Linux for zSeries

Early Experiences with 64-bit Linux
Agenda

- z/Architecture Overview
- Linux implementation for z/Architecture
- ABI changes
- Early experiences with ThinkBlue64
- Early experiences with SuSE system
- Early experiences with Redhat
Linux for zSeries

z/Architecture Overview
z/Architecture Overview

• z/Architecture is the next step in the evolution from the System/360 to the System/370, S/370-XA, ESA/370, and ESA/390.

• z/Architecture includes all of the facilities of ESA/390 except for the asynchronous-pageout, asynchronous-data-mover, program-call-fast, and vector facilities.
Four key features of z/Architecture include:

- It is a full 64-bit architecture that provides for 24, 31 and 64-bit coexistence.
- Intelligent Resource Director—Provides for an exclusive way to intelligently direct the processor and I/O resources to priority workloads running within the set of clustered LPARs.
- HiperSockets—An internal facility for z/Architecture that permits a TCP/IP network to be established between LPARs.
- License Manager Enablement—The z/Architecture includes capabilities that enable IBM's License Manager to run on z/OS and z900. This capability, when combined with HiperSockets, creates an ‘n-tier’ environment for e-business applications within a z900.
z/Architecture Overview

- 64 bit PSW
  - Bit 12 – ‘0’ specifies z/Architecture
- 64 bit control registers
- 16 IEEE/HFP registers
  - No need for software emulation
• 64 bit general registers
  – Can be operated upon as 64 or 32 bit entities

```c
#include <stdio.h>
int main(int argc, char **argv)
{
  union { long x; int y[2]; } longvar;

  longvar.x = -1;
  printf("%08X %08X %ld\n",longvar.y[0],longvar.y[1],longvar.x);
  __asm__ __volatile__ ("slr %0,%0" : "+d" (longvar.x) : : "cc");
  printf("%08X %08X %ld\n",longvar.y[0],longvar.y[1],longvar.x);
  __asm__ __volatile__ ("slgr %0,%0" : "+d" (longvar.x) : : "cc");
  printf("%08X %08X %ld\n",longvar.y[0],longvar.y[1],longvar.x);
}

FFFFFFFF FFFFFFFF -1
FFFFFFFF 00000000 -4294967296
00000000 00000000 0
```
z/Architecture Overview

- 64 bit addressing
  - 24 bit support
  - 31 bit support
  - Up to 3 levels of “Region Tables” to give:
    - 42, 53, 64 bit addressing
    - Use `samxx` instruction to switch addressing modes
- New term:
  - >16MB = “above-the-line”
  - >2GB = “above-the-bar”
z/Architecture Overview

- 32 bit Access Registers
- CCWs still only use 31 bit address fields
  - IDAL used for “above-the-bar”
• Prefix page now 8KB
• LOTS of new instructions
  – 64 bit versions of 32 bit ops: LG (load) = L (load)
  – Instructions to manipulate 32 bit entities: LGFR
  – Some new compiler-friendly: RLL/RLLG; ALC/ALCG
  – Address mode related: SAM24/31/64; TAM
  – Unicode support: CUUTF; TRE **
  – Enhanced relative branching: +/- 2GB branches
## z/Architecture Overview

- New old/new PSW locations

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>Old PSW</th>
<th>Old Base</th>
<th>Old Offset</th>
<th>New Base</th>
<th>New Offset</th>
<th>New PSW</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT</td>
<td>1004</td>
<td>130</td>
<td>07060001</td>
<td>80000000</td>
<td>00000000</td>
<td>00015F1A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1B0</td>
<td>04000001</td>
<td>80000000</td>
<td>00000000</td>
<td>00014D32</td>
<td></td>
</tr>
<tr>
<td>SVC</td>
<td>008E</td>
<td>140</td>
<td>0701C001</td>
<td>80000000</td>
<td>00000200</td>
<td>002618A6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1C0</td>
<td>04000001</td>
<td>80000000</td>
<td>00000000</td>
<td>0001406C</td>
<td></td>
</tr>
<tr>
<td>PRG</td>
<td>0004</td>
<td>150</td>
<td>07004001</td>
<td>80000000</td>
<td>00000000</td>
<td>00087C7A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1D0</td>
<td>04000001</td>
<td>80000000</td>
<td>00000000</td>
<td>00014AD6</td>
<td></td>
</tr>
<tr>
<td>MCH</td>
<td>0000</td>
<td>160</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00014DEA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1E0</td>
<td>04000001</td>
<td>80000000</td>
<td>00000000</td>
<td>00014C3A</td>
<td></td>
</tr>
<tr>
<td>I/O</td>
<td>0004</td>
<td>170</td>
<td>07060001</td>
<td>80000000</td>
<td>00000000</td>
<td>00015F1A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1F0</td>
<td>04000001</td>
<td>80000000</td>
<td>00000000</td>
<td>00014C3A</td>
<td></td>
</tr>
</tbody>
</table>
z/Architecture Overview

• Implemented on:
  – z900 (aka Freeway) processors
  – Hercules
    • No SIGA/SERVC - proprietary
  – Flex/ES (or should be in the future)

• Supported by:
  – z/VM
  – OS/390
  – Linux for zSeries
Linux for zSeries

Linux Implementation for z/Architecture
Linux for zSeries

- Based on 2.4 kernel
- Requires:
  - binutils
  - gcc
  - glibc
- Boots in 31 bit mode
- Switches to 64 bit mode fairly quickly
Linux – Intel Address Spaces

0xFFFFFFFF 4GB Himem

User Space Himem
(typically 0xC0000000 3GB)

Next
To
Run

Kernel

User Stack

Shared Libs

User Program
Data BSS
Text
Sections

0x00000000
Linux – S/390 Address Spaces

0x7FFFFFFFF 2GB Himem

User Stack

Shared Libs

User Program
  Data BSS
  Text
  Sections

Kernel

0x00000000
Linux – zSeries Address Spaces

0x3FFFFFFF 4TB
Himem

User Stack

Shared Libs

User Program
Data BSS
Text
Sections

Kernel

0x00000000
Linux for S/390 & zSeries

- A virtual address on S/390 is made up of 3 parts:

<table>
<thead>
<tr>
<th>Segment Index</th>
<th>Page Index</th>
<th>Byte Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>20</td>
<td>31</td>
<td></td>
</tr>
</tbody>
</table>

- On z/Architecture in Linux we currently make up an address from 4 parts:

<table>
<thead>
<tr>
<th>XXXXXXXXXX</th>
<th>Region Index</th>
<th>Segment Index</th>
<th>Page Index</th>
<th>Byte Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>22</td>
<td>33</td>
<td>41</td>
<td>52</td>
</tr>
</tbody>
</table>
Linux for zSeries

- 64-bit
- 4TB address spaces
  - 1 Region Table
  - Segment Table
  - Page Table
- 31-bit compatibility mode
  - Existing apps will run
  - Provided they can find their libraries!
  - Problems with some APIs (e.g. `shmctl()`)
  - Work done for co-existence: `/lib64` & `/lib`
zArchitecture Address Spaces

PGD → PGM → PGT

Region Table → Segment Table → Page Table

0x0000003FFFFFFFFF (4TB)

0x0000000000000000
Address Spaces

- Kernel runs in Primary Space mode
- User programs run in Home Space mode
- Copy to/from user just a MVC(L/E) in Access Register mode with AR set for kernel/user address spaces
- Compare this to some of the other elaborate schemes used
### Address Space Usage

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Permissions</th>
<th>Start Address</th>
<th>End Address</th>
<th>Size</th>
<th>User</th>
<th>Group</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000800000000-000000080008000</td>
<td>r-xp</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>5e:01</td>
<td>207901</td>
<td>/bin/more</td>
<td></td>
</tr>
<tr>
<td>0000000800008000-000000080009000</td>
<td>r-w-p</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>5e:01</td>
<td>207901</td>
<td>/bin/more</td>
<td></td>
</tr>
<tr>
<td>000000080009000-00000008000d000</td>
<td>rwxp</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>00:00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000200000000000-0000200000001b00</td>
<td>r-xp</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>5e:01</td>
<td>223562</td>
<td>/lib/ld-2.2.2.so</td>
<td></td>
</tr>
<tr>
<td>0000200000001b00-0000200000001d00</td>
<td>r-w-p</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>5e:01</td>
<td>223562</td>
<td>/lib/ld-2.2.2.so</td>
<td></td>
</tr>
<tr>
<td>0000200000001d00-0000200000001f00</td>
<td>r-w-p</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>00:00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000200000002400-0000200000002800</td>
<td>r-xp</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>5e:01</td>
<td>223625</td>
<td>/lib/libtermcap.so.2.0.8</td>
<td></td>
</tr>
<tr>
<td>0000200000002800-0000200000002900</td>
<td>r-w-p</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>5e:01</td>
<td>223625</td>
<td>/lib/libtermcap.so.2.0.8</td>
<td></td>
</tr>
<tr>
<td>0000200000002900-00002000000017000</td>
<td>r-xp</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>5e:01</td>
<td>223567</td>
<td>/lib/libc-2.2.2.so</td>
<td></td>
</tr>
<tr>
<td>00002000000017000-000020000000179000</td>
<td>r-w-p</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>5e:01</td>
<td>223567</td>
<td>/lib/libc-2.2.2.so</td>
<td></td>
</tr>
<tr>
<td>000020000000179000-00002000000017f000</td>
<td>r-w-p</td>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td>00:00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000300000000000-0000000000000000</td>
<td>rwxp</td>
<td>ffffffffffffffff</td>
<td>ffffffffffffffff</td>
<td>00:00</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
New Device Drivers

- Tape
  - 3490
  - Character and block
- 3270
  - Console
  - Standard terminal
- Cisco Routers
- Hipersockets
- FCP (SCSI)
Device Drivers

- CCWs must live “below-the-bar”
- Kernel supports memory requests for under the bar storage (GFP_DMA)
- Device drivers build CCW programs in this storage
- IDALs used to address “above-the-bar” storage
Linux for zSeries

ABI Changes
The Executable and Linkage Format Application Binary Interface (or ELF ABI), defines a system interface for compiled application programs. Its purpose is to establish a standard binary interface for application programs on LINUX for S/390 systems.
Application Binary Interface

- Defines (amongst other things):
  - Data formats
  - Byte layouts
  - Stack layouts
  - Process initialization
  - Register conventions
  - Routine linkage
  - Parameter passing
  - Returning results
Application Binary Interface

- Changes required for 64-bit support
  - Stack layouts
  - Routine prologues
  - Register conventions
  - Parameter passing
- Transparent for compiled applications
- Need to understand for such things as “FFI” or “JNI” or writing compilers
## Stack Frame Layouts

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Back chain (a 0 here signifies end of back chain)</td>
</tr>
<tr>
<td>4 8</td>
<td>EOS (end of stack, not used on Linux for S390)</td>
</tr>
<tr>
<td>8 16</td>
<td>Glue used in other linkage formats</td>
</tr>
<tr>
<td>12 24</td>
<td>Glue used in other linkage formats</td>
</tr>
<tr>
<td>16 32</td>
<td>Scratch area</td>
</tr>
<tr>
<td>20 40</td>
<td>Scratch area</td>
</tr>
<tr>
<td>24−63</td>
<td>GPR register save area</td>
</tr>
<tr>
<td>64−79</td>
<td>FPR4 &amp; FPR6 save area</td>
</tr>
<tr>
<td>96 160</td>
<td>Outgoing args (length x)</td>
</tr>
<tr>
<td>96+x 160+x</td>
<td>Possible stack alignment</td>
</tr>
<tr>
<td>96+x+y 160+x+y</td>
<td>alloca space of caller (if used)</td>
</tr>
<tr>
<td>96+x+y+z 160+x+y+z</td>
<td>Automatics of caller (if used)</td>
</tr>
</tbody>
</table>
31 Bit Co-existence

- ELF header indicates executable as:
  - S/390
  - 31 bit/64 bit
- Dynamic executables contain information regarding location of shared libraries
- ld.so.1 or ld.64 resolves information in elf header
31 Bit Co-existence

- Use `ldd` command to show what libraries your executable requires
- 31 bit apps cannot use 64 bit libraries
- `LD_LIBRARY_PATH` environment variable overrides internal specification of executable
- Can be set up globally or per application
- Look out for 2.1.3 glibc & 2.4 kernel disparities
31 Bit Co-existence

- SuSE have /lib64 and /lib
- Apps migrated from 31-bit will find their libraries
- Programs built on 64-bit system will look in /lib64
Linux for zSeries

ThinkBlue64 – Early Experiences
ThinkBlue64

- Redhat-like distribution
- 7.1 now available
- Download from [http://linux.zseries.org](http://linux.zseries.org)
- CDROM ISO image available
- 749 RPMS
- Starter system:
  - Kernel (tape or VM reader)
  - Initial RAMDISK (tape or VM reader)
  - Parameter file
• glibc-2.2
• Kernel 2.4.3 (2.4.5)
• Hard IRQ bug in ctcmain
• skb_buff problem with ctc
• Heaps of RPMS!
• Starting (using NFS):
  – Mount CDROM on another Linux system:
    ```shell
    mount -o loop ThinkBlue64-disc1.iso /mnt/cdrom
    ```
  – Add `/mnt/cdrom` to `/etc/exports` and restart NFS server
    ```shell
    /etc/rc.d/nfsserver restart
    ```
  # See exports(5) for a description.
  # This file contains a list of directories exported to other computers
  # It is used by rpc.nfsd and rpc.mountd.
  ```text
  /mnt/cdrom 10.20.45.7(rw;no_root_squash)
  ```
ThinkBlue64

- Starting
  - New option for 7.1
  - Mount CDROM on another Linux image
  - Use FTP option
• Upload starter components
• Punch to and boot from reader
• Answer questions:
  – IP connectivity
  – NFS server location
• Telnet to starter system
• Begin install of RPMS: ./install
ThinkBlue64

• Three panels of questions:
  – Disks to use and mount points: No swap
  – NFS server containing RPMS
  – [Repeat answers on IP addresses etc.]
  – Install begins
• Install process runs zilo
• Now boot from disk
ThinkBlue64

xterm

[jusanefe@dali007 - usanefes] dir

OV0_d15.0.4_src.zip ckit.data
TrueChargeProjects ckit
bash-2.04 jdiff
bash-2.04.tar.gz jdk-1.2.2-FCS-linux-s390-glibc-2.1.3.tar.gz

[jusanefe@dali007 - usanefes] w

12:33pm up 39 min. 2 users, load average: 0.07, 0.02, 0.00

USER TTY FROM LOGIN@ IDLE JCPU FCPU WHAT
usanefe pts/1 10:64.32.70:0.0 1:53pm 1:00s 0.02s 0.02s u
usanefe pts/0 10:64.32.70 1:20pm 5:19 0.52s 0.36s xterm

[jusanefe@dali007 - usanefes] cat /proc/cpuinfo

vendor_id : IBM/S390
processors : 1
bogomips per cpu: 748.74
processor 0: version = FF, identification = 087100, machine = 2064

[jusanefe@dali007 - usanefes] uname -rm

2.4.3-0.4.13wrd s390x
[jusanefe@dali007 - usanefes]
• Current work:
  – bash2 – fixed in 7.1
    • Problem with signal handling: Union of pointer and int
  – Regina ported
  – JDK 1.3 port ready for certification testing
    • Porting invokeNative_s390.S
    • Instructions: \texttt{sllg r1,r1,2} versus \texttt{sll r1,2}
  – Assessing requirements & efforts for SAG products
[usanefe@dali007 - usanefe] java -version
java version "1.3.0_02"
Java(TM) 2 Runtime Environment, Standard Edition (build Blackdown-1.3.0_02-FCS)
Classic VM (build Blackdown-1.3.0_02-FCS, native threads, nojit)
[usanefe@dali007 - usanefe] file
/usr/local/j2sdk/bin/s390x/native_threads/java
/usr/local/j2sdk/bin/s390x/native_threads/java: ELF 64-bit MSB executable, version 1, dynamically linked (uses shared libs), not stripped
ThinkBlue64

- Built glibc-2.2.3 – appears quite stable
  - Has make/swap-context APIs
  - Required for green-thread support of Java
- Built openMotif – appears to work
- Enhanced CPINT
  - 2.4 & 64-bit support
  - Ability to retrieve CP return code via `ioctl()`
  - Fixed a couple of bugs: passwords & buffer size
Early Experiences

64-bit SuSE System
SuSE System

- All externals/procedures as per SLES7
- 2.4.17+ kernel
- glibc 2.2.4+
- Hipersocket support
- Required “nopfault” on parmline
- Bug found in ucdsnmp
  - ssize_t versus int
- Worked perfectly with 8 CPUs and 3GB memory
- Problem with qdio driver - fixed
Some Problems – All Fixed

- X11 “funnies”
- `pthread_cancel` cleanup peculiarities
- signal handler recursion
- Support of `SA_SIGINFO`
- CTC buffersize set at 32K
  - `skb_buff()` failures
- `pthread_create` race condition
- `pfault` Ooopses (z/VM 4.2 fix required)
Things are changing fast…

- zfcp support
- gcc 3.1.1
Linux for zSeries

Redhat
Redhat

- 2.4.9+
- glibc-2.2.4-24
- Installed without a problem
- Configuration of hipersocket a bit of a task
- I’m Too used to YaST
Linux for zSeries

Miscellany
• Linux in a NSS (needs gcc-3.1.1)
  + #ifdef CONFIG_SHARED_KERNEL
  +     .org 0x100000
  + #else
  +     .org 0x10800
  + #endif
• Do a make image to avoid long wait caused by kernel disassembly
PFAULT Handling

#ifdef CONFIG_PFAULT
+       if (MACHINE_IS_VM) {
+           /* request the 0x2603 external interrupt */
+           if (register_external_interrupt(0x2603, pfault_interrupt) != 0)
+               panic("Couldn't request external interrupt 0x2603");
+           /*
+                * Try to get pfault pseudo page faults going.
+                */
+           if (pfault_init() != 0) {
+               /* Tough luck, no pfault. */
+               unregister_external_interrupt(0x2603,
+               pfault_interrupt);
+           }
+       }
#endif
Questions