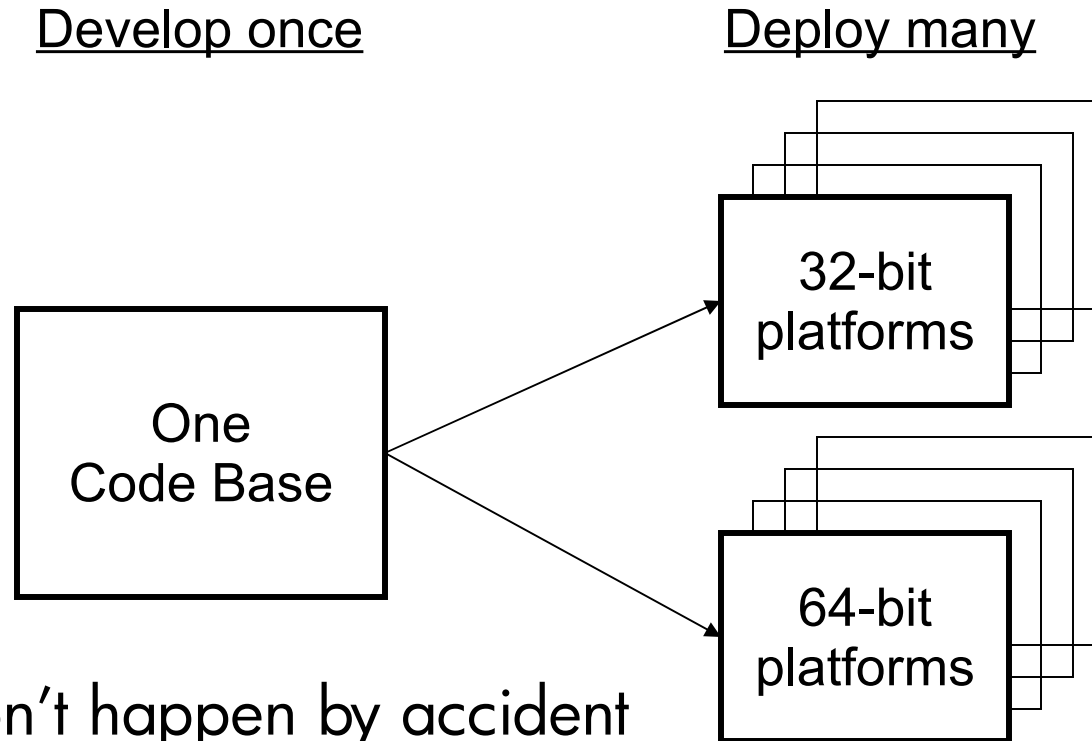


64-Bit Migration to Linux on Itanium Architecture



The Holy Grail



- This won't happen by accident
 - Rearchitect where necessary
 - Isolate platform dependencies
 - Use standard code practices

Differences from x86 Linux

- General user
 - Same look & feel
 - Can run huge apps with no problem
 - Runs most existing x86 apps:
 - realplayer, acroread, netscape, even some Windows apps (via WINE)
- Application developer
 - Vast majority of programs written in C, C++, and FORTRAN
 - just recompile
 - Java:
 - no portability issues: just run it
 - JVM availability: BEA JRockit, Sun JVM, or GCJ/IA-64
 - Assembler
 - Need to be re-written (in C/C++)

What is 64-bits?

- A 32-bit processor addresses 2^{32} bytes (4GB)
- A 64-bit processor addresses 2^{64} bytes (18EB)
 - **byte kilobyte megabyte gigabyte terabyte petabyte**
exabyte zettabyte yottabyte
- 64-bit cpu does 64-bit integer math in one go
- More data in memory, more data used per cycle
- Scalability! Performance!

What are the 32-bit and 64-bit programming models?

- LP64 (longs and pointers are 64-bits) all else is the same as ILP32
- Some derived types are 64-bits, i.e. filesystem types
- Sometimes you don't need 64 bits, but you need 33!
- What does ANSI specify?

Datatype	LP32	ILP32	LLP64	LP64	ILP64
char	8	8	8	8	8
short	16	16	16	16	16
_int32					32
int	16	32	32	32	64
long	32	32	32	64	64
long long			64		
pointer	32	32	64	64	64
fpos_t	32	32	64	64	64
off_t	32	32	64	64	64
size_t	32	32	64	64	64
ssize_t	32	32	64	64	64
time_t	32	32		64	64
ptrdiff_t	32	32	64	64	64

C/C++ Portability Hints

- LP64 data model
 - All 64-bit UNIX & Linux platforms use this data model
 - `sizeof(void*) = sizeof(long) = 8 bytes`
 - Rest is the same as for x86 (ILP32)
- How to prepare your code
 - Develop with change in mind
 - Don't be causal about data types
- Compiler Tips
 - gcc and Intel
 - Newer is usually faster
- Other stuff
 - Port your apps with LSB conformance in mind
 - Take advantage of existing multi-threading support

How do I prepare my code?

- Make your code 64-bit clean
 - Remember pointers and longs != int
 - casts of pointers to int
 - unions
 - use `uintptr_t` and `ptrdiff_t`
 - printf: use **%p** to print pointers, **%ld** for longs, **%Zu** for vals of `size_t`
 - check for inconsistent usage of int and long
 - If you interchange data types frequently, make a helper function
 - Remember to type constants
 - `0xFFFFFFFF00` is unsigned int, `0xFFFFFFFF00L` is signed long
- Take advantage of ISO/ANSI-9x

```
p = (char *)((intptr_t)p & 0xFFFFFFFFFC); WRONG
```

```
p = (char *)((intptr_t)p & ~0x3); RIGHT
```

Types

- Standards provide for a number of types the explicit size
 - `int32_t`, `uint32_t`, etc
 - `ssize_t`, `size_t`
 - `ptrdiff_t`, `intptr_t`
- Constants
 - Decimal constants are signed ints (32-bit) by default
 - Hex Constants are unsigned ints (32-bit)
 - `0xFFFFFFFF` unsigned int
 - `0xFFFFFFFFL` signed long, high bits not set on 64-bit
 - `0x7FFFFFFF` signed int

Function prototypes

- Return values in C are “`int`” unless specified
 - `malloc()` is not prototyped in `<malloc.h>`
- “`int`” values passed to prototyped functions, will be promoted as need by the compiler.
- C99 added support for `printf/scanf` for 64-bit values
 - `PRId32`, `PRId64`, `SCNd32`, `SCNd64`, etc

How do I prepare my code?

– use `sizeof()` and pointer arithmetic

```
char *p, *q;
```

```
p = malloc(4 * N);
```

```
q = (char *) ((int)p + 4 * N);
```

Wrong

```
p = malloc(sizeof(char *) * N);
```

Better

```
q = p + (N);
```

Best, don't use pointer arithmetic! This allows the compiler to flag the warning....

```
q = &p[N];
```

Some potential bugs

```
int i = -2;  
unsigned int k = 1;  
long n = i+k;
```

- What is n?
 1. -1
 2. 4294967295
 3. ?
- Why? How do you change it?

Some potential bugs

```
long m = 1 << 31;
```

- What is m?

- 1. 0x80000000

- 2. 0xFFFFFFFF80000000

- 3. 0x0000000080000000

- 4. ?

- Why? How do you change it?

Some potential bugs

```
struct _x {  
char s1[12];  
long l1;  
char s2 [16];  
};
```

- How big is struct _x? 32? 36? 40? 44?
- Where is _x.s2? +16? +20? +24?

Compiler Tips

- gcc for maximum portability:
 - gcc 4.x (e.g., gcc-4.1.1) preferred over older gcc3.x
 - gcc -Wall is your friend
 - Will warn you of questionable casts, (from pointer to “int”)
- Intel compilers for best IA-64 Linux performance:
 - Interoperates well with gcc-compiled binaries
 - -O2 or -O3 for best base-performance, PGO for peak performance
 - Free to use for open source developers
- Open64 Compilers
 - “New” compiler
 - Based on gcc with focus on Itanium performance
 - <http://www.open64.net>

Safe at 32 and 64. Yes/No?

```
{
int a1[2048];
int a2[2048];
int a3[2048];
long long* pa1 = (long long*) a1;
long long* pa2 = (long long*) a2;
long long* pa3 = (long long*) a3;
    for (int i = 0; i < sizeof(a1) / sizeof(long
long); ++i)
    {
        pa3[i] = pa1[i] & pa2[i];
    }
}
```

GCC

- -Wall -pedantic (-pedantic-errors)
- -Wpointer-to-int-cast
 - foo.c:6: warning: passing arg 1 of `foo' makes pointer from integer without a cast
 - test.c:12: warning: initialization makes integer from pointer without a cast
- -Wstrict-prototypes: make sure all functions have full prototypes
 - test.c:7: warning: function declaration isn't a prototype

GCC

- `-Wmissing-prototypes`: make sure all functions are declared in a header file
 - `test.c:7: warning: no previous prototype for 'foo'`
- `-Wimplicit` : warn about implicit declarations
 - `test.c:9: warning: implicit declaration of function 'malloc'`

Intel CC

- Free for non-commercial use, licensed for commercial use
- `icc -Wp64`
 - print diagnostics for 64-bit porting
 - `foo.c(6): warning #967: conversion from "int" to "int *"; sizes do not match`

Open64 Compilers

- <http://www.open64.net>
- Supports GCC switches
- Goal is better performance for Itanium

Secure Programming Lint

- <http://www.splint.org>
- `splint -checks -nolibs test.c`
 - `test.c:9:12: Unrecognized identifier: malloc`
 - `test.c:10:19: Variable k initialized to type int, expects unsigned int: 1`
 - `test.c:11:11: Variable n initialized to type int, expects long int: i + k`
 - `test.c:10:10: Variable m initialized to type int, expects long int: 1 << 31`

Source OS tools

- AIX
 - lint -t
- Tru64 UNIX
 - lint -Q
- HP-UX
 - lint +M2
 - HP STK tools
- Solaris
 - lint -errchk=longptr64,signext

Endianism does play a role

Low address

High address

Little Endian Byte 0 Byte 1 Byte 2 Byte 3 Byte 4 Byte 5 Byte 6 Byte 7

Big Endian Byte 7 Byte 6 Byte 5 Byte 4 Byte 3 Byte 2 Byte 1 Byte 0

For example, the value decimal 1025 (hexadecimal 0x0041) looks significantly different when examined as an array of bytes:

Little Endian

`char*[0] = 0x01`

`char*[1] = 0x04`

`char*[2] = 0x00`

`char*[3] = 0x00`

Big Endian

`char*[0] = 0x00`

`char*[1] = 0x00`

`char*[2] = 0x04`

`char*[3] = 0x01`

Summary – Steps to portability

- Be disciplined in your use of all data types
- Understand where 64-bits will make a difference
- Ensure that your 32-bit code works and builds correctly – before you start a port
- Understand and correct bad code
- Plan for change, odds are 64-bit targets aren't the last

Resources

- GCC/ICC/OpenCC – Use the compiler warnings
- SPLINT – Open source lint www.splint.org
- “Porting an Application to 64-bit Linux on HP Integrity Servers”
 - <http://www.hp.com/go/LinuxDev>
- “Building Applications with the Linux Standard Base”
 - http://www.linux-foundation.org/en/LSB_Book
- “Solaris to Linux Porting Guide”
 - <http://www.hp.com/go/LinuxDev>

Programs

- DSPP/Linux Expertice Center
 - <http://www.hp.com/partners/LEC>
- HP Itanium Developer Workshops
 - 3 days of hands on porting of your app with HP experts to help
 - Walk away with an Itanium and a completed port!
 - <http://www.hp.com/go/ItaniumDeveloperWorkshops>

Questions?

