OpenSolaris for System Z Prototype Release Notes

Document Number OSSZ-3140-02

October 17, 2008

Sine Nomine Associates
43596 Blacksmith Square
Ashburn, VA 20147

This document provides release notes for the OpenSolaris for System Z prototype.

First edition, September, 2008

Contents

OpenSolaris for System Z Prototype Release Notes

Applicable Release Identifier

This document applies to the prototype release delivered in September 2008.

Questions, comments or problems should be directed via electronic mail to support@sinenomine.net or via phone to +1 866 252 1923 (toll free) or +1 703 912 7023 if outside the US or Canada.

Feature Summary

- System boots from local disk
- System initializes and allows login from line-mode console and network-based login methods (telnet and ssh).
- Provides C compiler and linker functionality (see note).
- Provides functioning layer 2 network connectivity via a device driver implementing the DIAG2A8 specification provided by IBM.
- Provides IPv4 and IPv6 connectivity via the new network device driver.
- Implements common network services (ssh, FTP, http).
- Implements NFS client support.
- Provides network utilities (ssh, FTP, httpd)
- Provides symbolic debugger capability (gdb)

Features New in This Release

This release contains the following new functions:

- A functional `gdb` debugger is present in this release. See the notes in the application development section for limitations in the current release.

Known Issues In This Release

1. GDB Is Partially Functional

   Due to the complexity of tracking down several bugs in threading and device drivers, the development of `gdb` was hampered. The version of `gdb` delivered supports a number of useful features, including:

   - Setting and removing breakpoints in application execution.
   - Single-step of source programs
   - Displaying register contents on request
   - Instruction disassembly on request
   - Correct display of source lines if source files for application are available.

The following functions are not yet completely implemented:

- Watchpoints are not implemented.
- Display of variables and variable values within a routine scope are not complete and may cause display of random binary garbage under certain circumstances.

An additional release of `gdb` will be available shortly to correct some or all of these issues.

Release Notes

Pre-Installation Notes

1. This release must run on a z9-class processor or higher. It will install correctly on a non-z9 system, but will fail with illegal instruction traps if run on a non-z9 system.
2. Please ensure that you read the installation guide document carefully and have all the pre-requisites installed before reporting problem.
3. This release is supplied on DVD or as a download file that can be used to create a tape. If you have trouble reading the DVD, ensure that your workstation has a DVD reader.

Installation Notes

Creating an Install Tape from the AWSTAPE Image

The protype release is delivered on physical DVD, however both individual files and a AWSTAPE-format tape image are provided to allow customers to select their desired format. The tape image can be used to create a real physical tape by using the following process:

1. Locate the AWSTAPE folder on the DVD.
2. FTP the files in the folder to a SFS directory on your z/VM system. The files will need to be in SFS due to the fact that the AWSTAPE file is larger than most common 3390 volumes (if you have model 27 or model 54 3390 volumes, you'll need one to use a mindisk or T-DISK to hold the files). *Ensure that you transfer the files in binary mode and use the* `SITE FIX 80` *FTP subcommand before transferring the files*.Refer to the CMS TCPIP utilities documentation for more assistance using FTP.
3. Extract the DETAPE EXEC by running:

   ```
   EXEC DETAPEM EXTRACT
   ```

4. Mount a blank or scratch tape read/write at virtual address 181.

   Tape Will Be Overwritten

   Ensure that the tape you mount is either blank or contains nothing that you wish to save. The entire contents of the tape will be overwritten, including tape labels and other files.

   The tape must be at least a 3490E 600 ft volume. 3480 volumes are not large enough. 359x volumes of all types are supported.

5. Run DETAPE by typing the command at the command line.

The installation tape will be created automatically.

OpenSolaris Boot and Initialization Notes

Specifying a Boot Device and Root Filesystem Disk

Boot disks can be selected by IPLing the appropriate disk using the CP `IPL` command. Root filesystems must be at virtual address 200, but alternate volumes can be substituted by copying the minidisk to another minidisk using DDR, making the appropriate changes, and linking the new minidisk at virtual address 200. The /etc/system file and bootenv.rc that determine the selection of the boot volume is contained in the initial ramdisk image loaded from the NSS, and is not modifiable in the current release.A future version will document the process of recreating the initial ramdisk.

Operations Notes

Default Root Password

The password for the root account configured in the image is `f1shf4c3` (case sensitive). We recommend you change it immediately after installing your system.

SSHD Configuration

`sshd` is delivered in a secure configuration that does not permit direct root logins from remote users. This implies that for normal use, users and admins are expected to log in using their own userid and switch to root using `su` or `sudo`. While this configuration is normally the best practice, some sites may wish it to be more flexible. To achieve this, you can configure sshd to be less restrictive by logging in on the 3215 console and editing `/etc/ssh/sshd_config`. To permit logging in as root from a remote, remove the `PermitRootLogin no` line from this file.

Alternatively, you can create a non-root id for the user using the `passmgmt` and `useradd` commands. This is a much more secure approach, and is considerably more friendly to the 3215 environment. An example process for creating a non-root user named `testuser` might appear similar to the following:

```
passmgmt -a -h /export/home/testuser testuser
mkdir /export/home/testuser
chown testuser:other /export/home/testuser
passwd testuser
```

Preparing a Minidisk for Use with OpenSolaris

Minidisks used with OpenSolaris for System z should be formatted with the CMS `FORMAT` command and processed with the CMS RESERVE command to prevent accidental over-writing by CMS applications if the minidisk is accidentally accessed by a non-CMS system.

To format a minidisk with the CMS `FORMAT` command:

1. Log into a VM userid and IPL CMS.
2. Link the new blank minidisk to your virtual machine using the CP LINK command, eg: `CP LINK SOLDIST F001 F001 MR` (if the minidisk is defined to user SOLDIST at virtual address F001).
3. Run the CMS `FORMAT` command. An example might appear similar to the following:

```
format F001 f
DMSFOR603R FORMAT will erase all files on disk F(F001).
           Do you wish to continue? Enter 1 (YES) or 0 (NO).
1
DMSFOR605R Enter disk label:
BOOT
DMSFOR733I Formatting disk F
DMSFOR732I 100 cylinders formatted on F(F001).
R;
```

Figure 1. Formatting a Blank Minidisk for OpenSolaris Use

Following the format, DIAG 250 requires CMS RESERVE processing to operate correctly. Note that the CMS file name shown is not important, but it must be a valid CMS filename. We have been using a convention of the owning userid and the virtual address of the disk on that virtual machine, but you are free to choose your own scheme. OpenSolaris does not use or access any data via this name; it is only displayed if the disk is accessed in a CMS virtual machine and the LISTFILE command is used.

To reserve the minidisk (shown here accessed as CMS filemode F), use the following steps:

```
Ready; T=0.09/0.13 19:43:48
reserve soldist f000 f
DMSRSV603R RESERVE will erase all files on disk F(199).
           Do you wish to continue?  Enter 1 (YES) or 0 (NO).
1
DMSRSV733I Reserving disk F
Ready; T=0.01/0.01 19:44:07
l * * F
SOLDIST  F000     F6
Ready; T=0.01/0.01 19:44:12
```

Figure 2. Processing a Formatted Minidisk for OpenSolaris Use with RESERVE

The new minidisk is now ready for use by OpenSolaris, and will be recognized as available space when linked to the OpenSolaris guest.

Writing a Boot Loader on a Minidisk

Updating the Boot Loader Not Necessary in Default Install

If you install the protoype kit using the DDR process described in th installation guide, you do not need to perform this step unless you increase the size of the boot minidisk, or change its underlying disk technology (eg from FBA to ECKD, or vice versa).

Disks used for booting OpenSolaris systems need an additional preparation step to reserve a location on disk for the boot loader. The following steps provide that preparation. To prepare a disk for use in booting:

1. Create the minidisk using your local disk management procedures.

2. Link it to the OpenSolaris guest and format it using CMS `FORMAT` as described in section "Preparing a Minidisk for Use with OpenSolaris" on page 3.
3. Reformat the disk with the RECOMP option to reserve space on the device, for example:

   ```
   FORMAT cuuu F 48 ( RECOMP
   ```

   The 48 reserves 2 cylinders for the boot loader.
4. Run the SOLARIS exec provided in the default install. This exec will create and write the boot loader into the reserved area.

SOLARIS EXEC Updates 191 Disk In Target USER DIRECT Entry

As distributed, the SOLARIS EXEC expects a ECKD minidisk at virtual address 191 in the target user CP directory entry, and is hard-coded to update that disk only. When preparing a new boot disk, the administrator should substitute the new disk in the CP directory entry, prepare it as described above, and update it using SOLARIS EXEC to write the new boot loader.

Booting a Specific Boot Disk

A specific minidisk that has been prepared as described above can be booted using the CP `IPL` command for that virtual address, eg `IPL 191`.

Booting a Rescue/Recovery System

The default 191 minidisk also includes a kernel and ramdisk that does not automount the designated root filesystem. To use this recovery system, log into the OpenSolaris guest VM userid, IPL CMS, and type: `RECOVER`.

This will start a OpenSolaris system with the initial ramdisk mounted read-only as the root filesystem. You can then mount other filesystems and do repairs or other operations as needed.

Application Development Notes

Update Parameters for cw

The following parameters for the `cw` executable were changed or modified from the Solaris default.

`cw` Changed or Modified Parameters

Table 1.	
Parameter	**Description of Change/Modification**
-O	Default changed to -O1. The default value was changed to level 1 and expanded to allow multiple optimization levels.
-t	Changed to "-Wl -t". Disables duplicate warnings on link.
-xmodel=kernel	-ffreestanding -mcmodel=kernel -mno-red-zone ???? $
-Wu, -save_args	-msave-args ??? $
-xbuiltin=<b>	-fbuiltin (defaults to -fno-builtin if not present)
-xdebugformat=<format>	Ignore this parameter. Always use dwarf-z for gcc.
-xcrossfile=<n>	Ignore this parameter
-W0, -xdbggen=no%usedonly	-fno-eliminate-unused-debug-symbols, -fno-eliminate-unused-debug-type
-march=z990, -march=z9	Added flags to support z900 and z9 system types

The default C compiler flags for all OpenSolaris platforms are:

```
-fident -finline -fno-inline-functions -fno-builtin -fno-asm
-nodefaultlibs
```

The handling of -h and -o was extended to allow passing arguments to these two options. Processing of -M was also modified as GCC does not process map files in the same method as the Sun cc (see below).

Lint Command Line Processing Modified

`lint` command line arguments were modified to undefine the SPARC and Intel platform flags and define the appropriate s390x flags.

Mapfile Handling

The GNU toolchain does not handle version scripts and/or mapfiles in the same way as the Sun linker. This breaks the supplied build scripts and makefiles in libraries that use the 'mapfile' directive to define symbols. GCC treats this option as a no-op (as it sees no symbols) and does not produce an output file.

In this release, the workaround is a Perl script that scans the object files and produces a workable assembler source map file and the necessary makefiles to employ it. Libraries and other object builds in the source tree have a `Makefile.gnumapfile` included for each build that uses both a mapfile and other object files, and

`Makefile.gnumapfile.maponly` entry for libraries which consist of (and use) only a mapfile.

This situation rarely occurs in user-mode applications, however if your application requires it, appending the correct include to the end of the platform-specific makefile will correct the build. An example (taken from cmd/sgs/libdl):

```
# Add build for generated mapfile
include ../../../../Makefile.gnumapfile.maponly
```

Differences in GNU ld and Sun ld

When linking static/ar archives into a shared library, the OpenSolaris `ld` will include all objects and symbols regardless of whether they are actually used by the library itself. GNU `ld` requires supplying the `--whole-archive` command line option to enable this functionality.

The above procedure will handle any new symbols defined in the mapfiles. If you need actual versioning of the symbols (or 'local' symbol hiding), one can manually produce a GNU-compatible version-script of the mapfile. You must merge all the dependent mapfiles together (the Sun build tools can cascade through a concatenated list of mapfiles). Following the merge, you must manually (or via a script) remove any modifiers (anything that looks like ' = ...') present in the files. Finally, the sections need to be re-ordered in the reverse order of dependency (lowest-level first).

Note also that the use of '*;' in the Sun linker mapfile support is incompatible with the usage managed by the GNU linker. This situation typically only occurs in compiling kernel modules containing inline assembler and C in the same source file, but this usage may require manual intervention. If you encounter this problem, please contact your support organization for assistance.

GCC Optimization for Kernel Modules

There are open issues in GCC release 4.2.3 that affect all implementations of gcc for the s390x architecture, not just OpenSolaris. The following issues have been identified and patches applied to correct the problems shown in these entries, with permanent patches scheduled for inclusion in GCC release 4.3. The gcc binary installed on this release of OpenSolaris has interim patches applied that appear to correct the problem for userspace applications. Applications (such as kernel modules that contain substantial assembler components) may still exhibit optimization problems but the gcc developers need additional test cases to be certain the proposed fixes completely exterminate the problem. If problems occur, please submit a bug report to assist the gcc developers in isolating these problems. The workaround recommended by the gcc developers is to compile modules exhibiting optimization problems with the -O0 flag and contact your support organization to gather additional materials to assist in problem resolution.

The GCC bug tracker open issues related to these problems are:

- Invalid slgfi instruction emitted with -march=z9-109

  http://gcc.gnu.org/ml/gcc-patches/2007-12/msg00545.html

- Target macro to redefine 'm' constraint letter

  http://gcc.gnu.org/ml/gcc-patches/2008-03/msg01213.html

- Emit barrier INSN between stack pointer decrement and stack slot writes

  http://gcc.gnu.org/ml/gcc-patches/2007-12/msg00102.html

The issues do not appear to affect userspace code, however kernel developers should track these issues to ensure that problems do not ensue.

Additional GCC Notes

The following noncritical GCC features are not supported with the GCC toolchain that is included in this release:

- Code coverage analysis and branch profiling support is not completely tested.

  The `gcov`, branch profiling and the meta-optimization lookaside analysis (e.g., using the output from running a profiling build to improve branch determination) are not exhaustively tested in the test suite. The test suite components do not test this function correctly, but correcting and completing this feature is a "nice to have" function and will be deferred to a later release.

- Use of the `-fexception/unwind` options fails the tests present in the test suite.

  `-fexception/unwind` options used in the test suite programs failed to unwind the stack cleanly in the test cases. The programs compiled and ran correctly, but hit the end of the stack every time the test was executed. It is unclear whether this is a failure of the test or the compiler.

- Borland-style template (`-frepo`) support fails to build.

  The test case for the `-frepo` flag fails to build. The `-frepo` flag enables "Borland-style templating" which is rarely used for Unix applications (the flag enables compatibility with the MS DOS-based Borland C compiler's somewhat ideosyncratic support for function templates and prototypes). Code that requires `-frepo` may fail to compile, however this code is likely to fail for other reasons and should be examined to determine whether ANSI standard function prototypes could be used in place of the non-standard Borland prototypes.

- `isinf` builtin is not supported for Solaris, developers should use the macro version.

  This limitation is true for all flavors of Solaris and OpenSolaris; the isinf test failed on both OpenSolaris on x86 and on a commercial Solaris 9 and 10 SPARC system, and there are open references on the failure available by trivial Google search. There does not seem to be active development for this flaw in the GCC community, but we feel that we should note the failure for reference.

Linker Internal Debugging Support Not Available For System Libs

The Sun runtime linker requires certain Solaris-specific `ld` functionality to allow debugging symbols to load correctly. The specific functionality required is related to lazy loading of symbol data where the Sun linker produces a `.SUNW_syminfo` section in the object module output and sets `POSFLAG_1` to `0x1 (LAZY)`.

This release is based on (and uses) GNU `ld` to produce system libraries, so support for LD_DEBUG and related steps/processes for system library code does not work. This does not affect normal application runtime linker operation, nor other forms of debugging.

Sun is in the process of producing and releasing a more modular system linker as part of a redesign of the Solaris utilities. We expect a future release of the Solaris `ld` to allow better support for non-Sun-supported architectures, which will allow the system libraries to be built and linked with Solaris `ld`, resolving this problem.

Device Driver Notes

Interrupt Handling

For OpenSolaris on System z, interrupt handling follows the "autovectored interrupt" approach described in the Solaris device model as closely as possible. Two changes to this model were necessary:

- On System z, when an interrupt is presented, information specific to that interrupt is stored in various locations in low core. These locations in low core are fixed, and some are shared by interrupts of different priority levels. Since interrupt handlers are called with interrupts enabled (allowing higher level interrupts to occur), there is a risk of the information being overwritten by another interrupt.

  Therefore, when an interrupt occurs, and before interrupts are re-enabled, the low core information is collected and supplied to the interrupt handler via the second function argument when the interrupt handler is called. This is a deviation from normal Solaris autovec interrupt handling since the Solaris model assumes that the handler can use both arguments for any purpose.

- As is the case for interrupt handlers on other Solaris platforms, interrupt handlers on System z are expected to return `DD_INTR_CLAIMED` or `DDI_INTR_UNCLAIMED` to indicate whether the interrupt has been processed or not.

  On System z, claiming an interrupt will prevent any further processing by other handlers at the same priority level. This is an additional deviation from normal autovec interrupt processing.

Device Driver Types

Device drivers for OpenSolaris on System z take one of two distinct types:

hybrid Hybrid drivers take advantage of services provided by the z/VM hypervisor and have complete freedom to utilize z/VM CP APIs directly, or a mixture of the z/VM and CCW layer APIs provided as part of the OpenSolaris port. Examples of this type of driver include the DIAG 250 DASD driver (an exclusively z/VM API), and the DIAG 2A8 network device driver (both z/VM CP and CCW layer APIs).

native Native device drivers communicate directly with a device, without assistance from z/VM APIs. These device drivers must manage all aspects of device control, including command initiation, data transfer, device interruptions, and error handling. The CCW layer APIs are used to simplify parts of this interaction. The 3215 console driver provided in this kit is an example of this type of driver.

In either case, additional drivers should be designed to follow the Solaris DDI device driver interface specification provided by Sun in the "Writing Device Drivers" documentation. This is especially important for hybrid drivers, as they often need to set up and handle interrupts directly. Use of the CCW API manages this aspect for native device drivers.

In this release, device drivers may not use the `ddi_set_driver_private()` function to store a private data pointer for the individual driver. This function is used by the CCW API to store device information specific to the CCW layer operation. A future release of the CCW API will relax this restriction.

The following aspects of the Solaris device model and interface are not applicable on System z:

- DMA processing
- Device contexts
- Power management
- Polling

CCW Nexus Driver

The CCW nexus driver is the parent of all I/O device drivers within the OpenSolaris for System z environment. This driver is responsible for detecting devices, creating device nodes and fielding device interrupts.

During system startup, the CCW nexus driver will query each subchannel looking for reachable devices, and will use the z/VM DIAG 210 interface to interrogate the device for characteristics and capabilities. The device class and type returned by DIAG 210 are then used to initialize the subchannel interrupt priority and create the associated device node. Once identified and registered, the appropriate device driver is bound to the node, and the `attach()` function provided by the device driver is called to initialize the device.

Note that at this point in the boot process, interrupts are not yet enabled, so if the device driver requires interrupt processing to correctly initialize and set up the device for use, the initialization in the driver must be deferred until the first use of the `open()` driver function is performed. An alternative method to implement this type of initialization is to queue a soft interrupt, which will fire as soon as interrupts are actually enabled. This provides a notification that the driver may safely proceed with device initialization.

CCW API

The CCW API provides a variety of functions to simplify working with System z devices. Most of these functions are wrappers around z/Architecture I/O instructions with a few additional functions to help create specific channel programs and manage I/O requests.

The CCW API also provides memory functions for allocating locked and contiguous physical memory below the 2G line (some z/VM APIs still require 31-bit memory areas), simplifying the process of obtaining such storage. Future releases of the CCW nexus driver will provide simulated DMA memory allocations as described by the DDI documentation.

The CCW API layer resides within the CCW nexus, so if your device driver uses the CCW API, you will need to ensure that your driver includes one of the following:

1. The following variable definition, OR:

   ```
   char_depends_onffl" = "ccwnex";
   ```

2. Adding these options to the LDFLAGs for your driver:

   ```
   -dy -Ndrv/ccwnex
   ```

Native Device Drivers

As discussed earlier, native device drivers interact directly with the device hardware and do not depend on z/VM APIs for assistance. While this does allow complete control of the device, it also implies added complexity for the device driver, especially for complex devices with detailed requirements for error handling.

When your driver is bound, the `attach()` function is called, and the driver should perform all the required setup tasks documented in the Sun DDI.

The CCW nexus driver maintains information related to native devices and active or last I/O commands. Much of this information is available to the driver via the CCW API functions which should be used whenever possible. However, if or when the driver calls `ccw_device_register()`, a pointer to the device information is returned.

If your driver needs to directly process interrupts, then the `ccw_device_set_handler()` function should be called in your `attach()` function to inform the CCW layer of the address of your interrupt handler routine.

When an interrupt fires, the associated `ccw_device_req` and `ccw_device` pointers will be passed to your interrupt handler. Interrupt handler routines should minimize time spent in the handler routine as interrupts for all devices of the same type and class are disabled while the handler is in control. The Sun DDI documentation describes several methods of minimizing time spent in interrupt handler routines.

Console Driver: The 3215 console driver is a good example of a native device driver. Refer to "uts/zSeries/io/ccw/con3215.c" in the source tree for the source code of this device driver.

The console driver (con3215) provides basic TTY functinality during periods where network access is not available (i.e., during boot, single user mode without network started, or when the network is not functioning).

The console device node is notated as follows:

`/devices/ccw/cnsl@0x`*`CUUU`*`:con3215`

where:

CUUU The z/VM virtual device address of the 3215 device.

The console driver is a streams-based driver, and the `ldterm` and `ttcompat` streams modules should be pushed onto the console stream via `/etc/inittab` during boot. These modules perform control character processing, and are needed to properly allow interrupting processes and other important functions.

During initialization, the console driver sets the z/VM terminal AUTOCR option to OFF, since the driver provides line end and linefeed processing internally. Re-enabling this option while the console driver is active within a virtual machine will cause output to be garbled and difficult to read due to multiple linefeed processing in both OpenSolaris and z/VM.

The console driver supports several configuration options. Sensible defaults for z/VM virtual machine operation are compiled into the executable, but driver configuration may be modified by creating the file `/kernel/drv/con3215.conf`. A default configuration file

is supplied and may be used as a base. Options are specified as keyword="value" pairs from the following table.

Table 2 (Page 1 of 2). Console Driver Options (/kernel/drv/con3215.conf)

Option Name	Default Value	Description
intr	"[c"	Key sequence used to send the interrupt signal. Equivalent to cntrl-C on an ASCII terminal.
quit	"[\"	Key sequence used to send the quit signal. Equivalent to cntrl-\ on an ASCII terminal.
erase	"[?"	Key sequence used to send the erase signal. Equivalent to cntrl-? on an ASCII terminal.
kill	"[u"	Key sequence used to send the kill signal. Equivalent to cntrl-u on an ASCII terminal.
eof	"%d"	Key sequence to indicate EOF. Equivalent to cntrl-D on an ASCII terminal.
eol	"<undef>"	Key sequence for sending EOL. Undefined.
eol2	"<undef>"	Key sequence for sending EOL2. Undefined.
swtch	"<undef>"	Key sequence for sending SWTCH. Undefined.
start	"[q"	Key sequence to indicate START character. Equivalent to cntrl-Q on an ASCII terminal.
stop	"[s"	Key sequence to indicate STOP character. Equivalent to cntrl-S on an ASCII terminal.
susp	"[z"	Key sequence to indicate suspend character. Equivalent to cntrl-Z on an ASCII terminal.
dsusp	"[y"	Key sequence to indicate DSUSP. Equivalent to cntrl-Y on an ASCII terminal.

rprnt	"[r"	Key sequence to indicate RPRNT. Equivalent to cntrl-R on an ASCII terminal.
flush	"[o"	Key sequence to indicate FLUSH. Equivalent to cntrl-o (oh) on an ASCII terminal.
werase	"[w"	Key sequence to indicate WERASE. Equivalent to cntrl-w on an ASCII terminal.
lnext	"[v"	Key sequence to indicate LNEXT character. Equivalent to cntrl-v on an ASCII terminal.

Hybrid Device Drivers

Hybrid device drivers are similar in function and creation to native device drivers, and much of the information presented for native device drivers applies to hybrid drivers as well. The major difference between the two is that hybrid drives use a combination of the CCW API layer functions and the z/VM CP API functions, rather than only the CCW API layer functions.

Not all the z/VM CP API functions trigger device interrupts via the I/O subsystem, so hybrid drivers may need to utilize the functions in the Sun DDI interface to create and manage private device-specific interrupt handlers to compensate. If the z/VM CP API *does* generate device interrupts, you may use the CCW API interrupt handling to process these interrupts. The DIAG 250 and DIAG 2A8 device drivers provide examples of both cases. The DIAG 250 driver uses only the z/VM APIs, and the DIAG 2A8 driver uses a combination of the two APIs.

DIAG 250 Disk Driver: All disk I/O for OpenSolaris on System z is performed using z/VM DIAGnose 250 instructions. The device driver to implement that function is named `diag250`, and the source for this driver is located i `uts/zSeries/io/ccw/diag250_hl.c`.

The `diag250` driver supports FBA, CKD and emulated FBA disks on SCSI technology. The driver supports physical block sizes of 512, 1024, 2048, and 4096.

Disk device nodes are notated as follows:

`/devices/ccw/dasd@0x`*`CUUU`*`:dasd`

where:

CUUU The z/VM virtual device address of the minidisk in the CP directory for the virtual machine.

The diag250 driver supports one configuration option. Sensible defaults for z/VM virtual machine operation are compiled into the executable, but driver configuration may be modified by creating the file `/kernel/drv/diag250.conf`.A default configuration file is provided and may be used as a base. Options are specified as keyword="value" pairs from the following table.

Table 3. DIAG250 Driver Options (/kernel/drv/diag250.conf)

Option Name	Default Value	Description
use-minidisk-cache	"1"	Indicates whether z/VM minidisk caching should be used or bypassed. A value of 1 indicates that caching should be used, 0 indicates that cache should be bypassed.
bounce-buffer-limit	16	Most data buffers are handed off to z/VM directly, however there are certain cases where the data must first be copied to a bounce buffer before it can be handed off to z/VM. This value specifies the number of 4K buffers that are preallocated by the device driver for this purpose.
io-queue-limit	16	Specifies the number of internal I/O structure to preallocate. These structures are used to service I/O requests. Specifying a large number h will improve I/O performance at the cost of additional working set and virtual machine memory consumption.

Refer to the `driver.conf` man page in the OpenSolaris for System z distribution for additional information on how to specify settings on a global or per-device basis.

DIAG 2A8 Network Driver: This device driver requires z/VM PTF UM32414 for APAR VM64466 if you plan to run OpenSolaris on z/VM 5.3. The equivalent Special Programming Enhancement (SPE) for z/VM 5.4 will be available in November, 2008. Please consult the z/VM documentation for assistance in ordering and installing this PTF.

User and API Notes

Sun Perl Build and Local Perl Build Are Different

While the OpenSolaris tree contains a Perl installation, this distribution also contains a complete local build of Perl separate from the Sun-installed one. The reason for the duplication is that Perl is a self-bootstrapping language and the default set of `make` rules delivered with the Sun-installed version of Perl cannot be built using a cross-compiler. Future releases will revert to a single installation when a complete Sun link-edit environment is available with the new modular multi-architecture linker.

Known Perl Issues

The build process for Perl includes a number of complex tests that make significant assumptions about the behavior and architecture that a particular operating system choice implies. The release of Perl included in `/usr/local/bin` is current and capable enough to support most normal uses of Perl as a automation and system management tool. Several of the build tests do not complete correctly. The major test failures are:

- Occasionally, Perl programs will generate "bad `free()`" messages. We are investigating the cause of this problem.

The supplied module passes 99.78% of the entire Perl test suite, and we are confident that it will produce correct output for a majority of common Perl usage.We have tested it with a large text processing application and it operates correctly. We anticipate consulting the Perl development team when we can provide public access to a system to help them to debug, and resolving the remainder of the test problems with their assistance.