INTER-PROCESS COMMUNICATION, ANALYSIS, GUIDELINES AND ITS IMPACT ON COMPUTER SECURITY

Zoran Spasov
T-Mobile Macedonia
Skopje, Macedonia

Ph.D. Ana Madevska Bogdanova
Institute of Informatics, FNSM
Skopje, Macedonia

ABSTRACT

In this paper we look at the inter-process communication (IPC) also known as inter-thread or inter-application communication from other knowledge sources. We will look and analyze the different types of IPC in the Microsoft Windows operating system, their implementation and the usefulness of this kind of approach in the terms of communication between processes. Only local implementation of the IPC will be addressed in this paper. Special emphasis will be given to the system mechanisms that are involved with the creation, management, and use of named pipes and sockets.
This paper will discuss some of the IPC options and techniques that are available to Microsoft Windows programmers. We will make a comparison between Microsoft remoting and Microsoft message queues (pros and cons).
Finally we will make some notes and remarks regarding several issues and concerns about the security of the local system when using these methods, in order to use this knowledge in building a system that will control processes within different desktop environments. At the end, we will give some conclusions about the implementation and use of the IPC methods, including local security guidelines.

I. INTRODUCTION

Interprocess communication (IPC) serves for the coordination of activities among cooperating processes. A simple example of the IPCs usability is the need of two processes to share some data or a single value. In order for the IPC to work, some way of communication between the processes is needed. The IPC commonly is used on local computers, but it is also possibly to utilize its functionality over the network.
Today's big and important role that various distributed systems play in modern computing environments imposes the need of using the IPC in a common and transparent manner. Systems for managing communication and synchronization between cooperating processes are essential to many modern software systems. For many years the IPC was mainly present on the UNIX-platform based systems, but in the past decade the usability in the Microsoft Windows systems is more present than before. This paper will discuss some of the IPC types and methods that are available and will describe the techniques available to Windows OS programmers. We will explain the special features and implementations with the help of some example code and tools available for process monitoring of Windows operating systems.
The advantages that the IPC brings do not come with some noticeable risks and security concerns about the local computer system. In this paper we will properly address these concerns and we will describe some guidelines for their implementations in a secure way.
Finally the conclusion will offer a summary of the available programming techniques and implementations for the Windows platforms. We will note the security risks and the best practices to avoid them.

II. INTER-PROCESS COMMUNICATION (IPC)

Inter-Process Communication (IPC) stands for many techniques for the exchange of data among threads in one or more processes - one-directional or two-directional. Processes may be running locally or on many different computers connected by a network. We can divide the IPC techniques into groups of methods, grouped by their way of communication: message passing, synchronization, shared memory and remote procedure calls (RPC). We should carefully choose the IPC method depending on data load that the communication will carry and some other factors like the type of data that is transferred or the bandwidth and the latency of the communication between the threads (processes).
There are many reasons of using this kind of communication, some of them are:

- Shared control over several processes
- Sharing of data
- Parallel processing and computation
- Modularity
- Reduce programming costs
- etc…

IPC may also be referred to as inter-thread communication and inter-application communication. IPC, mainly depends of access to the shared memory address space and we can easily say that IPC is one of the foundation stones for the memory isolation concept.
IPC techniques include File, Signal, Sockets, Semaphore, Pipes, Memory-mapped file, Mailslot, Remote Procedure Calls (RPC), Message passing etc. The most widely used methods along with programming techniques in Microsoft operating systems will be described in the continuing text.

A. Memory-mapped file

File mapping is a mechanism for one-way or bi-directional inter-process communication among two or more processes in the local machine. The file mapping works by mapping a file in the computer's memory address space and the processes can access it by a handle or by the name of that mapping object. The address space that is occupied by a memory-mapped file contains the contents of a file but in virtual memory and that logical address space belongs to the application itself. This access to the mapped file is transparent to the processes and in order to modify the file they read and write directly into the memory. With the newest .NET Framework, we can use managed code to access memory-mapped files in the same way that native Windows functions access memory-mapped files from the Win API. To share a file, the first process creates or opens a file by using the *CreateFile* function. Next, it creates a file mapping object by using the *CreateFileMapping* function, specifying the file handle and a name for the file mapping object. The names of events, semaphores, mutexes, waitable timers, jobs, and file mapping objects share the same namespace. Therefore, the *CreateFileMapping* and *OpenFileMapping* functions fail if they specify a name that is in use by an object of another type. To share memory that is not associated with a file, a process must use the *CreateFileMapping* function and specify *INVALID_HANDLE_VALUE* as the *File* parameter instead of an existing file handle. The corresponding file mapping object accesses to the memory are backed by the system paging file. We must specify a size greater than zero when we specify an *File* of *INVALID_HANDLE_VALUE* in a call to *CreateFileMapping* function. Processes that share files or memory must create views by using the *MapViewOfFile* or *MapViewOfFileEx* functions in order to access the mapped file (to the whole file or only to its segment). They must coordinate their access using semaphores, mutexes, events, or some other mutual exclusion techniques or an error of concurrent access will be thrown. We can classify the memory-mapped files into two categories:

- Persisted memory-mapped files. Sometimes there is a need for large memory-mapped files. Usually these files are associated with a source file on a disk. When there is no more input/output operation with the memory file, all data from the memory is written to the source file on the disk.
- Non-persisted memory-mapped files. These files are not associated with a source file on the disk so after the last process finishes with the operations on this file, the file is removed from memory by the garbage collector and all data is lost. These files are suitable for creating shared memory for further IPCs.

Earlier we mentioned the file can be accessed by views created to the whole file or to a part of it. Depending on what type of files we use, there are two types of view access: for the non-persisted ones we use stream view access and for persisted memory-mapped files - random access views. We can also create multiple views to the same part of the memory-mapped file, creating a concurrent access to the memory. The number of views may also depend from the size of the data we use, because depending of the computer's hardware, maximum memory space available for memory mapping, for example we have 2GB on 32bit computer.
The access control and partitioning to some number of pages to these memory-mapped files is delegated by the operating system's memory manager.
Fig. 1 shows how we can use more than one view per process in order to access files or parts of files.

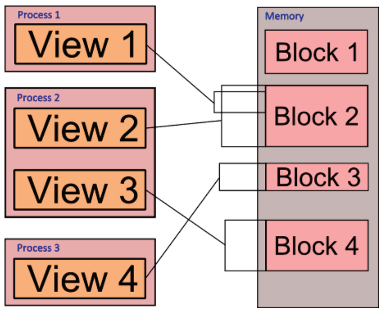


Figure 1: Views for access to memory blocks.

B. Pipes

Pipes are another method that provide a tool for interprocess communication. There are two types of pipes:

1) Anonymous pipes

In Unix environment, a pipeline is something very familiar to a shell script programmer: its a set of processes that are chained by their standard streams, one process output is another's input. That chain represents one anonymous pipe. This type of pipes can only be used on a local computer because they are not named and work only in direction. They are useful if we have intensive communication between threads because this kind of pipes does not introduce too much overhead in the communication but have limited services. They support only a single server instance and the pipe handles can be easily passed to the child process when it is created. In the .NET Framework, we can implement anonymous pipes by using the *AnonymousPipeServerStream* and *AnonymousPipeClientStream* classes.

2) Named pipes

On the named pipes we can look as some sort of add-on to the traditional type of pipes. By their functionality they are very similar to a FIFO queue. In the Mac operating system they are called sockets, which is different from a TCP socket. This concept is also present in Microsoft Windows and the Unix-like systems, but maybe the semantics are different. Previously we talked about unnamed or anonymous pipes, we need to mention that they are of short life, after the process termination they are deleted, as on the other hand named pipes are persistent and we need to take care about their deletion after they are no longer needed. Processes generally attach to the named pipe (usually appearing as a file) to perform inter-process communication (IPC). The main difference between unnamed and named pipes is that with latter we can have bi-directional communication from one server and one or more clients and this communication can exist on the local computer or over the computer network. Named pipes also support impersonation, which enables connecting processes to use their own permissions on remote servers. In the .NET Framework, we can implement named pipes by using the *NamedPipeServerStream* and *NamedPipeClientStream* classes. The named pipes concept is closely connected to the I/O subsystem and to the user it appears as nothing but another file system. The main reason for this impression is because named pipes are written as file system drivers. As they represent some kind of file system, they can be remotely accessed. We can use the Common Internet File System (CIFS) redirector to intercept file input/output (I/O) requests and direct them to a drive or resource on another networked computer.

C. *Microsoft message queues*

Microsoft Message Queue Server (MSMQ) enables easy approach for application developers to communicate with application programs quickly, reliably, and asynchronously by sending and receiving messages. MSMQ are present on the newer Windows operating systems and they work on the concept of message exchange between the applications. Everyone can be producer or consumer of messages. The underlying mechanism takes care of the message store and access to the messages. The producer creates the message carrying data to the destination application and places that message on early created named queue. The destination application takes the message whenever it is needed, even if the producer application (process) is down. That mechanism allows bi-directional and asynchronous communication between the processes. Because the MSMQ mechanism is separate system and it's not connected to the processes, we must take care of the messages in the queues and queues itself. This means we must delete the messages from the queues when they are not needed and also the message queues themselves. The asynchronous communication differs from most of the other methods. Communications are synchronous when the sender of a request must wait for a response or an acknowledgement from the receiver of the request before it continues with its work. With asynchronous communications, available through MSMQ, producers make requests to receivers and then can move on executing immediately. As mentioned earlier, with asynchronous communications, there are no requirements that a receiver must be running for the producer to make and send the message and this goes both ways, there is no requirement for the producer to be running in order the receiver to get the message.

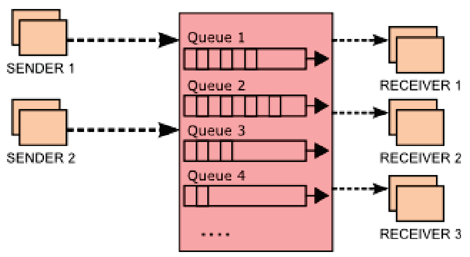


Figure 2: MSMQ functionality.

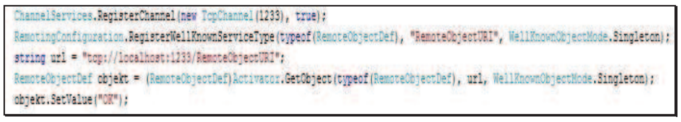
The base mechanism of MSMQ provides a very reliable way for the process to communicate and exchange data.

D. *Microsoft remoting*

Microsoft .NET remoting provides a framework that allows objects to communicate across different application environments. This concept provides the means of communications through channels that have activation and its own lifetime. The channels transport the messages from one process to another. We use different formatters for encoding and decoding the messages before they are placed on and transported by the channel. One of the advantages with this type of communication is using the XML encoding when there are different frameworks involved in the communication. The applications can use binary encoding as well, where performance is critical. All XML encoding uses the SOAP protocol in transporting messages from one process environment to the other. The security factor was primal in designing this method. A number of hooks are provided that allow channel sinks to gain access to the messages and serialized stream before the stream is transported over the channel.
As we mentioned earlier this method also provides the control of a object's lifetime. .NET remoting provides a number of activation models, but all of them fall into two categories:

- Client-activated objects
- Server-activated objects

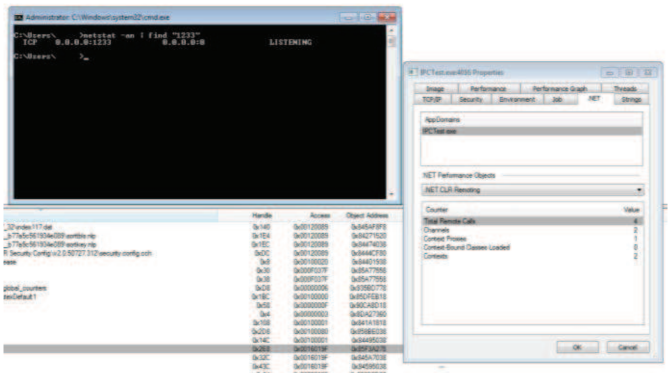
The first ones work on principle of a lease time. When this lease time expires, the object is collected by the garbage collector and it is destroyed. The other ones are activated by a server and they are also separated into two categories. We can select either the object to be "single call" or "singleton". The second one also works on the same principle of a lease time. The first type of object is destroyed after the first call to it was realized. They use local ports on which the client processes connect in order to read or write data.
This method will be presented in the next section through the example application in Microsoft .NET.
The class in .NET that contains the methods for creating, controlling, communicating and deleting this communication channels is located in the *System.Runtime.Remoting* library. The following code snippet written in C# language creates a tcp channel for communication on specified port, uses security wrapping and sets a string value to the seriazable object:



```
ChannelServices.RegisterChannel(new TcpChannel(1233), true);
RemotingConfiguration.RegisterWellKnownServiceType(typeof(RemoteObjectDef), "RemoteObjectURI", WellKnownObjectMode.Singleton);
string url = "tcp://localhost:1233/RemoteObjectURI";
RemoteObjectDef objekt = (RemoteObjectDef)Activator.GetObject(typeof(RemoteObjectDef), url, WellKnownObjectMode.Singleton);
objekt.SetValue("OK");
```

Figure 3: Code snippet for Microsoft remoting.

After executing the application we can see from the Windows process explorer in the "Sysinternals" toolkit that channel is created and it's listening on the designated port. This is presented in Fig. 4.



```
C:\Users\      >netstat -an | find "1233"
  TCP    0.0.0.0:1233           0.0.0.0:0              LISTENING

C:\Users\      >_
```

Figure 4: Sysinternals processes explorer.

III. Comparison between Microsoft message queues and remoting

When we are faced with the dilemma of choosing the right IPC method, the task is very difficult. By the popularity and especially the usability in the .NET framework, only two methods should be at top of our list. Although these mechanisms are relatively new they are the first choice in many situations. Microsoft message queue method has the following properties:

- bi-directional communication
- asynchronous exchange of data
- support of XML formatted messages
- control of the queues that are handled by the underlying mechanism

As with the remoting method, Microsoft has made a different approach that takes care of the functionality of the whole mechanism. The processes itself are responsible for creation, maintenance and deletion of communication channels. The method has these properties:

- bi-directional communication
- channels are established through ports
- asynchronous exchange of data
- dependency between the lifetime of the process that created the channel and the lifetime of that channel
- support of XML or seriazable objects

Depending of the environment, one method is preferred over the other. Microsoft remoting is better on client environments - there is no need of additional installation and configuration of software. It's very transparent and secure method for use and it's easy to set up. On the other hand MSMQs are preferable on central location of control like a server environment. By default Windows operating systems does not come with MSMQ preinstalled, configured and ready to use functionality. Also the server may require true asynchronous communication, meaning that it should continue to function normally and at the same time the other processes to access the data after process-creator's termination. MSMQ saves resources because the same queue, early created can be used again by another process for different purposes.

In terms of security readiness, the remoting IPC, because of ports usage can be more vulnerable to an attack than MSMQ, but in the recent versions of .NET framework, Microsoft has spend a great deal of time working to provide better security for the data exchange. We will discuss more about the security in the next section.

IV. Security issues and concerns of IPC usage

When IPC is implemented one has to take into consideration the security issues that are raised by its use. There are serious security risks with the IPC methods because these methods are tightly connected with the core functions of the operating system.

There are big number of different vulnerabilities that are available to an attacker by IPC misconfigurations. One of the introduced risks is execution of remote code. As some IPC methods use memory mappings techniques, an attacker can access the memory segment and overflow the address space which will cause the operating system to execute the code that is feed to this address space. They are many security checks and controls that are done by the operating systems and in the corporate environment – firewalls and IDS, etc., but there still a chance for the attacker to gain access to the local computer.

At present time and maybe in the past few years a great concern among the security community is the DoS (denial of service) attacks and especially Dynamic DoS attacks. Because of the nature of the IPC mechanism and the synchronous communication over sockets and channels of some of the IPC methods, they represent great opportunity for the attacker to exploit this weakness. There are three possible scenarios how an attacker can produce DoS attack using the IPC mechanisms:

- make numerous connections on the IPC channel's listening port
- interrupt the communication and with that the execution of the processes
- change the data, so the processes that communicate become irresponsive.

Because the communicating processes connect or listen on different ports, big number of connections can make the processes to be unable to connect and communicate with one another and thus making the whole application stop functioning (for synchronous communication) or some services to became not responsive. The second one also applies to the synchronous communication of the processes. In the third scenario, an attacker can change the data that passes or exists on the communication channel and make the processes to work with corrupted data and thus leading them to halt or stop responding.

Another possible way is the "man in the middle" attack, meaning that someone could be eavesdropping on the conversation between the processes and possibly will gain access to valuable and confidential data that is exchanged between the processes.

Many of the methods mentioned above can be achieved by using windows process hooks that inject dll library code into existing or new process, but they are not topic in this paper.
We will not go into detail in explanation of the possible attack techniques because of some ethical standards and law limitation. Instead we will make some suggestions in improving and eradicating some of the security issues.
If there is need for inter-process communication on the server environment it is recommended to use Microsoft message queues in order to allow the processes to communicate asynchronously. This way there is continuous functionality on the services. Or if it's decided to use other methods it's recommendable to use processes that do not wait for acknowledgment of the received data. If we exchange confidential data or some control messages with other process, the data should be encrypt and then decrypt at the receiver's end. If there is big volume of data communication the encryption/decryption process will introduce processing delay or drop in the computer performance. In that case we can implement some sort of local firewall for inbound connections or we can make some security tweaks on local operating system.

V. Conclusion

In this paper we have elaborated the need of inter-process communication, some of the IPC methods, mechanisms and their implementation. We looked at IPC methods – Microsoft message queues and Microsoft remoting and made short comparison of their properties. Finally we looked at the security requirements and concerns over using these methods and made some suggestion on how to improve the local security of the system. Each of the IPC mechanisms discussed has some advantages and some disadvantages; each of them is optimal solution for a particular problem for the application programmer.
The usability of these methods is very versatile, especially locally on the same computer but in different environments. Depending of the circumstances and the environments where the processes are executed, earlier we made some notes on how to choose the perfect IPC method in order to achieve our desired goal with regards to the computer security. Choosing which of the methods will be implemented is up to the individual application programmer. We must take into consideration the balance of the application performance and ease of the desired use, and the technical requirements of the application. We need to see what are our priorities and a selection can usually be made easily.
Overall, the inter-process communication is great for introducing reliable communication, maximum performance, great functionality, application modularity and support and also secure communication in our multi-process environment.

References

[1] Microsoft MSDN library, online material, *http://msdn.microsoft.com*.

[2] Zoran Spasov, "SMS Center", *graduate thesis, Subtitle: Message based communication, Institute of Informatics*, 2006.

[3] Ralph Davis, *Win32 Network Programming*, Reading, Massachusetts: Addison-Wesley Developers Press, 1996.

[4] Jim Beveridge and Robert Wiener, *Multithreading Applications in Win32,* Reading, Massachusetts: Addison-Wesley Developers Press, 1997.

[5] Leslie Lamport, *Interprocess Communication*, Technical Report, SRI International, March 1985.

[6] Jesse Burns, *Fuzzing Win32 Inter-Process Communication Mechanisms*, Las Vegas, 2006.

[7] James Naftel, Kim Williams and Scott McLean, *Microsoft .NET Remoting (Pro-Developer)*, September 2002

[8] Ingo Rammer and Mario Szpuszta, *Advanced .Net Remoting (Expert's Voice)*, April 2002