog (\_T(" Could not open pipe . GLE =% d\ n"), errorcode );
5. exit (1) ;

43 }

#### 44

1. *// Если все каналы заняты, жд e¨ м 20 секунд*
2. if (! Wait Named Pipe ( lpsz Pipename , 20000 ) ) {
3. log . loudlog (\_T(" Could not open pipe : 20 second wait timed out ."));
4. exit (2) ;

49 }

50 }

#### 51

1. *// Выводим сообщение о создании канала*
2. log . loudlog (\_T(" Successfully connected !"));
3. *// Цикл обмена данными с серверным процессом*
4. while (1) {
5. *// Получаем команду от сервера*
6. if ( Read File ( hPipe , chBuf , BUFSIZE \* sizeof ( TCHAR ), & readbytes , NULL )) {
7. log . loudlog (\_T(" Received from server : % s"), chBuf );
8. } else {
9. *// Если произошла ошибка, выводим ее код и завершаем работу приложения*
10. double errorcode = Get Last Error ();
11. log . loudlog (\_T(" Read File : Error % ld\ n"), errorcode );
12. \_ getch ();
13. break ;

65 }

1. *// В ответ на команду " exit" завершаем цикл обмена данными с серверным п роцессом*
2. if (! \_ tcsncmp ( chBuf , L" exit ", 4)) {
3. log . loudlog (\_T(" Processing exit code "));
4. break ;

70 }

71 }

1. *// Закрываем идентификатор канала*
2. Close Handle ( hPipe );

74

1. *// Завершение работы*
2. log . loudlog (\_T(" All tasks are done !"));
3. \_ getch ();
4. exit (0) ;
5. return 0;

80

}

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

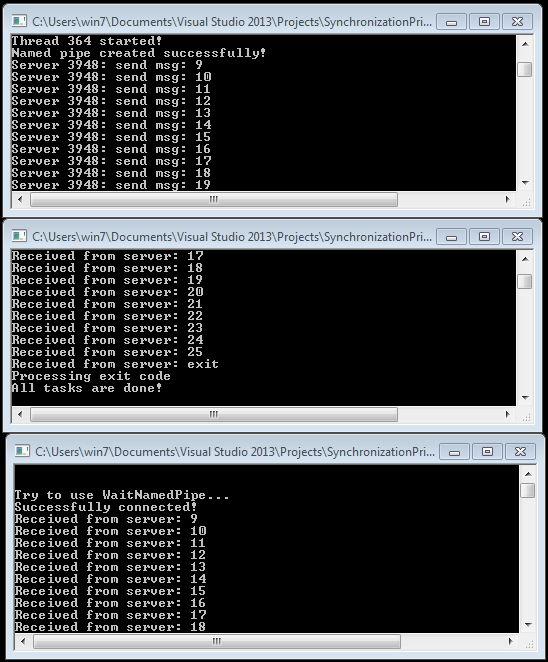


Рис. 12: Клиент-серверное приложение

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 раз- ные 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 ] Reader Writer Server 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 ...

1. [ 25 / 3 / 2015 19 : 49 : 32 ] Try to create named pipe on \\.\ pipe \ $$ My Pipe$$ [ 25 / 3 / 2015 19 : 49 : 32 ] Writer thread 3968 started ![ 25 / 3 / 2015 19 : 49 : 32 ] Writer thread 3948 started !
2. [ 25 / 3 / 2015 19 : 49 : 32 ] Named pipe created successfully !
3. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 32 ] Writer : Release SRW Lock
2. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 32 ] Writer : Release SRW Lock
2. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 33 ] Writer : Release SRW Lock
2. [ 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 !

1. [ 25 / 3 / 2015 19 : 49 : 33 ] Try to create named pipe on \\.\ pipe \ $$ My Pipe$$ [ 25 / 3 / 2015 19 : 49 : 33 ] Thread 1568 started !
2. [ 25 / 3 / 2015 19 : 49 : 33 ] Named pipe created successfully !
3. [ 25 / 3 / 2015 19 : 49 : 33 ] Waining for Slim Reader / Writer ( SRW ) Lock
4. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 33 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 33 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 33 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 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 !

1. [ 25 / 3 / 2015 19 : 49 : 34 ] Try to create named pipe on \\.\ pipe \ $$ My Pipe$$ [ 25 / 3 / 2015 19 : 49 : 34 ] Thread 364 started !
2. [ 25 / 3 / 2015 19 : 49 : 34 ] Named pipe created successfully !
3. [ 25 / 3 / 2015 19 : 49 : 34 ] Waining for Slim Reader / Writer ( SRW ) Lock
4. [ 25 / 3 / 2015 19 : 49 : 34 ] Waining for Slim Reader / Writer ( SRW ) Lock
5. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 34 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 34 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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 Lock Get 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

1. [ 25 / 3 / 2015 19 : 49 : 34 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 34 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 34 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 34 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 34 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 34 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 35 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 35 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 35 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 35 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 35 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 35 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 35 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 35 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 36 ] Waining for Slim Reader / Writer ( SRW ) Lock [ 25 / 3 / 2015 19 : 49 : 36 ]
2. 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

1. [ 25 / 3 / 2015 19 : 49 : 36 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 36 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 36 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 36 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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 ]

1. Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 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 Lock Client 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

1. [ 25 / 3 / 2015 19 : 49 : 37 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 37 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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

1. [ 25 / 3 / 2015 19 : 49 : 37 ] Waining for Slim Reader / Writer ( SRW ) Lock
2. [ 25 / 3 / 2015 19 : 49 : 37 ] Waining for Slim Reader / Writer ( SRW ) Lock
3. [ 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  25 | 19 : 49 : 37 ] | [ 25 / 3 / 2015 19 : 49 : 37 ] Get SRW Lock Client 1568: get | | msg : |
| 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 exit Event |  |  |
| 262 | [ 25 / 3 / 2015 | 19 : 49 : 37 ] | Time Manager finishing work |  |  |
| 263 | [ 25 / 3 / 2015 | 19 : 49 : 37 ] | Thread 364: Instance Thread exitting . |  |  |
| 264 | [ 25 / 3 / 2015 | 19 : 49 : 37 ] | Thread 1568: Instance Thread exitting . |  |  |
| 265 | [ 25 / 3 / 2015 | 19 : 49 : 37 ] | Thread 3968: Writer Thread exitting . |  |  |
| 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 exitting . |  |  |

### Листинг 42: Протокол работы клиента-читателя

271 [ 25 / 3 / 2015 19 : 56 : 7 ] Shutting down .

1 [ 25 / 3 / 2015 19 : 49 : 34 ] Reader Writer Client is starting .

2 [ 25 / 3 / 2015 19 : 49 : 34 ] Client is started !

3 [ 25 / 3 / 2015 19 : 49 : 34 ] Try to use Wait Named Pipe ...

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 осу- ществлять 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

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

# define \_ WINSOCK\_ DEPRECATED\_ NO\_ WARNINGS

# include

# include

< winsock 2 . h>

< list >

# include

" Logger . h"

# include

# include

# include

< time . h>

< stdio . h>

< strsafe . h>

# pragma comment ( lib , " Ws 2 \_ 32 . lib ")

struct CLIENT\_ INFO

{

SOCKET h Client Socket ;

*// Сокет*

struct sockaddr\_ in client Addr ;

};

\_ TCHAR sz Server IPAddr [] = \_T(" 127 . 0 . 0 . 1 "); *// server IP*

int n Server Port = 5050; *// server port*

std :: list < CLIENT\_ INFO \*> clients ; *// Все клиенты*

CRITICAL\_ SECTION cs С lients ;

*// Защита списка*

bool Init Win Sock 2 \_ 0 ();

BOOL WINAPI Client Thread ( LPVOID lp Data ); void send To All ( \_ TCHAR \* p Buffer );

*// Init log*

Logger mylog (\_T(" Net Reader Writer Server "));

int \_ tmain ( int argc , \_ TCHAR \* argv []) { if (! Init Win Sock 2 \_ 0 ()) {

1. double errorcode = WSAGet Last Error ();
2. mylog . loudlog (\_T(" Unable to Initialize Windows Socket environment , GLE =% d"), errorcode );
3. exit (1) ;

37 }

38 mylog . loudlog (\_T(" Windows Socket environment ready "));

#### 39

1. SOCKET h Server Socket ;
2. h Server Socket = socket (
3. AF\_INET , *// The address family. AF\_ INET specifies TCP/ IP*
4. SOCK\_ STREAM , *// Protocol type. SOCK\_ STREM specified TCP*
5. 0 *// Protoco Name. Should be 0 for AF\_ INET address family*

45 );

#### 46

1. if ( h Server Socket == INVALID\_ SOCKET ) {
2. mylog . loudlog (\_T(" Unable to create Server socket "));
3. *// Cleanup the environment initialized by WSAStartup ()*
4. WSACleanup ();
5. exit (2) ;

52 }

53 mylog . loudlog (\_T(" Server socket created "));

#### 54

1. *// Create the structure describing various Server parameters*
2. struct sockaddr\_ in server Addr ;

#### 57

1. server Addr . sin\_ family = AF\_ INET ; *// The address family. MUST be AF\_ INET*
2. size\_ t convtd ;
3. char \* p MBBuffer = new char [20];
4. wcstombs\_ s (& convtd , pMBBuffer , 20 , sz Server IPAddr , 20) ;
5. *// server Addr . sin\_ addr . s\_ addr = inet\_ addr ( p MBBuffer );*
6. server Addr . sin\_ addr . s\_ addr = INADDR\_ ANY ;
7. delete [] p MBBuffer ;
8. server Addr . sin\_ port = htons ( n Server Port );

#### 66

1. *// Bind the Server socket to the address & port*
2. if ( bind ( h Server Socket , ( struct sockaddr \*) & server Addr , sizeof ( server Addr

)) == SOCKET\_ ERROR ) {

1. mylog . loudlog (\_T(" Unable to bind to % s on port % d"), sz Server IPAddr , n Server Port );
2. *// Free the socket and cleanup the environment initialized by WSAStartup*

*()*

1. closesocket ( h Server Socket );
2. WSACleanup ();
3. exit (3) ;

74 }

75 mylog . loudlog (\_T(" Bind completed "));

#### 76

1. *// Put the Server socket in listen state so that it can wait for client connections*
2. if ( listen ( h Server Socket , SOMAXCONN ) == SOCKET\_ ERROR ) {
3. mylog . loudlog (\_T(" Unable to put server in listen state "));
4. *// Free the socket and cleanup the environment initialized by WSAStartup ()*
5. closesocket ( h Server Socket );
6. WSACleanup ();
7. exit (4) ;

84 }

85 mylog . loudlog (\_T(" Ready for connection on % s:% d"), sz Server IPAddr , n Server Port );

#### 86

1. HANDLE h Client Thread [2];
2. DWORD dw Thread Id [2];

89 for ( int i = 0; i != 2; ++ i) {

1. *// Start the client thread*
2. h Client Thread [ i] = Create Thread ( NULL , 0 ,
3. ( LPTHREAD\_ START\_ ROUTINE ) Client Thread ,
4. 0 , 0 , & dw Thread Id [ i]);
5. if ( h Client Thread [ i] == NULL ) {
6. mylog . loudlog (\_T(" Unable to create client thread "));

96 }

1. else {
2. Close Handle ( h Client Thread );

99 }

#### 100 }

101

102

1. *// инициализируем средство синхронизации*
2. Initialize Critical Section (& cs С lients );

#### 105

1. *// Start the infinite loop*
2. while ( true ) {
3. *// As the socket is in listen mode there is a connection request pending*

*.*

1. *// Calling accept( ) will succeed and return the socket for the request.*
2. CLIENT\_ INFO \* p Client Info = new CLIENT\_ INFO ;

#### 111

1. int n Size = sizeof ( p Client Info -> client Addr );
2. p Client Info -> h Client Socket = accept ( h Server Socket , ( struct sockaddr \*) & p Client Info -> client Addr , & n Size );
3. if ( p Client Info -> h Client Socket == INVALID\_ SOCKET ) {
4. mylog . loudlog (\_T(" accept () failed "));

#### 116 }

1. else {
2. wchar\_ t \* sin\_ addr = new wchar\_ t [20];
3. size\_ t convtd ;
4. mbstowcs\_ s (& convtd , sin\_addr , 20 , inet\_ ntoa ( p Client Info -> client Addr . sin\_ addr ), 20) ;
5. mylog . loudlog (\_T(" Client connected from % s:% d"), sin\_addr , p Client Info

-> client Addr . sin\_ port );

1. delete [] sin\_ addr ;

#### 123

1. Enter Critical Section (& cs С lients );
2. clients . push\_ front ( p Client Info ); *// Добавить нового клиента в список*
3. Leave Critical Section (& cs С lients );

#### 127 }

128 }

129

1. *// удаляем объект синхронизации*
2. Delete Critical Section (& cs С lients );

#### 132

1. closesocket ( h Server Socket );
2. WSACleanup ();
3. exit (0) ;

#### 136 }

137

1. bool Init Win Sock 2 \_ 0 () {
2. WSADATA wsa Data ;
3. WORD w Version = MAKEWORD (2 , 0);

#### 141

1. if (! WSAStartup ( wVersion , & wsa Data ))
2. return true ;

#### 144

145 return false ;

#### 146 }

147

1. BOOL WINAPI Client Thread ( LPVOID lp Data ) {
2. *// Chat loop:*
3. while (1) {
4. \_ TCHAR sz Buffer [ 124 ];
5. swprintf\_ s ( szBuffer , \_T("% d"), Get Current Thread Id () % rand ());
6. mylog . loudlog (\_T(" Server % d: % s"), Get Current Thread Id () , sz Buffer );
7. send To All ( sz Buffer );

155 Sleep (100 ) ;

#### 156 }

|  |  |  |
| --- | --- | --- |
| 157 |  | |
| 158 |  | return 0; |
| 159 | } |  |
| 160 |  |  |
| 161 | vo | id send To All ( \_ TCHAR \* p Buffer ) { |
| 162 |  | *// Пока мы обрабатываем список, его ни кто не должен менять!* |
| 163 |  | Enter Critical Section (& cs С lients ); |
| 164 |  | std :: list < CLIENT\_ INFO \* >:: iterator client ; |
| 165 |  | for ( client = clients . begin (); client != clients . end (); ++ client ) { |
| 166 |  | int n Length = ( lstrlen ( p Buffer ) + 1) \* sizeof ( \_ TCHAR ); |
| 167 |  | int n Cnt Send = 0; |
| 168 |  |  |

169 while (( n Cnt Send = send ((\* client ) -> h Client Socket , ( char \*) pBuffer , nLength , 0) != n Length )) {

|  |  |  |  |
| --- | --- | --- | --- |
| 170 |  | | if ( n Cnt Send == -1) { |
| 171 |  | | mylog . loudlog (\_T(" Error sending the client ")); |
| 172 |  | | break ; |
| 173 |  | | } |
| 174 |  | | if ( n Cnt Send == n Length ) |
| 175 |  | | break ; |
| 176 |  | |  |
| 177 |  | | p Buffer += n Cnt Send ; |
| 178 |  | | n Length -= n Cnt Send ; |
| 179 | } | |  |
| 180 |  | } | |
| 181 |  | Leave Critical Section (& cs С lients ); | |
| 182 | } |  | |

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

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

# define \_ WINSOCK\_ DEPRECATED\_ NO\_ WARNINGS

# include < winsock 2 . h>

# include " Logger . h"

# pragma comment ( lib , " Ws 2 \_ 32 . lib ")

\_ TCHAR sz Server IPAddr [ 20 ]; int n Server Port ;

*//*

*// server IP*

*server port*

bool Init Win Sock 2 \_ 0 ();

*// Init log*

Logger mylog (\_T(" Net Reader Writer Client "), Get Current Process Id ());

int \_ tmain ( int argc , \_ TCHAR \* argv []) {

1. \_ tprintf (\_T(" Enter the server IP Address : "));
2. wscanf\_ s (\_T(" %19 s"), sz Server IPAddr , \_ countof ( sz Server IPAddr ));
3. \_ tprintf (\_T(" Enter the server port number : "));
4. wscanf\_ s (\_T("% i"), & n Server Port );

#### 20

1. if (! Init Win Sock 2 \_ 0 ()) {
2. double errorcode = WSAGet Last Error ();
3. mylog . loudlog (\_T(" Unable to Initialize Windows Socket environment , GLE =% d"), errorcode );
4. exit (1) ;

25 }

26 mylog . quietlog (\_T(" Windows Socket environment ready "));

#### 27

1. SOCKET h Client Socket ;
2. h Client Socket = socket (
3. AF\_INET , *// The address family. AF\_ INET specifies TCP/ IP*
4. SOCK\_ STREAM , *// Protocol type. SOCK\_ STREM specified TCP*
5. 0); *// Protoco Name. Should be 0 for AF\_ INET address family*

#### 33

1. if ( h Client Socket == INVALID\_ SOCKET ) {
2. mylog . loudlog (\_T(" Unable to create socket "));
3. *// Cleanup the environment initialized by WSAStartup ()*
4. WSACleanup ();
5. exit (2) ;

39 }

40 mylog . quietlog (\_T(" Client socket created "));

#### 41

1. *// Create the structure describing various Server parameters*
2. struct sockaddr\_ in server Addr ;

#### 44

1. server Addr . sin\_ family = AF\_ INET ; *// The address family. MUST be AF\_ INET*
2. size\_ t convtd ;
3. char \* p MBBuffer = new char [20];
4. wcstombs\_ s (& convtd , pMBBuffer , 20 , sz Server IPAddr , 20) ;
5. server Addr . sin\_ addr . s\_ addr = inet\_ addr ( p MBBuffer );
6. delete [] p MBBuffer ;
7. server Addr . sin\_ port = htons ( n Server Port );

#### 52

1. *// Connect to the server*
2. if ( connect ( h Client Socket , ( struct sockaddr \*) & server Addr , sizeof ( server Addr )) < 0) {
3. mylog . loudlog (\_T(" Unable to connect to % s on port % d"), sz Server IPAddr ,

n Server Port );

1. closesocket ( h Client Socket );

57

58

59

60

61

62

63

64

65

WSACleanup ();

exit (3) ;

}

mylog . quietlog (\_T(" Connect "));

*// Main loop:*

while (1) {

\_ TCHAR sz Buffer [ 1024 ];

int n Length = recv ( h Client Socket , ( char

;

\*) szBuffer , sizeof ( sz Buffer ), 0)

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

if ( n Length > 0) {

sz Buffer [ n Length ] = ’\0 ’;

mylog . loudlog (\_T("% s"), sz Buffer );

\_ tprintf (\_T(" >> "));

}

}

closesocket ( h Client Socket );

WSACleanup (); exit (0) ;

}

bool Init Win Sock 2 \_ 0 () {

WSADATA wsa Data ;

WORD w Version = MAKEWORD (2 , 0);

if (! WSAStartup ( wVersion , & wsa Data )) return true ;

return false ;

}

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

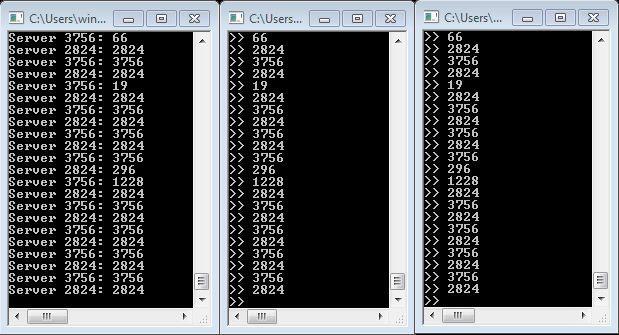


Рис. 13: Сетевая версия задачи читатели-писатели

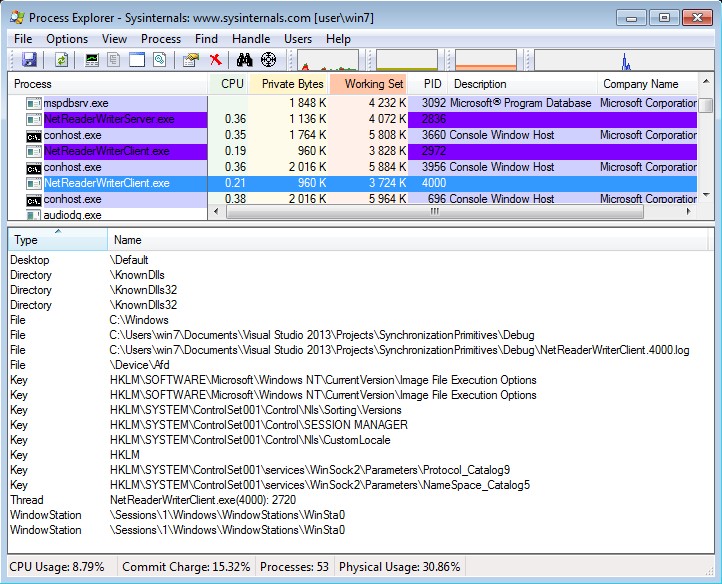


Рис. 14: Запуск process explorer для сетевой версии задачи читатели-писатели

# 6 Задача производители-потребители

Отличие этой задачи в том, что теперь каждый читатель может быть писателем. Фак- тически, 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 сни- жаться. 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 в проходит по этому списку и пересылает это событие остальным.

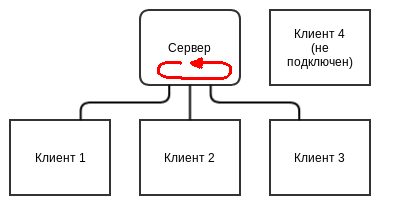


Рис. 15: Схема взаимодействия клиентом в сервером.

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 csСlients (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 беско- нечном цикле (стр 92);
* Как только происходит подключения, для работы с новым клиентом создаётся от-

дельный 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 бьется 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

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

# define \_ WINSOCK\_ DEPRECATED\_ NO\_ WARNINGS

# include

# include

< winsock 2 . h>

< list >

# include

" Logger . h"

# include

# include

# include

< time . h>

< stdio . h>

< strsafe . h>

# pragma comment ( lib , " Ws 2 \_ 32 . lib ")

struct CLIENT\_ INFO

{

SOCKET

struct

\_ TCHAR

h Client Socket ;

*// Сокет*

sockaddr\_ in client Addr ;

username [ 128 ];

*// Имя пользователя*

};

\_ TCHAR sz Server IPAddr []

int n Server Port = 5050;

= \_T(" 127 . 0 . 0 . 1 "); *// server*

*// server port*

*IP*

std :: list < CLIENT\_ INFO \*> clients ; *// Все клиенты*

CRITICAL\_ SECTION cs С lients ; *// Защита списка*

bool Init Win Sock 2 \_ 0 ();

BOOL WINAPI Client Thread ( LPVOID lp Data ); void send To All ( \_ TCHAR \* p Buffer );

*// Init log*

Logger mylog (\_T(" Full Reader Writer Server "));

int \_ tmain ( int argc , \_ TCHAR \* argv []) { if (! Init Win Sock 2 \_ 0 ()) {

1. double errorcode = WSAGet Last Error ();
2. mylog . loudlog (\_T(" Unable to Initialize Windows Socket environment , GLE =% d"), errorcode );
3. exit (1) ;

38 }

39 mylog . loudlog (\_T(" Windows Socket environment ready "));

#### 40

1. SOCKET h Server Socket ;
2. h Server Socket = socket (
3. AF\_INET , *// The address family. AF\_ INET specifies TCP/ IP*
4. SOCK\_ STREAM , *// Protocol type. SOCK\_ STREM specified TCP*
5. 0 *// Protoco Name. Should be 0 for AF\_ INET address family*

46 );

#### 47

1. if ( h Server Socket == INVALID\_ SOCKET ) {
2. mylog . loudlog (\_T(" Unable to create Server socket "));
3. *// Cleanup the environment initialized by WSAStartup ()*
4. WSACleanup ();
5. exit (2) ;

53 }

54 mylog . loudlog (\_T(" Server socket created "));

#### 55

1. *// Create the structure describing various Server parameters*
2. struct sockaddr\_ in server Addr ;

#### 58

1. server Addr . sin\_ family = AF\_ INET ; *// The address family. MUST be AF\_ INET*
2. size\_ t convtd ;
3. char \* p MBBuffer = new char [20];
4. wcstombs\_ s (& convtd , pMBBuffer , 20 , sz Server IPAddr , 20) ;
5. *// server Addr . sin\_ addr . s\_ addr = inet\_ addr ( p MBBuffer );*
6. server Addr . sin\_ addr . s\_ addr = INADDR\_ ANY ;
7. delete [] p MBBuffer ;
8. server Addr . sin\_ port = htons ( n Server Port );

#### 67

1. *// Bind the Server socket to the address & port*
2. if ( bind ( h Server Socket , ( struct sockaddr \*) & server Addr , sizeof ( server Addr

)) == SOCKET\_ ERROR ) {

1. mylog . loudlog (\_T(" Unable to bind to % s on port % d"), sz Server IPAddr , n Server Port );
2. *// Free the socket and cleanup the environment initialized by WSAStartup*

*()*

1. closesocket ( h Server Socket );
2. WSACleanup ();
3. exit (3) ;

75 }

76 mylog . loudlog (\_T(" Bind completed "));

#### 77

1. *// Put the Server socket in listen state so that it can wait for client connections*
2. if ( listen ( h Server Socket , SOMAXCONN ) == SOCKET\_ ERROR ) {
3. mylog . loudlog (\_T(" Unable to put server in listen state "));
4. *// Free the socket and cleanup the environment initialized by WSAStartup ()*
5. closesocket ( h Server Socket );
6. WSACleanup ();
7. exit (4) ;

85 }

86 mylog . loudlog (\_T(" Ready for connection on % s:% d"), sz Server IPAddr , n Server Port );

#### 87

1. *// инициализируем средство синхронизации*
2. Initialize Critical Section (& cs С lients );

#### 90

1. *// Start the infinite loop*
2. while ( true ) {
3. *// As the socket is in listen mode there is a connection request pending*

*.*

1. *// Calling accept( ) will succeed and return the socket for the request.*
2. CLIENT\_ INFO \* p Client Info = new CLIENT\_ INFO ;

#### 96

1. int n Size = sizeof ( p Client Info -> client Addr );
2. p Client Info -> h Client Socket = accept ( h Server Socket , ( struct sockaddr \*) & p Client Info -> client Addr , & n Size );
3. if ( p Client Info -> h Client Socket == INVALID\_ SOCKET ) {
4. mylog . loudlog (\_T(" accept () failed "));

#### 101 }

1. else {
2. HANDLE h Client Thread ;
3. DWORD dw Thread Id ;

#### 105

1. wchar\_ t \* sin\_ addr = new wchar\_ t [20];
2. size\_ t convtd ;
3. mbstowcs\_ s (& convtd , sin\_addr , 20 , inet\_ ntoa ( p Client Info -> client Addr . sin\_ addr ), 20) ;
4. mylog . loudlog (\_T(" Client connected from % s:% d"), sin\_addr , p Client Info

-> client Addr . sin\_ port );

1. delete [] sin\_ addr ;

#### 111

1. *// Start the client thread*
2. h Client Thread = Create Thread ( NULL , 0 ,
3. ( LPTHREAD\_ START\_ ROUTINE ) Client Thread ,
4. ( LPVOID ) p Client Info , 0 , & dw Thread Id );
5. if ( h Client Thread == NULL ) {
6. mylog . loudlog (\_T(" Unable to create client thread "));

#### 118 }

1. else {
2. Close Handle ( h Client Thread );

#### 121 }

122 }

123 }

124

1. *// удаляем объект синхронизации*
2. Delete Critical Section (& cs С lients );
3. closesocket ( h Server Socket );
4. WSACleanup ();
5. exit (0) ;

#### 130 }

131

1. bool Init Win Sock 2 \_ 0 () {
2. WSADATA wsa Data ;
3. WORD w Version = MAKEWORD (2 , 0);

#### 135

1. if (! WSAStartup ( wVersion , & wsa Data ))
2. return true ;

#### 138

139 return false ;

#### 140 }

141

1. BOOL WINAPI Client Thread ( LPVOID lp Data ) {
2. CLIENT\_ INFO \* p Client Info = ( CLIENT\_ INFO \*) lp Data ;

#### 144

1. Enter Critical Section (& cs С lients );
2. clients . push\_ front ( p Client Info ); *// Добавить нового клиента в список*
3. Leave Critical Section (& cs С lients );

#### 148

149 \_ TCHAR sz Buffer [ 1024 ];

150 \_ TCHAR sz Message [1024 + 255 + 128];

#### 151

1. int n Cnt Recv = 0;
2. int n Cnt Send = 0;

#### 154

1. *// Set username*
2. n Cnt Recv = recv ( p Client Info -> h Client Socket , ( char \*) szBuffer , sizeof ( sz Buffer ), 0);
3. if ( n Cnt Recv <= 0) {
4. mylog . loudlog (\_T(" Error reading username from client "));
5. return 1;

#### 160 }

161

1. sz Buffer [ n Cnt Recv ] = ’\0 ’;
2. String Cch Copy ( p Client Info -> username , sizeof ( p Client Info -> username ), sz Buffer );
3. swprintf\_ s ( szMessage , \_T(" System : % s has joined this chat "), p Client Info ->

username );

1. mylog . loudlog (\_T("% s"), sz Message );
2. send To All ( sz Message );

#### 167

1. *// Chat loop:*
2. while (1) {
3. n Cnt Recv = recv ( p Client Info -> h Client Socket , ( char \*) szBuffer , sizeof ( sz Buffer ), 0);
4. if ( n Cnt Recv > 0) {
5. *// Process message*
6. sz Buffer [ n Cnt Recv ] = ’\0 ’;

#### 174

1. *// Check , if its not QUIT*
2. \_ wcsdup ( sz Buffer );
3. if ( wcscmp ( szBuffer , \_T(" QUIT ")) == 0) {
4. swprintf\_ s ( szMessage , \_T(" System : % s has left this chat "), p Client Info -> username );
5. mylog . loudlog (\_T("% s"), sz Message );
6. send To All ( sz Message );

#### 181

1. Enter Critical Section (& cs С lients );
2. clients . remove ( p Client Info ); *// Удалить клиента из списка*
3. Leave Critical Section (& cs С lients );

#### 185

1. closesocket ( p Client Info -> h Client Socket );
2. delete p Client Info ;
3. return 0;

#### 189 }

190

1. *// Time*
2. struct tm newtime ;
3. \_\_ time 64 \_ t long\_ time ;
4. *// Get time as 64 - bit integer .*
5. \_ time 64 (& long\_ time );
6. *// Convert to local time.*
7. \_ localtime 64 \_ s (& newtime , & long\_ time );
8. *// Create message .*

199 swprintf\_ s ( szMessage , \_T(" [%02 d /%02 d /%04 d %02 d :%02 d :%02 d] % s: % s"), newtime . tm\_mday ,

1. newtime . tm\_ mon + 1 , newtime . tm\_ year + 1900 , newtime . tm\_hour ,
2. newtime . tm\_min , newtime . tm\_sec , p Client Info -> username , sz Buffer );

#### 202

203 mylog . loudlog (\_T("% s"), sz Message );

#### 204

205 send To All ( sz Message );

#### 206 }

1. else {
2. mylog . loudlog (\_T(" Error reading the data from % s"), p Client Info -> username );

#### 209 }

210 }

211

212 return 0;

#### 213 }

214

1. void send To All ( \_ TCHAR \* p Buffer ) {
2. *// Пока мы обрабатываем список, его ни кто не должен менять!*
3. Enter Critical Section (& cs С lients );
4. std :: list < CLIENT\_ INFO \* >:: iterator client ;
5. for ( client = clients . begin (); client != clients . end (); ++ client ) {
6. int n Length = ( lstrlen ( p Buffer ) + 1) \* sizeof ( \_ TCHAR );
7. int n Cnt Send = 0;

#### 222

1. while (( n Cnt Send = send ((\* client ) -> h Client Socket , ( char \*) pBuffer , nLength , 0) != n Length )) {
2. if ( n Cnt Send == -1) {
3. mylog . loudlog (\_T(" Error sending the data to % s"), (\* client ) -> username );

|  |  |  |  |
| --- | --- | --- | --- |
| 226 |  | | break ; |
| 227 |  | | } |
| 228 |  | | if ( n Cnt Send == n Length ) |
| 229 |  | | break ; |
| 230 |  | |  |
| 231 |  | | p Buffer += n Cnt Send ; |
| 232 |  | | n Length -= n Cnt Send ; |
| 233 | } | |  |
| 234 |  | } | |
| 235 |  | Leave Critical Section (& cs С lients ); | |
| 236 | } |  | |

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

1. # define \_ WINSOCK\_ DEPRECATED\_ NO\_ WARNINGS
2. # include < winsock 2 . h>
3. # include " Logger . h"

#### 4

5 # pragma comment ( lib , " Ws 2 \_ 32 . lib ")

#### 6

1. \_ TCHAR sz Server IPAddr [ 20 ]; *// server IP*
2. int n Server Port ; *// server port*

#### 9

1. bool Init Win Sock 2 \_ 0 ();
2. BOOL WINAPI a Reader ( LPVOID lp Data ); *// Чтение*

#### 12

1. *// Init log*
2. Logger mylog (\_T(" Full Reader Writer Client "), Get Current Process Id ());

#### 15

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. \_ tprintf (\_T(" Enter the server IP Address : "));
3. wscanf\_ s (\_T(" %19 s"), sz Server IPAddr , \_ countof ( sz Server IPAddr ));
4. \_ tprintf (\_T(" Enter the server port number : "));
5. wscanf\_ s (\_T("% i"), & n Server Port );

#### 21

1. if (! Init Win Sock 2 \_ 0 ()) {
2. double errorcode = WSAGet Last Error ();
3. mylog . loudlog (\_T(" Unable to Initialize Windows Socket environment , GLE =% d"), errorcode );
4. exit (1) ;

26 }

27 mylog . quietlog (\_T(" Windows Socket environment ready "));

#### 28

1. SOCKET h Client Socket ;
2. h Client Socket = socket (
3. AF\_INET , *// The address family. AF\_ INET specifies TCP/ IP*
4. SOCK\_ STREAM , *// Protocol type. SOCK\_ STREM specified TCP*
5. 0); *// Protoco Name. Should be 0 for AF\_ INET address family*

#### 34

1. if ( h Client Socket == INVALID\_ SOCKET ) {
2. mylog . loudlog (\_T(" Unable to create socket "));
3. *// Cleanup the environment initialized by WSAStartup ()*
4. WSACleanup ();
5. exit (2) ;

40 }

41 mylog . quietlog (\_T(" Client socket created "));

#### 42

1. *// Create the structure describing various Server parameters*
2. struct sockaddr\_ in server Addr ;

#### 45

1. server Addr . sin\_ family = AF\_ INET ; *// The address family. MUST be AF\_ INET*
2. size\_ t convtd ;
3. char \* p MBBuffer = new char [20];
4. wcstombs\_ s (& convtd , pMBBuffer , 20 , sz Server IPAddr , 20) ;
5. server Addr . sin\_ addr . s\_ addr = inet\_ addr ( p MBBuffer );
6. delete [] p MBBuffer ;
7. server Addr . sin\_ port = htons ( n Server Port );

#### 53

1. *// Connect to the server*
2. if ( connect ( h Client Socket , ( struct sockaddr \*) & server Addr , sizeof ( server Addr )) < 0) {
3. mylog . loudlog (\_T(" Unable to connect to % s on port % d"), sz Server IPAddr ,

n Server Port );

1. closesocket ( h Client Socket );
2. WSACleanup ();
3. exit (3) ;

60 }

61 mylog . quietlog (\_T(" Connect "));

#### 62

63 \_ TCHAR sz Buffer [ 1024 ] = \_T("");

#### 64

1. *// Choose username*
2. \_ tprintf (\_T(" Enter your username : "));
3. wscanf\_ s (\_T(" %1023 s"), szBuffer , \_ countof ( sz Buffer ));
4. int n Length = ( wcslen ( sz Buffer ) + 1) \* sizeof ( \_ TCHAR );

#### 69

1. *// send( ) may not be able to send the complete data in one go.*
2. *// So try sending the data in multiple requests*
3. int n Cnt Send = 0;
4. \_ TCHAR \* p Buffer = sz Buffer ;

#### 74

1. while (( n Cnt Send = send ( h Client Socket , ( char \*) pBuffer , nLength , 0) != n Length )) {
2. if ( n Cnt Send == -1) {
3. mylog . loudlog (\_T(" Error sending the data to server "));
4. break ;

79 }

1. if ( n Cnt Send == n Length )
2. break ;

#### 82

1. p Buffer += n Cnt Send ;
2. n Length -= n Cnt Send ;

85 }

#### 86

* 1. *// Запуск читающего треда*
  2. HANDLE ha Reader = Create Thread ( NULL , 0 ,
  3. ( LPTHREAD\_ START\_ ROUTINE ) aReader ,
  4. ( LPVOID )& h Client Socket , 0 , 0);
  5. if ( ha Reader == NULL ) {
  6. mylog . loudlog (\_T(" Unable to create Reader thread "));

93 }

#### 94

1. *// Chat loop:*
2. \_ tprintf (\_T(" Enter your messages or QUIT for exit .\ n"));
3. while ( wcscmp ( szBuffer , \_T(" QUIT ")) != 0) {

98 \_ tprintf (\_T(" >> "));

99 wscanf\_ s (\_T(" %1023 s"), szBuffer , \_ countof ( sz Buffer ));

#### 100

101 n Length = ( wcslen ( sz Buffer ) + 1) \* sizeof ( \_ TCHAR );

#### 102

1. *// send( ) may not be able to send the complete data in one go.*
2. *// So try sending the data in multiple requests*
3. n Cnt Send = 0;
4. p Buffer = sz Buffer ;

#### 107

1. while (( n Cnt Send = send ( h Client Socket , ( char \*) pBuffer , nLength , 0) != n Length )) {
2. if ( n Cnt Send == -1) {
3. mylog . loudlog (\_T(" Error sending the data to server "));
4. break ;

#### 112 }

1. if ( n Cnt Send == n Length )
2. break ;

#### 115

1. p Buffer += n Cnt Send ;
2. n Length -= n Cnt Send ;

#### 118 }

119

1. \_ wcsdup ( sz Buffer );
2. if ( wcscmp ( szBuffer , \_T(" QUIT ")) == 0) {
3. Terminate Thread ( haReader , 0);
4. break ;

#### 124 }

125 }

126

1. closesocket ( h Client Socket );
2. WSACleanup ();

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

exit (0) ;

}

bool Init Win Sock 2 \_ 0 () {

WSADATA wsa Data ;

WORD w Version = MAKEWORD (2 , 0);

if (! WSAStartup ( wVersion , & wsa Data )) return true ;

return false ;

}

BOOL WINAPI a Reader ( LPVOID lp Data ) {

SOCKET \* h Client Socket = ( SOCKET \*) lp Data ;

\_ TCHAR sz Buffer [ 1024 ]; int n Length = 0;

while (1) {

n Length = recv (\* h Client Socket , ( char \*) szBuffer ,

sizeof ( sz Buffer ), 0);

if ( n Length > 0) {

sz Buffer [ n Length ] = ’\0 ’;

mylog . loudlog (\_T("% s"), sz Buffer );

\_ tprintf (\_T(" >> "));

}

}

return 0;

}

### 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.

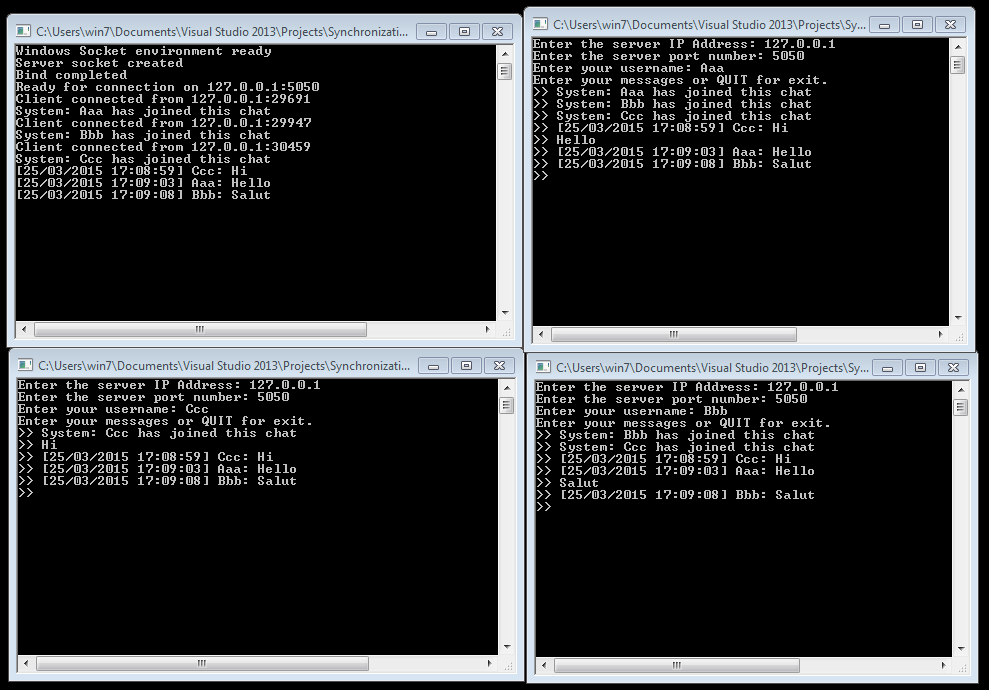


Рис. 16: Полноценный чат на Win-сокетах.

# 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

2

3

4

5

6

7

8

# include

# include

# include

< windows . h>

< conio . h>

< tchar . h>

# include

" Logger . h"

# define N

# define LEFT

5

*// всего философов*

( id+ N -1) % N *// левый сосед*

9 # define RIGHT ( id +1) % N *// правый сосед*

#### 10

1. # define THINKING 0 *// состояние размышления*
2. # define HUNGRY 1 *// состояние голода*
3. # define EATING 2 *// философ ест*

#### 14

1. int philosopher\_ state [ N]; *// состояние философов*
2. CRITICAL\_ SECTION crs [ N]; *// Объявление критических секций*

#### 17

1. DWORD WINAPI philosopher Thread ( LPVOID prm );
2. void take\_ forks ( int id);
3. void put\_ forks ( int id);
4. void think ();
5. void eat ();
6. void wait ();

#### 24

1. *// Init log*
2. Logger mylog (\_T(" Dining Philosophers Problem "));

#### 27

1. *// Размышления*
2. void think () {

30 Sleep (100 + rand () % 500) ;

31 }

#### 32

1. *// Еда*
2. void eat () {

35 Sleep (50 + rand () % 450) ;

36 }

#### 37

1. *// Голодное ожидание*
2. void wait () {

40 Sleep (50 + rand () % 150) ;

41 }

#### 42

1. *// симуляция жизни философа*
2. DWORD WINAPI philosopher Thread ( LPVOID prm ) {
3. int phil\_ id = ( int ) prm ;
4. while ( true ) {
5. think ();
6. *// Либо обе вилки, либо блокировка*
7. take\_ forks ( phil\_ id );
8. eat ();
9. *// Вернуть вилки на стол*
10. put\_ forks ( phil\_ id );

53 }

54 }

#### 55

1. void take\_ forks ( int id) {
2. bool done = false ;
3. while (! done ){
4. if ( rand () % 2) { *// left hand first*
5. mylog . quietlog (\_T(" Waining for Critical Section "));
6. Enter Critical Section (& crs [ id ]);
7. mylog . quietlog (\_T(" Get Critical Section "));

#### 63

1. philosopher\_ state [ id] = HUNGRY ;
2. mylog . loudlog (\_T(" Philosopher % d status HUNGRY "), id);

#### 66

1. if ( Try Enter Critical Section (& crs [ LEFT ])) {
2. if ( philosopher\_ state [ LEFT ] != EATING ) {
3. if ( Try Enter Critical Section (& crs [ RIGHT ])) {
4. if ( philosopher\_ state [ RIGHT ] != EATING ) {
5. philosopher\_ state [ id] = EATING ;
6. mylog . loudlog (\_T(" Philosopher % d status EATING "), id);
7. done = true ;

74 }

75 Leave Critical Section (& crs [ RIGHT ]);

76 }

77 }

78 Leave Critical Section (& crs [ LEFT ]);

79 }

1. mylog . quietlog (\_T(" Leave Critical Section "));
2. Leave Critical Section (& crs [ id ]);

82 }

1. else { *// right hand first*
2. mylog . quietlog (\_T(" Waining for Critical Section "));
3. Enter Critical Section (& crs [ id ]);
4. mylog . quietlog (\_T(" Get Critical Section "));

#### 87

1. philosopher\_ state [ id] = HUNGRY ;
2. mylog . loudlog (\_T(" Philosopher % d status HUNGRY "), id);

#### 90

1. if ( Try Enter Critical Section (& crs [ RIGHT ])) {
2. if ( philosopher\_ state [ RIGHT ] != EATING ) {
3. if ( Try Enter Critical Section (& crs [ LEFT ])) {
4. if ( philosopher\_ state [ LEFT ] != EATING ) {
5. philosopher\_ state [ id] = EATING ;
6. mylog . loudlog (\_T(" Philosopher % d status EATING "), id);
7. done = true ;

98 }

99 Leave Critical Section (& crs [ LEFT ]);

#### 100 }

101 }

102 Leave Critical Section (& crs [ RIGHT ]);

#### 103 }

1. mylog . quietlog (\_T(" Leave Critical Section "));
2. Leave Critical Section (& crs [ id ]);

#### 106 }

107

1. if (! done )
2. wait ();

#### 110 }

111 }

112

1. void put\_ forks ( int id) {
2. mylog . quietlog (\_T(" Waining for Critical Section "));
3. Enter Critical Section (& crs [ id ]);
4. mylog . quietlog (\_T(" Get Critical Section "));

#### 117

1. philosopher\_ state [ id] = THINKING ;
2. mylog . loudlog (\_T(" Philosopher % d status THINKING "), id);

#### 120

1. mylog . quietlog (\_T(" Leave Critical Section "));
2. Leave Critical Section (& crs [ id ]);

#### 123 }

124

1. int \_ tmain ( int argc , \_ TCHAR \* argv []) {
2. *// Массив потоков*
3. HANDLE allhandlers [ N];

#### 128

1. *// создаем потоки- читатели*
2. mylog . loudlog (\_T(" Create threads "));

131 for ( int i = 0; i != N; ++ i) {

1. mylog . loudlog (\_T(" Count = % d"), i);
2. *// создаем потоки- читатели, которые пока не стартуют*
3. if (( allhandlers [ i] = Create Thread ( NULL , 0 , philosopher Thread , ( LPVOID )i

, CREATE\_ SUSPENDED , NULL )) == NULL ) {

1. mylog . loudlog (\_T(" Impossible to create thread - reader , GLE = % d"), Get Last Error ());

136 exit ( 8000 ) ;

#### 137 }

138 }

139

140 *// инициализируем средство синхронизации*

141 for ( int i = 0; i != N; ++ i) {

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

Initialize Critical Section (& crs [ i]);

}

*// запускаем потоки на исполнение*

for ( int i = 0; i < N; ++ i) Resume Thread ( allhandlers [ i]);

*// ожидаем завершения всех потоков*

Wait For Multiple Objects (N, allhandlers ,

*// закрываем handle потоков*

for ( int i = 0; i < N; ++ i) Close Handle ( allhandlers [ i]);

TRUE , INFINITE );

*// удаляем объект синхронизации*

for ( int i = 0; i != N; ++ i) { Delete Critical Section (& crs [ i]);

}

*// Завершение работы*

mylog . loudlog (\_T(" All tasks are done !"));

\_ getch (); return 0;

}

### 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).

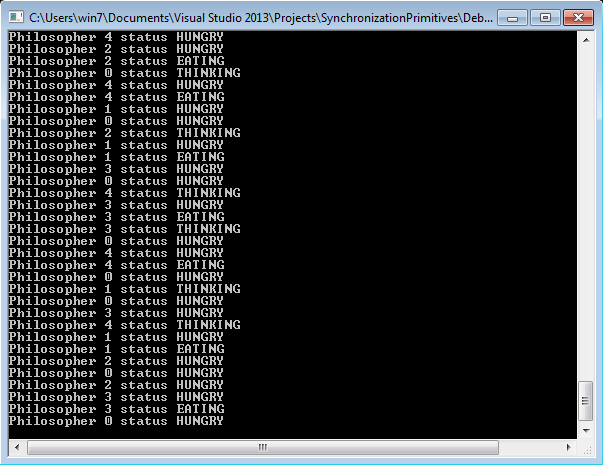


Рис. 17: Эмулятор жизни философа.

Листинг 48: Отрывок протокол работы эмулятора обедающих философов

1 [ 25 / 3 / 2015

2 [ 25 / 3 / 2015

3 [ 25 / 3 / 2015

4 [ 25 / 3 / 2015

5 [ 25 / 3 / 2015

6 [ 25 / 3 / 2015

7 [ 25 / 3 / 2015

8 [ 25 / 3 / 2015

9 [ 25 / 3 / 2015

10 [ 25 / 3 / 2015

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

Dining Philosophers Problem is starting .

Create threads

Count

Count Count Count Count

=

=

=

=

=

Waining

0

1

2

3

4

for Critical

Section

Get Critical Section

Philosopher 1 status

HUNGRY [ 25 / 3 / 2015

20 : 27 : 26 ] Waining

11

12

for Critical Section

[ 25 / 3 / 2015 20 : 27 : 26 ] Get Critical Section [ 25 / 3 / 2015 20 : 27 : 26 ] Philosopher 0 status

for Critical Section

[ 25 / 3 / 2015 20 : 27 : 26 ] Get Critical Section [ 25 / 3 / 2015 20 : 27 : 26 ] Philosopher 3 status

for Critical Section

[ 25 / 3 / 2015 20 : 27 : 26 ] Get Critical Section

HUNGRY [ 25 / 3 / 2015 20 : 27 : 26 ] Waining

13

14

HUNGRY [ 25 / 3 / 2015

20 : 27 : 26 ]

Waining

15

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

1. [ 25 / 3 / 2015 20 : 27 : 26 ] Philosopher 4 status HUNGRY
2. [ 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

1. [ 25 / 3 / 2015 20 : 27 : 26 ] Philosopher 2 status EATING
2. [ 25 / 3 / 2015 20 : 27 : 26 ] Leave Critical Section

#### 27

1. [ 25 / 3 / 2015 20 : 27 : 26 ] Philosopher 0 status EATING
2. [ 25 / 3 / 2015 20 : 27 : 26 ] Leave Critical Section
3. [ 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 ]

1. Waining for Critical Section [ 25 / 3 / 2015 20 : 27 : 26 ]
2. Philosopher 1 status HUNGRY [ 25 / 3 / 2015 20 : 27 : 26 ] Get Critical Section
3. [ 25 / 3 / 2015 20 : 27 : 26 ] Philosopher 3 status HUNGRYWaining for Critical Section

35 [ 25 / 3 / 2015 20 : 27 : 26 ] Get Critical Section

1. [ 25 / 3 / 2015 20 : 27 : 26 ] Philosopher 4 status HUNGRY
2. [ 25 / 3 / 2015 20 : 27 : 26 ] Leave Critical Section

#### 38

39 [ 25 / 3 / 2015 20 : 27 : 26 ] Leave Critical Section

#### 40

1. [ 25 / 3 / 2015 20 : 27 : 26 ] Leave Critical Section
2. [ 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

1. [ 25 / 3 / 2015 20 : 27 : 26 ] Philosopher 2 status THINKING
2. [ 25 / 3 / 2015 20 : 27 : 26 ] Leave Critical Section

#### 48

1. [ 25 / 3 / 2015 20 : 27 : 26 ] Leave Critical Section
2. [ 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

1. [ 25 / 3 / 2015 20 : 27 : 26 ] Philosopher 0 status HUNGRY
2. [ 25 / 3 / 2015 20 : 27 : 26 ] Philosopher 2 status EATING
3. [ 25 / 3 / 2015 20 : 27 : 26 ] Leave Critical Section

57

58 [ 25 / 3 / 2015

59 [ 25 / 3 / 2015

60 [ 25 / 3 / 2015

61 [ 25 / 3 / 2015

62 [ 25 / 3 / 2015

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

Philosopher 0 status EATING

Leave Critical Section

Waining for Critical

Get Critical Section Philosopher 1 status

Section

HUNGRY [ 25 / 3 / 2015 20 : 27 : 26 ] Waining

63

64

for Critical Section

[ 25 / 3 / 2015 20 : 27 : 26 ] Get Critical Section [ 25 / 3 / 2015 20 : 27 : 26 ] Philosopher 3 status

for Critical Section

HUNGRY [ 25 / 3 / 2015 20 : 27 : 26 ] Waining

65

66

67

68

69

70

71

[ 25 / 3 / 2015

[ 25 / 3 / 2015 [ 25 / 3 / 2015

20 : 27 : 26 ]

20 : 27 : 26 ]

20 : 27 : 26 ]

Get Critical Section

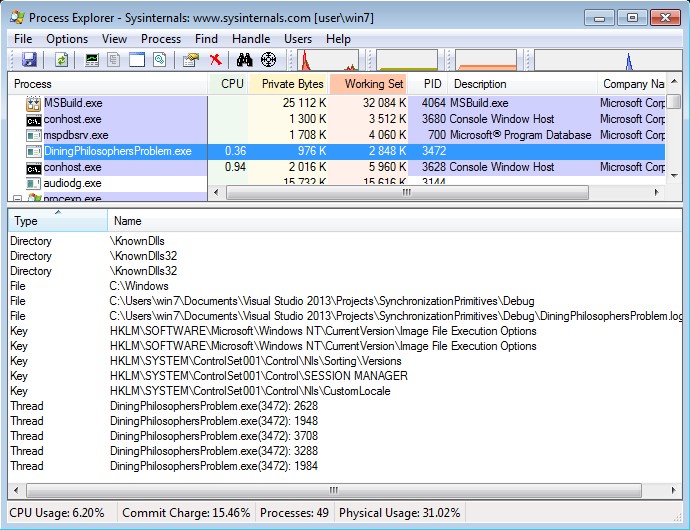
Philosopher 4 status

HUNGRY

Leave Critical Section

[ 25 / 3 / 2015 20 : 27 : 26 ] Leave Critical Section

[ 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 одновремен- ного 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 одно- временно смогут работать с ресурсом. Каждый раз, когда программа обращается к семафору, значение счетчика ресурсов семафора уменьшается на единицу. Когда значение счетчика ресурсов становится равным нолю, семафор недоступен.

* + Критические секции – это объект, который принадлежи 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 яв- ляются 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 отличии

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 использо- вана 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 межпотокового взаимо- действия 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.