Windows Kernel Internals
Windows Service Processes

David B. Probert, Ph.D.

Windows Kernel Development

Microsoft Corporation

What are services?

- Processes that run without needing an interactive logon
  - Services run without anybody logging on
  - Allow headless operation of machine
- Windows equivalent of UNIX daemons

NT Service Architecture

The Service Controller

- Started early in boot by Winlogon
- Responsible for enforcing service load order and dependencies
- Spawns all service processes
- Manages/watches all services on the local machine
  - Allows access to services via API calls
  - Guards access to services via access checks

NT Service Architecture

Service Processes

- Processes that host/implement one or more services
- Configured to run under a certain account
  - Can run interactively as LocalSystem
- Examples:
  - spoolsv.exe (Spooler, LocalSystem, interactive)
  - svchost.exe (generic host, any account)
  - services.exe (Eventlog, PlugPlay)

NT Service Architecture

Services

- Have a service name and display name
  - e.g., “PlugPlay” vs. “Plug and Play”
- Config info stored under …¥CCS¥Services¥<ServiceName>
- Follows service programming model
  - Implements ServiceMain and Handler(Ex) routine
  - Multiple services in-proc → each one implements a ServiceMain and Handler(Ex) routine

NT Service Architecture

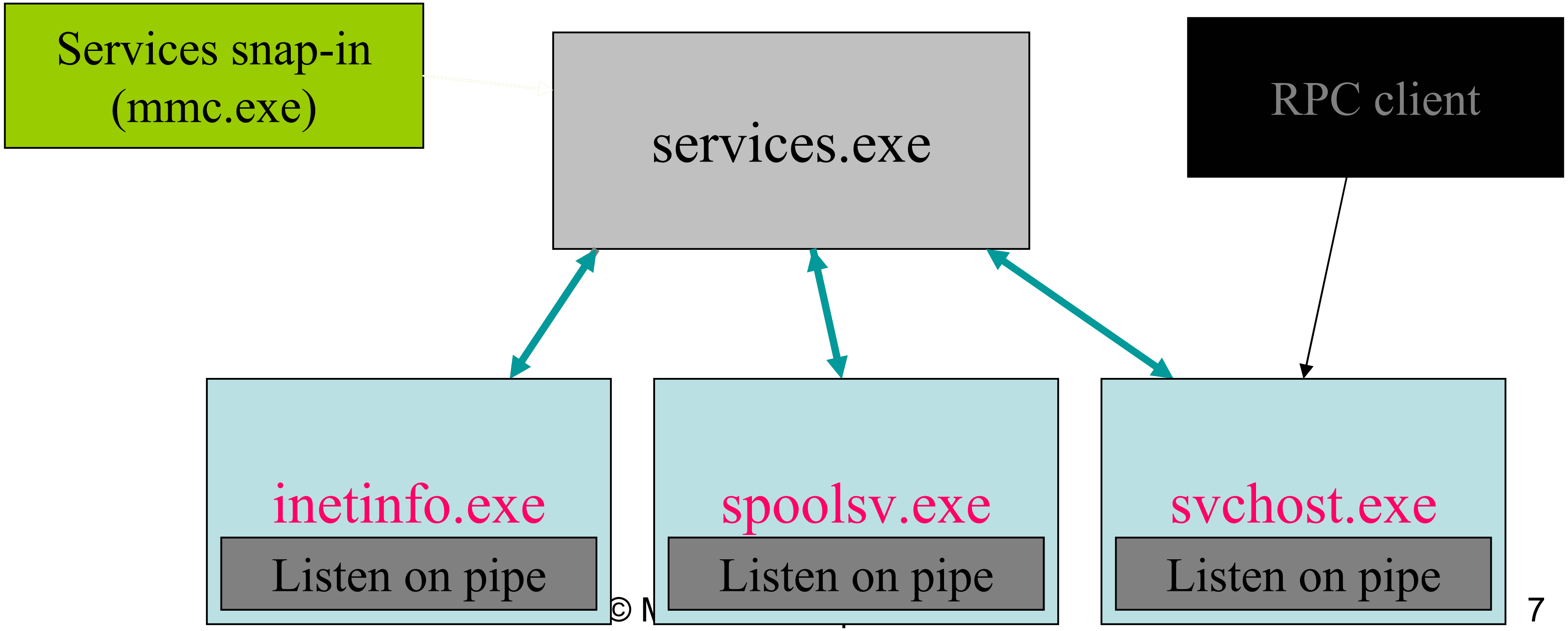
Service Control Programs (SCPs)

- Programs that call Service Controller APIs to manipulate services
  - Services MMC snap-in
  - sc.exe
  - net.exe (somewhat – provides start/stop only)
- SCPs call into the Service Controller, not the individual service processes

NT Service Architecture

How the pieces fit together

Services may have their own RPC interfaces/clients



Services snap-in (mmc.exe)

services.exe

RPC client

inetinfo.exe

Listen on pipe

spoolsv.exe

Listen on pipe

svchost.exe

Listen on pipe

The Service Controller

Starting services

- Service controller auto-starts services in group order
  - List at …¥CCS¥Control¥ServiceGroupOrder
  - Service may be configured as part of a group or ungrouped
  - Ungrouped services started last
- Service controller manages dependencies
  - Services may depend on other services or service groups
  - If dependent service (or service group) fails to start, SCM will fail start of service with ERROR_SERVICE_DEPENDENCY_FAIL
- Service Controller holds a critsec through entire auto-start process
  - Acquires/holds same critsec for each demand-start request
  - Allows SCM to enforce load-ordering
  - Means calls to StartService block until auto-start is complete

The Service Controller

Starting services and hang detection

- SCM waits for service to start if it is an auto-start service (being started as part of auto-start) or if it is a service on which a service being demand-started depends
- Service sets its dwWaitHint and dwCheckPoint via SetServiceStatus calls during its ServiceMain
  - dwCheckPoint → Current “stage” of service initialization
  - dwWaitHint → Estimated time to get to next checkpoint
- SCM uses a hang-detection scheme when waiting for a service to start (i.e., move out of the SERVICE_START_PENDING state)
  - Service gets 80 seconds plus its dwWaitHint to update its dwCheckPoint. If it doesn’t, SCM assumes service is hung and stops waiting for it (and kills the process if possible)

The Service Controller

Starting services (miscellaneous)

- Debugging service start
  - Configure the service process to start under the debugger piped out to the kd
  - Debugging using local debugger only (e.g., “ntsd” without “-d”) is difficult since the SCM will kill the service process if it takes more than 30 seconds to connect.
- Auto-start services have a significant performance effect
  - Many services starting up at boot leads to lots of I/O requests and contention over global resources (e.g., registry lock)
  - Can have a significant effect on boot time
  - If you can avoid making your service auto-start, do so

svchost.exe
How it works

- Individual services are configured to run in a particular instance of svchost.exe
  - Done through binary path associated with service (set when service is created or reconfigured)
  - Use “%SystemRoot%¥system32¥svchost.exe –k <instance name>”
- The list of services that can run in a particular process is static so list of services that run in an instance of svchost.exe must be well-known
  - Lists live at HKLM¥Software¥Microsoft¥Windows NT¥Svchost
- When svchost.exe starts up, it reads the list of services for the instance and sets a generic ServiceMain for each service
- Generic ServiceMain loads service DLL and then calls service’s actual ServiceMain routine (configured under the service’s Parameters key)

Writing a Service

Picking an account

- Win2K and earlier – service runs as LocalSystem with client LIB/DLL
  - Problem is that LocalSystem is too powerful
- Windows XP and beyond – service runs in new LocalService or NetworkService accounts
  - Greatly reduced privilege set
  - Have authenticated user access to objects (for the most part)
  - LocalService goes off-machine anonymously
  - NetworkService goes off-machine as machine account
  - Already instances of svchost.exe that run in these accounts (“LocalService” and “NetworkService” instances)
  - Configure account name of “NT AUTHORITY”¥LocalService or “NT AUTHORITY”¥NetworkService and empty password

Writing a Service

Performance Considerations

- Every process on the machine has a cost in memory (800K minimum working set vs. ~150K)
  - Rather than creating a new EXE for your service, run inside of a pre-existing instance of svchost.exe
- New threads have a cost in memory (each thread has stack pages that use up working set)
  - Rather than calling CreateThread for work items, use the NT thread pool APIs
- Avoid making your service auto-start if possible

Writing a Service

Being a good shared-process citizen

- Avoid APIs with process-wide effects
  - ExitProcess, ExitThread, TerminateThread
  - CoInitializeSecurity, RpcMgmt* APIs, etc.
- Avoid scary thread pool tricks
  - Blocking indefinitely during a work item
  - Returning pool thread in a different state
- Don’t unload your own service DLL (FreeLibraryAndExitThread)
- Don’t rely on running in a particular host process or instance of svchost

Writing a Service

Common Bugs During Service Start

- "Update" thread during service start
  - Service spins up a thread to loop while calling SetServiceStatus w/updated dwCheckPoint
  - If the ServiceMain hangs for real, no way for SCM to know. Boot hangs.
- Inaccurate dwWaitHint
  - Service may be killed when it's not actually hung (hint too small) or take too long to time out if actually hung (hint too large)

Writing a Service

Common Bugs During Service Start

- Trying to start another service from inside the ServiceMain
  - SCM holds global critsec when it waits for a service to start
  - StartService call needs that same critsec
  - Deadlock until service “times out”
- Implicitly depending on another service
  - Service polls for another service to be up and running in its ServiceMain
  - If load-ordering isn’t quite right (or is changed), the condition may never be met (e.g., polling on a service in a later load-order group)
  - If polling logic isn’t 100% correct (and it almost never is), other problems show up

Writing a Service

Common Bugs During Service Stop

- Service does clean-up after stopping
  - Service calls SetServiceStatus with SERVICE_STOPPED and then does some cleanup
  - As soon as the service reports that status, the SCM can start up a new instance of it. If the new instance starts while the old instance is still cleaning up, mayhem ensues

Writing a Service

Common bugs during Service Stop

- Shared-process service doesn’t clean up globals on stop or reinit globals on restart
  - Service runs in a process that doesn’t unload the service DLL when it stops (e.g., svchost)
  - Service is stopped/restarted. On restart, state of service based on stale globals is misleading.
- Service process does work after StartServiceCtrlDispatcher returns
  - Once last service in the process stops, SCM waits 30 seconds for process to exit before killing it

Writing a Service

Other Common Bugs

- Service takes too long in its control handler
  - One handler thread shared among all services in a process
  - SCM only waits 30 seconds for calls into the handler to return
  - If service wedges in its handler, handler thread is wedged for the entire process
- Modifying service config info in the registry directly
  - All service config info is stored in a known registry location, so app tweaks that info directly
  - The SCM doesn’t watch the service keys for changes – it reads/writes data to/from those keys at different times in response to API calls
  - Much more likely that this will hose the service rather than reconfigure it – use the SCM APIs

Discussion