Agenda

- Why PostgreSQL on Windows
- PostgreSQL for the Windows user
- Windows for the PostgreSQL user
- Advances in 8.3
Why PostgreSQL on Windows

- Isn’t Linux better?
  - Often, but not always

- Several scenarios
  - Developer laptops
  - Desktop database
  - Migration scenarios
  - Corporate policies
Agenda

- Why PostgreSQL on Windows
- PostgreSQL for the Windows user
- Windows for the PostgreSQL user
- Advances in 8.3
PostgreSQL for the Windows user

- Unix inheritance
  - Multi-process, not multi-thread
  - Shared memory
  - Requires “modern” windows (2000+, no FAT)
- Commandline!
  - psql, pg_dump, pg_dumpall, etc
  - pgAdmin3 to the rescue!
PostgreSQL for the Windows user

- Environment variables
- Configuration files
  - No registry
  - Edit with any text editor (e.g. notepad)
  - Edit with pgAdmin3
    - Still just a textfile
  - Signal server to reload (pause service)
- Security
  - Will *not* run with administrative privileges
Agenda

- Why PostgreSQL on Windows
- PostgreSQL for the Windows user
- Windows for the PostgreSQL user
- Advances in 8.3
It’s a brave new world

None of the normal tools

- No ps
- No kill
- No top
- No cron
- Nothing at all (almost)

Built-in Windows tools are generally bad
Architectural differences

- Runs as a service
  - Windows “version” of daemon
  - Started by "Service Control Manager"
  - Has it’s own login and session
- Logs to a file or eventlog
  - Startup errors *always* to eventlog
- CreateProcess() instead of fork()
  - Extra important to use connection pooling
Installing PostgreSQL

- Use the MSI
- External languages need to be in PATH: Perl, python, tcl etc
  - Path is per user or per system
- Or install from source
  - Complex, non-standard build environment
  - 8.3 supports MSVC, but still complex
Installing PostgreSQL - tips

- Turn off all unnecessary services
- Install data on dedicated filesystem
  - If possible, on dedicated spindles
- Use a junction or disk mount for xlog
- Mount with "noatime"
  fsutil behavior set disablelastaccess 1
- Disable 8.3 filename generation
  fsutil behavior set disable8dot3 1
Configuration parameters

- shared memory
  - Workload dependant
  - Smaller is better?!
- fsync methods
  - open_datasync (o_direct in 8.3)
  - fsync_writethrough
- log_destination
- lc_xyz
Get the right tools!

http://www.microsoft.com/technet/sysinternals

- Process Explorer (*the* tool for Win32)
- pstools
- Process monitor (debugging)
- Junction (tablespaces, xlog location)
Managing PostgreSQL

http://www.microsoft.com/technet/sysinternals
Library dependencies

- On Unix: "ldd"
### Unix:

```
root@svr1:~# ps axuwf |grep postgres
pgsql  1397  0.0  0.1  5360  1744 ?   S   2006  0:23    \_ postgres: logger process
pgsql  1399  0.0  0.3  15708  4160 ?   S   2006  2:29    \_ postgres: writer process
pgsql  1400  0.0  0.1  6360  1808 ?   S   2006  0:03    \_ postgres: stats buffer process
```
Process title

- Win32:
Process title

- **Win32:**

![Process Explorer Search](image)

- **Handle or DLL substring:** pgident

- **Process** | **Handle** | **Type** | **Handle or DLL**
- postgres.exe  | 2376 | Handle | \BaseNamedObjects\pgident: postgres: stats collector process
- postgres.exe  | 3900 | Handle | \BaseNamedObjects\pgident: postgres: writer process
Finding "stuck processes"

Unix:

```
root@svr1:~# strace -p 12771
Process 12771 attached - interrupt to quit
recv(7,  
```
Finding "stuck processes"

Win32:
Monitoring PostgreSQL

- Some things really didn’t change
- `pg_stat_xyz`
  - *Has not changed at all*
- PostgreSQL logs
  - `pg_data/*.log`
  - Don’t forget eventlog!
Monitoring PostgreSQL

- Performance Monitor
  - Standard Windows monitoring still applies!
- Monitoring the whole server
- Looking at individual backends
- Looking at the whole cluster
Monitoring PostgreSQL

- Performance
  - Standard Windows monitoring still applies!
- Monitoring
  - Monitoring the whole server
  - Looking at individual backends
  - Looking at the whole cluster
Monitoring PostgreSQL

- Performance
  - Standard
- Monitoring
- Looking at...
- Looking at...
Monitoring PostgreSQL

- Performance Monitor
  - Standard Windows monitoring still applies!

- Monitoring the whole server
- Looking at individual backends
- Looking at the whole cluster
Interesting counters to watch

- Number of processes
- New process creations
- Memory usage (private bytes, working set)
- Context switches / second
- Processor queue length
Interesting counters to watch

- I/O operations / second
- I/O bytes / second
- Physical disk queue length
- Logical disk\avg sec / read or write
  - xlog: <10ms
  - data: <50ms
- Logical disk\Disk transfers / sec
Monitoring with Process Explorer

- Add extra columns
  - Private bytes
    - Local backend memory
  - WS Shareable / WS Shared
    - Shared buffers + shared code
  - Virtual Size
    - Just address space
  - Context switch delta
    - Expensive on Windows!
Why PostgreSQL on Windows
PostgreSQL for the Windows user
Windows for the PostgreSQL user
Advances in 8.3
Main build is now built with Visual C++
  - More efficient binaries
  - Works with Windows Debugger Tools
  - Works with Visual Studio debugger and profiler
  - Support for detached symbols
Windows debugger support in 8.3

- Debugger backtrace of mingw build

![Debugger Backtrace Image]

 WARNING: Stack unwind information not available. Following frames may be wrong.

- ntdll!KiFastSystemCallRet
- postgres!pgwin32_waitforsinglesocket+0x217
- postgres!pgwin32_recv+0x8b
- postgres!secure_read+0x2d
- postgres!TouchSocketFile+0x93
- postgres!pq_getbyte+0x22
- postgres!PostgresMain+0x1058
- postgres!SubPostmasterMain+0x584
- postgres!main+0x568
- postgres+0x11e7
- postgres+0x1238
- kernel32!RegisterWaitForInputIdle+0x49
Windows debugger support in 8.3

- Debugger backtrace of msvc build
Windows debugger support in 8.3

- Debugger backtrace of msvc build
Windows debugger support in 8.3

- Local symbols supported!
Using windbg

- Step 0 – configure symbol directory
Using windbg

- Step 1 – attach to running backend
- Figured out pid using previous methods
Using windbg

PID 1068 - WinDbg: 6.6.0007.5

Command - PID 1068 - WinDbg: 6.6.0007.5

```
ModLoad: 76f60000 76f8c000 C:\WINDOWS\system32\WIDAP32.dll
ModLoad: 7d9c0000 7d1d5000 C:\WINDOWS\system32\SHELL32.dll
ModLoad: 77f60000 77fd6000 C:\WINDOWS\system32\SHLWAPI.dll
ModLoad: 10200000 10321000 C:\pgmsvc\bin\MSVCR80D.dll
ModLoad: 76390000 763ad000 C:\WINDOWS\system32\IMM32.DLL
ModLoad: 773d0000 774d3000 C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls
ModLoad: 5d090000 5d12a000 C:\WINDOWS\system32\comctl32.dll
ModLoad: 71a50000 71a8f000 C:\WINDOWS\system32\mswsock.dll
ModLoad: 662b0000 66308000 C:\WINDOWS\system32\hnetcfg.dll
ModLoad: 71a90000 71a98000 C:\WINDOWS\System32\wshtcpip.dll

(42c.190): Break instruction exception - code 80000003 (first chance)
eax=7ff68000 ebx=00000001 ecx=00000002 edx=00000003 esi=00000000 edi=00000005
eip=7c901230 esp=0148ffe8 ebp=0148ffe4 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=0038 gs=0000 efl=00000246
*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\WINC
ntdll!DbgBreakPoint:
7c901230 cc
```

O: 0x003

Ln 0, Col 0  Sys: <Local>  Proc 000:42c  Thrd 003:190  ASM  OVR  CAPS  NUM
Using windbg

- Step 3 – set a breakpoint

```plaintext
*** ERROR: Symbol file could not be found
ntdll!DbgBreakPoint:
7c901230 cc     int     3
0:003> bp int8in
*** WARNING: Unable to verify checksum
```
Using windbg

- Step 4 – run and hit breakpoint

- Step 5 – load whatever views are needed
Visual Studio debugger

- Much better actual debugger
- Work off symbols or source tree
- Suitable for development, not production
Thank you!

Questions?