Migration Kit for Solaris OS to Linux

Using the Endian Checking Portability Tool

Migration Kit for Solaris OS to Linux

Using the Endian Checking Portability Tool

Note: Before using this information and the product it supports, read the general information in “Notices” on page 19.

First Edition (September 2005)

This edition applies to the Endian Checking Portability tool and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. What the tool does 1

Chapter 2. Installing Endian Checking Tool 3
Prerequisites 3
Installation steps 3

Chapter 3. Using Endian Checking Tool 5
Running the tool 5
Post-processing 6
Understanding the output 6

Chapter 4. Endian Checking Tool restrictions 9

Chapter 5. Troubleshooting 11

Appendix A. Reporting defects to IBM 13

Appendix B. Messages 15

Appendix C. About endian differences 17

Notices 19
Trademarks 20

Chapter 1. What the tool does

The Endian Checking Tool (ECT) is similar in execution and principle to the **lint** code analysis tool, which is well-known to C developers. The tool performs an analysis of the application binary and sources, looking for code with potential chip architecture dependencies and problems in cross-hardware portability. The user will need to go through the list of such messages and fix the ones that are regarded by the user as significant issues.

It generates a report, which developers should take as a checklist of items to review in their source code. ECT will tend to err on the side of caution, flagging potential problems if it cannot determine conclusively whether they are actually problems or not.

This release of the tool includes the following capabilities:

- Ability to identify uses of ioctl system call
- Ability to identify uses of routines with potential endian issues
- Ability to check for ″long double″ usage (which has an inconsistent data size on Intel® (12 bytes) compared with most other platforms)
- Ability to check for potentially harmful uses of builtins
- Ability to check for consistent declarations of global variables

Chapter 2. Installing Endian Checking Tool

This chapter describes the prerequisites and the installation procedure for Endian Checking Tool.

You can choose beween the following installation methods:

- Install the included Solaris package file (ect.pkg), which by default will put everything in the /opt/mksl/ect directory.
- Install from the provided gzip'd tar file (or tarball) provided (ect.tar.gz).

If you do not have privileges or permission to install packages, the tarball will work best for you.

Prerequisites

Prior to installing, you will need to have some GNU tools installed on the machine. The quickest way is using the Freeware Companion CDs available at http://www.sun.com/software/solaris/freeware/. Verify that you have the following packages installed:

- SFWgawk gawk - Pattern scanning and processing language
- SFWgbin binutils - GNU binary utilities
- SFWgcc gcc - GNU Compiler Collection
- SFWgcc33 gcc-3.3.2 - GNU Compiler Collection
- SFWgcmn gcmn - Common GNU package
- SFWgdb gdb - GNU source-level debugger
- SFWgmake make - GNU make utility

A quick ″pkginfo | grep SFW″ will speed your checking.

Installation steps

You can install Endian Checking Tool by using a pkg file or a tar file. To install, follow these steps:

1. To install the ECT package file, log in as root and issue the following command on the command line:

   ```
   pkgadd -d ect.pkg
   ```

2. To expand the tarball, go to your chosen directory and issue the following command on the command line:

   ```
   gunzip -dc ect-solaris-beta.tar.gz | tar xvf -
   ```

   After you expanded the archive, you will have an ect directory (the subdirectories are explained later).

3. Once you have installed the package, or have expanded the tarball into your preferred location, you will need to set a few environment variables. The PATH variable must include the location of the GNU utilities, and, optionally, the location of ECT. Also, the LD_LIBRARY_PATH must be set correctly for the ECT executable files to work. For example, (using the bash shell, modify appropriately for your preferred shell):

```
export PATH=/opt/sfw/gcc-3/bin:/opt/sfw/bin:/usr/sfw/bin/:/usr/bin:/bin:/usr/sbin:/sbin
 export LD_LIBRARY_PATH=/opt/sfw/gcc-3/lib/
```

As mentioned, the PATH can include the location of ECT if you wish. You can either add /opt/mksl/ect/bin to the path (if you installed the package), add your ect/bin directory to the path (if you installed to a different directory or used the tarball) or add the path when you execute ECT (the tool handles that correctly), for example as follows:

```
/opt/mksl/bin/ect/ect.bash
```

or

```
/export/home/superdeveloper/ect/ect.bash
```

The following subdirectories and files are created when you install the tool:

ect/bin
Contains the executables. This is where ect.bash and ectgui.bash reside.

ect/data
Contains the report files, for example, filename.report

ect/doc
Contains the ErrorCodes and Welcome files.

ect/opp
Contains a report output post-processor, which can be used to organize the output for analysis. Instructions are available in the README in this directory.

ect/stats
Contains statistical data if ECTSTAT flag is set, for example, export ECTSTAT=/opt/ect/stats/ECT-timings.

Chapter 3. Using Endian Checking Tool

This chapter gives a brief introduction to the Endian Checking Tool and shows a typical porting workflow.

Before you begin: This tool will NOT operate properly if the source code and binaries to be examined are not properly built, set up, and in the correct locations. Read these instructions carefully.

1. Code must be built using the GCC compiler and the -gstabs+ and -save-temps switches. (This is critical as the default for -g is to use the DWARF-2 debug format)
2. Best results are achieved when the code is built using static (non-shared libraries) binding for analysis by the Endian Check Tool. This will allow the tool to properly check for endian errors in functions that are found in (normally) shared libraries. The ECT does not handle shared-library code at this time.
3. The source code must be available when you run Endian Checking Tool.
4. The resulting binaries must be located with respect to the sources in such a way that the debugging information (which includes where the source files are) is accurate. This would be the same as if you intended to debug your application using a debugger that also followed the source. This includes the .o and .i files created by the compile.
5. The executable file must be able to find all external shared libraries. The gnu debugger is used to obtain data without actually running the image file. The environment variable is called LD_LIBRARY_PATH.

   Note: Take care to set LD_LIBRARY_PATH correctly.
   If a library is not found in this path, an error message is written to the bottom right pane on the GUI, or to standard output if run in batch mode.
6. Prior to running, set the following environment variable: LANG=C

Running the tool

There are two ways to run the Endian Checking Tool:

- To run in command-line (batch) mode, issue the following command:

```
/opt/mksl/ect/bin/ect.bash <Executable File Spec> <Result File>
```

  At the conclusion of the run a clear text report <exename>.report is in the running directory. (Remember that if you used the tarball or relocated the package you will need to change the prefix path leading up to ect/bin, or just add it to your path.)
- You may also use the included GUI. You will need to verify that your current configuration allows for you to create X windows. The user interface requires X windows, so make sure your DISPLAY environment variable is set properly and that you can create X window objects (for example, try **xclock&**).

To run the Endian Checking Tool, follow these steps:

1. Start the Endian Checking Tool GUI by specifying the full path to the object (or by adding it to your PATH environment variable), for example:

```
/home/ect/bin/ectgui.bash
```

2. On the action bar, select **File –> Select** to choose the executable file you want to test. If there is data available from a prior run of the tool, it will be displayed in a tree format in the left pane and a message about the date and time of the run will be shown in the top right pane.
3. On the action bar, select **Action –> Process File** to start processing

In the bottom right pane, the progress of the tool analysis is shown. There are ten major steps in the processing, and progress is updated as each major step is completed. Note that the processing up to the 30% step is likely to consume 95% of the time.

Once processing is complete, the left pane of the GUI will display a tree with the results. The root will be the results file name, the first branch will contain source file names, and the ″leaves″ will contain references to items that the tool has tagged about the specified source files. Errors are displayed in red and warnings are displayed in green.

Clicking on a warning or error leaf will cause the source file to be displayed in an x-terminal that runs vi and you are placed on the appropriate line number. You may adjust the bin/editor.bash script to use the editor of your choice.

The ECT displays various percentage numbers to indicate its progress:

- 10% - Creates and populates the size table, gives size of all data types used.
- 20% - Creates the function tables from objdump and/or gdb's information function calls. The completion of this step is crucial for further processing.
- 30% - Creates the function reference tables. This is the longest running section of the tool.
- 40% - Checks for potentially problematic APIs.
- 50% - Ssearchs for ioctl usage.
- 60% - Validates function definitions/references match.
- 70% - Validates global variable declarations match.
- 80% - Finds and flags all instances of 'long double' data types.
- 90% - Finds and flags all instances of __builtin in the source.
- 100% - Writes the results to the data directory for use by the user or the GUI. Summary data is then created and displayed.

Post-processing

The .report in the running directory contains the details of potential cross-platform issues. The messages contained in this file should look like the ones in the doc/ErrorCodes.txt file. The .out file contains the messages displayed on the screen.

In the ″opp″ subdirectory is a report output post-processor, which can be used to organize the output for analysis. It can also help summarize duplicate error messages. Instructions on how to use the tool are contained in the README file in the ″opp″ subdirectory.

Understanding the output

ECT produces a report file and other intermediate files.

Report files contain the errors and warnings generated by the tool. As mentioned earlier, ECT acts like **lint** and displays both errors and warnings. You need to go through the listing and determine if any of the messages point out real problems or are simply flagged as things to be verified. If the examination of these errors and warnings indicate a real or significant problem, that issue must be corrected. The report file displays the file name which is being checked and the error number (for example, E30001 or E90001) along with the line number on which this error is reported. An example of E30001 is as follows:

```
/test/src/init.c – Line 199 E30001 Variable/parameter size mismatch arg
 2 size 4 in call to mystrncpy. (Defined in /test/src/init.c at line 190 size 1)
```

Reviewing the message reference section of this document, the definition of E30001 states that this is a variable or parameter size mismatch problem. In this case a size of 4 is expected, but a size of 1 is passed.

It is highly desirable to have all of the binaries run through the tool at one time to produce a consistent snapshot of the application. However, if the application becomes too large when statically linked, you might want to split the application up into sections if there are multiple binaries involved.

Chapter 4. Endian Checking Tool restrictions

This release of the tool includes the following restrictions:

1. Code may not have embedded Carriage Returns. The current symptom is that the application may loop endlessly due to attempting to count lines that do not exist.
2. Dynamic links will not be followed; dynamically linked libraries must be explicitly listed in the input source code to be checked. The tool does not follow calls for which there is no debugging information.
3. Nested calls to the same routine will appear to be a single call, for example:

   ```
   strstr(strstr(abc,"foo"),"bar");
   ```

   will result in only the outside call being found. Thus, endian errors for example, inside of nested calls will not be identified at present.
4. Intrinsic issues will not be identified if a macro expansion accounts for its use.
5. Endian checks on all assignments or usage are not performed.
6. Endian checks on macros, unions, and mixed-types are not performed.
7. Compatibilities with bit-field manipulations are not performed, and there is currently no examination of bit-field usages. Bit-fields can be manipulated individually with assignments or with masks. If the programmer mixes these mechanisms, it is likely to have different effects on platforms of different endian-ness. If however, the programmer always uses masks or always references the individual bit-field itself, then there is no issue. ″Compatibilities″ refers to this mixing of bit-field access mechanisms.
8. Currently the tool has limitations in parsing C++ code which includes Constructors, Destructors, and Operator Overloading.

Chapter 5. Troubleshooting

Problem: fatal: libstdc++.so.5: open failed: No such file or directory

Above message occurs repeatedly in the output, the tool fails to work.

Solution: The LD_LIBRARY_PATH variable is not set.

Problem: lct.bash: ggrep: command not found

Above message and tool does not run correctly.

Solution: Most likely the GUN tools are not in the path. Alternatively GNU grep is not installed. Verify that the appropriate GNU tools are installed per the installation section of this document. Verify that the tools are in the PATH prior to running.

Problem: File is not executable: or This tool expects to run on a binary executable, please adjust your usage and re-run

Similar problem as previous. If the grep command or the file command is not installed or in the path, ECT will not be able to run correctly.

Appendix A. Reporting defects to IBM

Errors should be reported to IBM® at the same location where you obtained this tool. Crisp explanations, small failing test cases can assist IBM in quicker resolution to the problem as well as minimize the development team needing to work with your source code. If you can resolve the problem, feel free to provide the code changes as well. This can also speed up the solution process.

Appendix B. Messages

Interactive messages are displayed in a message area at the bottom of the main window; error messages are displayed in a pop-up window.

W10001 Potentially harmful use of ioctl

Explanation: The ioctl system call typically has different sets of parameters that are very dependent on the underlying hardware. Please verify that its use here will work on this platform.

W20001 Potentially offensive API call

Explanation: A call is being made to a routine that has been identified as being a potential porting issue. Please verify the parameters and uses will work properly on this platform.

E30001 Variable/Parameter size mismatch

Explanation: A parameter, supplied by reference, in a call to a routine is of different size than is expected by the routine.

E30002 Variable/Parameter type mismatch

Explanation: A parameter, supplied by reference, in a call to a routine is of different type than is expected by the routine. For example, an integer value was supplied when a character was expected.

E40001 Global Variable size mismatch

Explanation: A global variable is defined as one size, and is declared as a different size in a separate module.

E40002 Global Variable type mismatch

Explanation: A global variable is defined as one type, and is declared as a different type in a separate module. For example:

```
module1.c:
 int myval;
 module2.c
 extern myval[4];
```

E40003 Typedef mismatch

Explanation: More than one definition of the same typedef has occurred

E40004 Structure mismatch

Explanation: More than one definition of the same structure has occurred

E50001 Variable assignment size mismatch

Explanation: A variable is attempted to receive a value from a variable or expression that is of different size.

E50002 Variable assignment type mismatch

Explanation: A variable is attempted to receive a value from a variable or expression that is not of the same type.

W60001 Potentially hazardous use of builtin

Explanation: Some intrinsic functions can operate differently on different platforms. Please vefify the code for this builtin will work properly on this platform.

W70001 Inconsistent Data Size on all platforms

Explanation: Intel uses 12 bytes for 'long double' others use 8 or 16, gcc on Solaris uses 16. Please verify the floating point usage of this variable is not going to be an issue in calculations or in file I/O.

W80001 Inconsistent Use of Bit fields

Explanation: In order to make the use of bit fields platform-independent, it is recommended that you either always look at the individual fields, or that you use masks to obtain the elements you are interested in. Please verify that your use of these bit fields is consistent.

E90001 Insufficent number of arguments supplied

Explanation: The caller to a function has not supplied the minimum number of expected parameters. Please adjust your code to match the function definition.

Appendix C. About endian differences

An example of code that would have endian changes flagged follows:

```
#include <stdio.h>

 // This is an example of Big-Endian and Little-Endian issue
 // The code will print different results based on the Hardware architecture.

 //  It will print "a=0x45, b=0x23" on xSeries (ia32, Little Endian)
 //  It will print "a=0x23, b=0x45" on Sparc, z/p/i Series (Big Endian)

 void testEndian( char& ra, char& rb) {
      union {
            short ab;
            struct {
                   char a;
                   char b;
             }inner;
       }var;

      var.ab = 0x2345;
      ra = var.inner.a;
      rb = var.inner.b;
 }

main() {
       char a,b;
       testEndian( a, b);
       printf( "a=0x%x, b=0x%x\n", a, b);
 }
```

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

- IBM

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java™ or all Java-based trademarks and logos, and Solaris are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux® is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.