Windows Kernel Internals
Lightweight Procedure Calls

David B. Probert, Ph.D.
Windows Kernel Development
Microsoft Corporation
Topics

• LPC overview
• !lp careless debugger extension
• Investigation checklist
• Debugging samples
LPC usage

• LPC is an internal interface for NT components.
• Communications between two user mode components (csrss and win32, winlogon and lsass)
• Communications between a user-mode process and a kernel-mode component (lsass and Security Reference Monitor)
• Local RPC
LPC Architecture

Server process

Kernel Address Space

Client process

Connection Port Handle

Connection port (named / unnamed)

Server Comm Port

Client Comm Port

Server View of Section

Shared Section

Client View of Section

© Microsoft Corporation
LPC ports

• Connection port (named / unnamed)
  – Created by the server side.
  – Used to accept connections, receive requests and to reply to messages

• Server communication port
  – The server receives a handle to server port each time a new connection is created.
  – Used to terminate a connection, to impersonate the client or to reply.

• Client communication port
  – The client receives a handle to a client port if the connection was successfully accepted.
  – Used to request/receive messages
LPC Data Transfer

• The message is temporary copied to kernel (< 256 bytes*)
• Using shared sections, mapped in both client and server address spaces
• The server can directly read from or write to a client address space
Creating an LPC server

- 1. Create a named connection port (NtCreatePort)
- 2. Create one or more working threads listening to requests on that LPC connection port (NtReplyWaitReceivePort)
Creating an LPC server – cont

```c
{ ...  
    If ( NtCreatePort(&SrvConnHandle, “LPCPortName”) ) {
        CreateThread ( ProcessLPCRequestProc)
    }

    ... 
}

ProcessLPCRequestProc ()
{

    ReplyMsg = NULL;
    while ( forever_or_so ){
        NtReplyWaitReceivePort( SrvConnHandle, ReplyMsg, ReceiveMsg )
        DoStuffWithTheReceivedMessage()
        ReplyMsg = PrepareTheReply ( IfAny )
    }
}

* Some servers launch an worker thread to process the request and reply to the client
```
Establishing an LPC connection

- The Client initiates a connection (NtConnectPort)
- The server receives a connection request message
- The server decides to accept/reject the connection and calls NtAcceptConnectPort
- The server wakes up the client (NtCompleteConnectPort)
Common issues

• Servers cannot send messages to clients that are not waiting for an LPC message
• If a server dies, the client is not notified unless it has threads waiting for a reply
• No timeout for the LPC wait APIs
LPC data structures

• LPC Port (paged)
  – Port type, connection & connected port, owning process, server process, port context

• LPC Message (paged)
  – MessageID, message type, ClientID

• Thread LPC fields (non-paged)
  – Wait state, request messageID, LCP port, received message id, port rundown queue

• Global data
  – LpcpNextMessageId, LpcpLock
LPC port object

• Object fields (name, ref count, type)
• Port type (connection, server comm, client comm)
• Connection and connected port
• Creator CID
• Message queue
• Port context
• Thread rundown queue
typedef struct _LPCP_PORT_OBJECT {
    ULONG Flags;
    struct _LPCP_PORT_OBJECT *ConnectionPort;
    struct _LPCP_PORT_OBJECT *ConnectedPort;
    LPCP_PORT_QUEUE MsgQueue;
    CLIENT_ID Creator;
    PVOID PortContext;
    ULONG MaxMessageLength;
    LIST_ENTRY LpcReplyChainHead;
    LIST_ENTRY LpcDataInfoChainHead;
    ...
}
LPC ports in EPROCESS

• DebugPort
  – Used to send debugger messages

• ExceptionPort
  – CsrCreateProcess assigns it to a win32 process

• SecurityPort
  – Used by lsass
LPC message format

- Kernel side (Port context, messages list)
- User side (PORT_MESSAGE)
  - Message type (request, reply, connection request, client died, port closed)
  - Message length, data offset
  - Client ID
  - Message ID
- Private data
typedef struct _LPCP_MESSAGE {
    union {
        LIST_ENTRY Entry;
    }
    PETHREAD RepliedToThread;
    PVOID PortContext;
    ...
    PORT_MESSAGE Request;
} LPCP_MESSAGE, *PLPCP_MESSAGE;
typedef struct _PORT_MESSAGE {
    CSHORT DataLength;
    CSHORT TotalLength;
    CSHORT Type;
    CSHORT DataInfoOffset;
    LPC_CLIENT_ID ClientId;
    ULONG MessageId;
    ULONG CallbackId;

    ...

    // UCHAR Data[];
} PORT_MESSAGE, *PPORT_MESSAGE;
More about LPC messages

• Where are messages to be found?
  – On the caller stack
  – In the port queue
  – In the thread pending the reply
• Can you tell how old a message is?
• Validating fields to detect corruptions
  – MessageID
  – Message type
  – Client ID
Typical message

Waiting for reply to LPC MessageId 000016df:

Pending LPC Reply Message:

```
e1a9d378: [e190e620,e1bd3008]
```

```
kd> dd e1a9d378
e1a9d378  e1bd3008 e190e620 00000000 00000000
e1a9d388  00000000 00000033 00cc009c 0000000a
e1a9d398  000007cc 00000784 000056df 00000000
e1a9d3a8  00000000 00000000 00000000 00000000
e1a9d3b8  00000000 00000000 e18e8ce0 00000000
```

```
1: kd> dc NT!LpcpNextMessageId l1
8025bafec  000027d8
```
The LPC fields in ETHREAD

- **LpcReplyChain**
  - To wake up a client if a server port goes away
- **LpcReplySemaphore**
  - It gets signaled when the reply message is ready
- **LpcReplyMessageId**
  - The message ID at which the client is waiting a reply
- **LpcReplyMessage**
  - The reply message received
- **LpcWaitingOnPort**
  - The port object currently used for a LPC request
- **LpcReceivedMessageId**
  - The last message ID that a server received
!lpc KD debugger extension

- !lpc message [MessageId]
- !lpc port [PortAddress]
- !lpc scan PortAddress
- !lpc thread [ThreadAddr]
- !lpc PoolSearch
Analyzing the LPC connection

• Get the information from the client thread
  – Use !thread to get the messageld and the communication port

• Find the server process
  – Use !lpc message to find the server thread/process working on this message
  – Use !lpc port to identify the connection port

• Check the server connection state
  – Semaphore state, message queue

• Look at what is doing the server thread
Client waiting for reply

• Recognizing the state
  – !thread will display:
    - WAIT state \texttt{WrLpcReply}
    - “Waiting for reply to LPC MessageId x”
    - “Current LPC port y”

• What’s next
  – Use !lpc to find the server thread / process / port
  – See if the server:
    • Didn’t receive the request
    • The server received but it didn’t reply
Common server problems

• The server is not servicing the port
  – All server threads are busy with some other requests (or deadlocked)
  – The server is suspended by the debugger
• The server replied to a wrong client
• The reply failed, and the server didn’t managed the result
• The server replied/impersonated using a wrong port
Discussion