Porting D to a non-Windows non-Posix platform

Igor Česi, Ubisoft Paris

D@Ubisoft

- Why are we interested
  - Compilation speed
  - Memory safety
- POC objectives
  - Port to a target platform
  - Evaluate dev tools
  - Estimate above points (if possible)

Porting

- Porting process
  - Make it compile (stubbing)
  - Run (ideally tests) and check what is not working
  - Fill in the gaps and try again

Agenda

- Compiler
- druntime
- Phobos

- POC results

Compiler

Choosing your compiler

- Depending on the hardware platform the choice can be large or very limited
  - x86/x64 – all available compilers
  - Exotic hardware – limited choice (gdc and/or ldc)
- Choice can be natural given the target software platform
  - Platform provider uses llvm/clang or gcc
  - Platform provider is MS

Choosing your compiler

- In my case provider uses LLVM/Clang => LDC is natural choice
- But does not provide LLVM libraries necessary to compile LDC
- Platform modifications are closed source
- Need to start from upstream LLVM and do necessary modifications myself

Compiling the compiler

- Problem: Choose the right triple
  - Choosing a well known architecture (linux/android) can bring too many things which your platform does not support
  - Unknown-unknown is not valid for LDC
  - My choice: unknown-haiku, brings a smallest part of suppositions on OS, good start in my case
  - Other choice: add support for your platform in LLVM, a separate problem, will be probably done in the future (when all works)

Compiling the compiler

- Instructions to build a compiler on the net are quite good and precise

Testing the compiler

- Simple ‘C’ like code with ‘betterC’ switch is a good way to see if your compiler works, and if it can be run on the target platform
- Opportunity to validate the debugger
  - Source debugging
  - Breakpoints
  - Watch points
  - Etc…

About versions

- LDC @ v1.4.0
- Frontend, druntime, Phobos @ 2.074.1

Some specific problems

- ‘Command line length’ problem when compiling LDC
  - (move source code up the folder hierarchy is a ‘sad but true’ solution)
- Bug in CMakeFiles.txt when compiling LDC with Visual Studio (and not Ninja) (link flags separator problem)
  - Check if this problem still exists and propose a patch

Dev environment

- Visual D
  - target toolchain is Windows/Visual Studio based
  - Use C++ projects (and not visual D projects) to simplify and match the expected final usage
- Need to add support for target platform
  - End result makes no modifications to VisualD (only property sheet additions which can be installed separately to ImportBefore/ImportAfter folders for the target platform)
- Learned how to debug MSBuild scripts in VS (it is possible!)

druntime

Quest for unit tests

- Get the minimum building to be able to run unit tests
  - Minimum is BIG!
    - Memory allocations
    - Exceptions
    - TypeInfo
    - ModuleInfo
    - stdc
    - And all the dependencies…

Brute force approach…

- Brute force strategy
  - Compile a file with –unittest and try to fix compilation/link problems
  - Stub complicated stuff
  - Try to run and see where it crashes
  - Fix and repeat

Brute force approach… Failure!

- Ended up with the crash in dynamic loader/linker, much before the main is reached. No sources for it, hard to figure out what was wrong.

One step at the time…

- One step at the time strategy
  - Have something compiling and running all the time
  - Start with empty C++ main()
  - Implement rt_init()/rt_term() step by step
  - Stub whatever has a lot of dependencies
  - Introduce new things one at the time

One step at the time…

- Quickly validated simple stuff
  - extern(C) functions
  - D interfaces to platform SDK
  - C style allocations (using gcstub)
- Used custom tests and/or debugger for validation

One step at the time…

- ‘new’ allocation brings in a lot of dependencies
  - object, lifetime, TypeInfo, …
- Generously stubbed all complex stuff to get it compile and run…

Dynamic loader/linker strikes again!

- Hit by the crash in the dynamic loader/linker again!
  - Able to compare running and crashing version
  - Problem => wrong relocation model (static instead of PIC)
  - Solution => patch the compiler to make PIC default for my platform

Quest for unit tests… almost there!

- Exceptions/Asserts are easy to get working once ‘new’ is functional
- Need working ModuleInfo to be able to find existing unit tests …

ModuleInfo

- To get ModuleInfo you need a help of the compiler (and linker)
  - Check RegistryStyle class in LDC for existing options (legacy, ELF, Darwin)
  - Check TargetOptions for choice between global constructors/destructors and init_array

ModuleInfo

- druntime ‘sections’ implementation must match the choice in the compiler
  - Several (different) implementations exist in druntime (for different platforms)
  - Hopefully one matches your platform (not my case unfortunately)

One step at the time… Success!

- Unit tests work!

Filling the gaps

- How to find platform specific code?
  - static assert(false, “Unsupported platform”) indicates which files need attention
    - Does not cover all the cases however (some files have defaults which might not be applicable to your case)
  - Once file identified, need to analyze the whole file for platform specific bits
    - There is usually only one static assert per file
    - Can be challenging for bigger files
    - Do not forget to grep for ‘version’ keyword

GC, threads, TLS…

- GC needs virtual memory, threads and sections to work

GC, threads, TLS…

- GC needs virtual memory, threads and sections to work
  - Virtual memory => use malloc/free for now

GC, threads, TLS…

- GC needs virtual memory, threads and sections to work
  - Threads => must be able to Suspend/Resume
    - Not in the public interface in my case (blocker!)
    - Luckily available as symbols, exposed via custom headers (huh)

GC, threads, TLS...

- GC needs virtual memory, threads and sections to work
  - Sections => must be able to register BSS and TLS memory with GC
    - Your linker might expose necessary symbols (__bss_start/end, __tdata_start/end, etc.)
    - Your C runtime might expose __tls_get_addr() to obtain TLS address for a thread
    - Check existing implementations too, they might match your case

Extern(C++) mangling problem

- C++ uses substitution for namespaces while mangling
- DMD will not use substitution if the symbol comes from a different module
  - Missing feature or a bug?
- Workarounds
  - Put everything in the same file
  - Use pragma(mangle) to adjust the mangling for problematic functions/methods

Porting druntime - recap

- Main challenges
  - Getting unit tests running
  - Sections implementation for GC

Phobos

80/20 rule

- LOC
  - 80% of code in Phobos is either platform agnostic or has a simple dependency to platform specific code (e.g. ascii.d)
  - 20% of code is platform specific
- Time
  - 20% of time spent to get 80% of Phobos working
  - 80% of time spent to get the platform specific code ported and working

Platform specific code

- Two categories
  - Platform supports the feature (easy)
  - Platform does not or partially supports the feature (time consuming, there are choices to make)
    - Drop support, implement partially or emulate fully?
    - Porting cost vs planned usage
    - Is needed by other Phobos packages/features?

Partial support examples

- std/datetime

Partial support examples

- std/datetime
  - Platform supports only dates between year 2000 and 2100

Partial support examples

- std/datetime
  - Platform supports only dates between year 2000 and 2100
  - Date and TZ implementations are possible but limited

Partial support examples

- std/datetime
  - Platform supports only dates between year 2000 and 2100
  - Date and TZ implementations are possible but limited
  - Many unit tests to adapt or disable (use dates outside the supported range)

Partial support examples

- std/datetime
  - Platform supports only dates between year 2000 and 2100
  - Date and TZ implementations are possible but limited
  - Many unit tests to adapt or disable (use dates outside the supported range)
  - Will it really be used?
  - Other code depends on it (file.d), must have at least minimal implementation

Partial support examples

- File/path – getcwd()

Partial support examples

- File/path – getcwd()
  - unistd version exists but is not supported (always returns nullptr)

Partial support examples

- File/path – getcwd()
  - unistd version exists but is not supported (always returns nullptr)
  - Does not have the same meaning as in general purpose OS
    - Executable starts with no storage space available/mounted

Partial support examples

- File/path – getcwd()
  - unistd version exists but is not supported (always returns nullptr)
  - Does not have the same meaning as in general purpose OS
    - Executable starts with no storage space available/mounted
  - Could be (probably) fully emulated
    - Cost vs usage
  - Something must be implemented because of dependencies

Random bits

- Some code is crashing the compiler (needs further investigation)
  - Some unit tests in std/format.d
  - std/outbuffer.d
- Return of malloc(0) is implementation defined
  - numeric.d (MakeLocalFft()) expects a valid pointer which is not guaranteed!

Porting Phobos - recap

- Main challenge
  - Finding the balance between invested time and expected usage when porting partially supported platform specific features

POC results

No blockers!

- Running D on target platform is possible
- Dev tools exist and work

POC continues…

- Presented work represents ~3-4 man/months
- Finish porting to target platform
- Increase dev tools comfort and reliability
- Collect more data on the compilation speed
  - Will need real production code if possible

Questions?

Thank you!