Solaris to Linux Porting Guide

Ulrich Drepper
Red Hat, Inc.

1325 Chesapeake Terrace
Sunnyvale
California
94089
drepper@redhat.com

Solaris to Linux Porting Guide
by Ulrich Drepper

Revision History

Revision 1.1 , 2000-7-12
Minor changes to the legal notices.
Revision 1.0 , 2000-7-7
First published version.

Table of Contents

1. Introduction 7

About this Guide 7

2. Development Tools 9

Language Support 9
An Alternative Way 9
C Compiler Features 10
Invoking the Compiler 10
Language Extensions 22
Linker Invocation 25

3. System Interfaces 29

Interfaces Missing on Linux 29
Differing Interfaces Between Solaris and Linux 32
Limited Implementations 32
Linux Development Environment Namespace Issues 33

Chapter 1. Introduction

With the emergence of Linux as a viable computing platform, applications written for other Unix platforms are being ported to Linux. This guide will help you do this task. The focus is on porting applications from Solaris (in particular, on porting from Solaris running on 32-bit SPARC processors to Linux running on Intel IA-32 processors). However, if an application follows the standard programming interfaces, you can use this guide to port from other platforms.

Generally speaking, porting is quite simple. Because Solaris is a certified Unix implementation, it has passed the conformance tests of the Unix copyright holders. Linux is also designed with conformance to the Unix standard in mind. Although Linux has not undergone the conformance testing (due to the costs involved), chances are that if the programmer used only the set of interfaces covered by the Unix standard, you can reuse the code without any changes.

The porting problems that you can expect occur in several different areas:

- You cannot port from some special versions of the Solaris OS, such as Trusted Solaris. There is not yet an equivalent to these special versions. However, as Sun has not updated these versions of the OS since version 2.5.1, it is not likely that you would be attempting such a port.
- The tools used on the different platforms are different. As Sun does not develop tools for Linux, programmers who are using Sun's tools on Solaris have to switch their tools when doing the work on Linux. This can introduce additional problems beyond the differences in the use of the tools. The languages the compilers accept are slightly different (that is, they have different extensions on top of the base language).
- The programming interfaces differ. While both operating systems are designed to follow standards, differences in the implementations and different states of the implementations are unavoidable. The programming environment is regulated by a common standard (POSIX.2), but there is still room for differences and extensions.

About this Guide

The following discussion is based on the 7.x series of the Red Hat Linux distribution. This series features the 2.4 version of the Linux kernel and the 2.2 version of the GNU C library. Comparisons with older versions of either package are not discussed in this paper. On the Solaris side, it is sufficient to discuss the latest version of the OS, Solaris 8.

Where useful, we discuss upcoming developments on the Linux side. When deficiencies in the Linux system are mentioned, take into account that the development of Linux proceeds very rapidly and the described deficiencies might already be solved. If a particular problem has not yet been solved, this need not prevent you from continuing. Because all of the core operating system is available under an Open Source license, you can either make appropriate changes yourself or contract out the changes. Red Hat's custom engineering services are available for such projects.

Chapter 2. Development Tools

The most notable difference for programmers going from Solaris to Linux is the change in the development tools. The SPARCworks compilers from SunSoft, which are the compilers predominantly used in Solaris development, are not available on Linux. Linux developers predominantly use the GNU compiler collection (GCC). The difference is not necessarily a problem, as the quality of the development tools on the Linux side is equally high. It is only the use of SPARCworks-specific features is truly a problem.

Language Support

Every program that uses only portable language features should not have problems.

C

There should not be any problems at all with compatibility of C programs. The Linux C development and runtime environment is compatible with all the latest standards.

C++

As all compilers are catching up with the latest language standards, incompatibilities due to compilers being at different stages of the race are unavoidable. The compilers available on Linux are very well positioned in the conformance race and changes happen daily. If you require the latest C++ compiler that supports the most language features, it might be worthwhile getting a support contract from Red Hat[1] for the compiler tools.

FORTRAN

One bigger problem is the support for FORTRAN. There is a GNU FORTRAN compiler available and comes with the standard Red Hat Linux distribution, but it currently supports only FORTRAN77. The more recent variants (FORTRAN90, FORTRAN95) are not supported at all. There are commercial FORTRAN compilers available for Linux, but you will need to purchase one.

Java

The Java support on Linux is very good. There are several JDK implementations available for Linux and, with those, you can execute Java bytecode binaries. Even the performance is comparable. In addition to the option of using bytecode, it is also possible to use the GNU Java compiler to generate native code for the IA-32. The resulting executable is many times faster than the interpreted byte code, even if compiled just-in-time for execution.

An Alternative Way

One way to avoid the troubles with the changing development tools is to use the GNU compilers on Solaris. This has been an option since the earliest days of the GNU compiler and Solaris has always been one of the best supported platforms.

Many companies are taking advantage of this possibility because it allows retargetting the applications even beyond SPARC and IA-32. The GNU compiler is available for all modern processors that have at least a 32-bit architecture, whether these are for desktop, server, or embedded systems. This also includes support for the different operating systems for those hardware types.

For companies considering porting to Linux, switching to using the GNU compiler on the known Solaris platform makes the port much easier. Once the application can be generated on Solaris using the new tools, you can then attempt to compile on Linux. This will be easier because only API issues (not language issues) can impede progress.

C Compiler Features

If applications were never deployed on other platforms, the code will certainly contain some dependency on the platform on which it was developed. In this section we will discuss the features that are related to the compiler. Information about the system libraries will be given in a later section.

For the compiler we have to handle two compatibility issues:

Invoking the compiler

The command line options to select different modes differs significantly. Especially for writing highly optimized code, it is necessary to know some of the options.

Language Features

Both the SPARCworks compiler and gcc have extended the C language. gcc does this far more, but because the direction of porting is from Solaris to Linux, this is not an issue.

Invoking the Compiler

In general, there is no standard for the form of command line options of compilers. This leads to wide variations among the different compilers to a point where almost no option is the same on all platforms.

The SPARCworks compiler and gcc agree on the form and function for the following options:

Table 2-1. Common C Compiler Options

`-c`	tells the compiler to compile, but not link
`-o` *`FILE`*	specify output file
`-I` *`DIR`*	add include search directory
`-L` *`DIR`*	add library search directory
`-l`*`name`*	search and add a library with name `libname.a` (or `.so`)
`-A`*`name[(token)]`*	define ISO C assertion
`-D`*`name[=val]`*	define preprocessor macro
`-U`*`name`*	undefine preprocessor macro

`-c`	the object files are not linked together, object files are created.
`-g`	tell compiler to emit debugging information
`-E`	the compiler performs only the preprocessing and writes the result to standard output or to the file specified with -o
`-S`	instructs the compiler to compile, but not assemble, the code
`-w`	inhibits printing warnings

All the other options the compilers understand are either understood by one side and have no equivalent or are named differently. In the remainder of this section we will cover the most important of these options. To ease the transition to Linux, the list is sorted by the names of the options of the SPARCworks compiler. Names of the option the GNU compiler understands are given in parenthesis if their function does not exactly match that of Sun's compiler.

Table 2-2. Differing Option of the C Compilers

SPARCworks Option	gcc Option	Description
`-#`	`-v`	The compiler shows the invocations of all components on standard output. The gcc equivalent shows the invocations on standard error. Sometimes the tools the compiler invokes are not the last in the chain. gcc, for instance, normally invokes a tool named `collect` instead of invoking the linker directly. To see the invocation of the linker (done by collect), add the `-Wl,-v` to gcc's command line.
`-###`		Similar to -#, but the stages are not actually executed. There is no equivalent in gcc.
`-Bstatic`	`-static`	Tells the compiler to statically link the application. That is, the link editor will look only for files named `lib*.a`.
`-Bdynamic`	(default)	Tells the compiler to link the application dynamically. That is, the compiler will look first for files named `lib*.so` and, if no such file exists, it will look for files named `lib*.a`.

SPARCworks Option	gcc Option	Description
`-dy/-dn`	(implicit)	The compiler generates a dynamic executable. That is, the runtime linker is used to finish generating the executable. On Linux this happens implicitly. If an object is linked against any shared object, it is a dynamic binary. Similarly, of an object is linked without the use of any shared object, the program is linked statically and the runtime linker is not needed.
`-dalign`	`-malign-double`	The compiler is allowed to use double-word load and store assembly instructions. This is a SPARC-specific option that is tailored for the SPARC instruction set. The `-malign-double` option of gcc serves a similar purpose. This tells the compiler to possibly waste some memory (especially stack space) to ensure the double and long double variables are always accessed aligned.
`-err=warn`	`-Werrors`	Treat all warnings as errors.
`-erroff=tag`	(`-w`)	Disable printing specific warnings based on their tag number. There is no direct equivalent in gcc, but you can inhibit all warnings completely with `-w`.
`-errtags=yes`		Show error message tags. There is no gcc equivalent.

SPARCworks Option	gcc Option	Description
`-fast`	`-O`	The compiler enables options to maximize execution speed of the compiled application. gcc controls the optimizations to be performed using the `-O` option. With `-O2`, all optimizations but one are enabled. You can combine this with several machine-dependent options, such as `-fomit-frame-pointer`, `-malign-double`, `-fstrict-aliasing`, `-fargument-noalias-global`, and others. Using `-O3` adds a more aggressive inlining optimization. This does not always lead to improvements because the possibly increased code size and the additional cache misses could undo the advantages of inlining.
`-fd`	`-Wstrict-prototypes`	The compiler warns about K&R-style function declarations and definitions.
`-flags`	`--help -v`	Print summary of available options. When using gcc, use the options `--help -v` to get a summary for all available compilers (includes C++, Java, FORTRAN and others, if they are available).
`-fnonstd`		The floating-point arithmetic hardware is initialized in a non-standard way and signals are used to notify the program of raised exceptions. These signals will cause the application to dump a core if not caught. There is no direct equivalent for gcc. The program can cause the ISO C99 function to modify the floating-point control word of the FPU (for example, using `fesetenv`). It is also possible to use the special symbol LIB_VERSION. Linking a program with an object file containing a definition of a variable with this name enables you to select the mode in which the compiler works (possible values are those from the enum _LIB_VERSION_TYPE in <`math.h`>).

SPARCworks Option	**gcc Option**	**Description**
`-fns`		Part of what -fnonstd does (enable the non-standard arithmetic).
`-fround=`*`mode`*		Select the rounding mode for the program. The default IEEE mode is round-to-nearest. This is also the case on Linux, but there is no option to select a different mode. Use the ISO C99 interface `fesetround` to select the appropriate rounding mode.
`-fsimple=`*`[012]`*		Values other than 0 (which is the default) allow the compiler to make simplifying assumptions about floating-point arithmetic that might leads to accuracy loss or to different behavior (if the programs depend on exceptions raised). With the option `-ffast-math`, gcc also performs Architecture-dependent simplifications that can violate IEEE 754.
`-fsingle`		The compiler evaluates expressions using float arithmetic instead of double. This might improve execution on SPARC. For the IA-32 compiler there is in most cases no difference. To force evaluating all expressions as float, you can force the compiler to treat double as float by using the option `-fshort-double`, but this is not really an equivalent.
`-ftrap=`*`mode`*		Turns on trapping for the specified floating-point conditions. On Linux. use the portable ISO C99 interfaces.
`-G`	`-shared`	The linker creates a shared object instead of an executable.
`-H`	`-M/-MM`	Print path name of each of the files being compiled. Options with similar effects exist on Linux. You can use the `-M` and `-MM` options can be used to write this information out in a form usable to generate `make`(1) rules.
`-h` *`name`*	`-Wl,--soname, name`	Enables you to assign a name to the generated shared object. With gcc this has to be communicated to the linker using the option `-Wl,--soname,name` where the last component *`name`* is the name you want to give the shared object.

SPARCworks Option	**gcc Option**	**Description**
`-keeptmp`	`-save-temps`	Temporary files (such as assembly files) are not removed.
`-KPIC`	`-fPIC`	Tells the compiler to generate position-independent code. This option allows for the maximum possible number of symbols (as opposed to `-kpic`). With gcc the option is `-fPIC`. There is also an equivalent `-fpic` option but for IA-32 this option makes no difference at all.
`-Kpic`	`-fpic`	See `-KPIC`. The gcc equivalent is `-fpic`.
`-misalign`	(implicit)	Tells the compiler that some data is misaligned and therefore conservative load and store instructions must be used. This is not necessary on IA-32 because the processor handles alignment by itself.
`-misalign2`		Similar to `-misalign`. It also does not apply for IA-32.
`-mr`		Removed all comments from the .comment section. This is not handled by gcc.
`-mtsafe`	`-pthread`	Passes -D_REENTRANT to the preprocessor and adds `-lthread` to the `-mt` linker line. On Linux, `-pthread` behaves the same except that `-lpthread` is added to the linker line.
`-native`	(`-b`)	Directs the compiler to generate code for the native platform. There is no direct flag to do this with gcc. However, if you know the platform name, you can use the `-b` flag to select the appropriate target compiler.

SPARCworks Option	gcc Option	Description
`-nofstore`		Tells the compiler not to convert a floating-point value to a smaller type if the value is needed again immediately. Instead the value is left in a floating-point register, which means the precision of the expression is higher. gcc has the flag `-ffloat-store` which has the opposite meaning. The non-standard behavior is the default; to get true IEEE 754 results on x86, use the `-ffloat-store` flag.
`-noqueue`		Queue compile request until license is available. This is completely unnecessary on Linux because there is no license.
`-P`		Compiler preprocesses only the corresponding C files. There is no equivalent with gcc.
`-p`	`-p/-profile`	Prepare program for profiling with gprof. It also links with `-qp` special versions of the `libc` and `libm` libraries. There is a `-p` option for gcc on Linux/IA-32 as well. It also prepares the binary for profiling, but it does not cause the profiling versions of the libraries to be linked in. To link in the profiling versions of the `libc` library, use the `-profile` option. Note that this handles only `libc`, not `libm`.
`-R` *`dirlist`*	`-Wl,-rpath,` *`dirlist`*	The compiler instructs the linker to add the dirlist as the runtime search path for the dynamic linker to the binary.
`-s`	`-Wl,-S/(-Wl,-s)`	Remove all symbolic debugging information from the output file. With gcc one has to use `-Wl,-S`. There is also the option of using `-Wl,-s` to remove all symbol information.
`-V`	`-v`	Print information about version of each tool as they get started. When using gcc this information is contained in the output of `-v`.

SPARCworks Option	gcc Option	Description
`-v`	`-Wall`	The compiler will perform more semantic checks and performs lint-like tests. You can achieve this with gcc by using the `-Wall` option and possibly other `-W` options that are not included in `-Wall`.
`-Wc,`*`arg`*	`-Wc,`*`arg`*	Tells the compiler to pass arg as an argument to the tool named by c. This option exists in the same form on gcc, but the tools have different names and not all tools are supported by gcc. The tools that have equivalents in gcc are: Assembler: SPARCworks uses the name fbe or gas, while gcc uses a. Linker: SPARCworks uses the name ld, while gcc uses l. Preprocessor: SPARCworks uses the name cpp, while gcc uses p. No other tools have equivalents.
`-X[a|c|s|t]`		Select various degrees of compliance with ISO C. There are no different levels of the tests in gcc, but the `-pedantic` option is available to perform stricter tests. The `-traditional` option allowed you to test for K&R C, which is equivalent to the `-Xs` option, but it is not really possible to use this option on Linux anymore. The C library is an ISO C library and conflicts in many cases with the `-pedantic` option.

`-x386`	`-mcpu=i386/`
	`-march=i386`
		Optimizes for the i386 processor. The equivalent option For gcc is `-mcpu=i386`. The resulting code will run on all x86 processors that are capable of supporting the mode.
		To compile code that is even better optimized for the given architecture (but probably will not run on older processors), use the `-march=i386` option.

`-x486`	`-mcpu=i486/`
	`-march=i486`
		Similar to `-x386`, but for the i486 processor.

SPARCworks Option	gcc Option	Description
`-xa`	`-a/-ax`	Insert code for basic block invocation counting. For gcc this option is `-a`. Instead of using the `tcov` tool, you must use the GNU version `gcov` to process the output. To get even more detailed information, use the `-ax` option to enable jump profiling as well.
`-xarch=`*`name`*	`-march=`*`name`*	Selects a specific instruction set to be used. Often allowing the instruction sets of more modern processor variants means that the program can be optimized better. With gcc the equivalent option is `-march`. For x86, the possible names are i386, i486, i586, and i686.
`-xautopar`		Turns on the automatic parallelization. There is no equivalent for gcc.
`-xcache`		Defines cache properties for the optimizer. There is no equivalent for gcc.
`-xchip=`*`name`*		Selects the target processor. This is handled by `-march` in gcc. (See `-xarch`.)
`-xcrossfile`		Enable optimization and inlining across source files. There is no equivalent for gcc.
`-xdepend`		Collects information for inter-iteration data dependencies. Some of these optimizations might be performed by gcc with some of its optimization passes, but there is no specific option.
`-xe`	`-fsyntax-only`	Perform syntax checks only.
`-xexplicitpar`		Loops that are explicitly marked are parallelized. There is no equivalent in gcc.
`-xF`		Enable use of the Analyzer tool. There is no equivalent in gcc.
`-xhelp=what`	`--help -v`	Shows help information about what. You can use `--help -v` with gcc to get help about the flags.
`-xildon/-xildoff`		Enable/Disable incremental linker. There is no equivalent in gcc.

SPARCworks Option	gcc Option	Description
`-xinline=`*`fct, ...`*		Inline only the functions specified in the list.
`-xlibmieee`		Math routines return IEEE 754 style return values for exceptional cases. This is the default (`-lieee` can be used to, if necessary, overwrite conflicting selections) and the `-ffast-math` flag must be used.
`-xlibmil`	`-ffast-math`	Some math library functions are inlined. The gcc equivalent, `-ffast-math`, might introduce different results due to different precision.
`-xlicinfo`		Retrieve license information. Does not apply to gcc because there are no licenses to get.
`-xloopinfo`		Give information about loops that are parallelized.
`-xM`	`-M`	Generate makefile dependencies.
`-xM1`	(`-MM`)	Same as `-M`, but dependencies on files in /usr/include are not reported. The option `-MM` does something similar for gcc, but gcc avoids dependencies for files from the system directories. System directories do not always include `/usr/include` and often do include other directories.
`-xnolib`	`-nostdlib`	Does not automatically link in any libraries.
`-xnolibmil`	(default)	Prevents inlining math functions. This is the default in gcc and can be enforced with `-fno-fast-math`.
`-xO`	`-O`	Select the optimization level.
`-xP`		Print prototypes for K&R function definitions. There is no gcc equivalent.
`-xparallel`		Automatically parallelize loops, but also let the user specify what to do.
`-xpg`	`-pg`	The files are compiled with preparation for `gprof`-based profiling.
`-xprofile= collect`	`-finstrument-functions`	Instruct the compiler to generate code for collecting profiling information.
`-xprofile= use`	`-fbranch-probabilities`	Use the information from various runs of a program compiled with `-xprofile=collect`.

SPARCworks Option	gcc Option	Description
`-xprofile= tcov`	`-a`	The program will emit information which then can be examined using the `tcov` tool. To achieve the same result with gcc, use the `-a` option to instrument the code, then use `gcov`.
`-xreduction`		The compiler performs reductions for automatic parallelization. gcc performs strength reduction optimizations; however, they are not geared towards automatic parallelization.
`-xregs=names`	`-ffixed-<reg>/` `-fcall-used-<reg>/` `-fcall-saved-<reg>`	Specifies the usage of registers for the generated code. With gcc you can use the options `-ffixed-<reg>`,`-fcall-used-<reg>` , and `-fcall-saved-<reg>` to specify the use of certain registers.
`-xrestrict`	(see comments)	Enables you to specify handling of pointed-valued function parameters. By default no assumptions are made about the restrictiveness. With `-xrestrict=all` you can tell the compiler that all parameters should be considered restricted. gcc has the options `-fargument-noalias` to do the same as `-xrestrict=all`. With `-fargument-noalias-global` you tell the compiler that parameters are not even aliasing global data. Note that programmers should start using the ISO C99 keyword restrict to mark function parameter which are not aliasing other values. gcc recognizes the keyword and performs appropriate optimizations.
`-xsafe=mem`		Allows the compiler to generate speculative loads. The IA-32 processors do not provide such a feature.
`-xsb`		Instructs the compiler to generate additional symbol tables for the source code browser. This is specific to Sun's IDE and therefore has no equivalent in gcc.
`-xsbfast`		Similar to `-xsb`.

SPARCworks Option	**gcc Option**	**Description**
`-xsfpconst`	(`-fshort-double`)	Unsuffix floating point variables are treated as float instead of the default double. gcc provides the option `-fshort-double` which tells gcc to use the same size for double as for float. This is not the same but if double values are not used in the compilation unit, it is close enough.
`-xspace`	`-Os`	Perform only optimizations that do not increase the code size.
`-xstrconst`	(default)	String literals are inserted into the read-only data section. This is the default with gcc. To get the default behavior of the Sun compiler, use the option `-fwritable-strings`.
`-xtarget=`*`name`*	`-b`	Select the target platform for the compilation. With gcc it is necessary to know the name of the target and then use the option -b to pass the information to the compiler.
`-xtemp=`*`dir`*	(see comments)	Set directory for temporary files. With gcc, set the environment variable TMPDIR to the name of the directory you want to use.
`-xtime`	`-Q`	The compiler reports the time and resources used for the compilation.
`-xtransition`		The compiler warns about differences between K&R and ISO C. There is no option to explicitly enable this. gcc warns in general about ISO C rule violations. To not be warned, you must use the `-traditional` option to explicitly allow K&R behavior.
`-xunroll=`*`n`*	`-funroll-loops/-funroll-all-loops`	Instructs the compiler to unroll loops. You can use the option `-funroll-loops` to instruct gcc to unroll loops with known iteration counts. To unroll loops with unknown iteration counts, use `-funroll-all-loops`.
`-xvpara`		Involves warnings about a SPARCworks compiler specific `#pragma` and therefore is not supported by gcc.

SPARCworks Option	gcc Option	Description
`-Yc,`*`dir`*	(`-B`)	Specifies that the component c of the compiler can be found in directory dir. There is no gcc option that exactly matches this option. However, gcc has an option `-B` that takes a directory name as the parameter. The given directory will be searched for any of the components binaries. It is possible to repeat the `-B` parameter and so provide more than one directory with binaries.
`-Zll`		Creates a lock_lint database for the lock_lint tool. Because this is a Sun-specific tool there is no equivalent on gcc.
`-Zlp`		Creates a lock_lint database for the looptool tool. Because this is a Sun-specific tool there is no equivalent on gcc.
`-Ztha`		Creates a lock_lint database for the thread analyzer tool. Because this is a Sun-specific tool there is no equivalent on gcc.

Language Extensions

The SPARCworks compiler as well as gcc extend the C language. In the following we are describing the extensions of the SPARCworks compiler and possible equivalences on the gcc-side. The extensions are exclusively in the form of `#pragmas`.

gcc generally does not use `#pragmas`. The reason is that the old, pre-ISO C99, form of pragmas cannot be generated in macros. This changed with the introduction of _Pragma in ISO C99. Anyhow, gcc uses the keyword __attribute__ which can be used to add information to a definition or declaration of an object. The __attribute__ keyword is followed by two opening parenthesis. The reason for this is that it enables you to define a macro

```
#define __attribute__(ignore)
```

which can be used if the compiler does not understand attributes. More on the syntax can be found in the following table and the gcc manual. All the keywords (like __attribute__ and __aligned__) are given in the form with two leading and two trailing underscore characters. They are also available without underscores or with only two leading underscores. But these names potentially conflict with names in the user programs or the system. The safest possible solution is to use the names used here.

One has to be careful not to miss a use of `#pragma` in a converted program since gcc simply ignores #pragmas it does not know. Only when the `-Wall` option is used will it warn about ignored `#pragmas`.

`#pragma align` *`integer`* `(`*`variable`*`[,`*`variable`*`])`

The variables mentioned in the list are all aligned in memory at an address congruent to *`integer`* .

With gcc one has to mark the variable with an attribute:

```
double d __attribute__ ((__aligned__ (16)));
```

`#pragma init (`*`fct`*`[,`*`fct`*`])`

The functions named in the list are marked as constructors which are run before the program transfers control to the user code.

gcc uses the following syntax to define such a function:

```
void
__attribute__ ((__constructor__))
fct (void)
{
   ...
}
```

`#pragma fini (`*`fct`*`[,`*`fct`*`])`

The functions named in the list are marked as destructors which are run before the program terminates.

gcc uses the following syntax to define such a function:

```
void
__attribute__ ((__destructor__))
fct (void)
{
   ...
}
```

`#pragma weak `*`name`*`[,`*`name`*`]`

This `#pragma` can be used to specify that a symbol is created weak. gcc also recognizes this `#pragma`. It is nevertheless advised to use the attribute form:

```
int a __attribute__ ((__weak__)) = 1;
```

`#pragma redefine_extname `*`oldname newname`*

This pragma enables you to assign to a C symbol a different external linkage name. This is also possible with gcc when using the following syntax:

```
extern int oldname (int) __asm__ ("newname");
```

`#pragma ident "string"`

The SPARCworks compiler puts the provided string in the .comment section. With gcc one must use

```
#ident "string"
```

`#pragma int_to_unsigned fct`

Marks the function which returns an unsigned value as returning an int. There is no gcc equivalent.

`#pragma nomemorydepend`

Instructs the compiler that there are no memory dependencies for any of the iterations of a loop.

There is no single flag which can be used to signal to gcc this fact. But by using the restrict keyword for participating pointer and arrays, the compiler can automatically deduce this information.

`#pragma no_side_effect (fct)`

Declares that the named function has no side effects of any kind. gcc has a finer-grained mechanism for the user to use.

If a function has no effects except the return value and the return value depends only on the parameters and the values of global variables, one can declare such a function as pure:

```
extern int square (int) __attribute__ ((__pure__));
```

Note that the declaration and not the definition is marked. This is necessary since the generated code is not changing. The compiler can possibly optimize uses of the function which then can be subject of common subexpression elimination.

One step further go functions which entirely depend only on their parameters and also have their return value as the only effect. These functions can be marked as const:

```
extern int abs (int) __attribute__ ((__const__));
```

`#pragma pack(n)`

This `#pragma` specifies that the named structure is packed, i.e., laid out without padding. The gcc way of expressing this is:

```
struct foo
{
  ...
} __attribute__ ((__packed__));
```

This form will pack the structure to the most compact form. But gcc enables you to express even more. One can force individual structure members to be packed while other members are aligned in the normal way. The syntax for this is similar to the following:

```
struct foo
{
  char c;
  short int a __attribute__ ((__packed__));
  short int b;
};
```

In this case the member `a` is not aligned but instead follows immediately the member `a` in memory at offset 1. The member `b` is aligned and follows on offset 4.

`#pragma pipeloop(`*`n`*`)`

The SPARCworks compiler allows you to specify using the #pragma the minimum dependence distance of the loop-carried dependence. There is no equivalent in gcc.

`#pragma unknown_control_flow(`*`name`*`[,`*`name`*`])`

Specifies names of functions which violate normal control flow properties.

There is no general way to express this with gcc. But it is possible to mark functions which never return so that the compiler does not have to generate code to deal with the return program flow. This happens with the attribute noreturn:

```
extern void abort (void) __attribute__ ((__noreturn__));
```

This helps in situations like

```
if (condition)
  abort ();
else
  some_other_function();
```

where no code has to be generated for the if branch to jump behind the else branch.

`#pragma unroll(`*`n`*`)`

This `#pragma` allows the programmer to suggest to the compiler an unroll factor for a loop. There is no equivalent in gcc. Future versions of gcc will include support for software pipelining when some similar way to instruct the compiler will be needed.

Linker Invocation

Next to the compiler, the linker is the most important tool. It controls the final form of the program code and can improve the code generation significantly. Both Sun's and the GNU linker accept numerous options. We'll explain in the following list the most important options of the SPARCworks linker which are not available with the same name in the GNU linker and relate them, if possible, to functionality of the GNU linker. It is also advised that you read the documentation for the GNU linker to find out about the functionality which is not available in Sun's linker.

Table 2-3. Linker Options

SPARCworks Option	gld Option	Description

SPARCworks Option	gld Option	Description
`-a`	`-static`	This option enables the default behavior in the static mode. The linker is creating an executable and undefined symbols cause error messages. GNU ld has the option -static which also enables this behavior.
`-b`	(see comment)	If this option is given the linker does not generate special `-fPIC` relocations for accessing for symbols in shared object which would allow the code to be shared. Instead it creates faster, direct references which cause the text section to become non-sharable. There is no option for the GNU linker to achieve this. It can be achieved, though, by not compiling the source code with the option `-fPIC/-fpic`.
`-G`	`-shared`	Generate a shared object. The equivalent for the GNU linker is `-shared`.
`-i`		Ignore the LD_LIBRARY_PATH environment variable. There is no equivalent for the GNU linker.
`-m`	(`-M`)	Print a linker map. The `-M` option prints something comparable but with a different format and slightly different content.
`-s`	`-S/-s`	This option instructs the linker to strip symbolic information from the output file. The GNU linker has a finer grain for this functionality. If the `-s` option is used, the GNU linker strips all symbols from the file. This is much more than Sun's linker does. To get the equivalent of the `-s` option, one has to use `-S` with the GNU linker, which removes only the debugging information.
`-t`		This option allows you to turn off warnings about multiply defined symbols that are not the same size. There is no equivalent in the GNU linker.

SPARCworks Option	gld Option	Description
`-B eliminate`		This option causes the linker to eliminate all symbols not assigned to any version from the symbol table. There is no equivalent in the GNU linker.
`-B group`		Establishes a shared object and its dependencies as a group. There is no equivalent in the GNU linker.
`-B local`		This option causes the linker to reduce symbols not assigned to a version to local. With the GNU linker this can only be achieved using the wildcard matching in a version definition file.
`-B reduce`		Reduces symbolic information according to version specification when generating a relocatable object. There is no equivalent in the GNU linker.
`-D` *`token`*`[,`*`token`*`]`		The linker prints debugging information according to the tokens. There is no equivalence in the GNU linker.
`-h` *`name`*	`-soname` *`name`*	Set name as the shared object name. With the GNU linker the option `-soname` must be used.
`-I` *`name`*	`--dynamic-linker` *`name`*	Set name as the interpreter in the program header of an executable. The equivalent option for the GNU linker is `--dynamic-liner` *`name`*.
`-M` *`file`*	`--version-script` *`file`*	Specifies that the linker should use the named file as the mapfile. The GNU linker uses the option `--version-script`.
`-N` *`string`*		Causes the linker to add a DT_NEEDED entry with the given string as value to the dynamic section of the generated object. This cannot directly be achieved using the GNU linker. Instead one will have to link against a shared object with this string is the name.
`-P` *`auditlib`*		Specifies audit library. There is no equivalent in the GNU linker.
`-Qy`	(ignored)	Sun's linker adds a string identifying the linker to the comment section of the generated binary. This option is ignored for compatibility by the GNU linker. To emulate the behavior one could create such a string in the input files (using the `#ident` directive) or use the linker script to add it automatically.

SPARCworks Option	gld Option	Description
`-R` *path*	`-rpath` *path*	This option specifies the search direction to the runtime linker. The GNU linker uses the option `-rpath`. **Note:** Since the DT_RPATH tag of the ELF binary format is deprecated and replaced with a DT_RUNPATH tag, it is likely that at some point the GNU linker will have an option `-runpath`.
`-z allextract/-z defaultextract /-z weakextract`		These options allow you to modify the rules for extracting members from an archive. There is no equivalent in the GNU linker.
`-z combreloc`		Combines multiple relocation sections into one. There is currently no equivalent in the GNU linker.
`-z defs`	`--no-undefined`	Undefined symbols at the end of the linking process will cause a fatal error. This option is ignored by the GNU linker and the user has to use the option `--no-undefined` to achieve the same results.
`-z endfiltee`		Marks a terminal filter object. There is no equivalent in the GNU linker.
`-z initfirst`		Marks the object as having its runtime initialization function to be run first. There is no equivalent in the GNU linker.
`-z lazyload/-z nolazyload`		Enables and disables respectively lazy loading of dynamic dependencies. There is no equivalent in the GNU linker.
`-z loadfltr`		If this option is used when creating a shared object and the object is used by a filter, the dynamic linker knows that the object must be processed immediately at startup time. There is no equivalent in the GNU linker.
`-z nodlopen`		Marks a shared object as not being available to `dlopen`. The same option is understood by the GNU linker.
`-z nodelete`		Marks a shared object as not deletable even so that it does not get removed even if the program tells the system to unload the object. The same option is understood by the GNU linker.

Notes

1. http://www.redhat.com/services/gnupro/gnupro_plus.html

Chapter 3. System Interfaces

The system interfaces, as defined by the standard libraries such as `libc`, `libm`, and `libpthread`, make Linux appears as an almost complete Unix system. There are only a few function families and individual functions missing and a very small number of functions have a different, possibly limited behavior.

In this section we will first introduce the functions which are entirely missing on Linux. We will only cover the standard interfaces and a few very important and widely used other interfaces.

We will discuss interfaces which are different or have a limited functionality (mostly on the Linux side). Knowing about this can save a lot of debugging time.

But before we start, a few more words on the standards. Solaris is a certified Unix implementation and Linux is also modeled after Unix. Logically the most standard is the Unix standard, specified by the OpenGroup. This huge document governs almost all interfaces the standard libraries provide. This is at least true on the Solaris side, Linux has several extension. To enable the Unix interface, the C preprocessor macro _XOPEN_SOURCE must be defined to the value 500. This selects the interface of the fifth revision of the Unix standard. Adventurous people can already use the interface of the sixth revision by setting the symbol _XOPEN_SOURCE to the value 600.

The Unix standard is in large parts based on the POSIX standards ISO 9945-1 and 9945-2 (aka IEEE 1003.1 and IEEE 1003.2) with their numerous extensions. This means that Unix systems also implicitly implement POSIX.

Below POSIX, mainly because of history, further interfaces are available. These are for BSD and SVID systems. Today none of these last interfaces should be used directly since they are long obsolete. For programming one should either use the POSIX or the Unix interface.

One last interface is specified and this is the GNU interface. It includes everything Unix does and more. The extra interfaces not included in the Unix specification are not portable and one should know exactly when and why to use them.

Interfaces Missing on Linux

The Linux system interface lacks at the time of this writing some interfaces which are available on Solaris. These fall into two categories: non-standard interfaces and standard interface.

Into the former category falls one entire special and popular library: Solaris' non-POSIX thread library. The file in question is `libthread.so` (`libthread.a`) and the system header is <`thread.h`>.

The interface of this thread library can to some extent be mapped to functions in the POSIX thread library (which Solaris also provides). But there are exceptions. The first table provided here maps functions where there exists a corresponding interface in the POSIX thread library of Linux.

Table 3-1. Solaris Thread vs. POSIX Thread Library

Solaris Thread Library	Linux POSIX Thread Library
`thr_create()`	`pthread_create()`

Solaris Thread Library	Linux POSIX Thread Library
`thr_exit()`	`pthread_exit()`
`thr_getprio()`	`pthread_getschedparam()`
`thr_getspecific()`	`pthread_getspecific()`
`thr_join()`	`pthread_join()`
`thr_keycreate()`	`pthread_key_create()`
`thr_kill()`	`pthread_kill()`
`thr_self()`	`pthread_self()`
`thr_setprio()`	`pthread_setschedparam()`
`thr_setspecific()`	`pthread_setspecific()`
`thr_sigsetmask()`	`pthread_sigmask()`
`thr_yield()`	`sched_yield()`
`mutex_destroy()`	`pthread_mutex_destroy()`
`mutex_init()`	`pthread_mutex_init()`
`mutex_lock()`	`pthread_mutex_lock()`
`mutex_trylock()`	`pthread_mutex_trylock()`
`mutex_unlock()`	`pthread_mutex_unlock()`
`cond_broadcast()`	`pthread_cond_broadcast()`
`cond_destroy()`	`pthread_cond_destroy()`
`cond_init()`	`pthread_cond_init()`
`cond_signal()`	`pthread_cond_signal()`
`cond_timedwait()`	`pthread_cond_timedwait()`
`cond_wait()`	`pthread_cond_wait()`

No corresponding interfaces exist for the following functions:

`thr_suspend()`

The `thr_suspend()` function is stops the thread specified by a parameter. This is a big problem since the function cannot take into account mutex and other synchronization objects the thread is currently allocating. But this can lead to deadlocks since other, not stopped threads might depend on the synchronization objects.

For this reason the suspend function was not added to the POSIX standard (and it also gets removed from specifications such as the Java thread library).

`thr_continue()`

This interface is used to continue a stopped thread. Since `thr_suspend()` is rejected it is unnecessary for the same reasons to define a `thr_continue()` equivalent.

`thr_main()`

This interface can be used to determine whether the current thread is the main thread or not. There is no equivalent in the POSIX thread library and also not in Linux's implementation.

`thr_min_stack()`

There is no corresponding interface in the Linux threads implementation. But the Unix standard requires a symbol PTHREAD_STACK_MIN to be defined.

`thr_getconcurrency()`
`thr_setconcurrency()`

These interfaces are only necessary for m-on-n implementation (m user level threads on top of n kernel threads, m >= n). Since the Linux thread library is currently not designed in this way, this interface serves no purpose.

The situation with the semaphore interfaces is similarly. Sun has its own special implementation which in parts can be mapped to the POSIX interfaces. Just as for the thread stuff, Sun provides besides the own interface a POSIX semaphore interface.

One major difference between the Solaris and the Linux interface is that Solaris defines these functions in `librt` while on Linux they are in `libpthread`.

Table 3-2. Solaris Semaphore vs. POSIX semaphore Library

Solaris Semaphore Library	Linux POSIX Semaphore Library
`sema_destroy()`	`sem_destroy()`
`sema_init()`	`sem_init()`
`sema_post()`	`sem_post()`
`sema_trywait()`	`sem_trywait()`
`sema_wait()`	`sem_wait()`

Further differences in the interfaces of the thread library result from missing functionality. This will be covered in the next section.

One other big area where there is, to some extent, no equivalent functionality is STREAMS and TLI (transport layer interface). The STREAMS network interfaces are not available by default on Linux systems. However, the C library provides the interface but they will always fail unless the kernel extension implementing STREAMS is available.

If the program to be ported uses STREAMS, probably the best solution is to rewrite the networking part to use the basic POSIX socket interface. This does not only make the program use a POSIX interface, it also will improve the portability. Besides, Sun is moving in this direction as well and is keeping STREAMS only for compatibility.

If rewriting is not an option or if a short-term solution is needed, information about the Linux STREAMS implementation can be found at the Linux STREAMS[1] site.

The TLI implementation which is part of the Linux STREAMS implementation may or may not be compatible with the SysV specification. Red Hat has no experience whatsoever with this code and we are not advising to use it.

Differing Interfaces Between Solaris and Linux

Some of the interfaces available on Solaris and Linux differ despite having same name and same purpose. This can happen for three reasons:

- limitations of the implementation (mainly on the Linux side)
- different interpretation of the standard
- extension on top of a standard implementation

In the remainder of this section we will outline a few of the functions falling into this category. This list will most probably be incomplete. In most cases if a Linux implementation differs from a Solaris implementation, the differences were unintentional and will be removed when reported. Therefore if you have found a difference, it was probably not previously known. We will concentrate here mainly of those differences where functionality is missing because the underlying system does not support the operation.

Limited Implementations

No Process-Shared Synchronization Objects

The functionality which will probably be missed most when coming from Solaris are sharable synchronization objects. This means mutexes, semaphores, and conditional variables which can be shared between different processes (not threads, this of course works).

Calls to `pthread_mutexattr_setpshared`, `pthread_rwlockattr_setpshared`, and `pthread_condattr_setpshared` will fail if the second parameter is set to PTHREAD_PROCESS_SHARED. This functionality is not implemented in the thread library because the kernel implementation is lacking some features. Once this kernel limitation is lifted the functionality is available. Therefore it is not a good idea to completely disable all uses of these functions for Linux. Instead a check of the return value at runtime should be used to determine whether the functionality is available or not.

This same problem exists for standard interfaces which are not yet supported by Solaris (like `pthread_barrierattr_setpshared()`).

The only exception is the implementation of the spinlocks. The second parameter of `pthread_spin_init` can be set to PTHREAD_PROCESS_SHARED and the function will not fail.

Signal in Threaded Application

The Linux POSIX thread implementation is using individual processes for each thread. These are not completely separated processes but instead since they have to shared things like virtual memory, file descriptors and the like.

However, the kernel does not really know the difference between threads and processes. Therefore it does not handle the delivery signals correctly. There is no single process ID (each thread has its own which is another difference from the POSIX standard) and the kernel is delivering the signal to that thread.

There is currently no easy way out of this. Programs which depend on signal delivery will still work but all the signals are received by exactly the thread with the process ID that was used.

This is the status at the time of this writing. It might be that this problem is already worked around since it is a quite high-priority problem and will be worked on when possible.

Linux Development Environment Namespace Issues

Great care has been taken to ensure the namespace of the development environment is clean and compliant with the individual standards. This means that only the functions specified by the standards are made available when the appropriate feature-select macro for the standard is selected, and extended interfaces are only enabled when explicitly requested.

The available and useful feature select macros are those in the following table. They have to be defined (as C preprocessor macros) before including the first system header. The best way to do this is to add, e.g., -D_GNU_SOURCE to the commandline of the compiler.

`_ISOC99_SOURCE`

This macro selects makes all the functionality of the ISO C99 standard available.

> **Note:** The GNU C library includes all ISO C99 functionality. This introduces two kinds of problems: a) traditionally used interfaces might be reused (example: the `nan` function, there was a `nan` symbol on some implementation denoting the NaN value); b) silent changes in the implementation. The latter is especially bad but unavoidable. One often hit problem is the change in `strtod` et.al to allow the hexadecimal floating-point number notation which suddenly lets `strtod` accept expressions it would not have before.

`_POSIX_SOURCE`

Signals that POSIX.1 functions should be used. This is not very useful and `_POSIX_C_SOURCE` should be used instead.

`_POSIX_C_SOURCE`

This macro should be set to a value representing the date of the revision of the POSIX version one wants to use. Currently the last officially supported version is identified by the value 199506L. To select this revision one should add `-D_POSIX_C_SOURCE=199506L` to the command line.

There are more revisions of the standard coming along and it will become necessary to set the macro to higher values. But the work of the standards committee is not yet finished. See the description of `_XOPEN_SOURCE` for more information.

`_XOPEN_SOURCE`

This macro can be used to select between the interfaces of the various revisions of the X/Open Portability Guides (XPG) and the Single Unix Specification. The GNU C library implements only the XPG4 and Single Unix Specification interfaces.

Normally everybody wants to set this macro to the value 500 which selects the Single Unix interface. But it is also possible to set this macro to 600. This will select the interfaces for the next revision of this specification. This is especially interesting since the next revision of this specification will be unified with the POSIX standard. I.e., the value 600 will select also all the new POSIX interfaces. Some of these new interfaces are critical for high-performance applications and therefore this option is useful even though the specification is not yet finished.

`_LARGEFILE_SOURCE64`
`_FILE_OFFSET_BITS=`*`m`*

These macros enable the interfaces agreed on by all Unix vendors at the Large File Summit. They exist to enable 32-bit systems, which traditionally use types which limit them to files of sizes up to 2GB, to support large files. There are two modes in which this can happen.

The first mode it to define `_LARGEFILE_SOURCE64` which makes an additional set of functions and data types available. The new data types are ino64_t, off64_t, blkcnt64_t, fsblkcnt64_t, fsfilcnt64_t, and all composed types which contain at least one element with of the basic forms of the former types (e.g., struct stat contains an element of type off_t and therefore exists a type struct stat64).

The naming of the functions to use with these types follows the same scheme. There exists a `lseek64()` function which takes and returned off64_t values. There also is a `open64` functions which opens a file ready for these large file operations.

The second mode does not require cleanly written applications to be rewritten. Define `-D_FILE_OFFSET_BITS=64` on the compiler command line enables a mode in which the `*64()` functions and appropriate types completely replace the old interfaces. I.e., off_t is suddenly a 64-bit type and `lseek()` takes and returns such a value.

For more information about this extension, read the Large File Summit document which is available, for instance, as part of the Single Unix Specification.

`_GNU_SOURCE`

If this macro is defined, all of the previously mentioned interface plus several GNU-specific are made available. This provides the most convenient programming environment but programs are becoming less portable. Therefore care should be taken when writing applications which are expected to run on other systems as well.

To ensure programs are not accidentally using interfaces which they should not use, it is important to select the appropriate feature select macro and then use the `-Wall` option to catch all occurrences of functions without prototypes. All such cases (unless they are error in the user code) point to possible problems since a symbol or function outside the selected interface is used.

At link time the GNU C library already tries to make sure that internal interfaces cannot be used. This is done by not exporting these interfaces and therefore not providing the linker to bind the application to these symbols. There are two problems:

1. Not all interfaces can be hidden. Some interfaces are needed by other shared objects implementing the system libraries (such as the thread library). This does not mean that these interfaces can be used. Since all system libraries are released in synch this means that the internal, but exported, interfaces can change since the necessary changes can be performed in all system libraries.

   The general rule is that user programs must not use any of the interfaces whose name is in the system namespace. This means, all symbols with a leading underscore character must be used. There are only a very few exceptions:

   `_tolower()`
   `_toupper()`

   > Traditional Unix interface that, on older systems, is faster than `tolower()` and `toupper()` resp. There is no reason to use this interface on modern systems.

   `_exit()`
   `_Exit()`

   > The variation of the `exit()` interface is needed in some special situations. The second form is the one chosen for the same functionality by the ISO committee.

   `_setjmp()`
   `_longjmp()`

   > These are the old 4.3BSD compatible names for the modern `sigsetjmp()` and `siglongjmp()` interfaces where the second parameter is nonzero.

   No other symbol starting with an underscore character should be used directly by the application.

2. Linking statically will not show the same behavior since the symbol export restrictions only work with shared objects. Every single symbol of the `libc` is available when linking statically. This is not a problem since all dependencies are added to the application (and therefore changes in the `libc` implementation are not effecting the application) but one of course has all the disadvantages of static linking.

Notes

1. http://www.gcom.com/LiS/