Dual Trace Communication Event Analysis

by

Huihui Nora Huang
B.Sc., Nanjing University of Aeronautics and Astronautic, 2003
M.Sc., Nanjing University of Aeronautics and Astronautic, 2006

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Dual Trace Communication Event Analysis

by

Huihui Nora Huang
B.Sc., Nanjing University of Aeronautics and Astronautic, 2003
M.Sc., Nanjing University of Aeronautics and Astronautic, 2006

Supervisory Committee

Dr. German. Supervisor Main, Supervisor
(Department of Same As Candidate)

Dr. M. Member One, Departmental Member
(Department of Same As Candidate)

Dr. Member Two, Departmental Member
(Department of Same As Candidate)

Dr. Outside Member, Outside Member
(Department of Not Same As Candidate)

Supervisory Committee

ABSTRACT

Contents

Supervisory Committee ii

Abstract iii

Table of Contents iv

List of Tables vi

List of Figures vii

Acknowledgements viii

Dedication ix

1 Introduction 1

2 Methodology 2

- 2.1 Define the Problem 2
- 2.2 Define the Scope 3
- 2.3 Obtain Background Knowledge 3
- 2.4 Obtain Background Knowledge 3

3 Channel Modeling 4

- 3.1 Assembly Calling Convention 4
- 3.2 Modeling 5
  - 3.2.1 Named pipes Channel 5
  - 3.2.2 MQMS Channel 7
  - 3.2.3 TCP/UDP Socket Channel 7
  - 3.2.4 HTTP Channel 7

4 Event Analysis Prototype 8

4.0.1 User Defined Communication Type 8
4.0.2 Communication Event Searching 13
4.0.3 Matching Event Visualization and Navigation 14
4.1 Some LaTeXExamples 14
4.1.1 Preamble 14
4.1.2 Document Body 15
4.2 How to Number Pages 15
4.3 How to Create a Title Page 15
4.4 How to Create an Abstract 16
4.5 How to Create a Table of Contents 16
4.6 How to Create Sections 16
4.7 How to Create a List 17
4.8 How to Insert Tables, Figures, Captions, and Footnotes 17
4.8.1 Tables 17
4.8.2 Figures 19
4.8.3 Captions 21
4.8.4 Footnotes 21
4.9 How to Alter Font 21
4.9.1 Type Style 21
4.9.2 Type Size 21
4.10 Math Mode 22
4.11 Formatting Extras 22

5 Experiments 24
5.0.1 Test and Verification Design 24
5.0.2 Result 24

6 Evaluation, Analysis and Comparisons 27

7 Conclusions 29

A Additional Information 30

Bibliography 31

List of Tables

Table 3.1 Functions for communication type definition of synchronous named pipe 6
Table 3.2 Functions for addition communication type definition of asynchronous named pipe 7

Table 4.1 Page Numbering Styles 15
Table 4.2 Table Example 18

List of Figures

Figure 2.1 Break one send/multiple receive into multiple send/receive events 2

Figure 4.1 Add function to a Communication type from Functions View . 10
Figure 4.2 Dialog to input information for a function adding to a communication type 11
Figure 4.3 New View: Communication Type View 12
Figure 4.4 Right Click menu to navigate to send and receive event in the traces 12

Figure 5.1 Defined clientsend and serversend communication types in Communication View 25
Figure 5.2 the search result of clientsend communication type 25
Figure 5.3 the search result of the serversend communication type 25
Figure 5.4 instruction view and memory view updated correctly 26

ACKNOWLEDGEMENTS

I would like to thank:

my cat, Star Trek, and the weather, for supporting me in the low moments.

Supervisor Main, for mentoring, support, encouragement, and patience.

Grant Organization Name, for funding me with a Scholarship.

I believe I know the only cure, which is to make one's centre of life inside of one's self, not selfishly or excludingly, but with a kind of unassailable serenity-to decorate one's inner house so richly that one is content there, glad to welcome any one who wants to come and stay, but happy all the same in the hours when one is inevitably alone.

Edith Wharton

DEDICATION

Just hoping this is useful!

Chapter 1

Introduction

Many network application vulnerabilities occur not just in one application, but in how they interact with other systems. These kinds of vulnerabilities can be difficult to analyze. Dual-trace analysis is one approach that helps the security engineers to detect the vulnerabilities in the interactive software. A dual-trace consist of two execution traces that are generated from two interacting applications. Each of these traces contains information including CPU instructions, register and memory changes of the running application. Communication information of the interacting applications is captured as the register or memory changes on their respective sides. Our research focuses on building a model to visualize and analyze dual-trace which would be available for various kinds of communication type.

Chapter 2

Methodology

2.1 Define the Problem

The major problem being considered in dual-trace analysis is locating the communication events in the assembly traces from both sides and present them to the user. In order to do so, we first, have to develop a method to locate those events in the traces, and second, develop an user interface for accessing the communication event information.

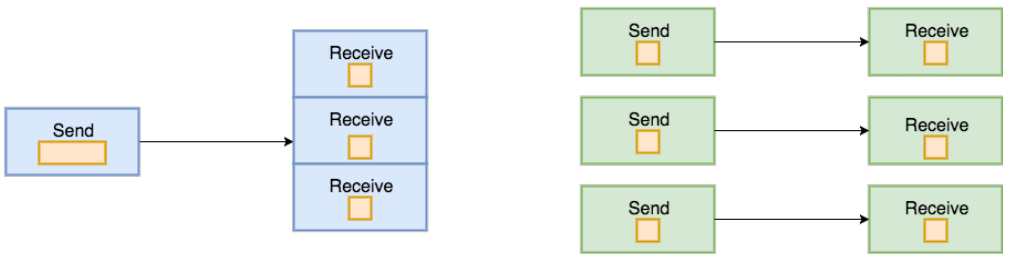


Figure 2.1: Break one send/multiple receive into multiple send/receive events

Before jumping into solving the problem we need to define what is a communication event. In this work, a communication event in the dual trace is defined as a successfully send and received message. There are four essential elements in this event: send function call, receive function call, the sent message in the sender's memory and the received message in the receiver's memory. However, in some cases, the send and receive operation are not always exactly paired. For example, the receiver buffer is small than the sender buffer, the receive function has to be called multiple times to receive one message from the sender. In this case, the one send V.S. multiple receives event is broken into multiple send V.S. receive events as shown in Figure 2.1.

2.2 Define the Scope

Named pipe, MAMQ, HTTP, TCP/UDP socket are targeted in this work since they are the most widely used ones in windows server/client service and are the ones used in the popular Window Communication Foundation.

2.3 Obtain Background Knowledge

In order to synchronize the messaging of both side of the traces, we need to investigate the communication methods to figure out how they looks like in the assembly level traces. There are so many communication methods exists in the real world and they are updating everyday. We are not covering all of them at a time, but start from narrowing down our study into some widely used communication methods. The built prototype is extendable to support other cases.

In these work, we had investigated the Windows APIs for setting up these communication channels and sending/receiving messages in these communication channels. By understanding these APIs, we modeled the communication channels in the assembly traces. And Then we build our prototype tool based on these models. In addition, the user can refer to the channel models to define the concerned communication type through the user interface we provide in the prototype tool. Defined communication type is later on used for locating the communication events in the dual-trace.

2.4 Model the Channels

2.5 Build the Prototype

2.6 Verify the Model and the Prototype Tool

2.7 Limitations

In this section, we specify the limitations of the current prototype and the reasons for them.

2.7.1 Event Status: Success or Fail

In current prototype, we only consider the success cases. For the Fail case, since the message was not successfully sent or received, there are high chance that they are not existed in the memory of the trace. From the assembly level trace, if the message was not traced in the memory, there is no way to match the sent/received message pair in the trace analysis.

2.7.2 Match Events Distinguishing

Distinguishing is considered when multiple clients connecting to the same server. Each connection is considered as an instance. In the server side all this instances have the same pipe name but different handler ID. However in the assembly trace level there is no way to match a client with it instance handler ID. In consequence, if the same content messages are being sent/received by different clients, when the user want to match the message pair between a client and the server, there is no way to distinguish the correct one from the assembly trace level. As a result, our tool will list all the matched content message event, regardless if it's from the interested client. The user can distinguish the correct ones for this client, if they have extra information.

2.7.3 Match Events Ordering

Ordering is considered when multiple messages with exactly the same content being send/receive between the client and server. If the channel is synchronous, the order of the event is always consist with the order they happen in both the sender and receive sides. However for the asynchronous channel, there are chance that the sent messages in the sender side's trace are out of order with the received messages in the received side's trace. Unfortunately, There is no way in the assembly level trace to match the exactly ones. As a result, our tool can only order the event based on the order they happen in the traces.

2.7.4 Buffer Sizes Of Sender and Receiver Mismatch

In current prototype, we only consider the success cases. For the Fail case, since the message was not successfully sent or received, there are high chance that they are not existed in the memory of the trace. From the assembly level trace, if the message was not traced in the memory, there is no way to match the sent/received message pair in the trace analysis.

Chapter 3

Channel Modeling

In this section we model four different communication channels based on the windows APIS. The modeling is simplified as much as possible to only get the enough information for the communication event locating. Necessary Assembly calling conventions will be introduced in the section for the purpose of understanding the modeling.

3.1 Assembly Calling Convention

Before we jumping into a specific communication channel, it is important to know some basic assembly calling convention. Calling Convention is different for operating system and the programming language. Since we are looking into the messaging methods being used in windows communication framework, and since our case study is running on a Microsoft* x64 system, we only list the Microsoft* x64 calling convention for interfacing with C/C++ style functions:

1. RCX, RDX, R8, R9 are used for integer and pointer arguments in that order left to right.
2. XMM0, 1, 2, and 3 are used for floating point arguments.
3. Additional arguments are pushed on the stack left to right. . . .
4. Parameters less than 64 bits long are not zero extended; the high bits contain garbage.
5. Integer return values (similar to x86) are returned in RAX if 64 bits or less.

6. Floating point return values are returned in XMM0.
7. Larger return values (structs) have space allocated on the stack by the caller, and RCX then contains a pointer to the return space when the callee is called. Register usage for integer parameters is then pushed one to the right. RAX returns this address to the caller.

3.2 Modeling

We model four base communication channels used in Windows operating system in this section. They are Named pipe, MQMS, TCP/UDP socket and HTTP channels

3.2.1 Named pipes Channel

A named pipe is a named, one-way or duplex pipe for communication between the pipe server and one or more pipe clients. All instances of the named pipe share the same pipe name, but each instance has its own buffers and handlers. In here we only consider one to one server/client pairs. One server to multiple clients scenario can always be broken into multiple server/client pairs. To locate a named pipe message event in dual-trace, we need to know how the channels are created as well as how the messages are send and received in the assembly traces. The creation of a named pipe will return the handler of that pipe. This handler will be used later on when messages are being sent or received. So we need to know the function calls for named pipe creation and message send/receive to locate the event in the traces. In the follow subsections, we will list the related functions for the named pipe channel for both synchronous mode and asynchronous mode. The create channel functions for both modes are the same but with different input parameters. The functions for send and receive message are also the same for both case. However, the operation of the send and receive functions are different for different mode. In addition, extra function are being called to check the status of message sending or receiving in asynchronous mode as well as the message content.

Synchronous

We list all the functions that needed to locate an messaging event in a dual-trace in Table3.2 for synchronous named pipe. The Channel Create Functions indicate how the channel being created in server and client sides, and the mattered parameters. For named pipes the channel create functions are different between in server and client. The parameter in RDX can indicate if the channel is opened as synchronous mode. The send or receive message functions are the same in server and client. When the channel is being created, the input file names for a channel at server and the client are the same, but the returned File Handler IDs are different. The send and receive function only use the handler to send and receive message to a specific channel.

Table 3.1: Functions for communication type definition of synchronous named pipe

	Channel Create Functions		Message Send Functions		Message Receive F	
	Function	Parameters	Function	Parameters	Function	Paramete
Sever	Create-NamedPipe	RAX: File Handler	WriteFile	RCX: File Handler	ReadFile	RCX: Fil
		RCX: File Name		RDX: Buffer Address		RDX: Buffer Ad
		RDX: Asyn/Syn				
Client	CreateFile	RAX: File Handler		R8: Buffer Length		R8: Buffe
		RCX: File Name		Stack: Overlap Pointer		Stack: Overlap
		RDX: Asyn/Syn				

Asynchronous

The functions used in Asynchronous mode for create channel, send and receive message are the same as those used in synchronous mode. However, the ReadFile and WriteFile functions run asynchronously when the channel is asynchronous. This means the function will return immediately, even if the operation has not been completed. If the operation is complete when the function returns, the return value indicates the success or failure of the operation. Otherwise the functions return zero and GetLastError returns ERROR_IO_PENDING. In this case, the calling thread must wait until the operation has finished. The calling thread must then call the GetOverlappedResult function to determine the results. This means besides looking for ReadFile and WriteFile function calls in the traces, the GetOverlappedResult function should be checked in the traces to get the full result of the ReadFile or WriteFile operations. Table**??** list the interested parameters of the GetOverlappedResult funciton. However, it's complicated to define the communication types with so many functions for only an event. In order to make the user interface clear and simple. We ask the user to make separate definition for the follow function with the send function as a new communication type. Table **??** shows the addition communication type for asynchronous mode.

Table 3.2: Functions for addition communication type definition of asynchronous named pipe

	Channel Create Functions		Message Send Functions		Message Receive F	
	Function	Parameters	Function	Parameters	Function	Pa
Sever	Create-NamedPipe	RAX: File Handler	WriteFile	RCX: File Handler	GetOverlapped-Result	R Fi
		RCX: File Name		RDX: Buffer Address		
		RDX: Asyn/Syn				
Client	CreateFile	RAX: File Handler		R8: Buffer Length		R O st
		RCX: File Name		Stack: Overlap Pointer		
		RDX: Asyn/Syn				

3.2.2 MQMS Channel

Synchronous

Asynchronous

3.2.3 TCP/UDP Socket Channel

3.2.4 HTTP Channel

Chapter 4

Event Analysis Prototype

In this section we discuss the design of the prototype of dual-trace analysis. This prototype consist of three main components: user interface for defining the communication type, algorithm of locating the communication events in the dual-trace, user interface and strategy to navigate the located events to the sender and receiver traces. We provide the background information of the design of each component as well as their detail design in each corresponding subsection.

4.0.1 User Defined Communication Type

In our design, we don't specify any predefined communication type but give the user ability to do that. By the user interface implemented, the user can defined their own communication type. This give the flexibility to the user to define what they are looking for. Each communication type consist of 4 system function calls. They are channel create/open in sender and receiver sides, sender's send message function and receiver's receive message function. By indicating the channel create/open functions in both sender and receiver sides, the tool can acquire the channel's identifiers. Later on the tool can match the send and received messages within a specific channel. The send and receive functions are used to located the event happened in the traces. The messages sent and received are reconstructed from the memory state when the send and receive functions are called and returned. The detail of the match algorithm will be discuss later.

Function Calls in the Traces

The called functions' name can be inspected by search of the symbolic name in the executable binary or any DLLs which used by the program at the time when it is traced. This functionality exists in the current Atlantis. By importing the DLLs and execution executable binary, Atlantis can list all called functions for the users in the Functions view. From this list, users can chose the interested functions and generate their interested communication type. In Figure4.1 there is an action item "Add to Communication type" in the right click menu of the function entry. Figure 4.2 shows the dialogue for entering the information for the adding function. As this figure shows, users can get the existing communication type list in the drop down menu. They can choose to add the current function to an exist communication type or they can add it to a new communication type by entering a new name. For the channel create/open function, the register holding the address of channel's name as input and the register holding the handle identification of the channel as output are required. For the send/receive function, the register holding the address of the send/receiver buffer, the register holding the length of the sending/receiving message and the register holding the channel's identification are required. As there are 4 functions for each communication type users have to repeat this add function to communication type action for 4 times to generate one communication type.

Communication Type Data Structure

The defined communication type will be stored in a xml file. The list below shows the data structure of one communication type.

```
<messageTypesData>
    <parentFolder>.tmp</parentFolder>
    <messageTypes>
        <messageType>
            <name>namedPipe_clientsend</name>
            <sendFunction>
                <associatedFileName>Client</associatedFileName>
                <name>WriteFile</name>
                <messageAddress>RDX</messageAddress>
                <messageLengthAddress>R8</messageLengthAddress>
                <channelIdReg>RCX</channelIdReg>
```

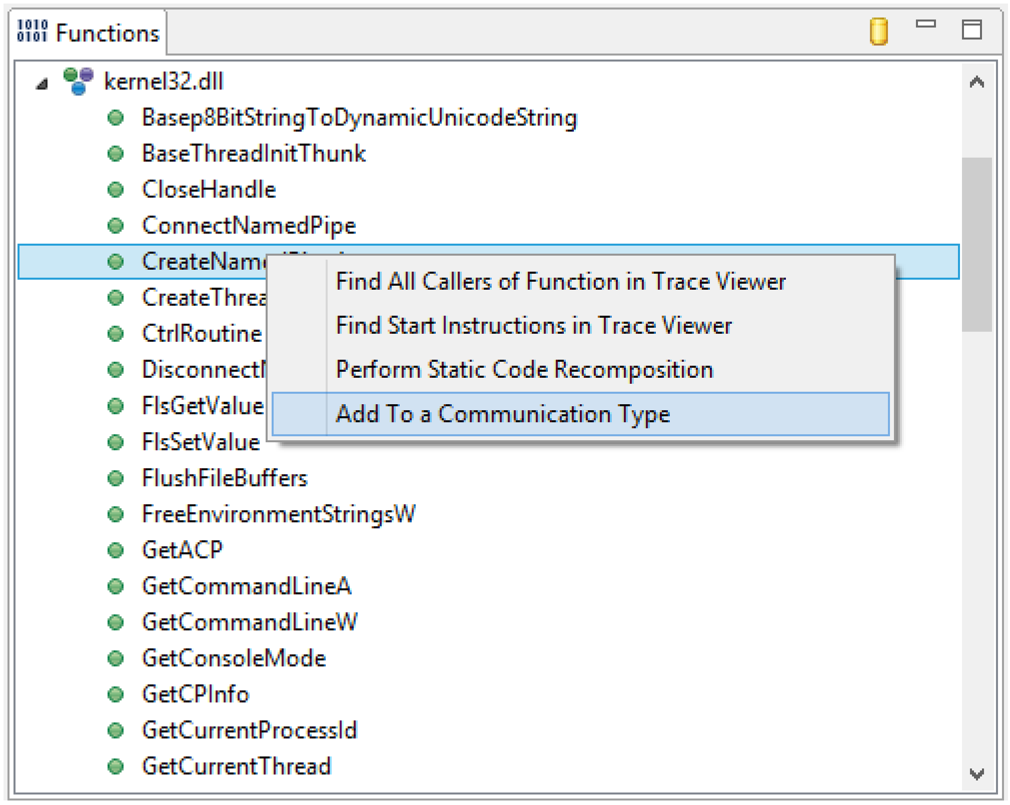
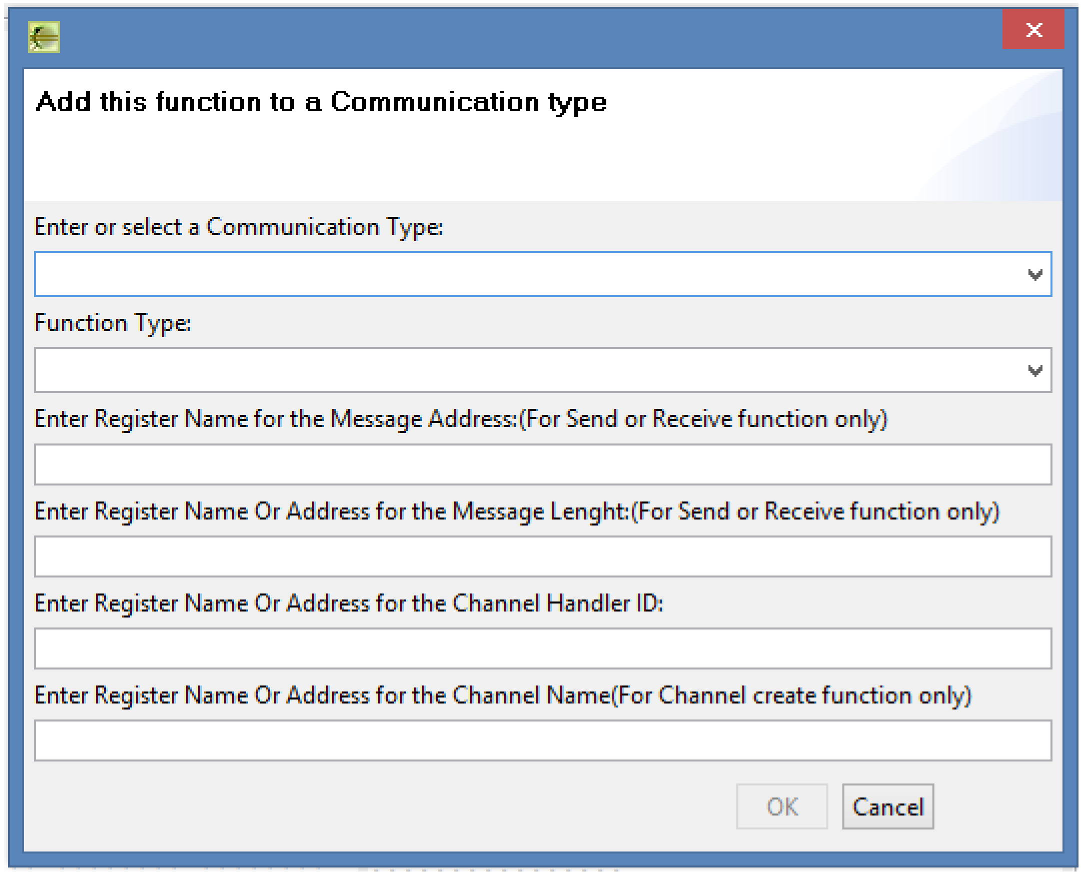


Figure 4.1: Add function to a Communication type from Functions View

```
</sendFunction>
<receiveFunction>
    <associatedFileName>Server</associatedFileName>
    <name>ReadFile</name>
    <messageAddress>RDX</messageAddress>
    <messageLengthAddress>R8</messageLengthAddress>
    <channelIdReg>RCX</channelIdReg>
</receiveFunction>
<sendChannelCreateFunction>
    <associatedFileName>Client</associatedFileName>
    <name>CreateFileA</name>
    <channelIdReg>RAX</channelIdReg>
    <channelNameAddress>RCX</channelNameAddress>
</sendChannelCreateFunction>
<receiveChannelCreateFunction>
```



Add this function to a Communication type

Enter or select a Communication Type:

Function Type:

Enter Register Name for the Message Address:(For Send or Receive function only)

Enter Register Name Or Address for the Message Lenght:(For Send or Receive function only)

Enter Register Name Or Address for the Channel Handler ID:

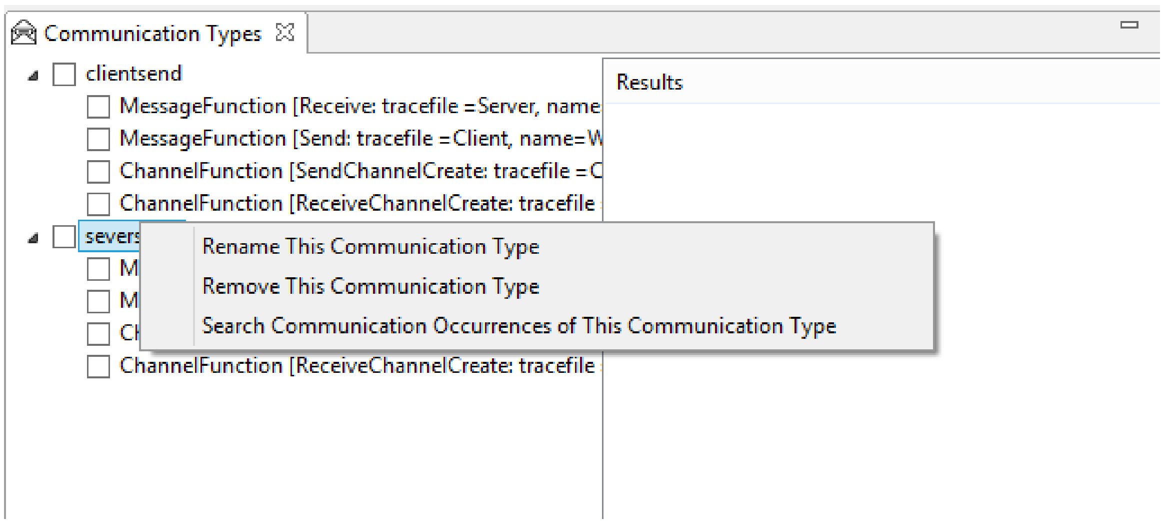
Enter Register Name Or Address for the Channel Name(For Channel create function only)

Figure 4.2: Dialog to input information for a function adding to a communication type

```
                <associatedFileName>Server</associatedFileName>
                <name>CreateNamedPipeA</name>
                <channelIdReg>RAX</channelIdReg>
                <channelNameAddress>RCX</channelNameAddress>
            </receiveChannelCreateFunction>
        </messageType>
    </messageTypes>
</messageTypesData>
```

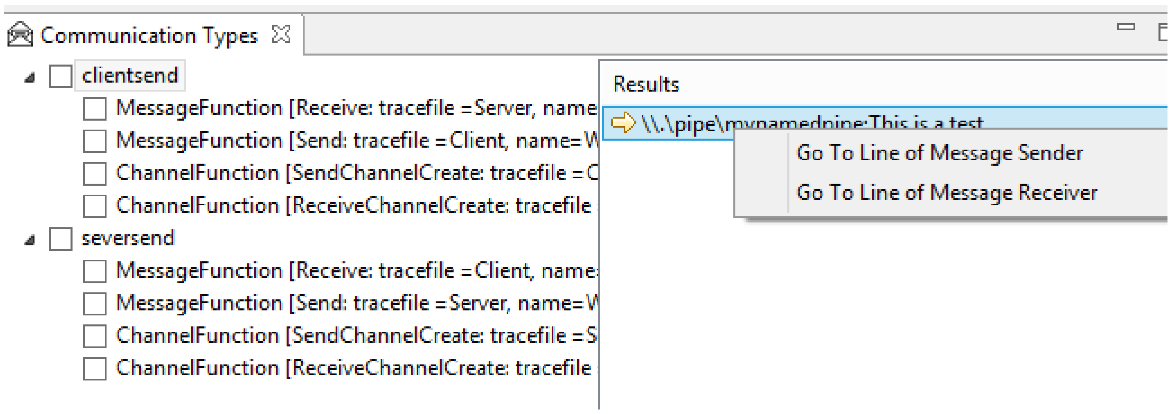
Communication Type View

A new view named Communication Types view is for the user defined communication types. All user defined communication type are stored in the .xml file and listed in communication type view when it's opened as shown in Figure 4.3. User can change the name of a communication type, remove an existing communication type or searching of the match message occurrences of selected communication type by selecting action item in the right click menu of an communication type entry. The matched messages are listed in the result window of the view. By clicking the entry of the search result, user can navigate to it's sender or receiver's corresponding instruction line as shown in Figure4.4. Message content in the memory view will be shown as well.


Communication Types
Results
Rename This Communication Type
Remove This Communication Type
Search Communication Occurrences of This Communication Type

clientsend
MessageFunction [Receive: tracefile =Server, name
MessageFunction [Send: tracefile =Client, name=W
ChannelFunction [SendChannelCreate: tracefile =C
ChannelFunction [ReceiveChannelCreate: tracefile
severs
M
M
Ch
ChannelFunction [ReceiveChannelCreate: tracefile

Figure 4.3: New View: Communication Type View



clientsend
MessageFunction [Receive: tracefile =Server, name
MessageFunction [Send: tracefile =Client, name=W
ChannelFunction [SendChannelCreate: tracefile =C
ChannelFunction [ReceiveChannelCreate: tracefile
seversend
MessageFunction [Receive: tracefile =Client, name
MessageFunction [Send: tracefile =Server, name=W
ChannelFunction [SendChannelCreate: tracefile =S
ChannelFunction [ReceiveChannelCreate: tracefile

\\.\pipe\mynamedpipe:This is a test

Figure 4.4: Right Click menu to navigate to send and receive event in the traces

4.0.2 Communication Event Searching

The communication event consists of the send message event in the sender side and receive message event in the receiver side. The communication event searching algorithm can be divided into three main steps: 1. search all channel create/open event in the sender and receiver side, save the handle id and corresponding channel name. 2. Search all message send and receive event in sender and receiver sides. 3. Matching the send/receive messages pair based on the channel names and message contents.

Record opened Channel

In this step the algorithm is supposed to search all the open channel both in the sender and receiver side. The found created channels are recorded in a map. The key of the map is the handler id of the channel and the value is the channel name. A channel in the sender and receiver sides will have different handler id but same channel name.

Search send and receive Message

All send message and receive message function calls will be found out in the trace. When a send function hit, the memory state of the hit instruction line will be reconstructed, and the message content can be get from the memory with the send message buffer address. When a receive function hit, the return line of that function is needed for getting the message content. The memory state of the function return line is reconstructed and the message content can be get from the reconstructed memory state with the receive message buffer address.

Matching the send/receive messages pair

After the created channel and send/receive message are found out in the sender and receiver side, a matching algorithm is used to match the send/receive message pairs.

Matching Event Data Structure

The matching event is stored in cache when the tool is running. Only the most recent search result is cached currently. If users need the previous result, they need to apply the search again. The matching Event consist of two sub-events, one is message send event while the other is message receive event. Both of these two sub-events are object of BfvFileMessageMatch. BfvFileMessageMatch is an Java class extends org.eclipse.search.internal.ui.text.FileMatch. FileMatch class containing the information needed to navigate to the trace file editor. In order to show the corresponding send/receive message in the memory view, the target memory address storing the message content is set in BfvFileMessageMatch. Two more elements: message and channel name are also set in BfvFileMessageMatch which are listed in the search result.

4.0.3 Matching Event Visualization and Navigation

The right click menu of an entry in the search result list has two action items: Go To Line of Message Sender and Go To Line of Message Receiver. Both of the action items allow users to navigate to the trace Instruction view. When the user click on these items, it will navigate to the corresponding trace sender or receiver trace instruction view. Meanwhile the memory view jumps to the target address of the message buffer, and the memory state is reconstructed so that the message content in that buffer will be shown in the memory view.

4.1 Some LaTeXExamples

A Latex document is composed of two parts: the Preamble, and the Document Body. The *Preamble* is the site for inclusion of all document set up commands: definition of new commands, inclusion of prebuilt packages, template declaration, etc. The *Body* is where the document content is placed.

4.1.1 Preamble

The Preamble refers to the input which precedes the documents contents. It is the area where the author determines the general template for the document using the \documentclass[*options*]{*doc style*} command. For example, \\documentclass[11*pt*]{*article*} declares that a document will follow the *article* document class, and have 11pt font.

If the document requires support of any library packages they must be included in the preamble using the \usepackage{*package name*} command. For example, \\usepackage{graphicx} is the command needed to include the graphicx package.

4.1.2 Document Body

The document body is the area which follows the Preamble. It is defined by the \begin{*document*} and \end{*document*} commands. The content of a Latex document is declared in the document body. Input which appears after the \end{document} command is ignored.

4.2 How to Number Pages

To number the pages of a document use the \pagenumbering{*style*} command. Numbering is defined in the documents preamble. There are several different *styles* to choose from.

Numbering Style	**Output**
\pagenumbering{arabic}	1, 2, 3, ...
\pagenumbering{roman}	i, ii, iii, ...
\pagenumbering{alph}	a, b, c, ...
\pagenumbering{Roman}	I, II, III, ...
\pagenumbering{Alph}	A, B, C, ...

Table 4.1: Page Numbering Styles

The numbering of pages for a thesis is, however, much more complex than for an article and, in fact, the *book* class has been adopted. Make changes to those settings only if you are really familiar with LaTeX.

4.3 How to Create a Title Page

A title page can be either on a separate page or integrated directly into the first page of the document. It is defined by three declarations, followed by the \maketitle command as illustrated below.

```
\title{Title of Paper}
\author{Author(s) of Paper}
\date{Publication Date}
\maketitle
```

The article document class defaults on an integrated title page. To make a separate title page, use the **titlepage** option with the \documentclass[*titlepage*]{*doc style*} command.

For this thesis style the title page has been completely formatted for you. Just insert the various names of people in the supervisory committee, the title, your name and so on in the location where the *dummy* entries exist right now and you will be done. I would suggest to avoid doing any other changes unless you are absolutely sure!

4.4 How to Create an Abstract

To create an abstract, place contents of abstract between the \begin{*abstract*} and \end{*abstract*} commands.

4.5 How to Create a Table of Contents

The \tableofcontents command automatically generates a table of contents from all section headers. The default behavior for the article document class is to produce an integrated table of contents. However, the document can be altered to generate the table of contents on a separate page using the \newpage command (see section Formatting Extras).

For this thesis template a special command has been added, namely the \textTOCadd. You can find it in the file *macros/style.tex*. It has to be explicitly called for an insertion into the Table of Contents and it is already in place appropriately for the existing sections and subsections.

4.6 How to Create Sections

Creating sections, subsections, and subsubsections is completed using the \section{Section Name}, \subsection{Subsection Name} and \subsubsection{Subsubsection Name} commands, respectively. Each sectional division is numerically labeled with respect to it's placement in the section hierarchy. For example, this section was defined with the code:

```
\section{How to Create Sections}
Creating sections, subsections, and ...
```

It is useful to give a label using the **\label** command to a section or subsection if a reference to it is made, so that the reference will be automatically updated should the structure of the document change.

4.7 How to Create a List

Lists can be either enumerated, non enumerated, or descriptive. Each element of a list is termed an 'item'.

1. enter the list environment with the \begin{*list style*} command.
2. define each item with the \item command for non\enumerated lists, or \item[*label*] for descriptive lists.
3. terminate list environment with the \end{*list style*} command.

4.8 How to Insert Tables, Figures, Captions, and Footnotes

The table and figure environments contain input blocks which cannot be split across pages. Rather than divide the input of either of these environments, the contents are relocated, or floated, to a location in the document which optimizes page layout with the surrounding document content.

4.8.1 Tables

Tables are created in the tabular environment. A single parameter is used to define the number of columns and item justification pertaining to each column. The single parameter is a combination of the following ones shown in Table 4.2.

\\ and & are used to define rows and columns, respectively. A table can either have the contents of its rows and columns lined or not. Each line used to construct the table must be individually specified, using | and \hline for vertical and horizontal lines, respectively.

Table 4.2 was generated with the following input:

loc	**Purpose**
l	left justified column
r	right justified column
c	centered column
|	vertical rule

Table 4.2: Table Example

```
\begin{center}
    \begin{tabular}{|l|l|} \hline
    l & left justified column   \\ \hline
    r & right justified column  \\ \hline
    c & centered column         \\ \hline
    $|$ & vertical rule         \\ \hline
 \end{tabular}
 \end{center}
```

You will want to include your table in the "List Of Tables" section at the beginning of your thesis. To do this you enclose the above table inside a table environment like so:

```
\begin{table}
    \begin{center}
    ...
    \end{center}
    \caption{Sentence describing table.}
    \label{unique:label}
\end{table}
```

The caption is the text that appears underneath the table. It should be short and precise. The label is a unique label that you can use to refer to the table within your document. You can use the `\ref{label}` to insert the table number into your text as in Table 4.2. In the example above you would use as in:

```
I am referring to Table \ref{unique:label}.
```

4.8.2 Figures

The first step to including an externally prepared image into a document, is to declare the graphixs package into the documents preamble. Integrating the image can be done using the figure environment. Enter and exit the figure environment with the **\begin{figure}**[*loc*] and **\end{figure}** commands, respectively. The *loc* dictates the placement of the included image, and can be any of the following:

h here: location in text where the environment appears

t top: top of the page

b bottom: bottom of the page

p page of floats: on a separate page with no text

For organizational purposes, it is best to have keep all figures in a folder together. I usually label the folder as "*Figures*" (with great creativity) and I placed it in the same directory as the topmost main *.tex* file. Include the image into the document with the \includegraphics[*dim*]{path to image} command. *dim* dictates the magnitude of the `height` or `width`. The image is scaled proportionally. An example and its resulting output follow below.

```
\begin{figure}[h]
\centering
\includegraphics[height = 1in]{LinuxPenguin.eps}
\caption{The Linux Penguin}
\end{figure}
```

Why is the output for the figure not shown? Because inserting figures into LaTeXis not that simple and it is highly dependent on the system you are using together with the type of figure. This is not the place to dwell upon the inconsistencies which can make your life difficult. Suffice it to say that the original LaTeXand its tools was geared to accept *.eps* files for figures and it still maintains that expectation if one compiles using a *Latex to dvi to (pdf or ps)* series of commands. On the other hand, if one uses the *Latex to pdf* direct path, then files of other types are perfectly fine (e.g. *pdf, jpg, gif, etc.*).

If you are interested, look at the actual file for this section namely "sec_latexhelp.tex" and consider the set of lines commented out just above this paragraph. There are two examples of insertion of figures, the first with the *.eps* version and the second with the *.pdf* version of the same picture (of a penguin). Delete the comments from one of the two sets and use the appropriate tools.

To refer to a figure, the same approach used for tables should be used, namely a `\ref{label}` command which includes the unique identifier label for that figure, as in:

```
I am referring to Figure \ref{unique:label}.
```

4.8.3 Captions

Captions for tables and figures are created using the **caption**{caption goes here}. Captions are automatically numbered with separate counters for tables and figures. **caption**{caption contents} can only be used in the Figure or Table environment.

4.8.4 Footnotes

Footnotes are inserted with the **footnote**{footnote contents} command. This footnote[1] is generated as follows:

```
...This footnote\footnote{this is a footnote} is generated...
```

4.9 How to Alter Font

4.9.1 Type Style

Roman Family is the default type style. The types style can be modified using the following commands.

Command	**Output**
\textit{Italic Characters}	*Italic Characters*
\textsl{Slanted Chartacters}	*Slanted Characters*
\textsc{Small Cap Characters}	SMALL CAP CHARACTERS
\textbf{Boldface characters}	**Boldface characters**
\textsf{Sans Serif Characters}	Sans Serif Characters
\texttt{Typewriter Characters}	`Typewriter Characters`

4.9.2 Type Size

The font size can be modified using the following commands.

[1] this is a footnote

Command	**Output**
`\tiny{tiny font}`	tiny font
`\scriptsize{scriptsize font}`	scriptsize font
`\small{small font}`	small font
`\normalsize{normalsize font}`	normalsize font
`\large{large font}`	large font
`\Large{Large font}`	Large font
`\huge{huge font}`	huge font
`\Huge{Huge font}`	Huge font

4.10 Math Mode

To incorperate mathematical content into a document, Latex provides three different environments: Displaymath, Math, and Equation. Brief descriptions for each environment, and environment short cuts are displayed in the table below.

Environment	**Function**	**Shortcut**
`math`	`displays an in-text formula`	`\ ( ...\ )`
`displaymath`	`displays an unnumbered formula`	`\ [ ...\ ]`
`equation`	`displays a numbered formula`	`N/A`

The following examples, using Einstein's famous $e \doteq mc^2$ equation, illustrate how to include a formula into a document.

```
...Einstein's famous \( e \doteq mc^{2} \) equation, illustrate...
```

```
\[e \doteq mc^{2}\]
```

$$e \doteq mc^2$$

```
\begin{equation}
   \doteq mc^{2}
\end{equation}
```

$$e \doteq mc^2 \tag{4.1}$$

4.11 Formatting Extras

The following table illustrates some formatting tips for perfecting the layout of a Latex document.

Command	**Purpose**
\hspace{*len*}	insert a horizontal space of length *len*
\vspace{*len*}	insert a vertical space of length *len*
\mbox{*text*}	ensure that *text* is not split over multiple lines
\\	new line
\newpage	start new page
\pagebreak	insert a page break
%	precedes comments

Chapter 5

Experiments

The case we used to test this prototype contains one named pipe synchronous channel between a server and a client. Client send a message to the server and server reply another message to the client.

5.0.1 Test and Verification Design

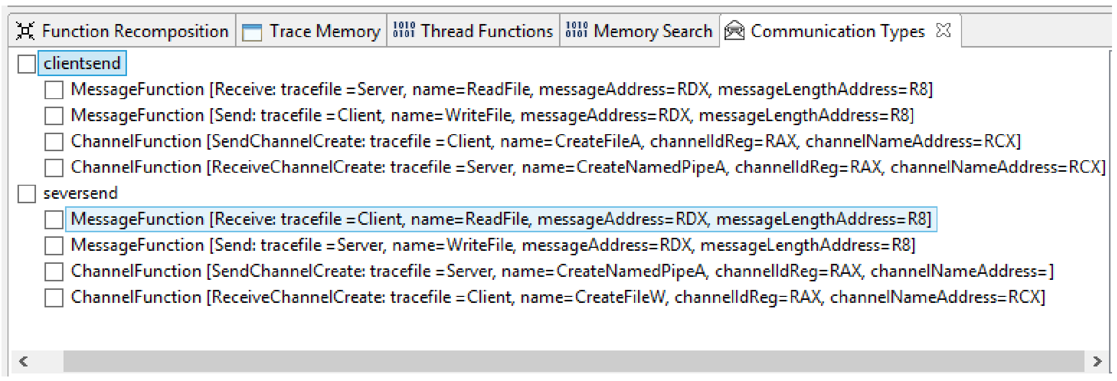
The test cases are designed to find all the messages from client to server and all the messages from server to client. Two end to end test cases are designed for both scenarios.

In each test case, there are three test steps: 1. define the communication type by adding channel creating functions and message send/receive functions of server and client sides. 2. search for the events of the defined communication type. 3. for the occurrence of the events, navigate to the trace instruction and memory view.

Verification points are specified for each step as: 1. verify the communication types with their functions are listed in the communication view. 2. verify the message events in the dual-trace can be found and listed in the search result view. 3. verify the navigation from the result entry to the instruction view of sender trace and receiver trace.

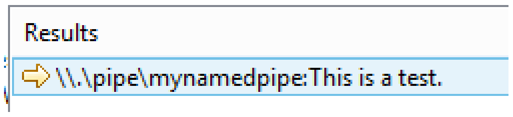
5.0.2 Result

We used the dual-trace provided by DRDC and follow the experiment and verification design to conduct this test. Figure5.1 shows that the user defined clientsend and serversend communication types are shown in the communication type view as well as the functions consist of the communication types. Figure5.3 shows the search result of clientsend communication type, while Figure5.3 shows the search result of the serversend communication type. By clicking the Go To Line of Message Sender and Go To Line of Message Receiver action items, instruction view and memory view updated correctly. Figure5.4 shows the server was sending out a message: This is an answer.



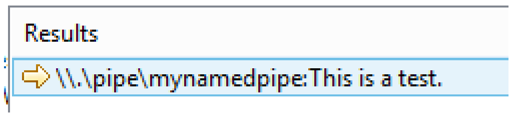
clientsend
MessageFunction [Receive: tracefile =Server, name=ReadFile, messageAddress=RDX, messageLengthAddress=R8]
MessageFunction [Send: tracefile =Client, name=WriteFile, messageAddress=RDX, messageLengthAddress=R8]
ChannelFunction [SendChannelCreate: tracefile =Client, name=CreateFileA, channelIdReg=RAX, channelNameAddress=RCX]
ChannelFunction [ReceiveChannelCreate: tracefile =Server, name=CreateNamedPipeA, channelIdReg=RAX, channelNameAddress=RCX]
seversend
MessageFunction [Receive: tracefile =Client, name=ReadFile, messageAddress=RDX, messageLengthAddress=R8]
MessageFunction [Send: tracefile =Server, name=WriteFile, messageAddress=RDX, messageLengthAddress=R8]
ChannelFunction [SendChannelCreate: tracefile =Server, name=CreateNamedPipeA, channelIdReg=RAX, channelNameAddress=]
ChannelFunction [ReceiveChannelCreate: tracefile =Client, name=CreateFileW, channelIdReg=RAX, channelNameAddress=RCX]

Figure 5.1: Defined clientsend and serversend communication types in Communication View



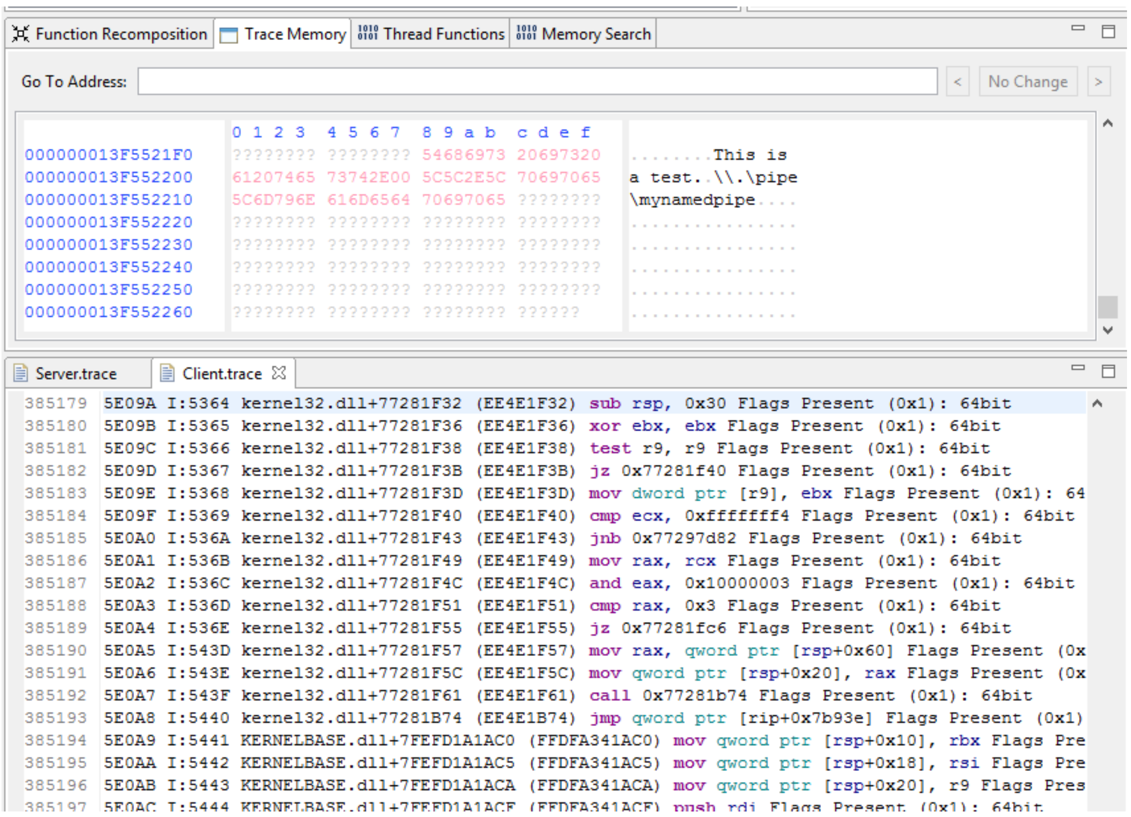
Results
\\.\pipe\mynamedpipe:This is a test.

Figure 5.2: the search result of clientsend communication type



Results
\\.\pipe\mynamedpipe:This is a test.

Figure 5.3: the search result of the serversend communication type



```
                  0 1 2 3  4 5 6 7  8 9 a b  c d e f
000000013F5521F0  ???????? ???????? 54686973 20697320   ........This is
000000013F552200  61207465 73742E00 5C5C2E5C 70697065   a test..\\.\pipe
000000013F552210  5C6D796E 616D6564 70697065 ????????   \mynamedpipe....
000000013F552220  ???????? ???????? ???????? ????????   ................
000000013F552230  ???????? ???????? ???????? ????????   ................
000000013F552240  ???????? ???????? ???????? ????????   ................
000000013F552250  ???????? ???????? ???????? ????????   ................
000000013F552260  ???????? ???????? ???????? ??????     ................
```

```
5E09A I:5364 kernel32.dll+77281F32 (EE4E1F32) sub rsp, 0x30 Flags Present (0x1): 64bit
5E09B I:5365 kernel32.dll+77281F36 (EE4E1F36) xor ebx, ebx Flags Present (0x1): 64bit
5E09C I:5366 kernel32.dll+77281F38 (EE4E1F38) test r9, r9 Flags Present (0x1): 64bit
5E09D I:5367 kernel32.dll+77281F3B (EE4E1F3B) jz 0x77281f40 Flags Present (0x1): 64bit
5E09E I:5368 kernel32.dll+77281F3D (EE4E1F3D) mov dword ptr [r9], ebx Flags Present (0x1): 64
5E09F I:5369 kernel32.dll+77281F40 (EE4E1F40) cmp ecx, 0xfffffff4 Flags Present (0x1): 64bit
5E0A0 I:536A kernel32.dll+77281F43 (EE4E1F43) jnb 0x77297d82 Flags Present (0x1): 64bit
5E0A1 I:536B kernel32.dll+77281F49 (EE4E1F49) mov rax, rcx Flags Present (0x1): 64bit
5E0A2 I:536C kernel32.dll+77281F4C (EE4E1F4C) and eax, 0x10000003 Flags Present (0x1): 64bit
5E0A3 I:536D kernel32.dll+77281F51 (EE4E1F51) cmp rax, 0x3 Flags Present (0x1): 64bit
5E0A4 I:536E kernel32.dll+77281F55 (EE4E1F55) jz 0x77281fc6 Flags Present (0x1): 64bit
5E0A5 I:543D kernel32.dll+77281F57 (EE4E1F57) mov rax, qword ptr [rsp+0x60] Flags Present (0x
5E0A6 I:543E kernel32.dll+77281F5C (EE4E1F5C) mov qword ptr [rsp+0x20], rax Flags Present (0x
5E0A7 I:543F kernel32.dll+77281F61 (EE4E1F61) call 0x77281b74 Flags Present (0x1): 64bit
5E0A8 I:5440 kernel32.dll+77281B74 (EE4E1B74) jmp qword ptr [rip+0x7b93e] Flags Present (0x1)
5E0A9 I:5441 KERNELBASE.dll+7FEFD1A1AC0 (FFDFA341AC0) mov qword ptr [rsp+0x10], rbx Flags Pre
5E0AA I:5442 KERNELBASE.dll+7FEFD1A1AC5 (FFDFA341AC5) mov qword ptr [rsp+0x18], rsi Flags Pre
5E0AB I:5443 KERNELBASE.dll+7FEFD1A1ACA (FFDFA341ACA) mov qword ptr [rsp+0x20], r9 Flags Pres
5E0AC I:5444 KERNELBASE.dll+7FEFD1A1ACF (FFDFA341ACF) push rdi Flags Present (0x1): 64bit
```

Figure 5.4: instruction view and memory view updated correctly

Chapter 6

Evaluation, Analysis and Comparisons

For a Master's research this chapter represents the critical part where **you** are truly evaluated to determine whether you should be given your degree. Even more so for a PhD. Consider carefully what the University calendar states regarding the expectations for a master's thesis, paraphrased here.

1. *A Masters thesis is an original lengthy essay.* The main implication here is that the essay is original, that is, it is completely newly written by you and does not contain any writings from others unless precisely quoted. Any paraphrased items must be cited.

2. *It must demonstrate that:*

   - students understand research methods;
   - students are capable to employ research methods;
   - students demonstrate command of the subject.

3. *The work may be based on:*

   - original data;
   - original exercise from scholarly literature;
   - data by others.

4. *The work must show that:*

- appropriate research methods have been used;
- appropriate methods of critical analysis supplied.

5. *The work must contain:*
    - evidence of some new contribution;
    - evidence of a new perspective on existing knowledge.

Only the last point uses the attribute *new* and it refers almost entirely to giving a new perspective and analysis, even if based on data from others. This truly implies that this current chapter on evaluation and analysis of results is the most important and must be written with care. You are demonstrating here that, even if given data and methods from others, your skills of critical judgment and analysis are now at the level that you can give professional evaluations.

Things are slightly different for a PhD. According to the Graduate Calendar:
a doctoral dissertation must embody original work and constitute a significant contribution to knowledge in the candidate's field of study. It should contain evidence of broad knowledge of the relevant literature, and should demonstrate a critical understanding of the works of scholars closely related to the subject of the dissertation. Material embodied in the dissertation should, in the opinion of scholars in the field, merit publication.

The general form and style of dissertations may differ from department to department, but all dissertations shall be presented in a form which constitutes an integrated submission. The dissertation may include materials already published by the candidate, whether alone or in conjunction with others. Previously published materials must be integrated into the dissertation while at the same time distinguishing the student's own work from the work of other researchers. At the final oral examination, the doctoral candidate is responsible for the entire content of the dissertation. This includes those portions of co-authored papers which comprise part of the dissertation.

The second paragraph makes it clear that one must emphasize what is new and different from others, without arrogance, yet without being too subtle either. The first paragraph implies that for a PhD it is required that one approached an important open problem and gave a new solution altogether, making chapters 3, 4, 5 all part of the body of research being evaluated. In fact at times even the problem may be entirely new, thus including chapter 2 in the examination. This is in contrast to a Master's degree where the minimum requirement is for chapter 5 to be original.

Chapter 7

Conclusions

My first rule for this chapter is to avoid finishing it with a section talking about future work. It may seem logical, yet it also appears to give a list of all items which remain undone! It is not the best way psychologically.

This chapter should contain a mirror of the introduction, where a summary of the *extraordinary* new results and their wonderful attributes should be stated first, followed by an executive summary of how this new solution was arrived at. Consider the practical fact that this chapter will be read quickly at the beginning of a review (thus it needs to provide a strong impact) and then again in depth at the very end, perhaps a few days after the details of the previous 3 chapters have been somehow forgotten. Reinforcement of the positive is the key strategy here, without of course blowing hot air.

One other consideration is that some people like to join the chapter containing the analysis with the only with conclusions. This can indeed work very well in certain topics.

Finally, the conclusions do not appear only in this chapter. This sample mini thesis lacks a feature which I regard as absolutely necessary, namely a short paragraph at the end of each chapter giving a brief summary of what was presented together with a one sentence preview as to what might expect the connection to be with the next chapter(s). You are writing a story, the *story of your wonderful research work.* A story needs a line connecting all its parts and you are responsible for these linkages.

Appendix A

Additional Information

This is a good place to put tables, lots of results, perhaps all the data compiled in the experiments. By avoiding putting all the results inside the chapters themselves, the whole thing may become much more readable and the various tables can be linked to appropriately.

The main purpose of an Appendix however should be to take care of the future readers and researchers. This implies listing all the housekeeping facts needed to continue the research. For example: where is the raw data stored? where is the software used? which version of which operating system or library or experimental equipment was used and where can it be accessed again?

Ask yourself: if you were given this thesis to read with the goal that you will be expanding the research presented here, what would you like to have as housekeeping information and what do you need? Be kind to the future graduate students and to your supervisor who will be the one stuck in the middle trying to find where all the stuff was left!

Bibliography

[1] Cliff Atkinson. *Beyond Bullet Points: Using Microsoft Office PowerPoint 2007 to Create Presentations That Inform*. Microsoft Press, 2008.

[2] Wayne Booth. *The Craft of Research*. University of Chicago Press, 2003.