Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)**

Кафедра технологий электронного обучения (ТЭО)

МЕТОДЫ И ТЕХНОЛОГИИ РАЗРАБОТКИ КЛИЕНТ-СЕРВЕРНЫХ ПРИЛОЖЕНИЙ

Методические указания к практическим занятиям
и организации самостоятельной работы аспирантов

2018

Кручинин Владимир Викторович, Морозова Юлия Викторовна
Методы и технологии разработки клиент-серверных приложений : Методические указания к практическим занятиям и организации самостоятельной работы аспирантов . – Томск: Томский государственный университет систем управления и радиоэлектроники. – 62 с.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Дисциплина «Методы и технологии разработки клиент-серверных приложений» является дисциплиной подготовки аспирантов направления «Информатика и вычислительная техника», по профилю «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей».

Целью данной дисциплины является: формирование навыков разработки и исследования программного обеспечения компьютерных сетей; повышение квалификации в области научных основ и применении программного обеспечения компьютерных сетей для решения фундаментальных научных и прикладных научно-технических проблем; получение знаний в области теории программирования, создание и сопровождение сетевых программных средств на основе различных подходов и технологий. Значение решения указанных проблем состоит в повышении эффективности и надежности процессов обработки и передачи данных и знаний в вычислительных машинах, комплексах и компьютерных сетях.

Основные задачи дисциплины:

1) подготовка научных и научно-технических публикаций и отчетов НИР в области системного программного обеспечения;

2) разработка подходов, технологий, алгоритмов и программных комплексов на основе модели клиент-сервер;

3) планирование процессов и ресурсов для решения задач в области прикладной математики, информатики и системного программного обеспечения;

4) использование методов разработки алгоритмического и программного обеспечения в научно-исследовательской, педагогической и производственно-технологической деятельности, включая разработку решений в области системного и прикладного программирования.

В результате освоения дисциплины аспирант должен:

1. Знать основные понятия и методы разработки сетевых приложений; языки программирования и системы программирования для разработки сетевых приложений.

2. Уметь применять принципы и методы анализа и разработки сетевых приложений; разрабатывать новые методы анализа и разработки программного обеспечения компьютерных сетей; анализировать, получать знания с помощью самостоятельной работы с печатными источниками; применять полученные теоретические знания при решении практических задач, строить простейшие модели в различных областях знаний; демонстрировать способность уметь работать самостоятельно, расширять свои математические знания и проводить математический анализ прикладных инженерных задач; разрабатывать модели, методы, алгоритмы, языки и программные инструменты для программного обеспечения компьютерных

сетей; разрабатывать модели и методы создания программ и программных систем для параллельной и распределенной обработки данных

3. Владеть способностью к участию в работах по разработке алгоритмического и программного обеспечения компьютерных сетей, комплексным исследованием научных и технических проблем с применением современной технологии математического моделирования; методами, алгоритмами и программными средствами для организации глобально распределенной обработки данных.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

2.1 Практическое занятие «Дочерние процессы. Совместный доступ нескольких процессов к одному файлу»

Цель

Изучение и получение практических навыков по созданию приложений, создающих и взаимодействующих с процессами.

Основные понятия

Процесс. Адресное пространство, первичный поток, переменные окружения, структуры STARTUP и PROCESSINFO, файл, каталог, распределенная файловая система.

Методические указания

Ниже приведены примеры написания программ:

1. Запуск процесса и ожидание его завершения

```
#include <windows.h>
#include <windowsx.h>
#include <tchar.h>
#include <process.h>

int WINAPI _tWinMain(HINSTANCE hinstExe, HINSTANCE, PTSTR
pszCmdLine, int) {
     STARTUPINFO si= { sizeof(si) };
     PROCESS_INFORMATION pi;
    //создаем процесс
     CreatePro-
cess(NULL,TEXT("NOTEPAD"),NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)
;
     //ждем завершения процесса
     WaitForSingleObject(pi.hProcess, INFINITE);

     CloseHandle(g_hThreads[g_nNumThreads]);

     MessageBox(NULL,"1","2",MB_OK);
   return(0);
}
```

2. Использование анонимных каналов для передачи информации между основным и дочерним процессами

```
//Основной процесс
// AnonymPipe.cpp
#include "stdafx.h"
#include <stdio.h>
#include <windows.h>

#define BUFSIZE 4096 //размер буфера
//объявление дескрипторов
HANDLE hChildStdinRd,
```

```
        hChildStdinWr,
        hChildStdinWrDup,
        hChildStdoutRd,
        hChildStdoutWr,
        hChildStdoutRdDup,
        hInputFile,
        hSaveStdin,
        hSaveStdout;
 //функции
BOOL CreateChildProcess(VOID);
VOID WriteToPipe(VOID);
VOID ReadFromPipe(VOID);
VOID ErrorExit(LPTSTR);
VOID ErrMsg(LPTSTR, BOOL);

DWORD main(int argc, char *argv[])
{
   SECURITY_ATTRIBUTES saAttr;
   BOOL fSuccess;

/* устанавливаем bInheritHandle flag, чтобы  дескрипоторы как-
нала наследовались */


   saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
   saAttr.bInheritHandle = TRUE;
   saAttr.lpSecurityDescriptor = NULL;

   // выполняет следующие шаги по перенаправлению STDOUT:
   //1.Сохраняем текущий STDOUT, чтобы потом восстановить его.
   //2. Создаем анонимный канал для STDOUT дочернего процесса.
   //3. Устанавливаем STDOUT для основного процесса
   /*4. Создаем ненаследуемый дескриптор, копируем и закрываем
наследуемый.*/

//Сохраняем текущий  STDOUT

   hSaveStdout = GetStdHandle(STD_OUTPUT_HANDLE);

// Создаем анонимный канал

  if (! CreatePipe(&hChildStdoutRd,&hChildStdoutWr,&saAttr,0))
      ErrorExit("Stdout pipe creation failed\n");

// Устанавливаем дескриптор для записи.

   if (! SetStdHandle(STD_OUTPUT_HANDLE, hChildStdoutWr))
      ErrorExit("Redirecting STDOUT failed");
```

```
/* Создаем ненаследуемый дескриптор для чтенияCreate и закры-
ваем наследуемый. */

    fSuccess = DuplicateHandle(GetCurrentProcess(),
hChildStdoutRd,
        GetCurrentProcess(), &hChildStdoutRdDup , 0,
        FALSE,
        DUPLICATE_SAME_ACCESS);
    if( !fSuccess )
        ErrorExit("DuplicateHandle failed");
    CloseHandle(hChildStdoutRd);

   // Шаги по перенаправлению STDIN дочернего процесса:
   //1.  Сохраняем текущий дескриптор STDIN.
   //2.  Создаем анонимный канал для STDIN  дочернего процесса
   //3.  Устанавливаем родительский STDIN для чтения из канала
   /*4.  Создаем ненаследуемый дескриптор для записи и закры-
ваем наследуемый */

// Сохраняем текущий дескриптор STDIN.
   hSaveStdin = GetStdHandle(STD_INPUT_HANDLE);

// Создаем анонимный канал для STDIN дочернего процесса.


   if (! CreatePipe(&hChildStdinRd, &hChildStdinWr, &saAttr,
0))
      ErrorExit("Stdin pipe creation failed\n");

// Устанавливаем родительский STDIN для чтения из канала.

   if (! SetStdHandle(STD_INPUT_HANDLE, hChildStdinRd))
      ErrorExit("Redirecting Stdin failed");

/* Создаем ненаследуемый дескриптор для записи и закрываем
наследуемый. */

   fSuccess = DuplicateHandle(GetCurrentProcess(),
hChildStdinWr,
      GetCurrentProcess(), &hChildStdinWrDup, 0,
      FALSE,                  // not inherited

      DUPLICATE_SAME_ACCESS);
   if (! fSuccess)
      ErrorExit("DuplicateHandle failed");

   CloseHandle(hChildStdinWr);

// Создаем дочерний процесс
   if (! CreateChildProcess())
```

```
      ErrorExit("Create process failed");

// После создания закрываем сохраненные STDIN и STDOUT.

   if (! SetStdHandle(STD_INPUT_HANDLE, hSaveStdin))
      ErrorExit("Re-redirecting Stdin failed\n");

   if (! SetStdHandle(STD_OUTPUT_HANDLE, hSaveStdout))

      ErrorExit("Re-redirecting Stdout failed\n");

/* Получаем дескриптор для файла, записанного в командной
строке. */
    if (argc > 1)
      hInputFile = CreateFile(argv[1], GENERIC_READ, 0, NULL,
         OPEN_EXISTING, FILE_ATTRIBUTE_READONLY, NULL);
   else
      hInputFile = hSaveStdin;

   if (hInputFile == INVALID_HANDLE_VALUE)
      ErrorExit("no input file\n");

//пишем в канал дочернего  процесса

   WriteToPipe();

/* Читаем из анонимного канала данные, записанные дочерним
процессом.*/


   ReadFromPipe();

   return 0;
}

//функция запуска дочернего процесса
BOOL CreateChildProcess()
{
   PROCESS_INFORMATION piProcInfo;
   STARTUPINFO siStartInfo;
 // устанавливаем структуру STARTUPINFO

   ZeroMemory( &siStartInfo, sizeof(STARTUPINFO) );
   siStartInfo.cb = sizeof(STARTUPINFO);

// создаем дочерний процесс

   return CreateProcess(NULL,
      "child", // командная строка
      NULL,   // атрибуты защиты
```

```
      NULL,
      TRUE,   //наследовать дескрипторы родительского процесса
      0,              // флаг создания
      NULL,           // использование родительских ресурсов
      NULL,           // использование родительских каталогов
      &siStartInfo,  //указатель на структуру STARTUPINFO
      &piProcInfo); /* указатель на структуру PRO-
CESS_INFORMATION */
}

//функция передачи данных
VOID WriteToPipe(VOID)
{
   DWORD dwRead, dwWritten;
   CHAR chBuf[BUFSIZE];

//Читаем данные из файла и пишем их в канал
   for (;;)
   {
    if (! ReadFile(hInputFile, chBuf, BUFSIZE, &dwRead, NULL)
||
         dwRead == 0) break;
//пишем в анонимный канал
      if (! WriteFile(hChildStdinWrDup, chBuf, dwRead,
         &dwWritten, NULL)) break;
   }
//читаем и пишем, пока dwRead больше нуля
// Закрываем канал для дочернего процесса
   if (! CloseHandle(hChildStdinWrDup))
      ErrorExit("Close pipe failed\n");
}

//Функция чтения из канала
VOID ReadFromPipe(VOID)
{
   DWORD dwRead, dwWritten;
   CHAR chBuf[BUFSIZE];
   HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);

// Close the write end of the pipe before reading from the
// read end of the pipe.
   if (!CloseHandle(hChildStdoutWr))
      ErrorExit("Closing handle failed");

// Read output from the child process, and write to parent's
STDOUT.

   for (;;)
   {
    if( !ReadFile( hChildStdoutRdDup, chBuf, BUFSIZE, &dwRead,
```

```
         NULL) || dwRead == 0) break;
    if (! WriteFile(hSaveStdout, chBuf, dwRead, &dwWritten,
NULL))
         break;
   }
}
 //функция выдачи сообщения об ошибке
VOID ErrorExit (LPTSTR lpszMessage)
{
   fprintf(stderr, "%s\n", lpszMessage);

   ExitProcess(0);
}
```

Дочерний процесс

Для запуска дочернего процесса необходимо создать exe-программу с именем child.exe. Эта программа в цикле читает из анонимного канала stdin и пишет в анонимный канал stdout.

```
#include "stdafx.h"
#include <windows.h>
#define BUFSIZE 4096

VOID main(VOID)
{
   CHAR chBuf[BUFSIZE];
   DWORD dwRead, dwWritten;
   HANDLE hStdin, hStdout;
   BOOL fSuccess;

//получаем дескрипторы stdout и stdin
   hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
   hStdin = GetStdHandle(STD_INPUT_HANDLE);
   if ((hStdout == INVALID_HANDLE_VALUE) ||
      (hStdin == INVALID_HANDLE_VALUE))
      ExitProcess(1);
   for (;;)
   {
   // Читаем из канала
   fSuccess = ReadFile(hStdin, chBuf, BUFSIZE, &dwRead, NULL);
      if (! fSuccess || dwRead == 0)
         break;
    MessageBox(NULL,chBuf,"1",MB_OK);
   // Пишем в канал
     fSuccess = WriteFile(hStdout, chBuf, dwRead, &dwWritten,
NULL);
      if (! fSuccess)
         break;
   }
}
```

3. Взаимодействие процессов через общую память.

Операционная система MS Windows позволяет отображать виртуальную память разных процессов на одну и ту же реальную, тем самым создает общую память для нескольких процессов. Для этого создается объект Mapping с помощью функции CtreatreMappingFile() и получается его адрес в виртуальном пространстве процесса с помощью функции MapViewOfFile(). Ниже приведен простой код программы, показывающий взаимодействие двух процессов через общую память.

```
#include <windows.h>

int PASCAL WinMain(HINSTANCE hCurInstance, HINSTANCE hPrevIn-
stance, LPSTR lpCmdLine, int nCmdShow)
{
    //Создаем мапированный объект
    HANDLE hmap=CreateFileMapping(
    (HANDLE)0xFFFFFFFF,
     NULL,
    PAGE_READWRITE  ,    // для чтения и записи
    0,    // старшие  32 бита блока памяти
    1024, // младшие 32 бита блока памяти
    "mapping"  //имя объекта мапирования
   );

   if(hmap==NULL){ //объект не создан
     MessageBox(NULL,"Error Create","",MB_OK);
     return 0;
   }
   LPVOID lpMapAddress;
// Handle to mapping object.
   lpMapAddress = MapViewOfFile(hmap,
// Read/write permission
    FILE_MAP_ALL_ACCESS,
    0,                    // Max. object size.
    0,                    // Size of hFile.
    0);                   // Map entire file.

if (lpMapAddress == NULL) {
    MessageBox(NULL,"Error View","",MB_OK);
}
    lstrcpy((char*)lpMapAddress,"Hello process");
   MessageBox(NULL,(char*)lpMapAddress,"1",MB_OK);
   if (!UnmapViewOfFile(lpMapAddress)) {
     MessageBox(NULL,"Error View","",MB_OK);
}
    CloseHandle(hmap);
   return 0;
}
```

Задание

1. Изучить функции:

- CreateProcess()
- CreateFile()
- WriteFile()
- ReadFile()
- CreatePipe()
- CreateFileMapping()
- CloseHandle()
- MapViewOfFile()

2. Изучить структуры:

- STARTUPINFO
- PROCESS_INFORMATION

3. Написать программы:

- Запуска процесса.
- Взаимодействия нескольких процессов для чтения и записи из одного файла в локальной сети.
- Обмена информацией между основным процессом и дочерним процессом с использованием анонимных каналов
- Взаимодействия процессов с использованием общей памяти.

Вопросы для самоконтроля

1. Дайте понятие процесса, приведите его основные свойства, структуру и описание функции запуска.

2. Дайте понятие потока, опишите содержимое контекста потока, функции создания потока и ее параметров.

3. Опишите предназначение и использование переменных окружения.

4. Как определить дескриптор процесса?

5. Как определить дескриптор первичного потока?

6. Как определить текущий каталог?

7. Дайте определение каналам (pipes).

8. В чем отличие именованных каналов от анонимных?

9. Перечислите основные функции для работы с каналами.

10. Дайте понятие «мапирование файлов».

11. Общая схема использования мапирования файлов для организации обмена данными между процессами.

12. Перечислите основные функции мапирования файлов.

2.2 Практическое занятие «Синхронизация потоков и процессов»

Цель

Изучение основ построения многопоточных и распределенных приложений, алгоритмов и структур данных синхронизации таких приложений. Получение навыков программирования простых примеров.

Основные понятия

Объекты синхронизации. Функции ожидания. События. Семафоры. Мьютексы. Ожидаемые таймеры. Сценарии управления потоками.

Методические указания

1. Синхронизация двух процессов: один пишет (Writer) в общую память, другой читает (Reader). Операции записи и чтения должны быть синхронизированы.

```
//процесс создает мапированную память, без указания на файл
// и пишет туда строку символов

#include <windows.h>
#include <process.h>

int PASCAL WinMain(HINSTANCE hCurInstance, HINSTANCE hPrevIn-
stance, LPSTR lpCmdLine, int nCmdShow){

    MessageBox(NULL,"Start Writer","1",MB_OK);
     HANDLE hEventWrit-
er=CreateEvent(NULL,FALSE,TRUE,"Process2MappingEventWriter");
    HANDLE hEventRead-
er=CreateEvent(NULL,FALSE,FALSE,"Process2MappingEventReader");

    HANDLE hmap=CreateFileMapping((HANDLE)0xFFFFFFFF,NULL,
    PAGE_READWRITE  ,    // возможна запись и чтение
    0,    // high-order 32 bits размер памяти
    1024, // low-order 32 bits размер памяти
    "mapping"  // Имя объекта (должно быть уникальным)
   );
   if(hmap==NULL) MessageBox(NULL,"Error Create","",MB_OK);

   LPVOID lpMapAddress;
   lpMapAddress = MapViewOfFile(hmap,
// дескриптор мапированного объекта.
    FILE_MAP_ALL_ACCESS,   // разрешение на чтение и запись
    0,                     // Max. object size.
    0,                     // Size of hFile.
    0);                    // Map entire file.

if (lpMapAddress == NULL) { MessageBox(NULL,"Error
View","",MB_OK); }
```

```
for(int i=0; i<10; i++){
     if(WaitForSingleObject(hEventWriter,INFINITE)==WAIT_OBJECT
_0)
     {
          //копируем в память
     *(int*)lpMapAddress=i;
      SetEvent(hEventReader);
       //MessageBox(NULL,"Write","1",MB_OK);
     }
}

   if (!UnmapViewOfFile(lpMapAddress)) {
     MessageBox(NULL,"Error View","",MB_OK); }
   CloseHandle(hmap);
   CloseHandle(hEventWriter);
   CloseHandle(hEventReader);
   MessageBox(NULL,"End process writer","1",MB_OK);
   return 0;
}

//процесс, читающий из мапированного объекта

#include <windows.h>

int PASCAL WinMain(HINSTANCE hCurInstance, HINSTANCE hPrevIn-
stance, LPSTR lpCmdLine, int nCmdShow){
HANDLE hMapFile;
LPVOID lpMapAddress;
char num[20];

//ждем команды на чтение
MessageBox(NULL,"Start Reader","1",MB_OK);

HANDLE hEventWriter, hEventReader;
while((hEventWriter=OpenEvent(EVENT_ALL_ACCESS,FALSE,"Process2
MappingEventWriter"))==NULL) Sleep(100);
while((hEventReader=OpenEvent(EVENT_ALL_ACCESS,FALSE,"Process2
MappingEventReader"))==NULL) Sleep(100);

//открываем мапированный объект по имени
hMapFile = OpenFi-
leMapping(FILE_MAP_ALL_ACCESS,FALSE,"mapping");
if (hMapFile == NULL) {  MessageBox(NULL,"Error
Open","",MB_OK); }

// создаем указатель на память
lpMapAddress = MapViewOfFile(hMapFile, //дескриптор
FILE_MAP_ALL_ACCESS, // разрешение на чтение и запись
    0,               // максимальный размер
    0,               // размер файла
```

```
    0);              // отображение
if (lpMapAddress == NULL) {
     MessageBox(NULL,"ErrAddr","2",MB_OK); }
//читаем из него строку, записанную первым процессом
for(int i=0; i<10; i++){
     if(WaitForSingleObject(hEventReader,INFINITE)==WAIT_OBJECT
_0)
     {
          //копируем в память
     wsprintf(num,"count = %d",*(int*)lpMapAddress);
      SetEvent(hEventWriter);
      MessageBox(NULL,num,"ReaderProcess",MB_OK);
     }
}

//все закрываем
 if (!UnmapViewOfFile(lpMapAddress)) {      Message-
Box(NULL,"Error View","",MB_OK);
}
 CloseHandle(hMapFile);
 CloseHandle(hEventWriter);
 CloseHandle(hEventReader);
 MessageBox(NULL,"End process","2",MB_OK);
 return 0;}
```

2. Синхронизация потоков в пользовательском режиме.

Приведенная ниже программа находит сумму вектора с помощью двух потоков. Суммирование производится параллельно. Для передачи начальных значений в поток используется структура TREATVECTOR. В качестве функции синхронизации используется InterlockedExchangeAdd.

```
#include "stdafx.h"
#include <windows.h>
#include <windowsx.h>
#include <tchar.h>
#include <process.h>

typedef struct {
     int size; //число элементов в векторе
     int *ptr; //указатель на первый элемент
     int nProcess; //номер потока
} TREATVECTOR;

//дескрипторы потоков
HANDLE hThread[2];
//значения параметров для двух потоков
TREATVECTOR v[2];
//вектор, сумму которого нужно найти
int vector[11]={ 1,2,3,4,5,6,7,8,9,10,11};

//общая переменная для двух потоков для накопления суммы
```

```
//выравненная по границе слова, без оптимизации

__declspec(align(4)) volatile long sum=0;

//функция потока
unsigned __stdcall ThreadSum(PVOID pvParam) {
   TCHAR str[20];
//преобразование указателя
   TREATVECTOR *v=(TREATVECTOR*)pvParam;
   for(int i=0; i<v->size; i++)
   {
//организуем атомарный доступ к переменной sum при добавлении
//sum=sum+v[i]
      InterlockedExchangeAdd(&sum,(long)v->ptr[i]);
     wsprintf(str,"Process =%d Sum=%ld",v->nProcess,sum);
     MessageBox(NULL,str,"Суммирование",MB_OK);
   }
   return(0);
}

int WINAPI _tWinMain(HINSTANCE hinstExe, HINSTANCE, PTSTR
pszCmdLine, int) {

     unsigned int dwThreadID;
    TCHAR str[20];
     //запускаем первый поток
     v[0].size=5;  //число элементов массива
     v[0].ptr=&vector[0]; //адрес начального элемента
     v[0].nProcess=0;
     hThread[0] = (HANDLE) _beginthreadex(NULL, 0, ThreadSum,
(PVOID) &v[0],0, &dwThreadID);
     //запускаем второй поток
     v[1].size=6;  //число элементов оставшейся части
     v[1].ptr=&vector[5]; //адрес начального элемента
     v[0].nProcess=1;
     hThread[1] = (HANDLE) _beginthreadex(NULL, 0, ThreadSum,
(PVOID) &v[1],0, &dwThreadID);
     //ждем завершения всех потоков (сумма подсчитана)
    WaitForMultipleObjects(2,hThread,TRUE, INFINITE);

    wsprintf(str,"sum=%ld",sum);
     MessageBox(NULL,str,"Итого",MB_OK);
     for(int i=0; i<2; i++) CloseHandle(hThread[i]);

   return(0);
}
```

2.2. Синхронизация потоков для задачи: один писатель, много читателей.

```
include <Windows.h>
#include <iostream.h>
#include <string.h>
```

```
#include <conio.h>
#define NREADER 5
HANDLE hThreadReader[NREADER]; //читатели
HANDLE hThreadWriter;          //писатель
HANDLE hAllRead;               //читать всем
HANDLE hReading[NREADER];      //читаем
HANDLE hReaded[NREADER];       //уже прочитали
volatile BOOL fShutDown;       //все закрыть

typedef struct {
     HANDLE hReading;
     HANDLE hReaded;
    int nReader;
} READERRARAMS;

READERRARAMS p[NREADER];

DWORD WINAPI WriteProc(LPVOID iValue)
{
     while(!fShutDown){
     if(WaitForMultipleObjects(NREADER,hReaded,TRUE,INFINITE)==
WAIT_OBJECT_0)
               {
              //писать
               MessageBox(NULL,"Writer","WRITE",MB_OK);
      for(int i=0; i<NREADER; i++) SetEvent(p[i].hReading);
      for(int i=0; i<NREADER; i++) ResetEvent(p[i].hReaded);
               }
     }
      return 0;
}

DWORD WINAPI ReadProc(LPVOID iValue)
{
     READERRARAMS *p=(READERRARAMS*)iValue;
     char snum[10];
     itoa(p->nReader,snum,10);
     while(!fShutDown){
if(WaitForSingleObject(p->hReading,INFINITE)==WAIT_OBJECT_0)
          {
               MessageBox(NULL,snum,"READ",MB_OK);
          //читать
          SetEvent(p->hReaded);
          }
     }
     return 0;
}

void InitProcess(){
     DWORD dwGenericThread;
     fShutDown=FALSE;
```

```
     for(int i=0; i<NREADER; i++)
//сброс вручную
hReaded[i]=p[i].hReaded=CreateEvent(NULL,TRUE,TRUE,NULL);
     for(int i=0; i<NREADER; i++)
          p[i].hReading=CreateEvent(NULL,FALSE,FALSE,NULL);
//автосброс
     hAllRead=CreateEvent(NULL,TRUE,TRUE,NULL);

hThreadWriter = CreateThread(NULL, 0, WriteProc, NULL, 0,
&dwGenericThread);
     for(int i=0; i<NREADER; i++){
          p[i].nReader=i+1;
          hThreadReader[i] = Cre-
ateThread(NULL,0,ReadProc,&p[i],0,&dwGenericThread);
     }
}

void DeleteProcess(){
 for(int i=0; i<NREADER; i++) CloseHandle(p[i].hReaded);
 for(int i=0; i<NREADER; i++) CloseHandle(p[i].hReading);
 for(int i=0; i<NREADER; i++) CloseHandle(hThreadReader[i]);
 CloseHandle(hAllRead);
 CloseHandle(hThreadWriter);
}
void main()
{
      InitProcess();
      MessageBox(NULL,"Close","END",MB_OK);
//while(!kbhit());
InterlockedExchangePointer((PVOID*)&fShutDown,(PVOID)TRUE);
      WaitForSingleObject(hThreadWriter,INFINITE);
       DeleteProcess();
       //getch();
     }
```

2.3. Задача о китайских философах

```
// Пять философов
// Philos.cpp :
#include "stdafx.h"
#include <windows.h>
//#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include "TimedMsgBox.h"
//#include <conio.h>
#define NPHILO 5
HANDLE hThreadPhilo[NPHILO]; //философы
HANDLE hStick[NPHILO];       //палочки
volatile BOOL fShutDown;     //переменная для закрытия циклов

typedef struct {
     HANDLE hStick[2];
```

```
     int nPhilo;
     int rand;
} PHILOPARAMS;

PHILOPARAMS p[NPHILO];

DWORD WINAPI PhiloProc(LPVOID iValue)
{
     PHILOPARAMS *p=(PHILOPARAMS*)iValue;

     TCHAR text[80];
     //itoa(p->nPhilo,snum,10);
     while(!fShutDown){
     if(WaitForMultipleObjects(2,p-> hStick, TRUE, 100)==
WAIT_OBJECT_0)
          {
               wsprintf(text,"Философ %d ест",p->nPhilo);
               TimeMsgBox(200,p->nPhilo*100+5,p->rand,text);
               //читать
               ReleaseMutex(p->hStick[0]);
               ReleaseMutex(p->hStick[1]);
               Sleep(2000);
          }
          else{
          wsprintf(text,"Философ %d говорит",p->nPhilo);
            TimeMsgBox(10,p->nPhilo*100+5,5,text);
          }
     }
     return 0;
}

void InitProcess(){
     DWORD dwGenericThread;
     srand(10);
     fShutDown=FALSE;
     for(int i=0; i<NPHILO; i++)
hStick[i]=CreateMutex(NULL,FALSE,NULL);
     for(int i=0; i<NPHILO; i++){
          if(i==0) {
               p[i].hStick[0]=hStick[NPHILO-1];
            p[i].hStick[1]=hStick[0];
          }
          else {
          p[i].hStick[0]=hStick[i-1];
          p[i].hStick[1]=hStick[i];
          }
          p[i].nPhilo=i;
          p[i].rand=rand()%20;
          hThreadPhilo[i] = Cre-
ateThread(NULL,0,PhiloProc,&p[i].hStick,0,&dwGenericThread);
     }
```

```
}

void DeleteProcess(){
     for(int i=0; i<NPHILO; i++){
          CloseHandle(hThreadPhilo[i]);
          CloseHandle(hStick[i]);
     }
}

void main()
{
      InitProcess();
       MessageBox(NULL,"Close","END",MB_OK);
       //while(!kbhit());
       InterlockedExchangePoint-
er((PVOID*)&fShutDown,(PVOID)TRUE);
      WaitForMultipleOb-
jects(NPHILO,hThreadPhilo,TRUE,INFINITE);
       DeleteProcess();
       //getch();
}
```

Функция TimeMsgBox предназначена для вывода ящика сообщений, который будет ждать заданное в TimeOut время.

```
//файл TimedMsgBox.h
int TimeMsgBox(int x, int y,int TimeOut,TCHAR *Caption);

typedef struct {
     TCHAR *Caption; //заголовок
     int SecLeft;    //всего прошло секунд
     int x;     //координаты
     int y;
     }  TIMEMSGBOXSTRUCT;

//файл TimedMsgBox.cpp

#include  "stdafx.h"
#define _WIN32_WINNT 0x500
#include <windows.h>
#include <tchar.h>
#include "TimedMsgBox.h"

#define ID_MSGBOX_STATIC_TEXT    0x0000ffff

VOID CALLBACK MsgBoxTimeout(PVOID pvContext, BOOLEAN fTimeout)
{

   TIMEMSGBOXSTRUCT *tmbxs=(TIMEMSGBOXSTRUCT*)pvContext;

//поиск окна в windows с заголовком tmbxs->Caption
```

```
   HWND hwnd = FindWindow(NULL, tmbxs->Caption);
   if(hwnd != NULL) { //если окно найдено
      TCHAR sz[100];
       //изменяем текст сообщения ящика сообщений
     wsprintf(sz, TEXT("You have %d seconds to respond"),
tmbxs->SecLeft--);
      SetDlgItemText(hwnd, ID_MSGBOX_STATIC_TEXT, sz);
   MoveWindow(hwnd, tmbxs->x, tmbxs->y,200,100,TRUE);

if(tmbxs->SecLeft == 0) { //если время истекло, закрываем ящик

         EndDialog(hwnd, IDOK);
      }
   } else {
    ;
   }
}
////////////////////////////////////////////////////////////
int TimeMsgBox(int x, int y, int TimeOut,TCHAR *Caption){

     TIMEMSGBOXSTRUCT tmbxs;
//заполняем структуру tmbxs.
    tmbxs.Caption=Caption;
    tmbxs.SecLeft=TimeOut;
    tmbxs.x=x;
     tmbxs.y=y;

   /* Создаем объект «ожидаемый таймер», с функцией обработки
через секунду MsgBoxTimeout*/
   HANDLE hTimerQTimer;
CreateTimerQueueTimer((PHANDLE)&hTimerQTimer, NULL, MsgBox-
Timeout, (PVOID)&tmbxs,1000, 1000, 0);

   // Отображаем время в ящик сообщений
   TCHAR sz[100];
   wsprintf(sz, TEXT("You have %d seconds to respond"),
tmbxs.SecLeft);
   MessageBox(NULL,sz,Caption, MB_OK);

   // Удаляем очередь
DeleteTimerQueueTimer((HANDLE)NULL, hTimerQTimer,(HANDLE)
NULL);
   return(0);   }
```

Задание

1. Создать и отладить распределенное приложение из двух процессов, которые читают и пишут в общую память.

2. Написать простую программу синхронизации потоков в пользовательском режиме.

3. Составить программу синхронизации потоков с использованием объектов ядра в схеме: один писатель — много читателей.

Вопросы для самоконтроля

1. Для чего нужна синхронизация потоков?
2. Укажите достоинства и недостатки синхронизации потоков в пользовательском режиме?
3. Для чего нужны функции WaitSingleObject, WaitMultipleObject?
4. Укажите особенности использования объектов «событие» для организации синхронизации потоков.
5. Проведите сравнительный анализ критических секций и мьютексов.
6. Алгоритм работы семафоров для учета ресурсов.
7. Что нужно изменить в первой программе, чтобы она работала для массива vector типа double?
8. Основные элементы сценария «один писатель — много читателей».
9. Что нужно изменить в программе 2.3 для получения статистики ожидания потоков?

2.3 Практическое занятие «Пул потоков для управления очередью»

Цель

Разработка пула потоков для управления очередью.

Основные понятия

Очередь, атомарный подступ, управление ресурсами, серверный поток, клиентский поток.

Методические указания

Рассмотрим подробнее реализацию данного примера.

```
#include <windows.h>
#include <windowsx.h>
#include <tchar.h>
#include <process.h>
#include "Resource.h"
//Описание очереди.
class CQueue {
public:
//структура элемента очереди
   struct ELEMENT {
      int m_nThreadNum,  //номер потока
      m_nRequestNum;      //номер запроса
      // другие данные
   };
   typedef ELEMENT* PELEMENT;
```

```
private:
   PELEMENT m_pElements;     //указатель на очередь
   int      m_nMaxElements;  //максимальный размер очереди
   HANDLE   m_h[2];          // дескриптор мьютекса и семафора
   HANDLE   &m_hmtxQ;        // ссылка на мьютекс
   HANDLE   &m_hsemNumElements; // ссылка на семафор m_h[1]

public:
   CQueue(int nMaxElements); //конструктор
   ~CQueue(); //деструктор
/*добавить элемент в очередь, время ожидания записано в
dwMilliseconds*/
   BOOL Append(PELEMENT pElement, DWORD dwMilliseconds);
/*удалить элемент из очереди время ожидания записано в dwMil-
liseconds*/
   BOOL Remove(PELEMENT pElement, DWORD dwMilliseconds);
};

//конструктор
CQueue::CQueue(int nMaxElements) //устананавливаем ссылки
   : m_hmtxQ(m_h[0]), m_hsemNumElements(m_h[1]) {
//распределяем память под очередь
   m_pElements = (PELEMENT) HeapAlloc(GetProcessHeap(), 0,
sizeof(ELEMENT) * nMaxElements);
//устанавливаем максимальный размер очереди
   m_nMaxElements = nMaxElements;
//создаем мьютекс и семафор
   m_hmtxQ = CreateMutex(NULL, FALSE, NULL);
   m_hsemNumElements = CreateSemaphore(NULL, 0, nMaxElements,
NULL);
}

//деструктор
CQueue::~CQueue() {
//закрываем объекты
   CloseHandle(m_hsemNumElements);
   CloseHandle(m_hmtxQ);
//освобождаем память
   HeapFree(GetProcessHeap(), 0, m_pElements);
}

// добавить элемент в очередь
BOOL CQueue::Append(PELEMENT pElement, DWORD dwTimeout) {
BOOL fOk = FALSE;
//ждем пока очередь занята (ждем освобождения мьютекса)
   DWORD dw = WaitForSingleObject(m_hmtxQ, dwTimeout);
   if (dw == WAIT_OBJECT_0) {
      // ожидание завершилось успешно, мьютекс захватили
      // увеличиваем число запросов в очереди
      LONG lPrevCount;
    fOk = ReleaseSemaphore(m_hsemNumElements, 1, &lPrevCount);
```

```
      if (fOk) { //увеличение состоялось
         //очередь не переполнена, заносим элемент в очередь
         m_pElements[lPrevCount] = *pElement;
      } else {

         // Ошибка: очередь переполнена
         SetLastError(ERROR_DATABASE_FULL);
      }

      // Освобождаем мьютекс
      ReleaseMutex(m_hmtxQ);

   } else {
      // Не дождались освобождения или другая ошибка
      SetLastError(ERROR_TIMEOUT);
   }

   return(fOk);
}

//Удаление элемента из очереди

BOOL CQueue::Remove(PELEMENT pElement, DWORD dwTimeout) {

 //ждем доступа к очереди (освобождения мьютекса)
// и наличия свободных мест в очереди
   BOOL fOk = (WaitForMultipleObjects(2,m_h,TRUE,dwTimeout)
      == WAIT_OBJECT_0);
   if (fOk) {
   //все в норме, в очереди есть элементы и мьютекс захвачен
      *pElement = m_pElements[0];
      //удаляем сдвигом влево
      MoveMemory(&m_pElements[0], &m_pElements[1],
         sizeof(ELEMENT) * (m_nMaxElements - 1));

      //освобождаем мьютекс
      ReleaseMutex(m_hmtxQ);

   } else {
      //не дождались освобождения
      SetLastError(ERROR_TIMEOUT);
   }

   return(fOk);
}

//Главная программа
// объявляем очередь
CQueue g_q(10);

// закрываем работу программы
```

```
volatile BOOL g_fShutdown = FALSE;  HWND g_hwnd;
// окно для вывода

// дескрипторы всех потоков
HANDLE g_hThreads[MAXIMUM_WAIT_OBJECTS];
int    g_nNumThreads = 0;

//Функция клиентского потока

unsigned __stdcall ClientThread(PVOID pvParam) {
//преобразуем pvParam в целое число (номер потока)
   int nThreadNum = PtrToUlong(pvParam);
//запрашиваем дескриптор окна клиента
   HWND hwndLB = GetDlgItem(g_hwnd, IDC_CLIENTS);
//организуем цикл для выдачи запросов
   for (int nRequestNum = 1; !g_fShutdown; nRequestNum++) {
      TCHAR sz[1024];
//организуем элемент очереди
      CQueue::ELEMENT e = { nThreadNum, nRequestNum };

  // пытаемся добавить в очередь
      if (g_q.Append(&e, 200)) {

//записать строку sz
 wsprintf(sz, TEXT("Sending %d:%d"), nThreadNum, nRequestNum);
      } else {

         // не помещено в очередь
         wsprintf(sz, TEXT("Sending %d:%d (%s)"), nThreadNum,
nRequestNum,
            (GetLastError() == ERROR_TIMEOUT)
               ? TEXT("timeout") : TEXT("full"));
      }

      // показать результат в ListBox
     ListBox_SetCurSel(hwndLB, ListBox_AddString(hwndLB, sz));
      Sleep(2500);   // ждем 2,5 секунды
   }

   return(0);
}

//Функция серверного потока
unsigned __stdcall ServerThread(PVOID pvParam) {
//получить номер серверного потока
   int nThreadNum = PtrToUlong(pvParam);
//получить дескриптор серверного окна
   HWND hwndLB = GetDlgItem(g_hwnd, IDC_SERVERS);

   while (!g_fShutdown) {
```

```
      TCHAR sz[1024];
      CQueue::ELEMENT e;

      //пытаемся удалить элемент из очереди
      if (g_q.Remove(&e, 5000)) {

         //если успешно удалили
         wsprintf(sz, TEXT("%d: Processing %d:%d"),
            nThreadNum, e.m_nThreadNum, e.m_nRequestNum);

         // серверный поток засыпает
         Sleep(2000 * e.m_nThreadNum);

      } else {
         //если запрос не обработан, выдаем TIMEOUT
         wsprintf(sz, TEXT("%d: (timeout)"), nThreadNum);
      }
      // выдаем сообщение в окно
      ListBox_SetCurSel(hwndLB, ListBox_AddString(hwndLB,
sz));
   }
   return(0);
}

//функция обработки события InitDialog
BOOL Dlg_OnInitDialog(HWND hwnd) {
  g_hwnd = hwnd; // используется потоками для вывода сообщений
   unsigned  dwThreadID;

   //создаем четыре клиентских потока
   for (int x = 0; x < 4; x++)
      g_hThreads[g_nNumThreads++] =
(HANDLE) _beginthreadex(NULL, 0, ClientThread, (PVOID)
(INT_PTR) x,0, &dwThreadID);

   // создаем два серверных потока
   for (int x = 0; x < 2; x++)
      g_hThreads[g_nNumThreads++] =
           (HANDLE)_beginthreadex(NULL, 0, ServerThread,
(PVOID) (INT_PTR) x,           0, &dwThreadID);

   return(TRUE);
}
//функция обработки WM_COMMAND

void Dlg_OnCommand(HWND hwnd, int id) {
   switch (id) {
      case IDCANCEL:
         EndDialog(hwnd, id);
         break;
   }
```

```
}

//оконная процежура диалога
INT_PTR WINAPI Dlg_Proc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam) {

   switch (uMsg) {
    case WM_INITDIALOG: return Dlg_OnInitDialog(hwnd);
      case WM_COMMAND:  Dlg_OnCommand(hwnd,LOWORD(wParam));
break;

   }
   return(FALSE);
}

//главная функция
int WINAPI _tWinMain(HINSTANCE hinstExe, HINSTANCE, PTSTR
pszCmdLine, int) {
//запускаем диалоговый ящик
   DialogBox(hinstExe, MAKEINTRESOURCE(IDD_QUEUE), NULL,
Dlg_Proc);
   InterlockedExchangePointer((PVOID*) &g_fShutdown, (PVOID)
TRUE);

   // ждем окончания всех потоков
   WaitForMultipleObjects(g_nNumThreads, g_hThreads, TRUE, IN-
FINITE);
//закрываем все дескрипторы
   while (g_nNumThreads--)
      CloseHandle(g_hThreads[g_nNumThreads]);

   return(0);
}
```

Задание

Организовать очередь, в которую *m* клиентских потоков заносят запрос, а *n* серверных потоков выбирают из нее запрос и обрабатывают его.

В качестве примера решения подобной задачи взята программа, написанная Джефри Рихтором[2].

В качестве очереди используется фиксированный одномерный масив. Синхронизация записи запроса в очередь и удаление обработанного запроса из очереди осуществляется с помощью мьютекса. Управление элементами очереди осуществляется с помощью семафора.

Вопросы для самоконтроля

1. Укажите назначение класса CQueue.
2. Каким образом присваиваются значения для ссылок m_hmtxQ и m_hsemNumElements?
3. Какое максимальное число элементов может быть в очереди?

4. В каком случае серверный поток будет находится в режиме простоя?

5. В каком случае клиентский поток будет находится в режиме ожидания?

6. Назначение вызовов функций Sleep в программе.

2.4 Практическое занятие «Клиент-серверные приложения»

Цель

Развитие, умений и навыков по разработке несложного сервера, основанного HTTP-протоколе. Использование стандартных сценариев клиент-серверных приложений. Построение простейшего WWW-сервера.

Методические указания

Для изучения данной программы необходимы знания языка C++ и STL.

Ниже приведен текст программы WWW-сервера. Описание классов:

```
class webserver { //основной класс
  public:
    struct http_request { //описание структуры запроса клиента

      http_request() : authentication_given_(false)
{//конструктор}

      Socket* s_;          //сокет клиента
      std::string method_;//метод “GET” или “PUT”
      std::string path_;     //путь
     //таблица пар <параметр><значение>
      std::map<std::string, std::string> params_;
      std::string accept_;
      std::string accept_language_;
      std::string accept_encoding_;
      std::string user_agent_;

      /* status_: результат обработки
         o  202 OK
         o  404 не найдено и т.д. */
      std::string status_;

      std::string auth_realm_;
    //ответ
      std::string answer_;

      /*  флаг authentication_given_ истина – когда клиент ввел
имя и пароль  */
      bool authentication_given_;
      std::string username_; //имя пользователя
      std::string password_;//пароль пользователя
```

```
    };
   //тип requet_func
    typedef void (*request_func) (http_request*);
//конструктор
    webserver(unsigned int port_to_listen, request_func);

  private:
//потоковая функция обработки запроса, статическая функция
    static unsigned __stdcall Request(void*);
//статический указатель на функцию генерации ответа
    static request_func request_func_;
};
```

Функция Get генерации html-страницы при обработке запроса клиента. На входе структура, описывающая конкретный запрос клиента.

```
void Get(webserver::http_request* r) {
//  Socket s = *(r->s_);
//устанавливаем значения строк
  std::string title;
  std::string body;
  std::string bgcolor="#ffffff";
  std::string links =
      "<p><a href='/red'>red</a> "
      "<br><a href='/blue'>blue</a> "
      "<br><a href='/form'>form</a> "
      "<br><a href='/auth'>authentication example</a> [use
<b>adp</b> as username and <b>gmbh</b> as password"
   "<br><a href='/header'>show some HTTP header details</a> ";
//анализируем строку path
  if(r->path_ == "/") {
    title = "Web Server Example";
    body  = "<h1>Простейший Web Server</h1>"
            "Взято у Rene's "  + links;
  }
  else if (r->path_ == "/red") {
    bgcolor = "#ff4444";
    title   = "You chose red";
    body    = "<h1>Red</h1>" + links;
  }
  else if (r->path_ == "/blue") {
    bgcolor = "#4444ff";
    title   = "You chose blue";
    body    = "<h1>Blue</h1>" + links;
  }
  else if (r->path_ == "/form") {
    title   = "Fill a form";

    body    = "<h1>Fill a form</h1>";
```

```
    body   += "<form action='/form'>"
              "<table>"
     "<tr><td>Field 1</td><td><input name=field_1></td></tr>"
     "<tr><td>Field 2</td><td><input name=field_2></td></tr>"
     "<tr><td>Field 3</td><td><input name=field_3></td></tr>"
              "</table>"
              "<input type=submit></form>";

//цикл для печати параметров
    for (std::map<std::string, std::string>::const_iterator
 i = r->params_.begin();    i != r->params_.end();   i++) {

      body += "<br>" + i->first + " = " + i->second;
    }
    body += "<hr>" + links;
  }
  else if (r->path_ == "/auth") {
    if (r->authentication_given_) {
      if (r->username_ == "adp" && r->password_ == "gmbh") {
         body = "<h1>Успешная аутентификация</h1>" + links;
      }
      else {
         body = "<h1>Неверный пароль, или  имя</h1>" + links;
         r->auth_realm_ = "Private Stuff";
      }
    }
    else {
      r->auth_realm_ = "Private Stuff";
    }
  }
  else if (r->path_ == "/header") {
    title   = "some HTTP header details";
    body    = std::string ("<table>")
+  "<tr><td>Accept:</td><td>" + r->accept_ + "</td></tr>" +
   "<tr><td>Accept-Encoding:</td><td>" + r->accept_encoding_ +
"</td></tr>" + "<tr><td>Accept-Language:</td><td>" + r-
>accept_language_ + "</td></tr>" +
    "<tr><td>User-Agent:</td><td>" + r->user_agent_ +
    "</td></tr>" +
              "</table>"
+
              links;
  }
  else {
    r->status_ = "404 Not Found";
    title      = "Wrong URL";
    body       = "<h1>Wrong URL</h1>";
    body      += "Path is : &gt;" + r->path_ + "&lt;";
  }
//формирование ответа
```

```
  r->answer_  = "<html><head><title>";
  r->answer_ += title;
  r->answer_ += "</title></head><body bgcolor='" + bgcolor +
"'>";
  r->answer_ += body;
  r->answer_ += "</body></html>";
}

//главная программа
int WINAPI WinMain(HINSTANCE hInstance,
HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
 {
  webserver(8080, Get);
}

//WebServer.h

#include <string>
#include <map>

class Socket;

//WebServer.cpp
#include <ctime>
#include <process.h>
#include <iostream>
#include <string>
#include <map>
#include <sstream>
#include "webserver.h"
#include "socket.h"
#include "UrlHelper.h"
#include "base64.h"

webserver::request_func webserver::request_func_=0;

//потоковая функция обработки запроса клиента

unsigned webserver::Request(void* ptr_s) {
//получаем и преобразуем указатель на Soket
  Socket s = *(reinterpret_cast<Socket*>(ptr_s));
//получаем строку от клиента
  std::string line = s.ReceiveLine();
  if (line.empty()) { //если строка пуста
    return 1;
  }
//создаем структуру http_request
  http_request req;
//определяем метод запроса
  if (line.find("GET") == 0) { //если это GET
```

```
    req.method_="GET";
  }
  else if (line.find("POST") == 0) { //если это POST
    req.method_="POST";
  }

  std::string path;
  std::map<std::string, std::string> params;

  size_t posStartPath = line.find_first_not_of(" ",3);
//разбор HTTP запроса
  SplitGetReq(line.substr(posStartPath), path, params);

//установка значений полей структуры запроса клиента
  req.status_ = "202 OK";
  req.s_      = &s;
  req.path_   = path;
  req.params_ = params;

  static const std::string authorization   = "Authorization:
Basic ";
  static const std::string accept          = "Accept: "
;
  static const std::string accept_language = "Accept-Language:
"    ;
  static const std::string accept_encoding = "Accept-Encoding:
"    ;
  static const std::string user_agent      = "User-Agent: "
;

  while(1) {
    line=s.ReceiveLine();

    if (line.empty()) break;

    unsigned int pos_cr_lf = line.find_first_of("\x0a\x0d");
    if (pos_cr_lf == 0) break;

    line = line.substr(0,pos_cr_lf);

    if (line.substr(0, authorization.size()) == authorization)
{
      req.authentication_given_ = true;
      std::string encoded = line.substr(authorization.size());
      std::string decoded = base64_decode(encoded);
      unsigned int pos_colon = decoded.find(":");
      req.username_ = decoded.substr(0, pos_colon);
      req.password_ = decoded.substr(pos_colon+1 );
    }
    else if (line.substr(0, accept.size()) == accept) {
```

```
      req.accept_ = line.substr(accept.size());
    }
    else if (line.substr(0, accept_language.size()) ==
accept_language) {
      req.accept_language_ =
line.substr(accept_language.size());
    }
    else if (line.substr(0, accept_encoding.size()) ==
accept_encoding) {
      req.accept_encoding_ =
line.substr(accept_encoding.size());
    }
    else if (line.substr(0, user_agent.size()) == user_agent)
{
      req.user_agent_ = line.substr(user_agent.size());
    }
  }

  request_func_(&req);

  std::stringstream str_str;
  str_str << req.answer_.size();

  time_t ltime;
  time(&ltime);
  tm* gmt= gmtime(&ltime);

static std::string const serverName =
     "RenesWebserver (Windows)";

  char* asctime_remove_nl = std::asctime(gmt);
  asctime_remove_nl[24] = 0;

  s.SendBytes("HTTP/1.1 ");

  if (! req.auth_realm_.empty() ) {
    s.SendLine("401 Unauthorized");
    s.SendBytes("WWW-Authenticate: Basic Realm=\"");
    s.SendBytes(req.auth_realm_);
    s.SendLine("\"");
  }
  else {
    s.SendLine(req.status_);
  }
  s.SendLine(std::string("Date: ") + asctime_remove_nl + "
GMT");
  s.SendLine(std::string("Server: ") +serverName);
  s.SendLine("Connection: close");
  s.SendLine("Content-Type: text/html; charset=ISO-8859-1");
  s.SendLine("Content-Length: " + str_str.str());
```

```
  s.SendLine("");
  s.SendLine(req.answer_);

  s.Close();


  return 0;
}
//конструктор класса webserver
webserver::webserver(

//порт сервера для прослушивания запросов клиентов
unsigned int port_to_listen,
  request_func r //функция обработки запроса
) {
//организуем серверный сокет
  SocketServer in(port_to_listen,5);
/*устанавливает значение статического указателя функции гене-
рации ответа*/
  request_func_ = r;
//организуем цикл по обработке запросов клиентов
  while (1) {
    Socket* ptr_s=in.Accept(); //принять вызов от клиента
    unsigned ret;
//запустить поток для обработки запроса клиента
    HANDLE hHandle = reinter-
pret_cast<HANDLE>(_beginthreadex(0,0,Request,(void*)
ptr_s,0,&ret));
//закрыть указатель потока
    CloseHandle(hHandle);
  }
}
```

Классы, предназначенные для описания различных типов сокетов. Структура описывает тип сокета (TCP или UDP):

```
enum TypeSocket {BlockingSocket, NonBlockingSocket};
class Socket {
public:

  virtual ~Socket(); //деструктор
  Socket(const Socket&); //конструктор копирования
  Socket& operator=(Socket&); //операция присвоения

  std::string ReceiveLine(); //функция приема строки
  std::string ReceiveBytes();//функция приема байт
//закрыть сокет
  void   Close();
//передать строку клиенту
  void   SendLine (std::string);
//передать байты клиенту
```

```
  void   SendBytes(const std::string&);
protected:
//класс сокета для сервера
  friend class SocketServer;
//класс сокета для выбора
  friend class SocketSelect;

  Socket(SOCKET s);
  Socket();

  SOCKET s_;

  int* refCounter_;

private:
  static void Start(); //подключение библиотеку сокетов
winsock.dll
  static void End(); //удаление библиотеки сокетов winsock.dll
  static int  nofSockets_;
};
//класс сокетов для клиента
class SocketClient : public Socket {
public:
//клиентский конструктор
  SocketClient(const std::string& host, int port);
};

//класс сокетов для сервера
class SocketServer : public Socket {
public:
//сервекрный конструктор
  SocketServer(int port, int connections, TypeSocket
type=BlockingSocket);

  Socket* Accept();
};
//класс сокетов для выбора
class SocketSelect {
  public:
    SocketSelect(Socket const * const s1, Socket const * const
s2=NULL, TypeSocket type=BlockingSocket);

    bool Readable(Socket const * const s);

  private:
    fd_set fds_;
};
```

Функции классов для организации и манипулирования сокетами:

```
int Socket::nofSockets_= 0;

void Socket::Start() { //грузим библиотеку winsock.dll
  if (!nofSockets_) {
    WSADATA info;
    if (WSAStartup(MAKEWORD(2,0), &info)) {
      throw "Could not start WSA";
    }
  }
  ++nofSockets_;
}

void Socket::End() {
  WSACleanup();
}

//конструктор
Socket::Socket() : s_(0) {
  Start(); //загрузка winsock.dll
//создаем сокет для работы с протоколом TCP
  s_ = socket(AF_INET,SOCK_STREAM,0);
  if (s_ == INVALID_SOCKET) {
    throw "INVALID_SOCKET";
  }
  refCounter_ = new int(1);
}

//конструктор, при условии, что сокет уже есть
Socket::Socket(SOCKET s) : s_(s) {
  Start();
  refCounter_ = new int(1);
};
//деструктор
Socket::~Socket() {
  if (! --(*refCounter_)) {
    Close();
    delete refCounter_;
  }
  --nofSockets_; //если число созданных сокетов равно нулю,
                 //закрываем библиотеку
  if (!nofSockets_) End();
}

//конструктор копирования
Socket::Socket(const Socket& o) {
  refCounter_=o.refCounter_;
  (*refCounter_)++;
  s_         =o.s_;
  nofSockets_++;
}
```

```
//операция присвоения
Socket& Socket::operator=(Socket& o) {
  (*o.refCounter_)++;
  refCounter_=o.refCounter_;
  s_         =o.s_;
  nofSockets_++;
  return *this;
}
//закрыть сокет
void Socket::Close() {
  closesocket(s_);
}
//принять байты
std::string Socket::ReceiveBytes() {
  std::string ret;
  char buf[1024];

  while (1) {
    u_long arg = 0;
//определяем число байт, которые надо прочитать
    if (ioctlsocket(s_, FIONREAD, &arg) != 0)
      break; //если ошибка

    if (arg == 0) //если число равно нулю
      break;
//должно быть не больше размера буфера
    if (arg > 1024) arg = 1024;
//читаем байты от клиентского сокета
    int rv = recv (s_, buf, arg, 0);
    if (rv <= 0) break;
    //копируем в строку t
    std::string t;

    t.assign (buf, rv);
//добавляем в строку res
    ret += t;
  }
   return ret;
}

//принять строку от клиента
std::string Socket::ReceiveLine() {
  std::string ret;
  while (1) {
    char r;
    //читаем побайтно
    switch(recv(s_, &r, 1, 0)) {
      case 0:// приняли 0 байт

        return ret;
```

```
      case -1: //ошибка
        return "";
     }
    //добавляем байт в строку
     ret += r;
    //если байт содержит возврат каретки, то выход
     if (r == '\n')  return ret;
   }
}
//послать строку клиенту
void Socket::SendLine(std::string s) {
  s += '\n';
  send(s_,s.c_str(),s.length(),0);
}
//послать байты клиенту
void Socket::SendBytes(const std::string& s) {
  send(s_,s.c_str(),s.length(),0);
}

//конструктор для серверного сокета
SocketServer::SocketServer(int port, int connections,
TypeSocket type) {
  sockaddr_in sa;

  memset(&sa, 0, sizeof(sa));

  sa.sin_family = PF_INET;
  sa.sin_port = htons(port);
  s_ = socket(AF_INET, SOCK_STREAM, 0);
  if (s_ == INVALID_SOCKET) {
    throw "INVALID_SOCKET";
  }

  if(type==NonBlockingSocket) {
    u_long arg = 1;
    ioctlsocket(s_, FIONBIO, &arg);
  }

  /* связывает сокет с адресом  сервера */
  if(bind(s_, (sockaddr *)&sa, sizeof(sockaddr_in)) == SOCK-
ET_ERROR) {
    closesocket(s_);
    throw "INVALID_SOCKET";
  }
  // определяем очередь ожидания
  listen(s_, connections);
}

//выполняем прием запроса от клиента
Socket* SocketServer::Accept() {
```

```
  SOCKET new_sock = accept(s_, 0, 0);
  if (new_sock == INVALID_SOCKET) {
    int rc = WSAGetLastError();
    if(rc==WSAEWOULDBLOCK) {
      return 0;
    }
    else {
      throw "Invalid Socket";
    }
  }
  /*прием запроса успешный, создаем объект Socket для обработ-
ки клиентского запроса*/
  Socket* r = new Socket(new_sock);
  return r;
}

//создание сокета на стороне клиента
SocketClient::SocketClient(const std::string& host, int port)
: Socket() {
  std::string error;

  hostent *he;
  if ((he = gethostbyname(host.c_str())) == 0) {
    error = strerror(errno);
    throw error;
  }

  sockaddr_in addr;
  addr.sin_family = AF_INET;
  addr.sin_port = htons(port);
  addr.sin_addr = *((in_addr *)he->h_addr);
  memset(&(addr.sin_zero), 0, 8);

  if (::connect(s_, (sockaddr *) &addr, sizeof(sockaddr))) {
    error = strerror(WSAGetLastError());
    throw error;
  }
}
//обработка неблокированных запросов
SocketSelect::SocketSelect(Socket const * const s1, Socket
const * const s2, TypeSocket type) {
  FD_ZERO(&fds_);
  FD_SET(const_cast<Socket*>(s1)->s_,&fds_);
  if(s2) {
    FD_SET(const_cast<Socket*>(s2)->s_,&fds_);
  }

  TIMEVAL tval;
  tval.tv_sec  = 0;
  tval.tv_usec = 1;
```

```
  TIMEVAL *ptval;
  if(type==NonBlockingSocket) {
    ptval = &tval;
  }
  else {
    ptval = 0;
  }

  if (select (0, &fds_, (fd_set*) 0, (fd_set*) 0, ptval) ==
SOCKET_ERROR)
    throw "Error in select";
}

//определение наличия данных
bool SocketSelect::Readable(Socket const* const s) {
  if (FD_ISSET(s->s_,&fds_)) return true;
  return false;
}
```

Вспомогательные функции для разбора URL:

```
 //UrlHelper.cpp
#include "UrlHelper.h"
#include "Tracer.h"
#include "stdHelpers.h"
#include <windows.h>
#include <sstream>
#include <iostream>

//Функция определения протокола
bool RemoveProtocolFromUrl(std::string const& url,
std::string& protocol, std::string& rest) {
  TraceFunc("RemoveProtocolFromUrl");
  Trace(std::string("url='")+url+"'");
  std::string::size_type pos_colon = url.find(":");
  if (pos_colon == std::string::npos) { //двоеточие не найдено
    rest = url;
    return false;
  }
//найдено, но нет названия протокола
  if (url.size() < pos_colon + 2) {
    rest = url;
    return false;
  }

  if (url[pos_colon+1] != '/' ||
      url[pos_colon+2] != '/')  { //ищем две косых черты
    rest = url;
    return false;
  }

  protocol = url.substr(0,pos_colon); //берем имя протокола
```

```
//пропускаем три символа( '://' )
  rest = url.substr(3+pos_colon);
  return true;
}

void SplitGetReq(std::string get_req, std::string& path,
std::map<std::string, std::string>& params) {
  TraceFunc("SplitGetReq");

  // Удаляем символы перевода строки и возврата каретки
  if (get_req[get_req.size()-1] == '\x0d' ||
      get_req[get_req.size()-1] == '\x0a')
      get_req=get_req.substr(0, get_req.size()-1);

  if (get_req[get_req.size()-1] == '\x0d' ||
      get_req[get_req.size()-1] == '\x0a')
      get_req=get_req.substr(0, get_req.size()-1);

  // удаляем символы HTTP/1.x
  if (get_req.size() > 7) {
    if (get_req.substr(get_req.size()-8, 7) == "HTTP/1.") {
      get_req=get_req.substr(0, get_req.size()-9);
    }
  }
//ищем вопрос
  std::string::size_type qm = get_req.find("?");
  if (qm != std::string::npos) { //символ вопроса найден
    //запоминаем строку параметров
    std::string url_params = get_req.substr(qm+1);
//выделяем путь из строки запроса
    path = get_req.substr(0, qm);

//Добавляем "&” в конец списка параметров
<parametr>=<value>&…&<parametr>=<value>&

       url_params += "&";
//ищем первый разделитель пары
       std::string::size_type next_amp = url_params.find("&");
//цикл по разбору списка пар
//пока не конец строки выделяем пару
    while (next_amp != std::string::npos) {
      std::string name_value = url_params.substr(0,next_amp);
//перемещаем указатель на начало следущей пары
      url_params = url_params.substr(next_amp+1);
//Перемещаем указатель на конец следующей пары
      next_amp  = url_params.find("&");
//ищем символ «равно»
      std::string::size_type pos_equal = name_value.find("=");
//выделяем имя
      std::string nam = name_value.substr(0,pos_equal);
```

```
//выделяем значение
      std::string val = name_value.substr(pos_equal+1);
//убираем плюс
      std::string::size_type pos_plus;
      while ( (pos_plus = val.find("+")) != std::string::npos)
{
        val.replace(pos_plus, 1, " ");
      }

      // изменям нотацию %xy
      std::string::size_type pos_hex = 0;
      while ( (pos_hex = val.find("%", pos_hex)) !=
std::string::npos ) {
        std::stringstream h;
        h << val.substr(pos_hex+1, 2);
        h << std::hex;
        int i;
        h>>i;
        std::stringstream f;
        f << static_cast<char>(i);
        std::string s;
        f >> s;
        val.replace(pos_hex, 3, s);
        pos_hex ++;
      }

params.insert(std::map<std::string,std::string>::value_type(na
m, val));
    }
  }
  else {
    path = get_req;
  }
}

//разбор URL (протокол, сервер, путь)
void SplitUrl(std::string const& url, std::string& protocol,
std::string& server, std::string& path) {
  TraceFunc("SplitUrl");
  RemoveProtocolFromUrl(url, protocol, server);

  if (protocol == "http") {
    std::string::size_type pos_slash = server.find("/");

    if (pos_slash != std::string::npos) {
      Trace("slash found");
      path   = server.substr(pos_slash);
      server = server.substr(0, pos_slash);
    }
    else {
```

```
      Trace("slash not found");
      path = "/";
    }
  }
  else if (protocol == "file") {
    path = ReplaceInStr(server, "\\", "/");
    server = "";
  }
  else {
    std::cerr << "unknown protocol in SplitUrl: '" << protocol
<< "'" << std::endl;
  }
}
```

Трассировщик сервера:

```
/*   Tracer.h */

#ifndef TRACER_H__
#define TRACER_H__
#ifdef DO_TRACE

#include <string>
#include <sstream>
#include <windows.h>

#define StartTrace(x)    TraceFunc_::StartTrace_(x)
#define Trace(x) dummy_____for_trace_func______.Trace_(x)
#define Trace2(x,y) dummy_____for_trace_func______.Trace_(x,y)
#define TraceFunc(x) TraceFunc_ dum-
my_____for_trace_func______(x)
#define TraceFunc2(x,y) TraceFunc_ dum-
my_____for_trace_func______(x,y)

class TraceFunc_ {
  std::string func_name_;
  public:
      TraceFunc_(std::string const&);
 TraceFunc_(std::string const&, std::string const& something);
   ~TraceFunc_();

    static void StartTrace_(std::string const& filename);

   template <class T>
   void Trace_(T const& t) {
     DWORD d;
     std::string indent_s;
     std::stringstream s;

     s << t;

     for (int i=0; i< indent; i++) indent_s += " ";
```

```
::WriteFile(trace_file_,indent_s.c_str(),indent_s.size(),&d,0);
::WriteFile(trace_file_, s.str().c_str(), s.str().size() ,&d, 0);
     ::WriteFile(trace_file_,"\x0a",1,&d,0);
   }

   template <class T, class U>
   void Trace_(T const& t, U const& u) {
     std::string indent_s;
     std::stringstream s;

     s << t;
     s << u;
     Trace_(s.str());
   }

   static int indent;
   static HANDLE trace_file_;
};

#else

#define StartTrace(x)
#define Trace(x)
#define Trace2(x,y)
#define TraceFunc(x)
#define TraceFunc2(x,y)

#endif

#endif  // TRACER_H__

#include "base64.h"
#include <iostream>

static const std::string base64_chars =
             "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
             "abcdefghijklmnopqrstuvwxyz"
             "0123456789+/";

static inline bool is_base64(unsigned char c) {
  return (isalnum(c) || (c == '+') || (c == '/'));
}

std::string base64_encode(unsigned char const*
bytes_to_encode, unsigned int in_len) {
  std::string ret;
  int i = 0;
  int j = 0;
  unsigned char char_array_3[3];
  unsigned char char_array_4[4];
```

```
  while (in_len--) {
    char_array_3[i++] = *(bytes_to_encode++);
    if (i == 3) {
      char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
      char_array_4[1] = ((char_array_3[0] & 0x03) << 4) +
((char_array_3[1] & 0xf0) >> 4);
      char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) +
((char_array_3[2] & 0xc0) >> 6);
      char_array_4[3] = char_array_3[2] & 0x3f;

      for(i = 0; (i <4) ; i++)
        ret += base64_chars[char_array_4[i]];
      i = 0;
    }
  }

  if (i)
  {
    for(j = i; j < 3; j++)
      char_array_3[j] = '\0';

    char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
    char_array_4[1] = ((char_array_3[0] & 0x03) << 4) +
((char_array_3[1] & 0xf0) >> 4);
    char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) +
((char_array_3[2] & 0xc0) >> 6);
    char_array_4[3] = char_array_3[2] & 0x3f;

    for (j = 0; (j < i + 1); j++)
      ret += base64_chars[char_array_4[j]];

    while((i++ < 3))
      ret += '=';

  }
  return ret;
}

std::string base64_decode(std::string const& encoded_string) {
  int in_len = encoded_string.size();
  int i = 0;
  int j = 0;
  int in_ = 0;
  unsigned char char_array_4[4], char_array_3[3];
  std::string ret;

  while (in_len-- && ( encoded_string[in_] != '=') &&
is_base64(encoded_string[in_])) {
    char_array_4[i++] = encoded_string[in_]; in_++;
    if (i ==4) {
```

```
      for (i = 0; i <4; i++)
        char_array_4[i] = base64_chars.find(char_array_4[i]);

      char_array_3[0] = (char_array_4[0] << 2) +
((char_array_4[1] & 0x30) >> 4);
      char_array_3[1] = ((char_array_4[1] & 0xf) << 4) +
((char_array_4[2] & 0x3c) >> 2);
      char_array_3[2] = ((char_array_4[2] & 0x3) << 6) +
char_array_4[3];

      for (i = 0; (i < 3); i++)
        ret += char_array_3[i];
      i = 0;
    }
  }

  if (i) {
    for (j = i; j <4; j++)
      char_array_4[j] = 0;

    for (j = 0; j <4; j++)
      char_array_4[j] = base64_chars.find(char_array_4[j]);

    char_array_3[0] = (char_array_4[0] << 2) +
((char_array_4[1] & 0x30) >> 4);
    char_array_3[1] = ((char_array_4[1] & 0xf) << 4) +
((char_array_4[2] & 0x3c) >> 2);
    char_array_3[2] = ((char_array_4[2] & 0x3) << 6) +
char_array_4[3];

    for (j = 0; (j < i - 1); j++) ret += char_array_3[j];
  }

  return ret;
}
```

Задание

1. Изучить HTTP-протокол.
2. Познакомиться с HTML.
3. Написать программу WWW-сервера, который выдает статические HTML-страницы.

Вопросы для самоконтроля

1. Раскройте понятие «протокол», укажите основное назначение протоколов IP, TCP, UDP.
2. Дайте определение понятию «сокет».
3. Перечислите основные функции для работы с сокетами.
4. Укажите последовательность вызовов функций для организации сокета на стороне клиента.

5. Укажите последовательность вызовов функций для организации сокета на стороне сервера.

6. Опишите основную схему организации сервера, основанного на сокетах.

7. Опишите основную схему организации клиента, основанного на сокетах.

8. Раскройте протокол HTTP.

9. Опишите структуру WWW сервера.

2.5 Практическое занятие «CGI-приложения для www-сервера»

Цель

Изучение основ и приобретение практических навыков по разработке CGI приложений.

Методические указания

Общая схема CGI приложения показана на рисунке 1.

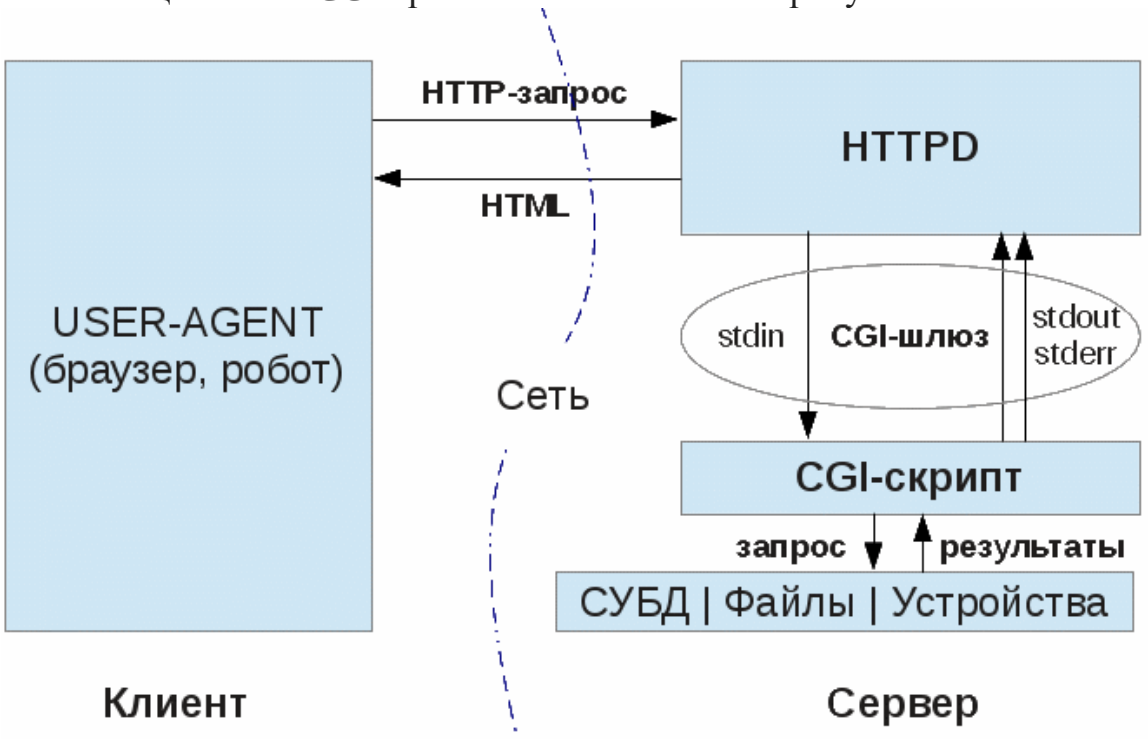

Рис. 1 – Общая схема CGI приложения

Задание

1. Выбрать и установить WWW-сервер, поддерживающий CGI-интерфейс.

2. Разработать структуру CGI-приложения.

3. Разработать интерфейсы для модулей CGI-приложения.

4. Разработать и отладить модули входящие в структуру CGI-приложения.

5. Провести комплексную отладку и разработать тестовый демонстративный пример.

Вопросы для самоконтроля

1. Что такое веб-сервер?
2. Перечислите преимущества CGI.
3. Перечислите недостатки CGI.
4. Чем CGI отличается от ISAPI?
5. Как работает протокол HTTP и для чего он нужен?

2.6 Практическое занятие «Сетевые приложения на языке Python»

Цель

Изучение основ и приобретение практических навыков по разработке клиент-серверных приложений на языке Python.

Методические указания

Сокеты

Для работы с сокетами в языке Python имеется пакет socket, который обеспечивает основные функции для работы с сокетами. Рассмотрим некоторые из них.

Константы

socket.AF_UNIX
socket.AF_INET
socket.AF_INET6
socket.SOCK_STREAM
socket.SOCK_DGRAM
socket.SOCK_RAW
socket.SOCK_RDM
socket.SOCK_SEQPACKET

Методы

- socket.socket(family=AF_INET, type=SOCK_STREAM, proto=0, fileno=None) – функция создания сокета;
- socket.create_connection(address[, timeout[, source_address]]) – создает соединение и взвращает (пару (host, port));
- socket.getaddrinfo(host, port, family=0, type=0, proto=0, flags=0) – преобразует информацию о порте и хосте в 5-элементный список: (family, type, proto, canonname, sockaddr).

Например

```
socket.getaddrinfo("example.org", 80,
proto=socket.IPPROTO_TCP) [(<AddressFamily.AF_INET6: 10>,
<SocketType.SOCK_STREAM: 1>, 6, '',
('2606:2800:220:1:248:1893:25c8:1946', 80, 0, 0)),
(<AddressFamily.AF_INET: 2>, <SocketType.SOCK_STREAM: 1>,
6, '', ('93.184.216.34', 80))]
```

- socket.gethostbyname(hostname) – возвращает имя хоста в формате адреса IPv4;

- socket.gethostname() – возвращает имя хоста на котором работает интерпретатор Питона;
- socket.gethostbyaddr(ip_address) – возвращает список содержащий следующие элементы (hostname, aliaslist, ipaddrlist);
- socket.getnameinfo(sockaddr, flags) преобразование структуры sockaddr в список (host, port);
- socket.accept() – принять соединение, возвращает пару (conn, address);
- socket.bind(address) – связывает сокет с хостом;
- socket.close() – закрывает сокет (аналогично файловой операции close);
- socket.connect(address) – принимает запрос на соединение с удаленным хостом;
- socket.getsockname() – возвращает собственный адрес сокета;
- socket.listen([backlog]) – ожидает соединения в режиме сервера;
- socket.recv(bufsize[, flags]) – принимает данные от сокета;
- socket.recvfrom(bufsize[, flags]) – принимает данные от сокета, возвращает пару (bytes, address);
- socket.send(bytes[, flags]) – посылает данные сокету;
- socket.sendto(bytes, address) – посылает данные сокету;
- socket.sendto(bytes, flags, address) посылает данные сокету;
- socket.sendfile(file, offset=0, count=None) – посылает файл сокету.

Создание сокетов

Рассмотрим следующий пример:

```
#распределяем память под структуры сокета и заполняем некото-
рые поля
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# производим запрос на соединение с серверомon по порту 80
s.connect(("www.python.org", 80))
```

Когда запрос на соединение выполнен можно посылать запросы на чтение и передачу данных. После выполнения операций чтения и записи данных необходимо закрыть сокет.

Схема сервера

В тех случаях, когда алгоритм web-сервера более сложен, тогда создается серверный сокет (сокет на стороне сервера). Например,

```
# создаем INET, STREAMing сокет
serversocket = socket.socket(socket.AF_INET, sock-
et.SOCK_STREAM)
# связываем сокет с хостом, по известному порту 80
serversocket.bind((socket.gethostname(), 80))
# ждем соединения с клиентом
serversocket.listen(5)
```

Необходимо отметить следующее:

1. Использование вызова метода socket.gethostname() позволяет сделать сокет видимым в сети. Если мы используем вызовы s.bind(('localhost', 80)) или s.bind(('127.0.0.1', 80)), то получим серверный сокет, видимый только на заданном компьютере.
2. Нижние номера портов зарегистрированы для “хорошо известных» сервисов (HTTP, SNMP и т.д.).
3. Вызов listen указывает библиотеки сокетов необходимость для сождание очереди запросов (как правило не более 5) до отказа на запрос соединения.

Основной цикл работы сервера по обработке запроса

```
while True:
    # принять запрос и  получить сокет клиента
    (clientsocket, address) = serversocket.accept()
    # выполнить предварительные действия для сокета клиента
    # запустить поток (процесс) обработки запроса клиента
    ct = client_thread(clientsocket)
    ct.run()
```

Задание

1. Выбрать схему клиент-серверного приложения.
2. Разработать структуру сервера.
3. Разработать структуру клиентского приложения.
4. Выбрать сервер баз данных.
5. Разработать и отладить модули сервера.
6. Разработать и отладить модули клиентского приложения.
7. Провести комплексную отладку и разработать тестовый демонстративный пример.

Вопросы для самоконтроля

1. Опишите функции и методы модуля socket
2. Опишите функции и методы модуля threading
3. Запишите общую структуру сервера, реализованного на сокетах.
4. Запишите организацию основного цикл сервера.
5. Запишите основные механизмы обмена данными.
6. Опишите способы организации мульти-поточности
7. Раскройте механизмы синхронизации потоков с помощью модуля threading
8. Раскройте отличия объектов объекта семофор от объекта события

2.7 Практическое занятие «Сетевые приложения на языке Java»

Цель

Получение практических навыков реализовывать сетевое приложение на языке Java.

Методические указания

Сервлеты – это компоненты приложений Java 2 Platform Enterprise Edition (J2EE), выполняющиеся на стороне сервера, способные обрабатывать клиентские запросы и динамически генерировать ответы на них. Сервлет может применяться, например, для создания серверного приложения, получающего от клиента запрос, анализирующего его и делающего выборку данных из базы данных, а также пересылающего клиенту страницу HTML, сгенерированную с помощью JSP на основе полученных данных.

Все сервлеты реализуют общий интерфейс Servlet. Для обработки HTTP-запросов можно воспользоваться в качестве базового класса абстрактным классом HttpServlet. Базовая часть классов JSDK помещена в пакет javax.servlet. Однако класс HttpServlet и все, что с ним связано, располагаются на один уровень ниже в пакете javax.servlet.http.

Жизненный цикл сервлета начинается с его загрузки в память контейнером сервлетов при старте либо в ответ на первый запрос. Далее происходят инициализация, обслуживание запросов и завершение существования.

Первым вызывается метод init(). Он дает сервлету возможность инициализировать данные и подготовиться для обработки запросов. Чаще всего в этом методе программисты помещают код, кэширующий данные фазы инициализации.

После этого сервлет можно считать запущенным, он находится в ожидании запросов от клиентов. Появившийся запрос обслуживается методом service() сервлета, а все параметры запроса упаковываются в объект ServletRequest, который передается в качестве первого параметра методу service(). Второй параметр метода – объект ServletResponse. В этот объект упаковываются выходные данные в процессе формирования ответа клиенту. Каждый новый запрос приводит к новому вызову метода service(). В соответствии со спецификацией JSDK, метод service() должен уметь обрабатывать сразу несколько запросов, т.е. быть синхронизирован для выполнения в многопоточных средах. Если же нужно избежать множественных запросов, сервлет должен реализовать интерфейс SingleThreadModel, который не содержит ни одного метода и только указывает серверу об однопоточной природе сервлета. При обращении к такому сервлету каждый новый запрос будет ожидать в очереди, пока не завершится обработка предыдущего запроса.

После завершения выполнения сервлета контейнер сервлетов вызывает метод destroy(), в теле которого следует помещать код освобождения занятых сервлетом ресурсов.

Интерфейсом Servlet предусмотрена реализация еще двух методов: getServletConfig() и getServletInfo(). Первый возвращает объект типа ServletConfig, содержащий параметры конфигурации сервлета, а второй – строку, описывающую назначение сервлета.

При разработке сервлетов в качестве базового класса в большинстве случаев используют не интерфейс Servlet, а класс HttpServlet, отвечающий за обработку запросов HTTP.

Класс HttpServlet имеет реализованный метод service(), служащий диспетчером для других методов, каждый из которых обрабатывает методы доступа к ресурсам. В спецификации HTTP определены следующие методы: GET, HEAD, POST, PUT, DELETE, OPTIONS и TRACE. Наиболее часто употребляются методы GET и POST, с помощью которых на сервер передаются запросы, а также параметры для их выполнения.

При использовании метода GET (по умолчанию) параметры передаются как часть URL, значения могут выбираться из полей формы или передаваться непосредственно через URL. При этом запросы кэшируются и имеют ограничения на размер. При использовании метода POST (method=POST) параметры (поля формы) передаются в содержимом HTTP-запроса и упакованы согласно полю заголовка Content-Type. По умолчанию в формате: <имя>=<значение>&<имя>=<значение>&...

Однако форматы упаковки параметров могут быть самые разные, например: в случае передачи файлов с использованием формы enctype="multipart/form-data".

В задачу метода service() класса HttpServlet входит анализ полученного через запрос метода доступа к ресурсам и вызов метода, имя которого сходно с названием метода доступа к ресурсам, но перед именем добавляется префикс do: doGet() или doPost(). Кроме этих методов могут использоваться методы: doHead(), doPut(), doDelete(), doOptions() и doTrace(). Разработчик должен переопределить нужный метод, разместив в нем функциональную логику.

Задание

1. Изучить технологию Java Servlets.
2. Описать жизненый цикл работы сервлетов.
3. Разработать сервлет.
4. Разработать web-интерфейс и связать его с сервлетом.
5. Разработать сервлет, который принимает данные от пользователя и выводит соответствующие сообщения. Реализовать адаптивный тест из цепочки в 3 – 4 вопроса.
6. Готовое веб-приложение разместить на сервере Tomcat (http://tomcat.apache.org/download-80.cgi).
7. Упаковать приложение и развернуть на сервере.

Вопросы для самоконтроля

1. Что такое сервлет?
2. Какие еще существуют технологии, похожие на сервлеты?
3. Какова структура каталогов web-приложения?
4. Какой класс является базовым для сервлетов?

5. Каков жизненный цикл у сервлета?
6. Каким образом послать ответ клиенту?

2.8 Практическое занятие «Анализ сетевых приложений и сетевого оборудования»

Цель

Мониторинг сетевых приложений с помощью программы Zabbix.

Методические указания

Имеется множество различного программного обеспечения, предназначенного для анализа компьютерных сетей и сетевого программного обеспечения: Zabbix, Nagios, Cacti и др.

Рассмотрим использование системы Zabbix.

Zabbix – свободно распространяемая система для комплексного мониторинга сетевого оборудования и серверов, сервисов. Состоит из следующих компонент:

1. Сервер мониторинга (ядро) – выполняет периодический опрос и получение данных, обрабатывает их, анализирует, также осуществляет запуск скриптов для рассылки оповещений. Может удаленно проверять сетевые сервисы, является хранилищем, в котором хранятся все конфигурационные, статистические и оперативные данные.
2. Прокси – собирает данные о производительности и доступности от имени Zabbix сервера. Все собранные данные заносятся в буфер на локальном уровне и передаются Zabbix серверу, к которому принадлежит прокси-сервер. Zabbix прокси является решением для централизованного удаленного мониторинга мест, филиалов, сетей, не имеющих локальных администраторов. Он может быть также использован для распределения нагрузки одного Zabbix сервера.
3. Агент – специальный демон, который запускается на отслеживаемых объектах и предоставляет данные серверу, осуществляя контроль локальных ресурсов и приложений (таких как жесткие диски, память, статистика процессора и т. д.) на сетевых системах, т.е. эти системы должны работать с запущенным Zabbix агентом.
4. Веб-интерфейс – средство визуального представления Zabbix, реализован на PHP, для запуска требует наличия веб-сервера.

Основные функции:

1. Сбор данных (проверки доступности и производительности, поддержка мониторинга с использованием SNMP (и трапперы, и поллеры), IPMI, JMX, VMware, пользовательские проверки, сбор желаемых данных с использованием пользовательских интервалов выполняются сервером/прокси и агентами).
2. Установка гибких порогов.
3. Установка множеств оповещений (отправку оповещений можно настроить, используя расписания эскалаций, получателей, типов оповеще-

ний, оповещения можно сделать информативными и полезными при использовании переменных макросов, автоматические действия, включающие в себя удаленные команды).

4. Построение графиков в режиме реального времени.
5. Возможности Веб-мониторинга.
6. Широкие возможности визуализации(возможность создавать пользовательские графики, что позволяет комбинировать множество элементов данных в одном месте карты сети, пользовательские комплексные экраны и слайд-шоу, отчеты, высокоуровневое (бизнес) представление наблюдаемых ресурсов.
7. Хранение данных истории(данные записываются в базу данных, настраиваемая история, встроенная процедура очистки истории.

Задание

1. Скачать Zabbix (www.Zabbix.com).
2. Установить и настроить.
3. Добавить новый узел.
4. Создать несколько элементов данных.
5. Добавить новый триггер.
6. Настроить в Zabbix уведомления оповещений.
7. Разработать веб-сценарий для исследования и мониторинга веб-сервера и ресурсов.
8. Создать распределённый мониторинг.

Вопросы для самоконтроля

1. Каковы основные цели мониторинга сетевого трафика?
2. Чем отличается мониторинг трафика от фильтрации?
3. Основные функции и возможности Zabbix.

2.9 Практическое занятие «Сетевые программы для обмена сообщениями между пользователями»

Цель

Разработка сетевых программ для обмена сообщениями между пользователями.

Методические указания

1. Разработка чата

Чатом называют онлайновую конференцию, в которой пользователи могут одновременно обмениваться сообщениями. Также чатом называют программу, организующую такой вид Интернет-коммуникаций.

Основные элементы чата:

1) регистрация пользователей (имя, пароль);
2) список активных пользователей;
3) архив сообщений;

2. Разработка форума

Форумом называют Интернет-конференцию, в которой обмен сообщениями не является одновременным. Форум предлагает набор разделов для обсуждения. Работа форума заключается в создании пользователями тем в разделах и последующим обсуждением внутри этих тем. Для форума характерно следующее деление: разделы – темы – сообщения. Сообщения имеют следующую структуру: «автор – тема – содержание – дата/время». Сообщение и все ответы на него образует ветку.

Обсуждение должно соответствовать теме. Отклонение от начальной темы обсуждения часто запрещено правилами поведения форума. За соблюдением правил следит модератор – участник, наделённый возможностью удалять чужие сообщения в определённом разделе или теме, а также контролировать доступ к ним отдельных участников.

Основные элементы форума:

1. Регистрация (создание базы данных пользователей) <имя-пароль-дополнительная информация>.
2. Главная страница, где представлены разделы форума.
3. Страница темы, сообщения и ответы на сообщения.

3. Разработка простой сетевой игры

Сервер обеспечивает логику изменения обстановки, некоторой совокупности объектов и трансляцию этой обстановки каждому клиенту.

Клиент (браузер) обеспечивает изменение состояния некоторого объекта обстановки и передает на сервер.

4. Программа мониторинга Интернета

Программа мониторинга Интернета предназначена для наблюдения за изменением публикаций в некоторой предметной области. Например, имеется организация «XXX», хотелось бы знать динамику изменений публикаций по поводу деятельности «XXX». Другой пример, в Интернете имеется некоторая услуга, хотелось бы иметь информацию о динамике изменений этой услуги.

Такая программа должна обеспечивать выполнение следующих функций:

1) формирование базы знаний условий мониторинга;

2) формирование начальной базы данных, отражающей начальное состояние исследуемой области;

3) текущий мониторинг Интернета, формирование базы данных на текущий момент;

4) управление текущим мониторингом, изменение частоты запуска, изменение сегментов Интернета и пр.

5. Разработка сетевой обучающей программы

Сетевая обучающая программа предназначена для обучения пользователей Интернета некоторой дисциплине. Дисциплина представлена на серверной стороне некоторым набором учебных модулей. Каждый модуль

отвечает за обучение конкретной теме и соответствует некоторому виду обучения: представление теории, тренажеры и виртуальные лабораторные работы, модули контроля знаний.

Основные функции программы:

1) регистрация пользователей;

2) выдача модуля в соответствии с моделью обучения;

3) ведение модели обучения.

6. Разработка сетевой тестирующей программы

Сетевая тестирующая программа обеспечивает тестирование знаний по некоторой дисциплины или специальности. На некотором WWW-сервере организуется база вопросов (в простейшем виде вопросы представлены в форме меню). Программа выполняет следующие функции:

1) регистрация пользователя;

2) выбор теста;

3) формирование множества вопросов;

4) тестирование;

5) формирование оценки и выдача;

6) формирование протокола.

7. Разработка простой информационной системы, основанной на интранете.

Простейшая информационная система обеспечивает автоматизацию документооборота некоторой фирмы. Основные функции такой системы:

1) ввод документов;

2) занесение документов в базу данных;

3) поиск документов в базе данных;

4) формирование разнообразных отчетов;

5) архивация ненужных документов в базе данных;

6) аутентификация и регистрация пользователей.

Достоинства интранета следующие: нет привязки клиента к компьютеру, нет привязки к операционной системе. Главный недостаток системы состоит в том, что вся обработка ложится на сервер. Наиболее предпочтительная схема реализации: Apache+PHP.

8. Разработка программы определения местоположения компьютера по IP-адресу.

Для построения такой программы необходимо использовать географическую базу. Например, базу городов, университетов, фирм и т.д. Далее, используя поисковые сервера, найдите связь между соответствующим географическим названием и некоторым сайтом. Затем провести анализ связи. Если эта связь существенна, то по имени сайта ищем его IP-адрес, используя приведенную ниже программу.

```
#include "..\winsock.h"
```

```
// файл заголовков Winsock
#define PROG_NAME "Simple DNS Lookup"
#define HOST_NAME "CERFNET.COM"
// может быть любым (настоящим именем компьютера
#define WINSOCK_VERSION 0x0101
// Необходим Winsock версии 1.1
#define PF_INET_LENGTH 4
// длина адреса в протоколах
//Интернет всегда равна  4 байтам
#define HOST_ADDR "129.79.26.27"

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
LPSTR lpszCmdParam, int nCmdShow)
{
WSADATA wsaData;
// сведения о реализации Winsock LPHOSTENT lpHostEnt;
// структура с информацией о
// сетевом компьютере Интернет
DWORD dwIPAddr;
// IP-адрес в виде беззнакового
// целого двойной длины LPSTR szIPAddr;
// IP-адрес в виде "десятичное
// с точкой"
if(WSAStartup(WINSOCK_VERSION, &wsaData))
MessageBox(NULL, "Could not load Windows SocketsDLL.",
ROG_NAME, MB_OK|MB_ICONSTOP);
else
// преобразуем имя сетевого хоста
{
lpHostEnt = gethostbyname(HOST_NAME);
if (!lpHostEnt) MessageBox(NULL, "Could not get IP address!",
HOST_NAME, MB_OK|MB_ICONSTOP);
else
// IP-адрес преобразуется в нотацию "десятичное
// с точкой"
{
szIPAddr = inet_ntoa
(*(LPIN_ADDR)*
(lpHostEnt->h_addr_list));
MessageBox(NULL, szIPAddr, lpHostEnt->h_name,
MB_OK|MB_ICONINFORMATION);
}
// Формат "десятичное с точкой" преобразуется в
// 32-разрядный IP-адрес
dwIPAddr = inet_addr(HOST_ADDR);
if (dwIPAddr == INADDR_NONE)
MessageBox(NULL,"Invalid Internet address!", HOST_ADDR,
MB_OK|MB_ICONSTOP);
else // Преобразуем IP-адрес
{
```

```
 lpHostEnt = gethostbyaddr((LPSTR)
&dwIPAddr,PF_INET_LENGTH,PF_INET);
 if (!lpHostEnt) MessageBox(NULL, "Could not get host name!",
HOST_ADDR, MB_OK|MB_ICONSTOP);
else
MessageBox(NULL, lpHostEnt->h_name,HOST_ADDR,
MB_OK|MB_ICONINFORMATION);
}
}
 WSACleanup();
// Программа освобождает занятые ресурсы
// и завершается
return(NULL);
}
```

Вопросы для самоконтроля

1. Что такое чат в сети Интернет?
2. Что такое форум в сети Интернет?
3. Что такое IP-адрес и как его определить?
4. Укажите достоинства интранета.

3 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

3.1 Общие положения

Целью самостоятельной работы является систематизация, расширение и закрепление теоретических знаний, использование материала, собранного и полученного в ходе самостоятельной подготовки к практическим занятиям.

Самостоятельная работа включает в себя подготовку к практическим занятиям, проработку лекционного материала и подготовку к контрольным работам, проработку тем дисциплины, вынесенных на самостоятельное изучение.

3.2 Проработка лекционного материала

Изучение теоретической части дисциплин призвано не только углубить и закрепить знания, полученные на аудиторных занятиях, но и способствовать развитию у студентов творческих навыков, инициативы и организовать свое время.

Проработка лекционного материала включает:

- чтение студентами рекомендованной литературы и усвоение теоретического материала дисциплины;
- знакомство с Интернет-источниками;
- подготовку ответов на вопросы по различным темам дисциплины в той последовательности, в какой они представлены.

Планирование времени, необходимого на изучение дисциплин, студентам лучше всего осуществлять весь семестр, предусматривая при этом регулярное повторение материала.

Материал, законспектированный на лекциях, необходимо регулярно прорабатывать и дополнять сведениями из других источников литературы, представленных не только в программе дисциплины, но и в периодических изданиях.

При изучении дисциплины сначала необходимо по каждой теме прочитать рекомендованную литературу и составить краткий конспект основных положений, терминов, сведений, требующих запоминания и являющихся основополагающими в этой теме для освоения последующих тем курса. Для расширения знания по дисциплине рекомендуется использовать Интернет-ресурсы; проводить поиски в различных системах и использовать материалы сайтов, рекомендованных преподавателем.

3.3 Подготовка к практическим занятиям

Проведение практических занятий включает в себя следующие этапы:

- постановку темы занятий и определение задач практического занятия;

• определение порядка практического занятия или отдельных его этапов;

• непосредственное выполнение практического задания студентами и контроль за ходом занятий;

• подведение итогов практического занятия и формулирование основных выводов;

• оформление отчета и защиты практического задания (демонстрация работы и ответы на вопросы по теме занятия).

При подготовке к практическим занятиям необходимо заранее изучить методические рекомендации по его проведению. Обратить внимание на цель занятия, на основные вопросы для подготовки к занятию, на содержание темы занятия.

Если в процессе практического занятия или над изучением теоретического материала у студента возникают вопросы, разрешить которые самостоятельно не удается, необходимо обратиться к преподавателю для получения у него разъяснений или указаний.

ЛИТЕРАТУРА

1. Эммерих В. Конструирование распределенных объектов. – М.: Мир, 2002. – 510 с.

2. Рихтер Дж. Windows для профессионалов: создание эффективных приложений с учетом специфики 64-разрядной версии Windows. – СПб.: Питер, 2001. – 752 с.

3. Вильямс А. Системное программирование в Windows 2000 для профессионалов – СПб.: Питер. – 624 с.

4. Семенов Ю.А. Сети Интернет. Архитектура и протоколы. – М.: Блик плюс, 1998. – 424 с.

5. Разработка Web-приложений на Microsoft Visual Basic .NET и Microsoft Visual *C#* .NET. – М.: Издательско-торговый дом «Русская редакция», 2003. – 660 с.

6. Стивенс У. Р. Unix. Разработка сетевых приложений. – СПб.: Питер, 2003. – 1088 с.

7. Блинов, И. Н. Java 2: практ. рук. / И.Н. Блинов, В.С. Романчик. – Мн.: УниверсалПресс, 2005. – 403 с.

8. Tomcat Server. – URL : http://tomcat.apache.org/download-80.cgi (дата обращения: 13.06.2018).

9. Морозова, Ю. В. Методы и технологии разработки клиент-серверных приложений: Учебное пособие [Электронный ресурс] / Ю. В. Морозова, В. В. Кручинин. – Томск: ТУСУР, 2018. – 106 с. – Режим доступа: https://edu.tusur.ru/publications/7922