TRANSITIONING FROM WINDOWS TO POSIX

DIFFERENCES

- gcc vs. g++
- No initialization
- No “socket” type
- Separate header files
- errno & error constants
- File handles vs. sockets
- Non-blocking IO differences

gcc vs. g++

Issue
Visual Studio just has one compiler, and picks the compilation mode (C vs. C++) based on filename. GNU has separate compilers for straight-C and C++.

Warning Sign
Error messages like “undefined reference to __gxx_personality_v0”.

Solution
Use the right compiler for your project--gcc for C, g++ for C++.

Google is your friend.

No Initialization or Cleanup

Issue
Winsock is a separate library that has to be loaded and unloaded. POSIX sockets are integral to the OS, so there’s no equivalent to WSAStartup() or WSACleanup().

Warning Sign
Error messages like “WSAStartup was not defined in this scope”.

Solution
Surround platform-specific code with conditional blocks:

```
#ifdef WIN32
// Do Windows things
#else
// Do *nix things
#endif
```

Don’t Repeat Yourself! Hide platform-specific code behind consistent APIs within your program.

No “socket” Type

Issue
POSIX sockets are simply ints; there is no “socket” type.

Warning Sign
Error messages like “socket does not name a type”.

Solution
Create a conditionally-compiled typedef for socket.

```
#ifndef WIN32
typedef int socket;
#endif
```

Or, in C++, hide sockets entirely behind a custom class.

Separate Header Files

Issue
All of Winsock is in one header file. POSIX socket code is spread across several.

Warning Sign
Error messages like “socket was not defined in this scope”.

Solution
Typing “man *function*” in a command shell will tell you what header file you’re missing.

Make one “sockets.h” header that includes all the others.

```
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
```

errno and Error Constants

Issue
The POSIX equivalent of WSAGetLastError() is a global variable, errno.

Error constants are different--strip the “WSA” part off.

Warning Sign
Error messages like “WSAECONNRESET was not defined in this scope”.

Solution
Create your own “GetLastError()” function, wrapping a conditional-compilation block.

Define your own error codes for the errors you care about… or better yet use an enum for clarity and type-safety.

File Handles vs. Sockets

Issue
POSIX sockets are the same as file handles. Existing POSIX code may take advantage of this.

Also, the same close() call destroys both, unlike Winsock.

Warning Sign
Error messages like “closesocket was not defined in this scope”.

Solution
When reading POSIX code, understand that read() is equivalent to recv().

When writing portable code, avoid usage like that.

Call close() instead of closesocket()... and again, hide this behind an internal API.

Non-blocking IO Differences, Part I

Issue
As with close() vs. closesocket(), you have to call a general-purpose function rather than ioctlsocket().

You can *mostly* rely on ioctl()... mostly.

Warning Sign
Error messages like “ioctlsocket was not defined in this scope”.

Solution
You can call ioctl() with the exact same parameters as ioctlsocket(), on virtually every *nix you encounter.

Formal POSIX standard specifies fcntl(), which is inconvenient.

Hide this from the rest of your program.

continued...

Non-blocking IO Differences, Part II

Issue
The POSIX standard defines the “call would block” response as EAGAIN, not EWOULDBLOCK. But older Unixes still use EWOULDBLOCK.

Warning Sign
You expect EWOULDBLOCK, but you get some unrecognized other error value.

Solution
Whenever you check for EWOULDBLOCK, also check for EAGAIN at the same time.

The compiler knows what values these actually represent, and will optimize away the duplication wherever possible.

As always, hide this from the rest of your program.

SUMMARY

- Understand the gcc toolchain
- Use conditional compilation
- Hide calls, types and return values/error codes behind your own API
- While you’re at it, solve the getaddrinfo() blocking problem as well.