Пермский филиал федерального государственного автономного образовательного учреждения высшего образования   
«Национальный исследовательский университет   
«Высшая школа экономики»

*Факультет социально-экономических и компьютерных наук*

Ивакин Александр Дмитриевич

**Отчет о лабораторной работе №2**

**«Мэйлслоты»**

студента образовательной программы бакалавриата «Программная инженерия»

по направлению подготовки *09.03.04 Программная инженерия*

Преподаватель

Доцент каф. ИТБ, к.ф-м.н.

Е.Б. Замятина

Пермь, 2024 год

Оглавление

[Оглавление 2](#_Toc179844928)

[Задание 1 3](#_Toc179844929)

[Задание 2 4](#_Toc179844930)

[Задание 3 5](#_Toc179844931)

[Задание 4 6](#_Toc179844932)

[Листинг 8](#_Toc179844933)

[Форма клиента 8](#_Toc179844934)

[Форма сервера 10](#_Toc179844935)

[Kernel32 12](#_Toc179844936)

[Enums 13](#_Toc179844937)

[Constants 14](#_Toc179844938)

Задание 1

На рисунке 1 представлен запуск севера и клиента на одной машине.

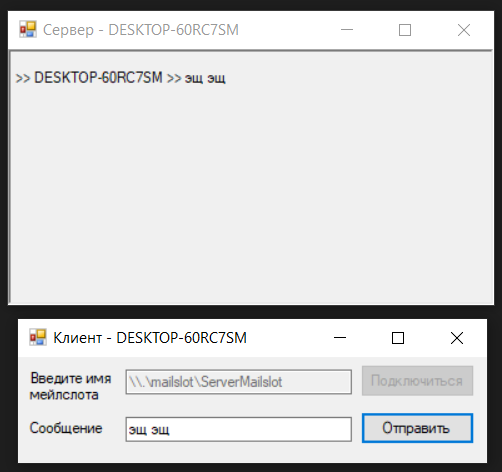


Рис. 1 – запуск сервера и клиента на одной машине

Для запуска сервера и клиента на разных машинах необходимо, чтобы данные машины были в одной локальной сети.

Мною были испробованы компьютеры, находящиеся в компьютерном классе общежития, но по каким-то причинам исходные приложения не работали. Вероятнее всего приложения не работали, потому что администратор сети запретил обмен сообщения между компьютерами.

Задание 2

На рисунке 2 представлен запуск трёх клиентов и трёх серверов, клиенты по очереди отправляли числа в порядке возрастания, начиная с 1 и заканчивая до 9.

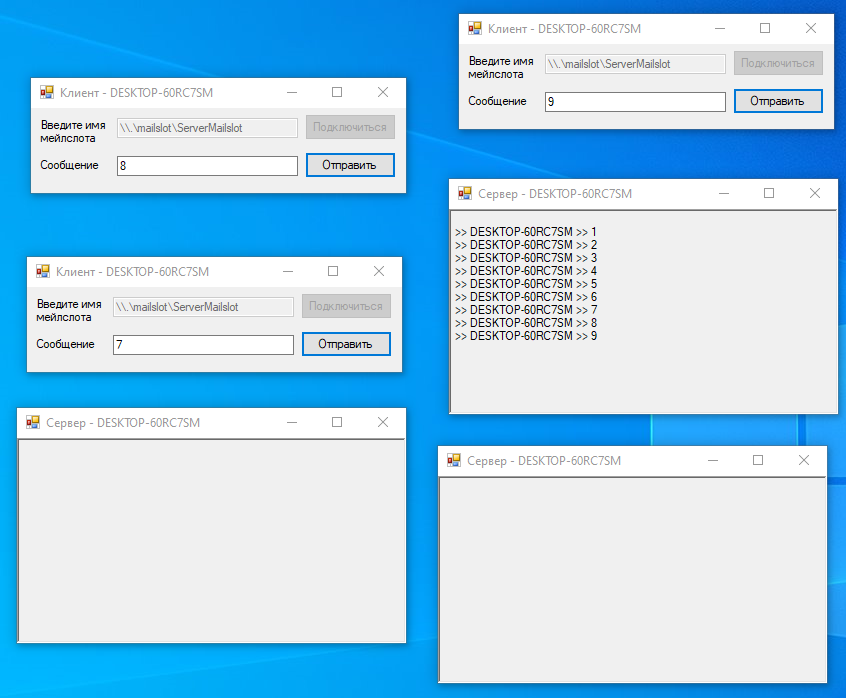


Рис 2. – Запуск трёх клиентов и трёх серверов

Сказать, что приложения вообще перестали работать – нельзя, они стали работать некорректно. Сообщения всех клиентов ушли на один сервер, который был создан самым первым.

Можно предположить, что это связано с тем, что на одной машине нельзя создавать несколько мейлслотов с одинаковыми именами. Первый сервер успешно создал мейлслот, а последующие не смогли создать мейлслоты с таким же именем. Это можно проверить, отдебажив, увидеть, что дескрипторы мейлслотов последующих серверов равны -1.

Задание 3

На рисунках 4 и 5 представлен модифицированный интерфейс клиента.

Перед отправкой сообщений клиенту стало необходимо ввести свой логин и авторизоваться, по умолчанию логином является имя машины клиента.

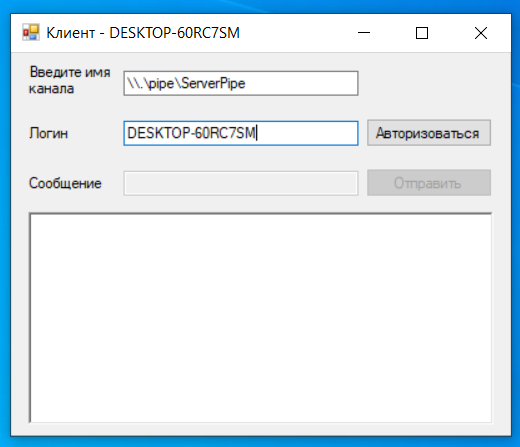


Рис 4. – Интерфейс клиента перед авторизацией

После авторизации у клиента появляется возможность писать и отправлять сообщения.

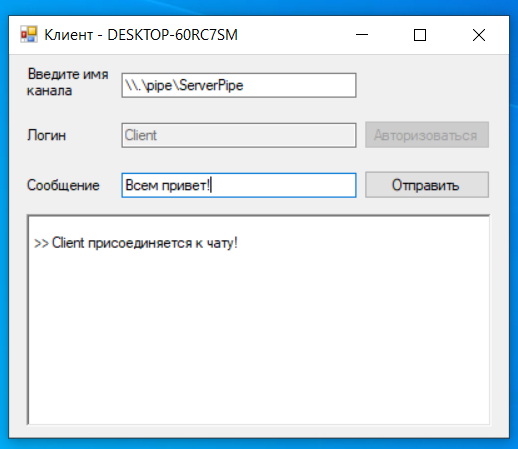


Рис 5. – Интерфейс клиента после авторизации

Задание 4

На рисунках 6 и 7 представлен чат между четырьмя клиентами.

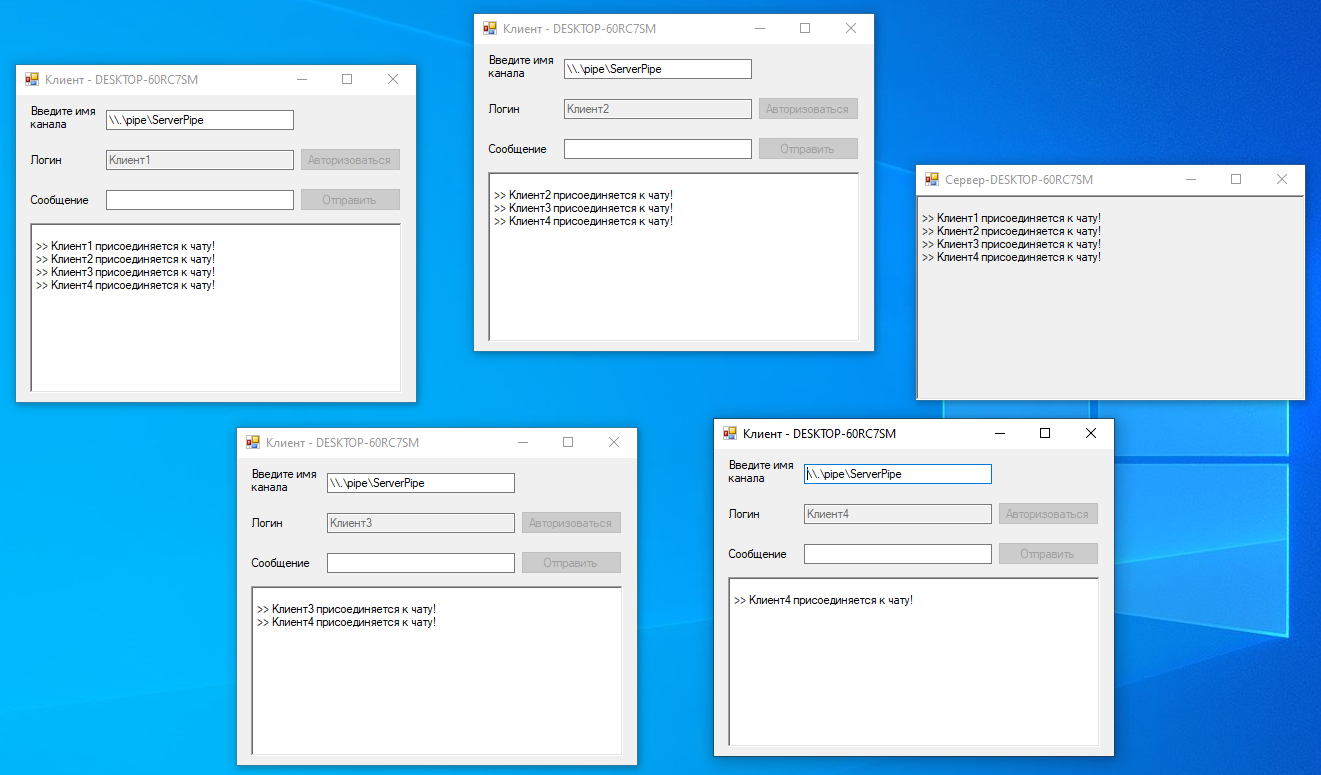


Рис 6. – Присоединение всех клиентов к чату

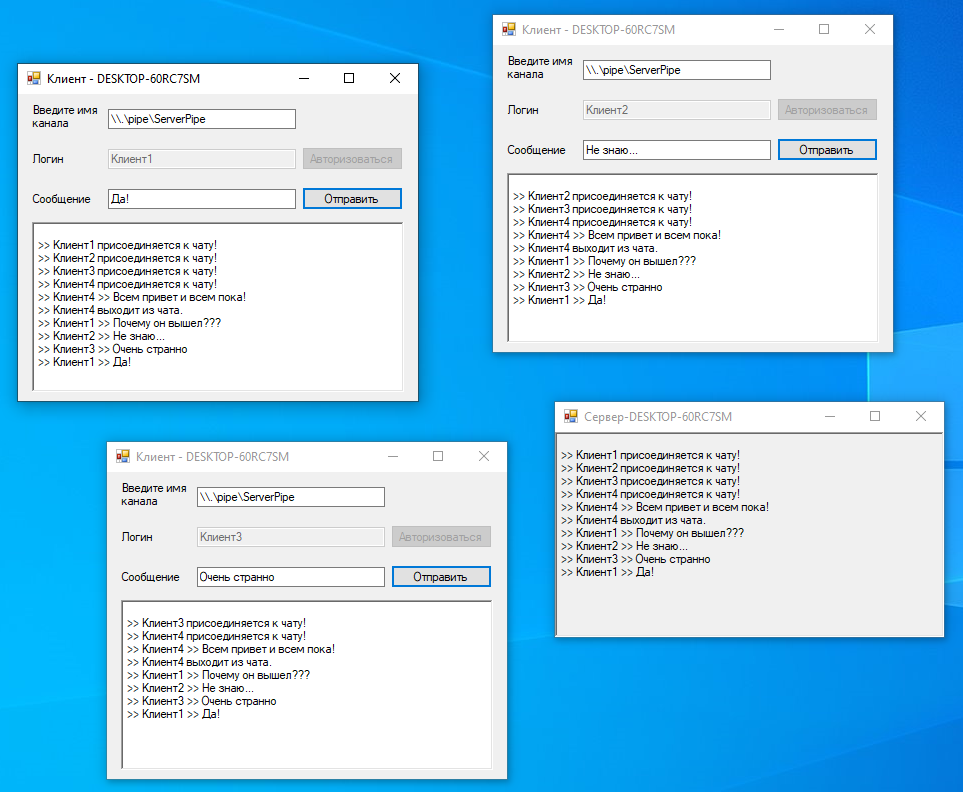


Рис 7. – Диалог клиентов после присоединения всех к чату

После авторизации клиент начинает видеть чат и сообщения, которые клиент сам отправляет и сообщения других клиентов, которые были отправлены после присоединения исходного клиента.

Помимо самих сообщений клиентов также есть сообщения о вступлении и выходе клиентов из чатов.

Клиенты создают свои именованные каналы, через которых сервер отправляет им новые сообщения в чате. Сервер хранит каналы клиентов, добавляет каналы при присоединении клиентов к чату и удаляет их при выходе из чата.

Листинг

Форма клиента

using System;

using System.Text;

using System.Windows.Forms;

using System.Net;

using PipesInfrastructure;

using System.Threading;

namespace Pipes

{

public partial class frmMain : Form

{

private int \_pipeClientHandle;

private readonly string \_pipeName;

private readonly Thread \_thread;

private bool \_continue = true;

public frmMain()

{

InitializeComponent();

var hostName = Dns.GetHostName();

Text += " - " + hostName;

\_tbLogin.Text = hostName;

\_pipeName = $"\\\\.\\pipe\\ClientPipe";

\_thread = new Thread(ReceiveMessage);

\_thread.Start();

}

private void ReceiveMessage()

{

uint realBytesReaded = 0;

while (\_continue)

{

if (Kernel32.ConnectNamedPipe(\_pipeClientHandle, 0))

{

Kernel32.FlushFileBuffers(\_pipeClientHandle);

byte[] buff = new byte[1024];

Kernel32.ReadFile(hFile: \_pipeClientHandle,

lpBuffer: buff,

nNumberOfBytesToRead: 1024,

lpNumberOfBytesRead: ref realBytesReaded,

lpOverlapped: 0);

var msg = Encoding.Unicode.GetString(buff);

\_rtbMessages.Invoke((MethodInvoker)delegate

{

if (msg != string.Empty)

\_rtbMessages.Text += "\n >> " + msg;

});

Kernel32.DisconnectNamedPipe(\_pipeClientHandle);

Thread.Sleep(500);

}

}

}

private void FrmMain\_FormClosing(object sender, FormClosingEventArgs e)

{

\_continue = false;

if (\_pipeClientHandle != -1)

Kernel32.CloseHandle(\_pipeClientHandle);

\_thread?.Abort();

}

private void TbLogin\_TextChanged(object sender, EventArgs e)

{

\_btLogin.Enabled = \_tbLogin.Text.Length != 0;

}

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

{

\_pipeClientHandle = Kernel32.CreateNamedPipe(

lpName: \_pipeName + \_tbLogin.Text,

dwOpenMode: Constants.PIPE\_ACCESS\_DUPLEX,

dwPipeMode: Constants.PIPE\_TYPE\_BYTE | Constants.PIPE\_WAIT,

nMaxInstances: Constants.PIPE\_UNLIMITED\_INSTANCES,

nOutBufferSize: 0,

nInBufferSize: 1024,

nDefaultTimeOut: Constants.NMPWAIT\_WAIT\_FOREVER,

lpSecurityAttributes: 0);

SendMessage(\_tbLogin.Text + " присоединяется к чату!");

\_btLogin.Enabled = false;

\_tbLogin.Enabled = false;

\_tbMessage.Enabled = true;

}

private void TbMessage\_TextChanged(object sender, EventArgs e)

{

\_btnSend.Enabled = \_tbMessage.Text.Length != 0;

}

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

{

SendMessage(\_tbLogin.Text + " >> " + \_tbMessage.Text);

}

private void SendMessage(string message)

{

uint bytesWritten = 0;

byte[] buff = Encoding.Unicode.GetBytes(message);

int pipeHandle = Kernel32.CreateFile(lpFileName: \_tbPipe.Text,

dwDesiredAccess: Enums.EFileAccess.GenericWrite,

dwShareMode: Enums.EFileShare.Read,

lpSecurityAttributes: 0,

dwCreationDisposition: Enums.ECreationDisposition.OpenExisting,

dwFlagsAndAttributes: 0,

hTemplateFile: 0);

Kernel32.WriteFile(hFile: pipeHandle,

lpBuffer: buff,

nNumberOfBytesToWrite: Convert.ToUInt32(buff.Length),

lpNumberOfBytesWritten: ref bytesWritten,

lpOverlapped: 0);

Kernel32.CloseHandle(hObject: pipeHandle);

}

private void FrmMain\_FormClosing\_1(object sender, FormClosingEventArgs e)

{

if (\_pipeClientHandle != 0)

SendMessage(\_tbLogin.Text + " выходит из чата.");

\_continue = false;

if (\_pipeClientHandle != -1)

Kernel32.CloseHandle(\_pipeClientHandle);

\_thread?.Abort();

}

}

}

Форма сервера

using System.Text;

using System.Windows.Forms;

using System.Net;

using System.Threading;

using PipesInfrastructure;

using System;

using System.Collections.Generic;

namespace Pipes

{

public partial class frmMain : Form

{

private readonly int \_pipeServerHandle;

private readonly string \_pipeServerName;

private readonly string \_pipeClientLogin;

private readonly Thread \_thread;

private bool \_continue = true;

private readonly HashSet<string> clientLogins = new HashSet<string>();

public frmMain()

{

InitializeComponent();

\_pipeServerName = "\\\\.\\pipe\\ServerPipe";

\_pipeClientLogin = "\\\\.\\pipe\\ClientPipe";

\_pipeServerHandle = Kernel32.CreateNamedPipe(

lpName: \_pipeServerName,

dwOpenMode: Constants.PIPE\_ACCESS\_DUPLEX,

dwPipeMode: Constants.PIPE\_TYPE\_BYTE | Constants.PIPE\_WAIT,

nMaxInstances: Constants.PIPE\_UNLIMITED\_INSTANCES,

nOutBufferSize: 0,

nInBufferSize: 1024,

nDefaultTimeOut: Constants.NMPWAIT\_WAIT\_FOREVER,

lpSecurityAttributes: 0);

var hostName = Dns.GetHostName();

Text += "-" + hostName;

\_thread = new Thread(ReceiveMessage);

\_thread.Start();

}

private void ReceiveMessage()

{

uint realBytesReaded = 0;

while (\_continue)

{

if (Kernel32.ConnectNamedPipe(\_pipeServerHandle, 0))

{

Kernel32.FlushFileBuffers(\_pipeServerHandle);

byte[] buff = new byte[1024];

Kernel32.ReadFile(hFile: \_pipeServerHandle,

lpBuffer: buff,

nNumberOfBytesToRead: 1024,

lpNumberOfBytesRead: ref realBytesReaded,

lpOverlapped: 0);

var msg = Encoding.Unicode.GetString(buff);

\_rtbMessages.Invoke((MethodInvoker)delegate

{

if (msg != string.Empty)

\_rtbMessages.Text += "\n >> " + msg;

var words = msg.Split();

var clientLogin = words[0];

switch (words[1])

{

case "присоединяется":

clientLogins.Add(clientLogin);

SendMessageClients(buff);

break;

case ">>":

SendMessageClients(buff);

break;

case "выходит":

SendMessageClients(buff);

clientLogins.Remove(clientLogin);

break;

}

});

Kernel32.DisconnectNamedPipe(\_pipeServerHandle);

Thread.Sleep(500);

}

}

}

private void SendMessageClients(byte[] buff)

{

foreach (var item in clientLogins)

{

uint BytesWritten = 0;

var pipeClientHandle = Kernel32.CreateFile(lpFileName: \_pipeClientLogin + item,

dwDesiredAccess: Enums.EFileAccess.GenericWrite,

dwShareMode: Enums.EFileShare.Read,

lpSecurityAttributes: 0,

dwCreationDisposition: Enums.ECreationDisposition.OpenExisting,

dwFlagsAndAttributes: 0,

hTemplateFile: 0);

Kernel32.WriteFile(hFile: pipeClientHandle,

lpBuffer: buff,

nNumberOfBytesToWrite: Convert.ToUInt32(buff.Length),

lpNumberOfBytesWritten: ref BytesWritten,

lpOverlapped: 0);

Kernel32.CloseHandle(hObject: pipeClientHandle);

}

}

private void FrmMain\_FormClosing(object sender, FormClosingEventArgs e)

{

\_continue = false;

if (\_pipeServerHandle != -1)

Kernel32.CloseHandle(\_pipeServerHandle);

\_thread?.Abort();

}

}

}

Kernel32

using System.Runtime.InteropServices;

namespace PipesInfrastructure

{

public static class Kernel32

{

[DllImport("kernel32.dll")]

public static extern int CreateNamedPipe(string lpName,

uint dwOpenMode,

uint dwPipeMode,

uint nMaxInstances,

uint nOutBufferSize,

uint nInBufferSize,

int nDefaultTimeOut,

uint lpSecurityAttributes);

[DllImport("kernel32.dll")]

public static extern bool ConnectNamedPipe(int hNamedPipe,

int lpOverlapped);

[DllImport("kernel32.dll")]

public static extern bool DisconnectNamedPipe(int hPipe);

[DllImport("kernel32.dll", SetLastError = true, CharSet = CharSet.Auto)]

public static extern int CreateFile(string lpFileName,

Enums.EFileAccess dwDesiredAccess,

Enums.EFileShare dwShareMode,

int lpSecurityAttributes,

Enums.ECreationDisposition dwCreationDisposition,

int dwFlagsAndAttributes,

int hTemplateFile);

[DllImport("kernel32.dll")]

public static extern bool ReadFile(int hFile,

byte[] lpBuffer,

uint nNumberOfBytesToRead,

ref uint lpNumberOfBytesRead,

int lpOverlapped);

[DllImport("kernel32.dll")]

public static extern bool WriteFile(int hFile,

byte[] lpBuffer,

uint nNumberOfBytesToWrite,

ref uint lpNumberOfBytesWritten,

int lpOverlapped);

[DllImport("kernel32.dll")]

public static extern byte FlushFileBuffers(int hPipe);

[DllImport("kernel32.dll", SetLastError = true)]

[return: MarshalAs(UnmanagedType.Bool)]

public static extern bool CloseHandle(int hObject);

}

}

Enums

using System;

namespace PipesInfrastructure

{

public static class Enums

{

[Flags]

public enum EFileAccess : uint

{

GenericRead = 0x80000000,

GenericWrite = 0x40000000,

GenericExecute = 0x20000000,

GenericAll = 0x10000000

}

[Flags]

public enum EFileShare : uint

{

None = 0x00000000,

Read = 0x00000001,

Write = 0x00000002,

Delete = 0x00000004

}

public enum ECreationDisposition : uint

{

New = 1,

CreateAlways = 2,

OpenExisting = 3,

OpenAlways = 4,

TruncateExisting = 5

}

[Flags]

public enum EFileAttributes : uint

{

Readonly = 0x00000001,

Hidden = 0x00000002,

System = 0x00000004,

Directory = 0x00000010,

Archive = 0x00000020,

Device = 0x00000040,

Normal = 0x00000080,

Temporary = 0x00000100,

SparseFile = 0x00000200,

ReparsePoint = 0x00000400,

Compressed = 0x00000800,

Offline = 0x00001000,

NotContentIndexed = 0x00002000,

Encrypted = 0x00004000,

Write\_Through = 0x80000000,

Overlapped = 0x40000000,

NoBuffering = 0x20000000,

RandomAccess = 0x10000000,

SequentialScan = 0x08000000,

DeleteOnClose = 0x04000000,

BackupSemantics = 0x02000000,

PosixSemantics = 0x01000000,

OpenReparsePoint = 0x00200000,

OpenNoRecall = 0x00100000,

FirstPipeInstance = 0x00080000

}

}

}

Constants

namespace PipesInfrastructure

{

public static class Constants

{

public const uint PIPE\_ACCESS\_DUPLEX = 0x00000003;

public const uint PIPE\_TYPE\_BYTE = 0x00000000;

public const uint PIPE\_TYPE\_MESSAGE = 0x00000004;

public const uint PIPE\_WAIT = 0x00000000;

public const uint PIPE\_UNLIMITED\_INSTANCES = 255;

public const int NMPWAIT\_WAIT\_FOREVER = -1;

public const uint PIPE\_OPEN\_MODE = 0x00000003;

public const int MAILSLOT\_WAIT\_FOREVER = -1;

}

}