

Linux Porting Experiences

David Boyes
Session L330



Agenda

- Is Linux the right platform?
- When NOT to do a port
- Overview of Linux development environment
- System and development tool changes
- System and development tool semantics
- Case Studies



Is Linux the Right Platform?

- Often this isn't a question we can influence – we need an application in the Linux environment NOW.
- Still, questions are valid – Linux isn't the right solution every time.



Applications Well Suited to Linux Ports

- Tools with existing Linux ports on other architectures
- Tools existing in the commercial Unix world that do not depend on vendor-specific HW or function
- Tools depending on stream data processing (ie, filters)




When To Try to Avoid A Port

- The application uses internal OS interfaces
- The application is geometry or HW configuration sensitive in ways that cannot be handled by typedef or casting at the source level
- The application uses vendor extensions to APIs outside of 4.3/4.4BSD or SVID semantics




Overview of Linux Development Environment

- Four areas of developer tools:
 - Editing and text processing
 - Compilers and object code management
 - Source code management
 - Deployment and packaging tools



Overview

- Linux development suite based heavily on GNU Project tools
- Primarily text-based environment (few GUI-based developer tools)
- Usual edit-compile-test cycle holds.



Overview

- Representative applications:
 - emacs
 - gmake
 - gcc/g++/glibc
 - binutils
 - cvs/rcs
 - RPM



Overview

- GNU tools are often substantial enhancements to vendor-supplied development tools and syntax may be different or enhanced.
- Some program options may not make sense on all Linux architectures, or worse yet, do something different. This is accepted behavior in the Linux world.



Common Linux Dev Model

- Where possible, use of autoconf is desirable.
- Makefiles are mandatory
- Use of CVS repository is highly recommended




Changes to System Dev Tools

- In general, Linux development tools are SVID and POSIX compliant
- Notable exceptions:
 - make (Berkeley extensions)
 - bison/flex (substantial extensions to BSD syntax)
 - gcc/g++ (strict ANSI C/C++ syntax)
 - C library (memory management functions)




Circumventions

- Make
 - Careful reordering of dependencies can eliminate requirement for BSD extensions
 - Note that default build rules for .c and .o have changed from BSD and SVID sources
 - Autoconf, autoconf, autoconf....



Circumventions

- Bison/flex
 - Most BSD grammars work without change
 - -traditional flag to flex allows lax BSD syntax for incomplete LLR grammars
 - Full SVID grammars often need some tweaking to adapt to bison syntax - f=strength-reduce -traditional flags help here.



Circumventions

- gcc/g++
 - Time to convert to ANSI C
 - -traditional -fstrength-reduce permits most BSD syntax
 - (advantage) cross-compilers!
 - Many vendors now shipping gcc as default C compiler in place of proprietary compiler (see Sun and acc as an example)



Circumventions

- C library
 - Some functions require kernel support (bzero, memcpy) – SuSE defect in 2.2.16 kernel.
 - Assume SVID/POSIX interfaces when possible.



Case Study: ISP Accounting

- 10 linked C applications driven by Perl script
- Moved from Solaris to Linux for S/390
- Output: monthly/weekly usage summaries
- Use of a lex grammar to parse syslog output



Case Study: ISP Accounting

- Step 1: Verify lex grammar

```
flex -syntax-only plex.flex
```



Case Study: ISP Accounting

- Step 2: Compile and link applications
 - Use existing makefile (no BSD extensions)

```
make -f Makefile > /tmp/foo &
```



Case Study: ISP Accounting

- Step 3: Verify Perl version and libraries
 - Required Perl 5.004, Perl 5.04 present.

```
perl -v
```



Case Study: ISP Accounting

- Step 4: Test and Implement

```
./report
```



Case Study: OpenAFS

- Kernel interface module
- Client/server
- Very, very OLD code
- Deep ties to internal kernel structures
- Probably a bad choice for a port, but we didn't know it at the time...8-)



Case Study: OpenAFS

- Compile Modules
 - Debug SuSE non-export of kernel memcpy function.
 - make takes literally hours to process due to ancient dependency checking.
 - Dealing with:

```
/* This code will be removed in the rewrite in 1989 */
```

in many, many places...



Case Study: OpenAFS

- Compile Modules
 - Linux pthreads implementation incompatible with CMU pthreads.
 - Assumption of int = char representation.
 - '1' and x'00001' are NOT identical on S/390 (fixing improper casting of pointer)
 - Fencepost errors in buffer management



Case Study: OpenAFS

- Compile Modules
 - Interface to setjmp() internal task buffer organized very differently on S/390 than other platforms
 - Implementation of context switch assembler routines (Linux assembler is NOT OS linkage compliant).




Summary

- Linux porting is straightforward with well-written code.
- Most development skills on major Unix variants transfer w/o change.
- Once a Linux port on any architecture is done, it is likely to be directly useful on other architectures due to peer review.




Questions



Contact Information

David Boyes
dboyes@sinenomine.net
+1 703 783 0438





Presentation Foils

- www.sinenomine.net/downloads/vmvse