|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждениевысшего профессионального образования «Московский технологический университет»  МИРЭА | | |

Институт Информационных Технологий

Кафедра Корпоративных Информационных Систем

**ОТЧЕТ**

по Лабораторной Работе №4

на тему

«Виды архитектур сокетных пулов»

по дисциплине

«Разработка клиент-серверных приложений»

Выполнил студент группы ИКБО-08-18 Валяев Д.А.

Принял старший преподаватель Мирзоян Д.И.

Выполнено «\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 г.

Зачтено «\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 г.

Москва 2020

**Задание**

Изучить особенности работы сокетов в различных режимах, используя различные подходы к организации их опроса

**Теоретическая часть**

Что такое «блокирующий» и «неблокирующий» вводы/выводы

Разберёмся с видами ввода/вывода на примере сетевого сокета, через который пользователь интернета соединяется с сайтом, и загружает на него картинку для аватара. «Синхронный» и «блокирующий» ввод/вывод - это самый простой и обычный ввод/вывод, на котором пишется большая часть сайтов:

* Открыть файл
* Начать его считывать
* Ждать, пока файл считается
* Закрыть файл
* Вывести содержимое файла на экран

В случае с нашим сокетом это будет:

* Начать прослушивать сокет
* Считать из него первую порцию данных картинки
* Ждать, пока придёт вторая порция данных картинки
* Считать из него вторую порцию данных картинки
* Ждать, следующую порцию данных картинки
* ...
* Установить картинку в качестве аватара

При этом в коде нашей программы возникает «блокировка», во время которой поток простаивает, хотя мог бы заняться чем-нибудь полезным. Для решения этой задачи был придуман «синхронный» и «неблокирующий» ввод/вывод:

* Начать прослушивать сокет
* Если нет новых данных, остановить прослушивание сокета
* Считать данные с сокета, если они имеются
* перестать прослушивать сокет

Если эти шаги выполнять в цикле, пока не будет считана последняя порция данных картинки, то мы опять же в итоге получим всю картинку полностью. С тем лишь отличием, что в этом цикле, помимо считывания данных с сокета, мы можем делать что-нибудь ещё полезное, а не простаивать под «блокировкой». Например, можно было бы считывать данные ещё и с другого сокета.

Существует ещё «асинхронный» ввод/вывод. Это когда мы вешаем на сокет «функцию обратного вызова» из нашего кода, которая будет вызываться операционной системой каждый раз, когда на этот сокет будет приходить очередная порция данных картинки. И дальше уже забываем о прослушивании этого сокета вообще, отправляясь делать другие дела. «Асинхронный» ввод/вывод, как и «синхронный», делится на «блокирующий» и «неблокирующий».

Асинхронная событийная модель

Нужна была новая архитектура для подобного класса приложений. И в такой ситуации, как нельзя кстати, подошла «асинхронная событийная модель». В основе её лежат «событийный цикл» и шаблон «reactor».

«Событийный цикл» представляет собой бесконечный цикл, который опрашивает «источники событий» на предмет появления в них какого-нибудь «события». Опрос происходит с помощью библиотеки «синхронного» ввода/вывода, который, при этом будет являться «неблокирующим».

То есть, во время очередного витка «событийного цикла», наша система проходит последовательно по всем дескрипторам, и пытается считать из них «события»: если таковые имеются, то они возвращаются функцией чтения в нашу систему; если же никаких новых событий у дескриптора нет, то он не станет «блокировать» и ждать появления «события», а сразу же возвратит ответ: «новых событий нет».

Событием может быть приход очередной порции данных на сетевой сокет, или считывание новой порции данных с жёсткого диска: в общем, любой ввод/вывод. Например, когда вы загружаете картинку на хостинг, данные туда приходят кусками, каждый раз вызывая событие «новая порция данных картинки получена».

Источником событий в данном случае будет являться «дескриптор» того самого TCP-сокета, через который вы соединились с сайтом по сети.

Второй компонент новой архитектуры, как уже было сказано, – это шаблон «reactor». Суть этого шаблона заключается в том, что код сервера пишется не одним большим куском, который исполняется последовательно, а небольшими блоками, каждый из которых вызывается, когда происходит связанное с ним событие. Таким образом, код представляет собой набор множества блоков, задача которых состоит в том, чтобы «реагировать» на какие-то события.

**Выполнение работы**

При выполнении работы сначала были реализованы клиентское приложение, которое будет использоваться для всех реализованных серверов и блокирующий сервер для полинга массива сокетов.

При посылании 10000 сообщений среднее время на ответ составило 0.28 мс. Затраты ресурсов представлены ниже.

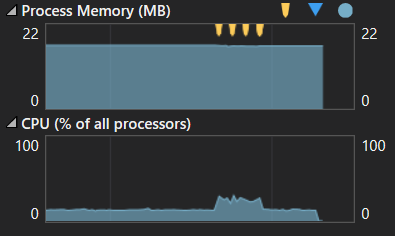


Рис. 1 – Потребление ресурсов блокирующим сервером

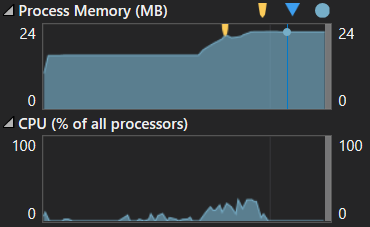


Рис. 2 – Потребление ресурсов клиентом при посылании на блокирующий сервер.

Далее был реализован неблокирующий сервер.

При посылании 10000 сообщений среднее время на ответ составило 0.268 мс. Затраты ресурсов представлены ниже.

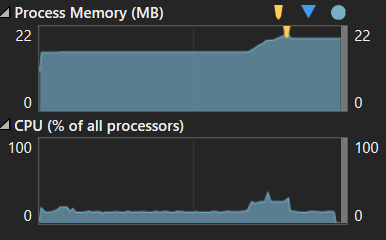


Рис. 3 – Потребление ресурсов неблокирующим сервером

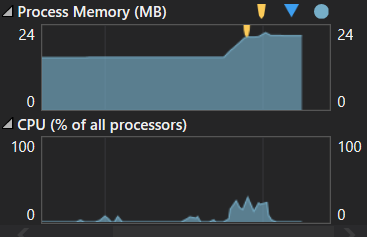


Рис. 4 – Потребление ресурсов клиентом при посылании на неблокирующий сервер.

Затем использовался сервер, использующий потоки для обработки I/O

При посылании 10000 сообщений среднее время на ответ составило 0.077 мс. Затраты ресурсов представлены ниже.

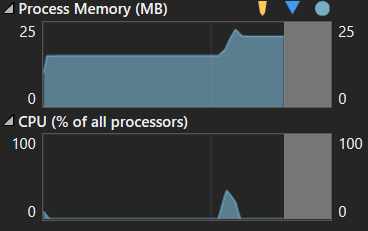


Рис. 5 – Потребление ресурсов сервером использующий потоки

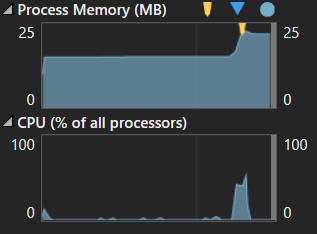


Рис. 6 – Потребление ресурсов клиентом при посылании на сервер, использующий потоки.

Затем использовался сервер, использующий несколько потоков для обработки I/O

При посылании 10000 сообщений среднее время на ответ составило 0.081 мс. Затраты ресурсов представлены ниже.

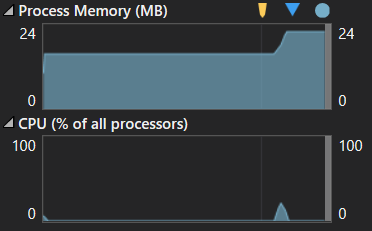


Рис. 7 – Потребление ресурсов сервером использующий несколько потоков

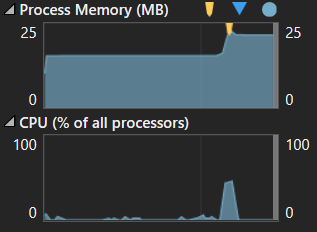


Рис. 8 – Потребление ресурсов клиентом при посылании на сервер, использующий несколько потоков.

**Заключение**

В результате данной лабораторной работы были реализованы четыре приложения, осуществляющие различные подходы реализации полинга, а также было реализовано клиентское приложение осуществляющее отправку сообщений на серверы. По результату выполнения работы были продемонстрированы временные и иные характеристики различных подходов

**Исходный код**

**1 клиент**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab4\_client

{

public partial class Form1 : Form

{

System.Diagnostics.Stopwatch watch;

IPHostEntry ipHostEntry;

IPAddress ipAddress;

Socket trans\_tcp;

SocketAsyncEventArgs e = new SocketAsyncEventArgs();

double times = 0;

double time = 0;

byte[] buffer = new byte[16384];

public Form1()

{

InitializeComponent();

ipHostEntry = Dns.Resolve(Dns.GetHostName());

ipAddress = ipHostEntry.AddressList[0];

watch = new System.Diagnostics.Stopwatch();

}

int count;

private void button1\_Click(object sender, EventArgs e)

{

times += (int)numericUpDown1.Value;

count = 0;

watch.Start();

for (int i = 0; i < numericUpDown1.Value; i++)

{

System.Threading.ThreadPool.QueueUserWorkItem(sendReceive);

}

while (count < numericUpDown1.Value)

System.Threading.Thread.Sleep(1);

watch.Stop();

textBox1.AppendText(watch.ElapsedMilliseconds.ToString() + "\r\n");

time += (int)watch.ElapsedMilliseconds;

textBox2.Text = (time / times).ToString();

watch.Reset();

}

public void sendReceive(object state)

{

Socket trans\_tcp = new Socket(SocketType.Stream, ProtocolType.IP);

trans\_tcp.Connect(new IPEndPoint(ipAddress, 2002));

trans\_tcp.Send(buffer, buffer.Length, SocketFlags.None);

int rcv = 0;

do

{

rcv += trans\_tcp.Receive(buffer, buffer.Length, SocketFlags.None);

} while (rcv < buffer.Length);

Interlocked.Increment(ref count);

trans\_tcp.Disconnect(false);

}

private void button2\_Click(object sender, EventArgs e)

{

times = 0;

time = 0;

textBox2.Text = "";

}

}

}

**2 Неблокирующий сервер**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab4\_nonBlock

{

public partial class Form1 : Form

{

IPHostEntry ipHostEntry;

IPAddress ipAddress;

Socket acceptSocket;

List<Socket> accepted;

Thread acceptThread;

Thread readThread;

SocketAsyncEventArgs e = new SocketAsyncEventArgs();

byte[] buffer = new byte[128\*1024];

public Form1()

{

InitializeComponent();

accepted = new List<Socket>();

ipHostEntry = Dns.Resolve(Dns.GetHostName());

ipAddress = ipHostEntry.AddressList[0];

acceptSocket = new Socket(AddressFamily.InterNetwork,

SocketType.Stream,

ProtocolType.Tcp);

acceptSocket.Blocking = false;

acceptSocket.Bind(new IPEndPoint(IPAddress.Any, 2002));

acceptSocket.Listen(1000);

acceptThread = new Thread(new ThreadStart(acceptListen));

acceptThread.Start();

readThread = new Thread(new ThreadStart(pollAcceped));

readThread.Start();

}

public void acceptListen()

{

Socket newConnection = null;

do

{

if (acceptSocket.Poll(-1, SelectMode.SelectRead))

{

newConnection = acceptSocket.Accept();

newConnection.Blocking = false;

lock (accepted)

{

accepted.Add(newConnection);

}

}

} while (newConnection != null);

}

public void pollAcceped()

{

while (true)

{

lock (accepted)

{

for (int i = 0; i < accepted.Count; i++)

{

if (accepted[i].Poll(-1, SelectMode.SelectRead))

{

int x = accepted[i].Receive(buffer, 16384, SocketFlags.None);

if (x != 0)

{

accepted[i].Send(buffer, x, SocketFlags.None);

}

else

{

accepted.Remove(accepted[i]);

i--;

}

}

}

}

}

}

}

}

**3 Блокирующий сервер**

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Threading;

namespace Lab4\_serv

{

public partial class Form1 : Form

{

IPHostEntry ipHostEntry;

IPAddress ipAddress;

Socket acceptSocket;

List<Socket> accepted;

Thread acceptThread;

Thread readThread;

System.Diagnostics.Stopwatch watch;

byte[] buffer = new byte[128 \* 1024];

public Form1()

{

InitializeComponent();

accepted = new List<Socket>();

ipHostEntry = Dns.Resolve(Dns.GetHostName());

ipAddress = ipHostEntry.AddressList[0];

acceptSocket = new Socket(AddressFamily.InterNetwork,

SocketType.Stream,

ProtocolType.Tcp);

acceptSocket.Bind(new IPEndPoint(IPAddress.Any, 2002));

acceptSocket.Listen(1000);

acceptThread = new Thread(new ThreadStart(acceptListen));

acceptThread.Start();

readThread = new Thread(new ThreadStart(pollAcceped));

readThread.Start();

}

public void acceptListen()

{

Socket newConnection = null;

do

{

if (acceptSocket.Poll(-1, SelectMode.SelectRead))

{

newConnection = acceptSocket.Accept();

if (newConnection != null)

{

lock (accepted)

{

accepted.Add(newConnection);

}

}

}

} while (newConnection != null);

}

public void pollAcceped()

{

while (true)

{

lock (accepted)

{

for (int i = 0; i < accepted.Count; i++)

{

if (accepted[i].Poll(-1, SelectMode.SelectRead))

{

int x = accepted[i].Receive(buffer, 16384, SocketFlags.None);

if (x != 0)

{

accepted[i].Send(buffer, x, SocketFlags.None);

}

else

{

accepted.Remove(accepted[i]);

i--;

}

}

}

}

}

}

}

}

**4 Сервер с использованием потоков**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lab4\_thread

{

public partial class Form1 : Form

{

IPHostEntry ipHostEntry;

IPAddress ipAddress;

Socket acceptSocket;

List<Socket> accepted;

Thread acceptThread;

Thread readThread;

SocketAsyncEventArgs e = new SocketAsyncEventArgs();

byte[] buffer = new byte[128 \* 1024];

public Form1()

{

InitializeComponent();

accepted = new List<Socket>();

ipHostEntry = Dns.Resolve(Dns.GetHostName());

ipAddress = ipHostEntry.AddressList[0];

acceptSocket = new Socket(AddressFamily.InterNetwork,

SocketType.Stream,

ProtocolType.Tcp);

acceptSocket.Bind(new IPEndPoint(IPAddress.Any, 2002));

acceptSocket.Listen(1000);

acceptThread = new Thread(new ThreadStart(acceptListen));

acceptThread.Start();

}

public void acceptListen()

{

Socket newConnection = null;

do

{

if (acceptSocket.Poll(-1, SelectMode.SelectRead))

{

newConnection = acceptSocket.Accept();

if (newConnection != null)

{

System.Threading.ThreadPool.QueueUserWorkItem(pollAcceped, newConnection);

}

}

} while (newConnection != null);

}

private void pollAcceped(object acc)

{

Socket accepted = (Socket)acc;

if (accepted.Poll(-1, SelectMode.SelectRead))

{

int x = accepted.Receive(buffer, 16384, SocketFlags.None);

accepted.Send(buffer, x, SocketFlags.None);

}

}

}

}

**5 Сервер с использованием несколько потоков**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab4\_threads

{

public partial class Form1 : Form

{

IPHostEntry ipHostEntry;

IPAddress ipAddress;

Socket acceptSocket;

List<Socket> accepted;

Thread acceptThread;

Thread readThread;

List<Thread> acceptThreads;

SocketAsyncEventArgs e = new SocketAsyncEventArgs();

byte[] buffer = new byte[128 \* 1024];

public Form1()

{

InitializeComponent();

accepted = new List<Socket>();

ipHostEntry = Dns.Resolve(Dns.GetHostName());

ipAddress = ipHostEntry.AddressList[0];

acceptSocket = new Socket(AddressFamily.InterNetwork,

SocketType.Stream,

ProtocolType.Tcp);

acceptSocket.Bind(new IPEndPoint(IPAddress.Any, 2002));

acceptSocket.Listen(1000);

acceptThread = new Thread(new ThreadStart(acceptListen));

acceptThread.Start();

acceptThreads.Add(new Thread(new ThreadStart(acceptListen)));

acceptThreads.Add(new Thread(new ThreadStart(acceptListen)));

acceptThreads.Add(new Thread(new ThreadStart(acceptListen)));

acceptThreads.Add(new Thread(new ThreadStart(acceptListen)));

acceptThreads.Add(new Thread(new ThreadStart(acceptListen)));

foreach(Thread tr in acceptThreads)

{

tr.Start();

}

}

public void acceptListen()

{

Socket newConnection = null;

do

{

if (acceptSocket.Poll(-1, SelectMode.SelectRead))

{

newConnection = acceptSocket.Accept();

if (newConnection != null)

{

pollAcceped(newConnection);

}

}

} while (newConnection != null);

}

private void pollAcceped(object acc)

{

Socket accepted = (Socket)acc;

if (accepted.Poll(-1, SelectMode.SelectRead))

{

int x = accepted.Receive(buffer, 16384, SocketFlags.None);

accepted.Send(buffer, x, SocketFlags.None);

}

}

}

}