Linux for zSeries

Early Experiences with 64-bit Linux

Agenda

- z/Architecture Overview
- Linux implementation for z/Architecture
- ABI changes
- Building a kernel on a 31-bit system
- Building a file system from scratch
- Early experiences with ThinkBlue64
Linux for zSeries

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—A statement of direction 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.

- 64 bit PSW
  - Bit 12 – ‘0’ specifies z/Architecture
- 64 bit control registers
- 16 IEEE/HFP registers
  - No need for software emulation
z/Architecture Overview

- 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

- 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”

```
06440008  07000000
0000060000000000
```

- 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

### Implemented on:
- z900 (aka Freeway) processors
- Hercules
- Flex/ES (or should be in the future)

### Supported by:
- z/VM
- OS/390
- Linux for zSeries

---

<table>
<thead>
<tr>
<th></th>
<th>OLD</th>
<th>NEW</th>
<th>EXT 1004</th>
<th>SVC 008E</th>
<th>PRG 0004</th>
<th>MCH 0000</th>
<th>I/O 0004</th>
</tr>
</thead>
<tbody>
<tr>
<td>130</td>
<td>07060001</td>
<td>04000001</td>
<td>80000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00015F1A</td>
<td></td>
</tr>
<tr>
<td>1B0</td>
<td></td>
<td></td>
<td>80000000</td>
<td>00000000</td>
<td>00014D32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>0701C001</td>
<td>04000001</td>
<td>80000000</td>
<td>00000200</td>
<td>002618A6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1C0</td>
<td></td>
<td></td>
<td>80000000</td>
<td>00000000</td>
<td>0001406C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>07004001</td>
<td>04000001</td>
<td>80000000</td>
<td>00000000</td>
<td>00087C7A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1D0</td>
<td></td>
<td></td>
<td>80000000</td>
<td>00000000</td>
<td>00014AD6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E0</td>
<td></td>
<td></td>
<td>80000000</td>
<td>00000000</td>
<td>00014DEA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>07006001</td>
<td>04000001</td>
<td>80000000</td>
<td>00000000</td>
<td>00015F1A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1F0</td>
<td></td>
<td></td>
<td>80000000</td>
<td>00000000</td>
<td>00014C3A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Linux for zSeries

Linux Implementation for z/Architecture

- 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

Kernel

User Stack

Shared Libs

User Program
Data BSS
Text
Sections

Next
To
Run

User Space Himem
(typically 0xC0000000 3GB)

0x00000000

Linux – S/390 Address Spaces

0x7FFFFFFF 2GB Himem

User Stack

Shared Libs

User Program
Data BSS
Text
Sections

Kernel

0x00000000
Linux – zSeries Address Spaces

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 1 12 20</td>
<td></td>
<td>31</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>X</th>
<th>Region Index</th>
<th>Segment Index</th>
<th>Page Index</th>
<th>Byte Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 22 33 41</td>
<td>52 63</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright 2001 Neale Ferguson, Software AG. All Rights Reserved.
### 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!
zArchitecture Address Spaces

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

New Device Drivers

- Tape
  - 3490
- Character and block
- 3270
- Console
- Standard terminal
- Cisco Routers
Device Drivers

- CCWs must live “below-the-bar”
- Kernel supports memory requests for under the bar storage
- 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.

- 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>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Back chain (a 0 here signifies end of back chain)</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>EOS (end of stack, not used on Linux for S390)</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>Glue used in other linkage formats</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>Glue used in other linkage formats</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>Scratch area</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>Scratch area</td>
</tr>
<tr>
<td>24–63</td>
<td>48–127</td>
<td>GPR register save area</td>
</tr>
<tr>
<td>64–79</td>
<td>128–159</td>
<td>FPR4 &amp; FPR6 save area</td>
</tr>
<tr>
<td>96</td>
<td>160</td>
<td>Outgoing args (length x)</td>
</tr>
<tr>
<td>96+x</td>
<td>160+x</td>
<td>Possible stack alignment</td>
</tr>
<tr>
<td>96+x+y</td>
<td>160+x+y</td>
<td>alloca space of caller (if used)</td>
</tr>
<tr>
<td>96+x+y+z</td>
<td>160+x+y+z</td>
<td>alloca space 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
Linux for zSeries

Building your first kernel

Building your first Kernel

- Create a cross-compile environment
  - `binutils-2.10.0.8` + experimental patches
  - `gcc-2.95.2` + experimental patches
  - `glibc-2.2.2` + experimental patches
- Download kernel from `ftp.kernel.org`
- Apply patches
Building your first Kernel

- Update Makefile:
  - `ARCH := s390x`
  - `CROSS_COMPILE=s390x-ibm-linux`
  - `make menuconfig`
  - `make dep`
  - `make`
  - `make modules`
  - `make modules_install`

- Build silo:
  - `cd arch/s390x/tools/silo`
  - `make`

- Copy files to `/boot`:
  - `cp arch/s390x/boot/image /boot`
  - `cp arch/s390x/boot/*.boot /boot`
  - `cp System.map /boot`

- Run silo:
  - `cd /boot`
  - `<path-to-kernel-tools>/silo -f image -p parmfile -d /dev/dasdx`
Building your first Kernel

- Most things will work with your 31 bit executables and glibc-2.1.3
- Certain system interfaces have changed and are not supported by library routines
- Good enough to build everything you’ll need

Linux for zSeries

Building a file system from scratch
Building a 64-bit file system

- Use first kernel as a base
- Get “Building LFS from Scratch” from: http://linuxfromscratch.org
- Get second disk and mount as /root64
- Build ncurses using cross-compiler
- Follow instructions in LFS document to populate /root64

Building a 64-bit file system

- Basically a two-stage process:
  - Starter pack:
    - Build statically linked versions of core packages (inc. gcc)
    - Create traditional file layouts (i.e. /etc, /lib etc.)
    - Create /dev nodes – need to manually define /dev/dasdx
    - Create /etc contents (e.g. passwd, groups)
  - chroot to /root64
Building a 64-bit file system

- Production pack:
  - Build new glibc-2.2.2
  - Rebuild core packages - this time dynamically linked
  - Create and tailor startup scripts
  - Copy `/boot` contents & `/lib/modules`
  - Update parameter file and run `silo`
  - Run `depmod`
  - Reboot off the `/root64` disk

Linux for zSeries

ThinkBlue64 – Very Early Experiences
ThinkBlue64

- Redhat-like distribution
- 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

Starting:

- Mount CDROM on another Linux system:
  ```
  mount -o loop ThinkBlue64-disc1.iso /mnt/cdrom
  ```
- Add `/mnt/cdrom` to `/etc/exports` and restart NFS server

```
# 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.

/mnt/cdrom 10.20.45.7(rw,no_root_squash)

/etc/rc.d/nfsserver restart
```
ThinkBlue64

- 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
Current work:

- Bash2
  - Problem with signal handling: Union of pointer and int
- Regina ported!!!
- JDK 1.3
  - Porting invokeNative_s390.S
  - Instructions: `sllg r1, r1, 2` versus `sll r1, 2`
- Assessing requirements & efforts for SAG products
  - Tamino currently runs on Linux for S/390
  - Samba 2.2 has just gone GA on other platforms