Windows Kernel Internals
Object Manager & LPC

Dave Probert, Ph.D.
Advanced Operating Systems Group
Windows Core Operating Systems Division
Microsoft Corporation

Kernel Object Manager (OB)

- **Provides underlying NT namespace**
- **Unifies kernel data structure referencing**
- **Unifies user-mode referencing via handles**
- **Simplifies resource charging**
- **Central facility for security protection**

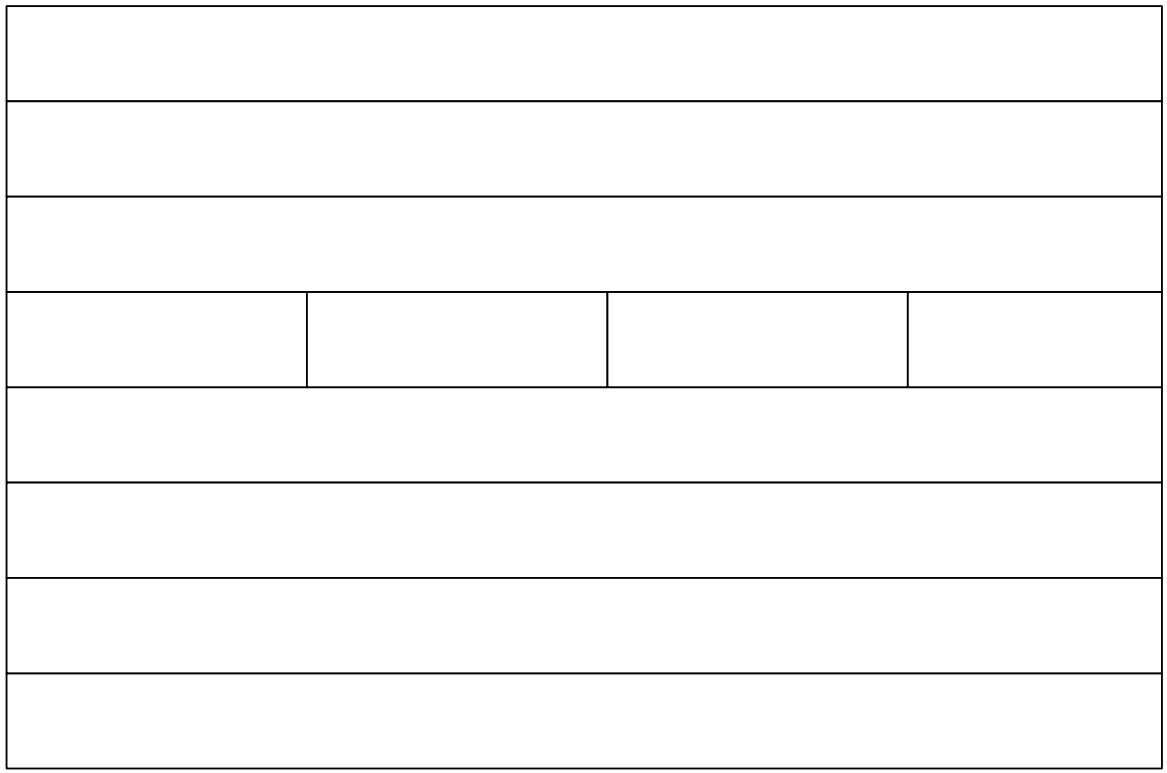
¥ObjectTypes

Adapter
Callback
Controller
DebugObject
Desktop
Device
Directory
Driver
Event
EventPair
File
IoCompletion
Job
Key
KeyedEvent
Mutant
Port
Process
Profile
Section
Semaphore
SymbolicLink
Thread
Timer
Token
Type
WaitablePort
WindowsStation
WMIGuid

¥ObjectTypes

Adapter
Callback
Controller
DebugObject
Desktop
Device
Directory
Driver
Event
EventPair
File
IoCompletion
Job
Key
KeyedEvent
Mutant
Port
Process
Profile
Section
Semaphore
SymbolicLink
Thread
Timer
Token
Type
WaitablePort
WindowsStation
WMIGuid

OBJECT_HEADER



Generic object services

- namespace ops: directories, symlinks
- NtQueryObject
- NtQuery/SetSecurityObject
- NtWaitForSingle/MultipleObjects
- ObOpenObjectByName/Pointer
- ObReferenceObjectbyName/Handle
- NtDuplicateObject
- NtClose
- ObDereferenceObject

OBJECT_DIRECTORY

OBJECT_DIRECTORY

OBJECT_DIRECTORY_ENTRY *pHashBuckets[ ]
Lock
pDeviceMap
SessionId

OBJECT_DIRECTORY_ENTRY

OBJECT_DIRECTORY_ENTRY *pChainLink
pObject

ObpLookupDirectoryEntry(pD, s)

```
object = NULL
idx = HASH(s)
pE = pD->HashBuckets[idx]
LockDirectoryShared(pD)
while (pE && !eqs(s, pE->Object->Name))
    pE = pE->pChainLink
if (pE)
    ObpReferenceObject(object = pE->Object)
UnlockDirectory(pD)
return object
```

Object Methods

OPEN: Create/Open/Dup/Inherit handle

CLOSE: Called when each handle closed

DELETE: Called on last dereference

PARSE: Called looking up objects by name

SECURITY: Usually *SeDefaultObjectMethod*

QUERYNAME: Return object-specific name

OKAYTOCLOSE: Give veto on handle close

Object Manager Types

Directory - namespace object

Implementation hardwired

SymbolicLink - namespace object

DeleteProcedure = ObpDeleteSymbolicLink

ParseProcedure = ObpParseSymbolicLink

Type **- represent object types**

DeleteProcedure = ObpDeleteObjectType

Object Manager lookups

ObpLookupObjectName(Name,Context)

- Search a directory for specified object name
- Use ObpLookupDirectoryEntry() on Directories
- Otherwise call object-specific ParseProcedure
  - Implements symbolic links (SymbolicLink type)
  - Implements file systems (DeviceObject type)

I/O Manager Types

Adapter - ADAPTER_OBJECT

Controller - CONTROLLER_OBJECT

Device - DEVICE_OBJECT

ParseProcedure = IopParseDevice

DeleteProcedure = IopDeleteDevice

SecurityProcedure = IopGetSetSecurityObject

Driver - DRIVER_OBJECT

DeleteProcedure = IopDeleteDriver

IoCompletion - KQUEUE

DeleteProcedure = IopDeleteIoCompletion

I/O Manager File Type

File - FILE_OBJECT

CloseProcedure = IopCloseFile
DeleteProcedure = IopDeleteFile
ParseProcedure = IopParseFile
SecurityProcedure = IopGetSetSecurityObject
QueryNameProcedure = IopQueryName

IopParseDevice

(DeviceObject, Context, RemainingName)

- Call SeAccessCheck()
- If (!*RemainingName) directDeviceOpen = TRUE
- For file opens, get Volume from DeviceObject
- Update references on Volume and DeviceObject
- Construct an I/O Request Packet (IRP)
- FileObject = ObCreateObject(IoFileObjectType)
- Initialize FileObject
- Initiate I/O via IoCallDriver(VolumeDevice, IRP)
- Wait for I/O to signal FileObject->Event
- Return the FileObject to caller

FILE_OBJECT

pDeviceObject
pVolumeParameterBlock
pFsContext/pFsContext2
pSectionObjectPointers
pPrivateCacheMap
FinalNTStatus
pRelatedFileObject

Flags
CurrentByteOffset
FinalNTStatus
nWaiters
nBusy
LockEvent
Event
pIOCompletionContext

Process/Thread Types

Job - JOB

DeleteProcedure = PspJobDelete

CloseProcedure = PspJobClose

Process - EPROCESS

DeleteProcedure = PspProcessDelete

Profile - EPROFILE

DeleteProcedure = ExpProfileDelete

Section - SECTION

DeleteProcedure = MiSectionDelete

Thread - ETHREAD

DeleteProcedure = PspThreadDelete

Token - TOKEN

DeleteProcedure = SepTokenDeleteMethod

Job methods - Close

PspJobClose - called by OB when a handle is closed

- Return unless final close
- Mark Job as closed
- Acquire the job's lock
- If job marked PS_JOB_FLAGS_CLOSE_DONE
  - Release the JobLock
  - Call PspTerminateAllProcessesInJob()
  - Reacquire the JobLock
- Acquire the job's MemoryLimitsLock
- Remove any completion port from the job
- Release the MemoryLimitsLock
- Release the JobLock
- Dereference the completion port

Job methods - Delete

PspJobDelete - called by OB at final dereference

Holding the Joblock callout to ntuser

Acquire the PspJobListLock

If part of a jobset then we are the job pinning the jobset

tJob = next job in set and remove current job

Release the PspJobListLock

If (tJob) ObDereferenceObjectDeferDelete (tJob)

If (Job->Token) ObDereferenceObject (Job->Token)

Free pool allocated for job filters

Unlink our JobLock from the global list

Synchronization Types

Event - KEVENT

EventPair - EEVENT_PAIR

KeyedEvent - KEYED_EVENT_OBJECT

Mutant - KMUTANT

DeleteProcedure = ExpDeleteMutant

Port - LPCP_PORT_OBJECT

DeleteProcedure = LpcpDeletePort

CloseProcedure = LpcpClosePort

Semaphore - KSEMAPHORE

Timer - ETIMER

DeleteProcedure = ExpDeleteTimer

Win32k.sys

Callback - CALLBACK_OBJECT

DeleteProcedure = ExpDeleteCallback

WindowsStation, Desktop

CloseProcedure = ExpWin32CloseProcedure

DeleteProcedure = ExpWin32DeleteProcedure

OkayToCloseProcedure = ExpWin32OkayToCloseProcedure

ParseProcedure = ExpWin32ParseProcedure

OpenProcedure = ExpWin32OpenProcedure

ObCreateObjectType

TypeName – mostly for debugging

DefaultsCharges – amount of memory usage to charge process

InvalidAttributes – restricts object instances, e.g. not PERMANENT

GenericMapping – maps object-specific access rights

ValidAccessMask – restricts requested access

MaintainHandleCount – maintain database for debugging

Dispatch procedures – open, close, delete, parse, queryname, …

Handle Table (Executive)

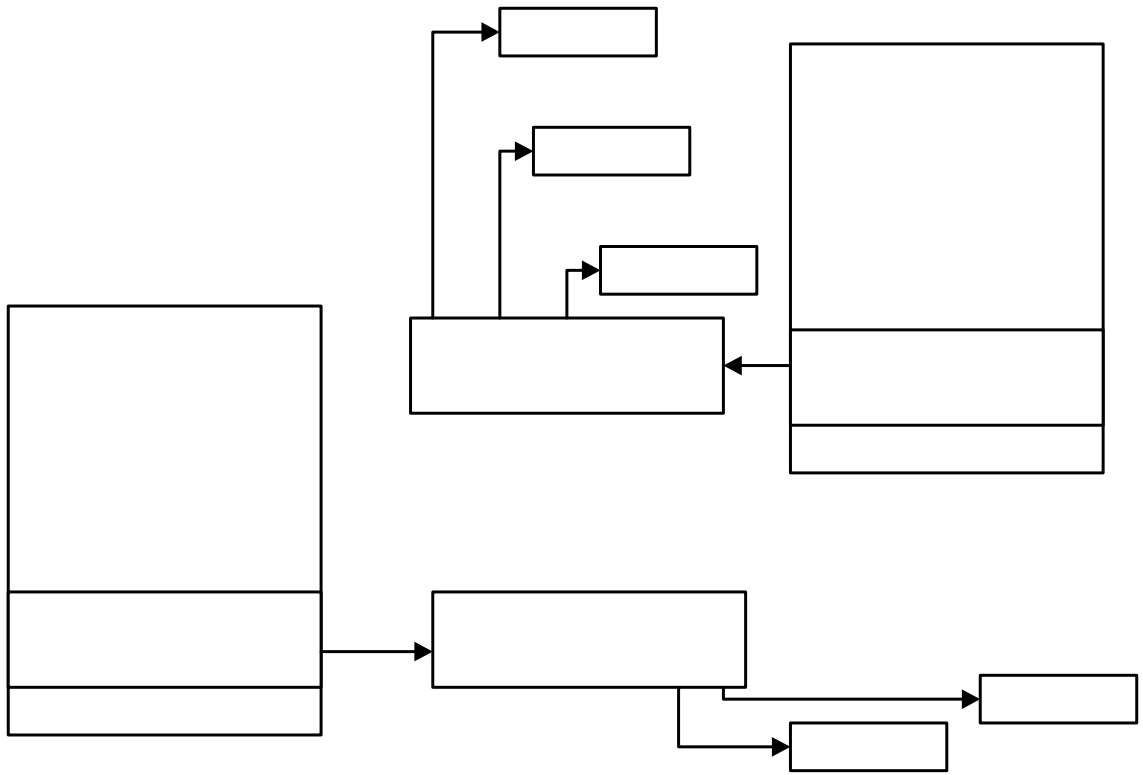
Efficient, scalable object index structure

One per process containing ‘open’ objects

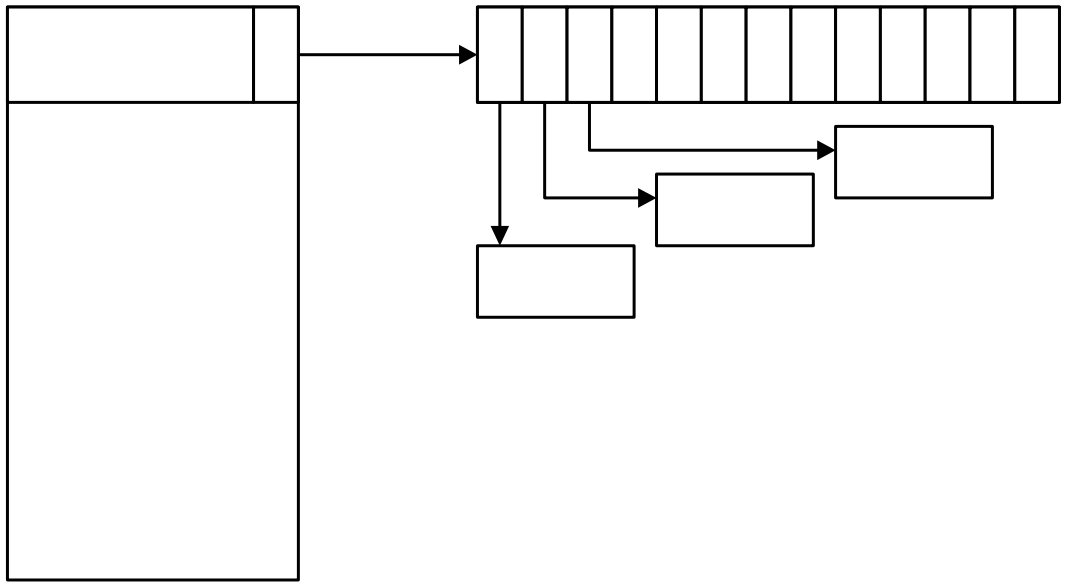
Kernel handle table (system process)

Also used to allocate process/thread IDs

Process Handle Tables

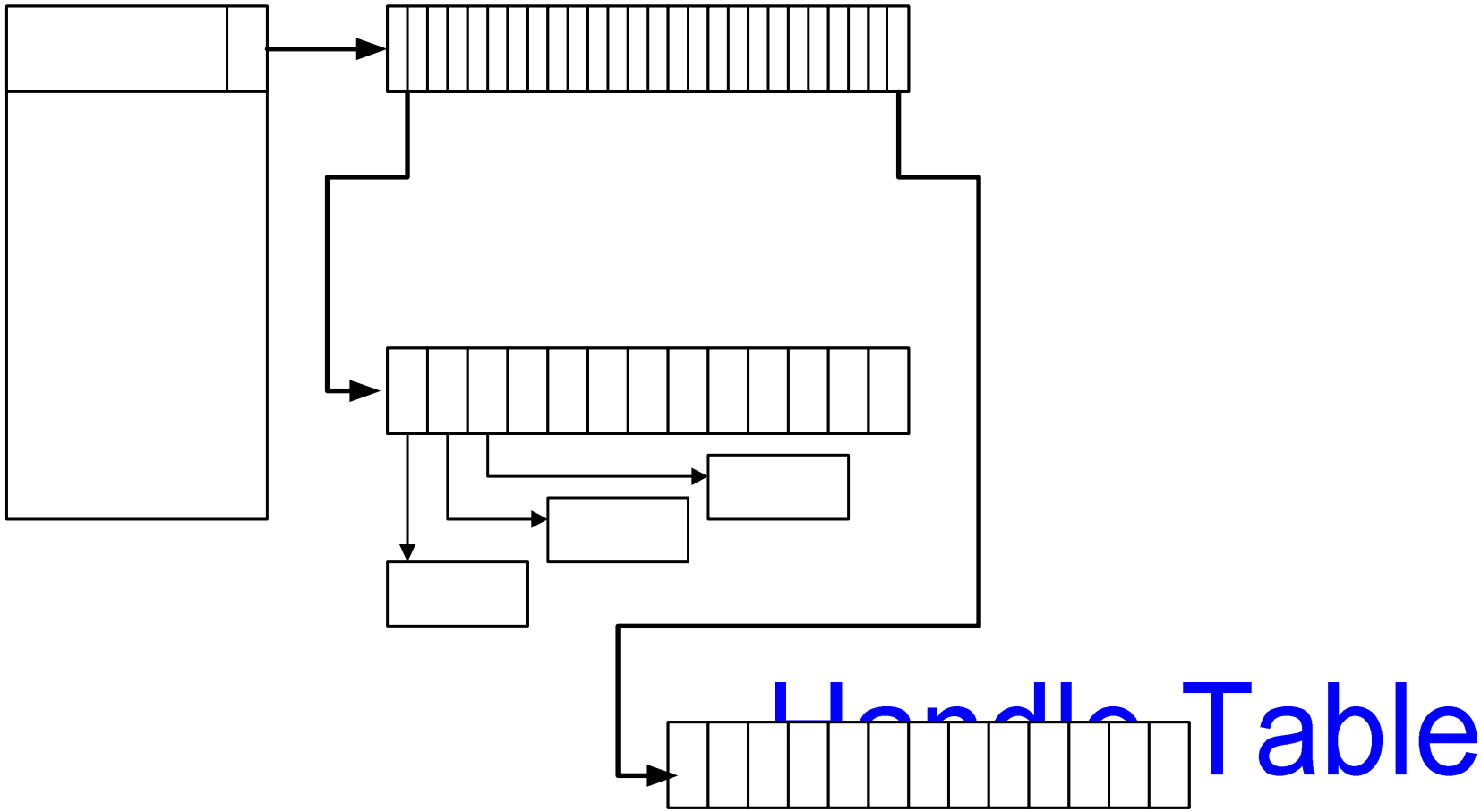


One level: (to 512 handles)

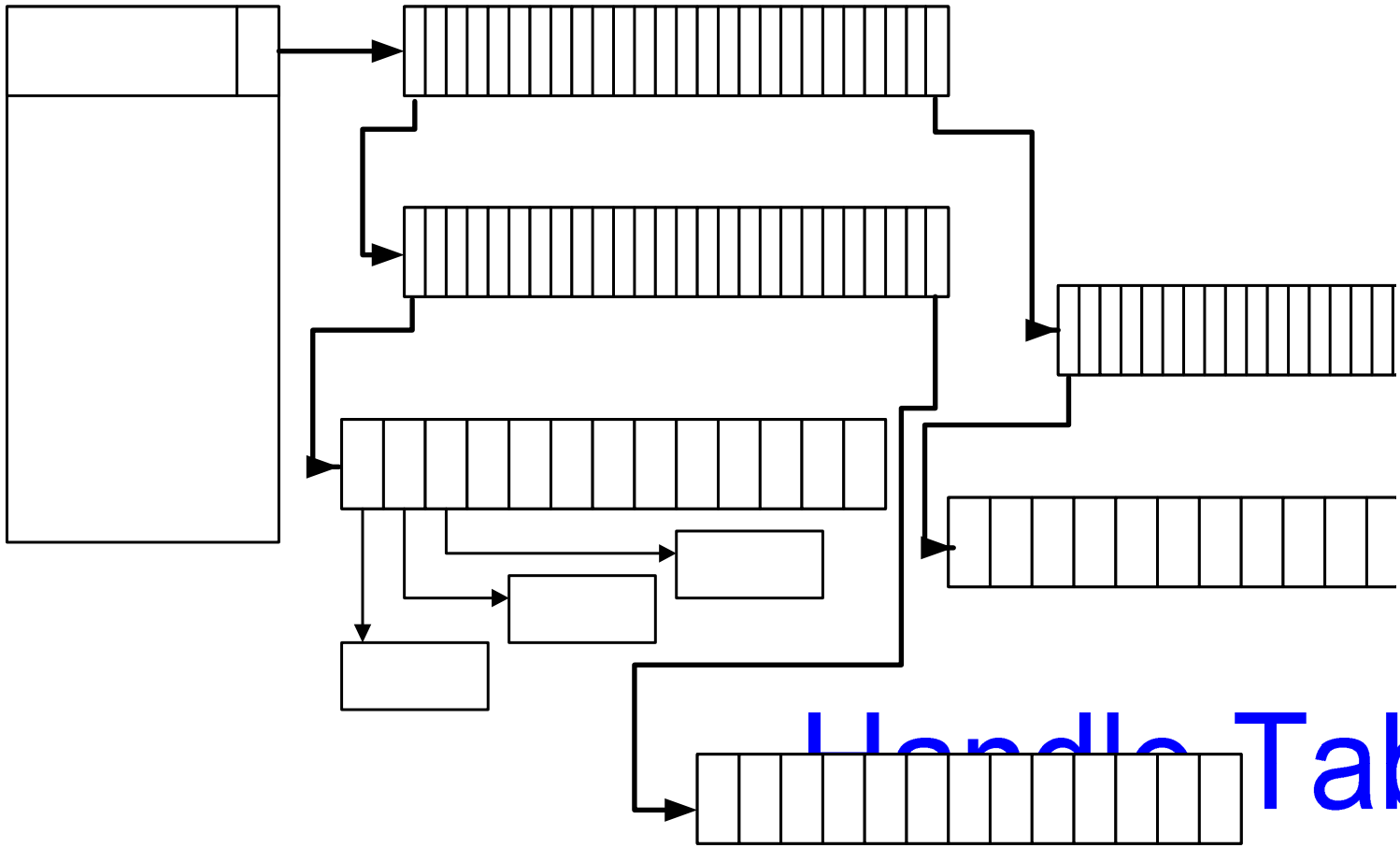


Handle Table

Two levels: (to 512K handles)



Three levels: (to 16M handles)



Handle Table Data Structure

TablePointer/Level	Points at handles
QuotaProcess	Who to charge
UniqueProcessId	Passed to callbacks
HandleTableLocks[N]	Locks for handles
HandleTableList	Global list of tables
HandleContentionEvent	Event to block on
DebugInfo	Stacktraces
ExtraInfoPages	Parallel table for audits
FirstFree/LastFree	The two handle free lists
NextHandleNeedingPool	Handles w/ memory
HandleCount	Handles in use

Handle Table Functions

ExCreateHandleTable – create non-process tables
ExDupHandleTable – called creating processes

ExSweepHandleTable – for process rundown
ExDestroyHandleTable – called destroying processes

ExCreateHandle – setup new handle table entry
ExChangeHandle – used to set inherit and/or protect
ExDestroyHandle – implements CloseHandle
ExMapHandleToPointer – reference underlying object

ExReferenceHandleDebugInfo – tracing handles
ExSnapShotHandleTables – handle searchers (oh.exe)

ExCreateHandle(table, entry)

NewHandleTableEntry = ExpAllocateHandleTableEntry()

KeEnterCriticalRegionThread()

*NewHandleTableEntry = *HandleTableEntry

ExUnlockHandleTableEntry()

KeLeaveCriticalRegionThread()

Object Manager Summary

- Manages the NT namespace
- Common scheme for managing resources
- Extensible method-based model for building system objects
- Memory management based on reference counting
- Uniform/centralized security model
- Support handle-based access of system objects
- Common, uniform mechanisms for using system resources

Lightweight Procedure Calls

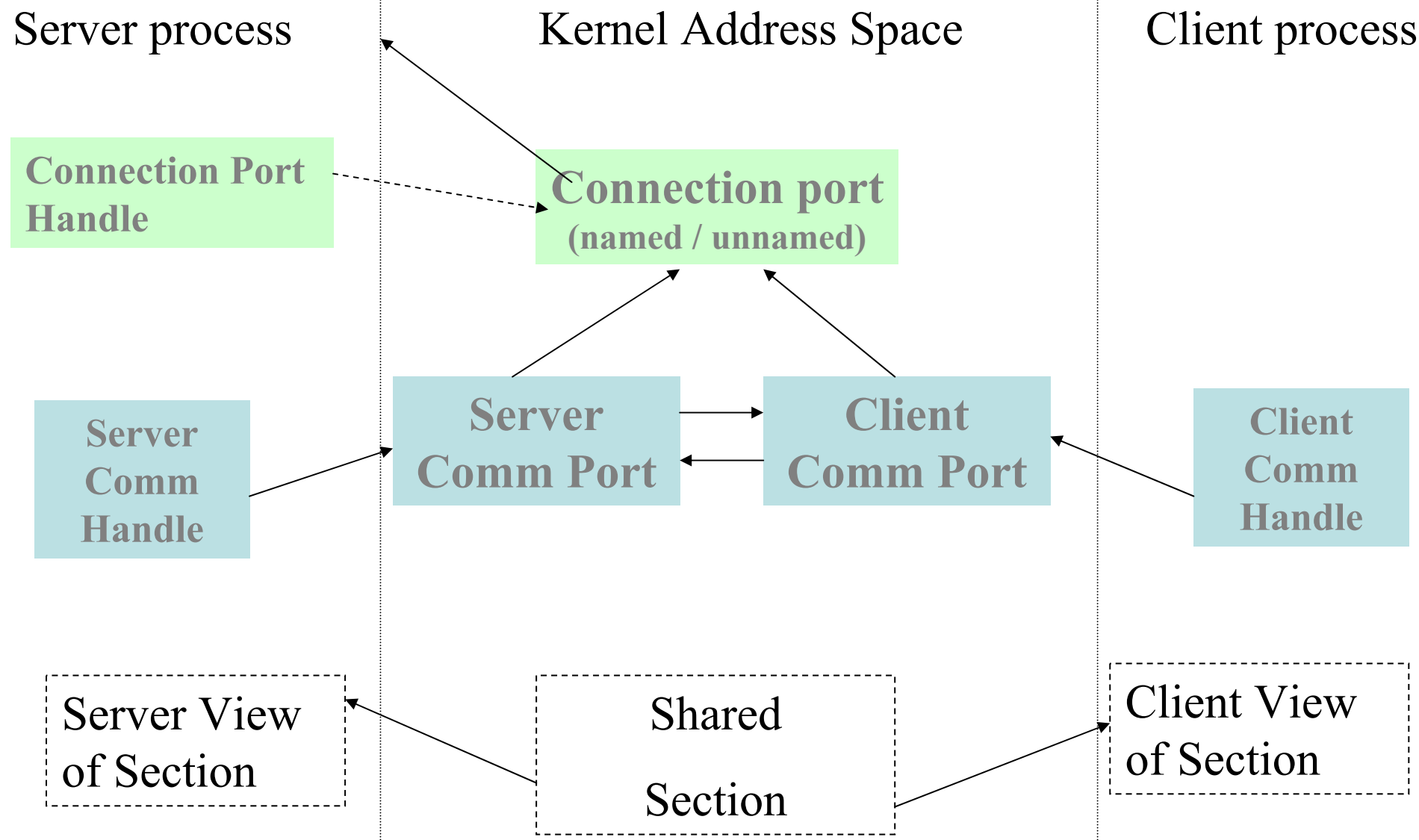
Most common local machine IPC

Built for subsystem communication

Local transport for RPC

- RPC also uses named pipes

LPC Architecture



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

LPC APIs

NtListenPort – server waits for connection request from client (wrapper for NtReplyWaitReceive)

NtAcceptConnectPort – accept/reject client connection request received by NtListenPort

NtCompleteConnectPort – server calls to wake up client after NtAcceptConnectPort

NtConnectPort – used by clients to connect to server ports

NtCreatePort – create a port and give a name in OB namespace

NtImpersonateClientOfPort – used by servers to impersonate client credentials

LPC APIs - 2

NtReplyWaitReceivePort – reply to a message and wait for next message

NtReplyPort – used by clients and servers to reply to messages

NtReplyWaitReplyPort – replies and then waits for a response

NtRead/WriteRequestData – copy message data to/from user buffer

NtRequestPort – send a message

NtRequestWaitReplyPort – send a message and wait for a response

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)

```
{… if ( NtCreatePort(&SrvConnHandle, “LPCPortName”) ) {
        CreateThread ( ProcessLPCRequestProc)
    } …
}
ProcessLPCRequestProc ()
{   ReplyMsg = NULL;
    while (forever_or_so)  {
        NtReplyWaitReceivePort( SrvConnHandle, ReplyMsg, ReceiveMsg )
        DoStuffWithTheReceivedMessage()
         ReplyMsg = PrepareTheReply ( IfAny )*
    }
}
```

* Some servers launch a 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

LPC Ports in Processes

DebugPort

- used to send debugger messages

ExceptionPort

- CsrCreateProcess assigns it to a win32 process

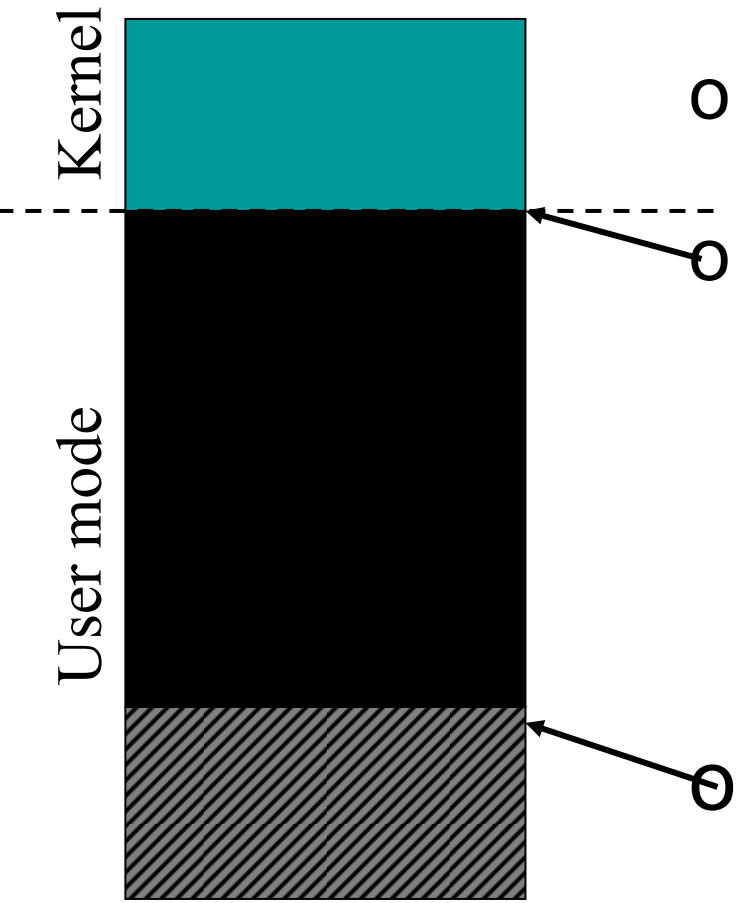
SecurityPort

- used by lsass (authentication system)

Where are messages found?

- on the caller stack
- in the port queue
- in the thread pending the reply

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

PORT_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;
```

LPC Fields in Threads

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

Discussion