Linux到Solaris™ 10操作系统
迁移指南

目录

第1章

简介

本文的目标在于为读者提供指南，帮助读者解决将基于Linux的应用程序迁移到Solaris™ 10操作系统时可能遇到的问题。通过使用支持信息，我们还讨论了两种环境在操作系统功能、架构、系统调用和应用程序开发环境方面的优缺点。此外，我们还提供了一些有关Sun提供的对迁移活动非常有用的工具的信息。

Solaris 10操作系统和Linux概述

Solaris 10操作系统是一种适用于任务关键型应用程序的操作系统，具有成熟、可靠、安全、高性能以及可伸缩等特点。它有着丰富的功能，其核心为与POSIX兼容的UNIX®系统，派生自Sun和AT&T联合创建的UNIX System V版本4（SVR4）。通常情况下，Solaris操作系统提供了丰富的与POSIX兼容的命令、实用工具以及库的集合。有关POS IX标准的2004修订版的详细信息，请访问：

http://www.opengroup.org/onlinepubs/009695399/toc.htm。

为了更好地与其他UNIX版本兼容，Solaris软件还绑定了BSD系列的UNIX版本中的命令、实用工具和库。Solaris操作系统既可在基于SPARC™的操作系统上运行，也可在基于x86/x64的系统上运行，既支持在全部两种处理器架构上的32位应用程序，也支持64位应用程序。Solaris操作系统已经被证明了能够遵循大多数领先的行业标准，如X/Open® XPG4、XTI、UNIX98、POSIX和FIPS。

最近Sun通过将Solaris源代码中的大部分开源，启动了一个名为OpenSolaris™的项目。这也是将商用品牌的UNIX开源的第一例。相关的开源社区已经开始运行，所开发的OpenSolaris操作系统已经可以使用。OpenSolaris项目对于社区开发来说是一件非常激动人心的事件，同时它也为开发人员和用户用他们创新的扩展来开发和改进操作系统提供了一个很好的机会。OpenSolaris操作系统还提供了Solaris 10操作系统所提供的许多前沿功能。因此，我们在本指南中讨论的大部分内容也适用于同样作为目标平台的OpenSolaris。

和OpenSolaris一样，Linux也是一种流行的开源操作系统，其用户基础也在大幅度增长。这种技术在多年前就已经成熟，并可提供一个可与POSIX几乎完全兼容的开发和部署环境。Linux的行为还能够不同程度地与BSD UNIX和SVR4兼容。近几年来，领先的Linux商们已经同意遵循Linux Standard Base（LSB）标准。遵循LSB确保了可迁移应用程序能够使用标准的API集。虽然Linux可在许多架构上运行，更常见的情况是Linux部署在x86/x64架构上。在本文

中，为了使文章简洁，我们将以Red Hat Enterprise Linux 3（RHEL3）和Red Hat Enterprise Linux 4（RHEL4）这两种Linux环境为例进行说明。

作为开源软件中之独占鳌头者，Sun在Solaris 10操作系统中内置了许多功能，使其成为实现与Linux互操作性的领先的平台。Sun以打包的二进制文件的形式为各个版本的Solaris操作系统提供了许多与Linux兼容的命令、实用工具和开发工具。其中的大部分都作为Solaris 10的一部分在Solaris Companion CD中发布，安装在`/usr/sfw`这个位置。您可从http://sun. com/freeware获得这些软件包的最新版本以及其他相关信息。一旦在安装了Solaris操作系统的计算机上安装了这些附加软件包，即使是在Solaris环境中，用户也可以获得与Linux相似的功能。
由于Solaris OS和Linux均能提供POSIX功能，而Solaris 10操作系统绑定了许多Linux互操作性功能，Linux和Solaris操作性实际上难分伯仲。因此，我们很自然地期望能够很容易地将Linux的应用程序移植到Solaris中。而且，由于最近的大多数Linux发布都是与LSB 2.0兼容的，所以这项任务看起来就更加简单。但是，实际移植应用程序的时候应该非常小心。需要附加操作的应用程序是：使用架构和操作系统特定的扩展、具有内核空间权限的访问、到设备的直接访问以及不受支持的编译器扩展等。不遵循可移植代码准则的应用程序需要进行更多的相关操作。

Sun帮助

Sun为将操作系统从Linux移植到Solaris操作系统提供了工具、准则和帮助。

在没有应用程序源代码的情况下，Sun提供了在使用Solaris操作系统的x86/x64系统上运行基于Linux x86的应用程序二进制的方法。这种免费的实用工具叫做`lxrun`，可从http://www.sun.com/software/linux/compatibility/lxrun/下载。

`lxrun`是位于Solaris和Linux x86二进制可执行文件之间的模拟器，它动态重新映射了系统调用，使它们无需修改即可运行在Solaris OS for x86平台上。 有关此实用工具的详细信息，请参见`lxrun`(1)网页。

但是，更好的是获取应用程序的完整移植，并创建Solaris操作系统的本机二进制文件。这就使得复杂应用程序和ISV产品能够最大程度上发挥Solaris 10产品提供的成熟平台的优点。在启动这种需要将现有Linux应用程序移植到Solaris 10操作系统的项目之前，标识出代码中可能存在问题的区域是十分有用的。

Sun提供了一个代码扫描工具来分析C/C++源代码并标识代码和功能兼容性问题。这个工具称为LinCAT（Linux Compatibility Assurance Toolkit），也叫Linux Analyzer for C/C++ Source Code。此工具基于Java™技术，可用于标识C/C++源代码文件中的移植问题。此实用工具可分析在将源代码文件由Linux移植到Solaris操作系统时可能带来的大多数问题。它还可帮助估计进行移植所需要完成的总工作量。

LinCAT能够标识可能出现的问题，并提供有关如下内容的信息：

- 函数调用的名称、位置以及使用频率
- 具有潜在移植问题的源代码
- 解决所选择的移植问题的解决方案

- 不兼容调用——通过查找函数调用数据库，提供有关解决方案的建议

在大型项目中，使用LinCAT可为开发人员节省大量的时间和精力。LinCAT的最新版本可从 http://developers.sun.com/prodtech/solaris/downloads/lincat/index.html下载。

LinCAT生成的报告还帮助评估所需完成的工作量。一旦标识了所需的更改，实际更改代码、编译代码以及生成应用程序二进制文件的工作就相对简单了。遵循软件开发的常规步骤（如生成、测试和发布），应用程序用于Solaris平台时其功能就不会受到任何影响。Sun为解决性能问题（如果存在）提供了动态跟踪（DTrace）等高质量工具。

在这种实际迁移类型的项目中，最好遵循系统化的，面向过程的方法。在附录G中，我们就将应用程序迁移到Solaris 10平台提供了一个典型的工作流程。

附加资源

Sun为开发人员和管理员提供了许多这方面的基于Web的附加资源。Sun还特别为Linux管理员提供了一些教程，内容涉及对Solaris操作系统和其他关联IT基础结构的深层次介绍。除了核心技术之外，这些教程还涉及系统、网络和应用程序管理。这些教程提供了非常有价值的信息，可帮助管理员快速了解从Linux迁移到Solaris操作系统时可能遇到的各种问题。同样，不熟悉Sun™ Studio 11软件等专业开发工具的开发人员也可以获得许多基于Web的资源。

有关从Linux迁移到Solaris平台的详细信息，请参见Sun的Web站点上关于迁移的部分：

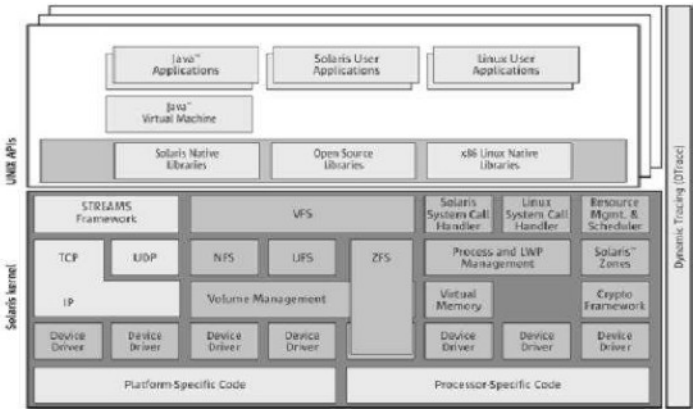
http://partneradvantage.sun.com/partners/migration/。

第2章

Linux与Solaris 10操作系统的关系

Solaris 10操作系统采用了模块化设计方式，因此也适用于新的处理器和硬件架构。这种设计还允许通过革命性的方式添加新服务。在Solaris 10版本中，早期Solaris版本的众所周知的属性（可靠性、可用性、安全性、可伸缩性、可管理性和互操作性）得到了进一步的发扬光大。您可通过如下地址找到一个提供了更多有关Solaris 10操作系统的信息的白皮书：http://www.sun.com/software/whitepapers/solaris10/classbyitself.pdf。

因此，Solaris 10操作系统是承载大型任务关键型的应用程序（尤其是当前在Linux平台上运行的应用程序）的最流行的平台之一。Sun在Solaris 10操作系统中内置了许多功能，以使其与Linux兼容。下图显示了Solaris平台支持的多种服务。



除了上述服务，Solaris 10操作系统还提供了成百上千的标准Linux命令、工具、实用工具和服务。生成这些程序包的开源代码库与Linux的相同。这些程序包作为Solaris附赠CD的一部分提供，安装在目录/usr/sfw下。这些程序包的最新版本以及相关信息也可从http://sun.com/freeware免费下载。

所有Solaris用户都可获取这些程序包以及Solaris的常用功能，它们为Solaris用户提供了一个很大的机会。用户可选择使用与Linux相似的程序包，从而极大地增加与Linux的可移植性。注意，您可能需要适当地改变外壳的环境变量（如PATH），以便为这些与Linux相似的命令和服务提供更高的优先级。

为了让您能够做出更加明智的选择，下面列出了各种类别的部分常用Linux工具，这些工具也可在Solaris 10操作系统中找到。

类别	工具名称
GNU命令集	所有用于访问、文件操作、编辑、通信，等等的常见UNIX命令
外壳	Bash
脚本编写	Bash、Perl、Python、Tcl/Tk、gawk和sed
GUI和GUI开发	GNOME、KDE和GTK+
GCC编译器和运行时库	C、C++、G77、glibc……
GNU软件开发工具	Ld、ldd、lorder、ar、elmdump、Gmake、ant、nm、gprof、RCS和CVS
GNU调试工具和派生工具	Gdb和DDB
Web服务	Apache、Tomcat和JBoss
Java服务	Sun JRE和Sun JDK™

如果Solaris 10操作工具已经就平台服务、命令、工具、库和软件开发环境提供了非常出色的Linux基准，那么将应用程序移植到Solaris操作系统这项任务剩下的工作就是对实际代码进行修改。如果两边使用的开发工具和编译器相同，则移植时与工具相关的工作的复杂度就非常低。

重要的一点是要注意到GNU Compiler Collection（GCC）和其他GNU工具在Linux和Solaris操作系统上均有提供。Sun提供的Sun Studio 11编译器和工具在Solaris和Linux平台上均可用，并且还免费提供证书。

但是，通常情况下开发人员喜欢在Linux端使用GCC工具链，而在Solaris端使用Sun提供的开发工具。而且，在某些情况下，进行迁移时可能要求在Solaris端使用更专业的工具。Sun Studio 11软件就是这样的工具，它在Solaris开发人员中更是流行。Sun Studio 11产品会在发布产品的同时提供自己的编译器和工具集。在这样的情况下，开发人员需要了解GNU产品和Sun Studio 11工具之间存在的一些微小的语义和句法差异。我们将在本文的后面部分介绍这些内容。

第3章

移植问题

处理器架构问题

低位优先与高位优先

与源平台和目标平台中的字节排序相关的问题会导致许多潜在的问题。在计算机内存中存储信息的方法以及字节在内存中的顺序称为字节顺序。当多字节数组字段中最大的字节（MSB）存储在最低的内存地址时，这个系统就称为高位优先。当多字节数组字段中最小的字节（LSB）存储在最低的内存地址时，这个系统就称为低位优先。Intel和AMD处理器都是低位优先，意味着最小的字节存储在最低的地址。相反，UltraSPARC®处理器则是高位优先。请参见下面的例子。

以低位优先格式在Intel或AMD处理器中存储0X01020304，假定采用4字节存储方式，如下所示：

00000100	00000011	00000010	00000001
内存地址： 100	101	102	103

以高位优先格式在UltraS PARC处理器中存储相同的数字0X01020304，假定采用4字节存储方式，如下所示：

00000001	00000010	00000011	00000100
内存地址： 100	101	102	103

Solaris操作系统即可用于高位优先模式（SPARC版本），又可用于低位优先模式（基于Intel或AMD的x86/x64版本）。字节顺序由平台使用的硬件委托管理。Solaris操作系统的一个优点就在于它以透明的方式处理字节顺序问题，因此ISV或开发人员不需要太多地考虑应用程序能在多久的时间内在Solaris OS for SPARC平台或Solaris OS for x86/x64平台（用于Intel或AMD）中可用。Sun的软件架构允许SPARC平台上的Solaris操作系统中的高位优先应用程序兼容运行在x86/x64平台（用于Intel或AMD）的Solaris 操作系统中。一般说来，不需要进行再造。但是，字节顺序的

差异也会导致问题，特别是当应用程序使用低级别代码，而该代码充分利用了最初为它编写的硬件功能时。

通常情况下，数值数据内的字节顺序对应用程序来说也是透明的，但下面所示的应用程序除外。

- **通过网络发送数据：**标准C库提供了以下例程：ntohl()、ntohs()、htonl()和htons ()，用于将数值数据在网络字节顺序和主机字节顺序之间进行转换。这些例程是在主机平台上实现的，用于处理主机字节顺序和网络字节顺序之间的转换，我们应始终使用这些例程隔离应用程序与基础架构。

- **使用指针访问数值数据的各个字节：**如果您使用数值数据类型的各个字节存储和检索值，则基础平台的字节顺序属性会使您得到不同的结果。在本例中，代码是不可移植的。

例如：

Intel或AMD平台（低位优先格式）上的值0X01020304

00000100	00000011	00000010	00000001

内存地址：　　100　　101　　102　　103

```
int flagVal = 0X01020304;

char *cptr = (char *)&flagVal;

printf("Val is %d\n", *cptr);
```

在本例中，低位优先计算机将显示4，而高位优先计算机将显示1。要使此代码可移植，您应使用各种字符类型的变量而不是数值数据类型进行存储。相反，您可以使用为每个值提供一个字段的结构变量，并使用结构字段名称获取或设置这些值。

应用程序数据存储

通常情况下，文件系统对于字节顺序来说是中立的，Solaris操作系统的SPARC和x86/x64版本（Intel或AMD）之间的文件交换是不成问题的。但存储可在不同平台之间共享的原始数据的应用程序就很成问题。例如，如果用于SPARC平台的Solaris操作系统上的应用程序使用原始格式将数据结构写入到文件中，则存储在这些文件中的数据将是字节顺序相关的数据。从基于Intel或AMD处理器的Solaris系统读取或写入这些数据文件将导致数据产生字节顺序方面的问题。简言之，存储在文件中的二进制数据（原始数据）通常不能在SPARC和x86/x64（Intel或AMD）平台之间进行转换。

存储可在不同平台之间共享的数据的应用程序可通过如下两种方式之一处理字节顺序问题：

- 使用文本文件或字符串将数据存储为应用程序定义的，与字节顺序无关的格式。

- 选择高位优先或低位优先约定中的一种，然后在必要时进行字节转换（可能要使用XDR等启用技术）。

对跨平台兼容性的需求可理解为大多数应用程序在高位优先和低位优先Solaris环境中运行了数年之久而未出现任何问题。这其中包括个人开发的应用程序，也包括供应商提供的大

多数数据库管理系统，包括Oracle、Informix和Sybase。

存储数据与对齐方式

不同的平台也具有不同的数据存储顺序，因此希望使用特定的存储顺序的代码就将是不可移植的。请看下面的例子：

```
struct  date
{
         char   yr;
         char  mon;
         char day;
         char x;    // struct var size is 4
} planned_date, end_date;

if ( (* (long * ) &end date) > (*(long *) &planned date))
{
         printf ("Sorry, You missed the deadline for migrating to Solaris \n");
}
```

本例中的代码假定year存储在month之前。这段代码在x86平台上能够正常工作，但无法在基于UltraSPARC技术的计算机上正常工作。要避免出现此类错误，可移植代码应分别比较结构变量的各个字段，而不要依赖存储顺序。例如：

```
if (compare_dates(end_date, planned_date) > 0)
{
        printf ("Sorry, You missed the deadline for migrating to Solaris \n");
}
int compare_dates (struct date date_to_compare, struct date date_compare_against)
{
         if (date to compare.year != date compare against.year)
                   return (datetocompare.year - date compare against.year);

         if (date_to_compare.month != date_compare_against.month)
                 return (date_to_compare.month - date_compare_against.month);

         if (date_to_compare.day != date_compare_against. day)
                 return (date_to_compare.day - date_compare_against.day);
```

平台不同结构中的字段号的对齐也有差异，从而导致填充要求的不同以及结构大小的差异。为了使代码可移植，应总是使用size of运算符访问结构大小。

读写结构

大多数程序使用标准C例程以二进制形式将数据作为完整结构读写到永久存储区中。这些例程需要知道正在读写的数据大小。由于不同的平台使用不同的对齐方式和数据类型大小，而结构大小又各不相同，因此应使用size of运算符指定在这些例程中读写的字节数。

例如，如果程序需要读取My Record类型的记录（已知数据总共有25字节），则不应使用如下所示的代码：

```
MyRecord myrecord;

fread(&myrecord, 25, 1, fp);
```

相反，您应使用下面的约定：

```
MyRecord myrecord;

fread(&myrecord, sizeof(MyRecord), 1, fp);
```

32位与64位问题

Solaris操作系统和Linux均可运行在32位和64位处理器上。UltraSPARC处理器是64位处理器。来自Intel和AMD的与x86兼容的处理器是32位处理器，默认的地址空间为4千兆字节。Intel EM64T和AMD64系列处理器是64位处理器，支持64位地址空间。但是，这些处理器也可以32位兼容模式运行，并可运行所有旧的32位兼容应用程序。Solaris 10操作系统具有处理其所在平台上的相关32位和64位问题的全部配置。这个问题仅在错误编写的应用程序对各种数据类型的大小以及相互转换进行了假设时才出现。

C语言规范没有指定任何整数数据类型的大小，而只是以如下方式描述了它们的相互关系：

```
sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof (long)
```

各个数据类型的大小由操作系统使用的数据类型模型确定。32位的操作系统使用ILP32数据模型，该模型将int、long和pointer数据类型的大小指定为32位（因此ILP32数据模型也是32位的）。

LP64数据类型模型这种业界标准用于64位操作系统。LP64数据类型模型为long和pointer数据类型指定的大小为64位，并且它不修改其他数据类型的大小。32位和64位操作系统下各种数据类型的大小关系如下所示。

C数据类型	ILP32数据类型（大小以位为单位）	LP64数据类型（大小以位为单位）
char	8	8
short int	16	16
int	32	32
long	32	64
float	32	32
double	64	64
pointer	32	64
enum	32	32
long long	64	64
long double	128	128

64位操作系统也支持32位操作系统API以便提供向后兼容性，应用程序不需要进行任何更改。将一个32位应用程序转换为64位应用程序时，理想情况下仅需要与64库重新兼容并重新链接。实际上，大多数应用程序都要在一定程度上更改和清除代码后才能同时在32位和64位操作系统上运行。

如果32位和64位API的版本不同，则最好使用`#if defs`为不同的代码段提供条件兼容。这比为两个操作系统维护不同的源代码文件要好，但是，考虑到可读性的问题，最好不要过多地使用这种方法。Solaris和Linux平台可为数据类型模型提供功能测试宏。有关Solaris操作系统的功能测试宏的详细信息，请参见Solaris 64-bit Developer's Guide（网址为：

http://docs.sun.com/app/docs/doc/816-5138）。

ILP32数据类型模型不区分`int`、`long`和`pointer`数据类型，因为它们都是32位长整型。但是，在LP64数据类型模型中，`long`和`pointer` 数据类型的大小更改为64位。因此，大多数代码问题均源自这些数据类型的错误使用。下面的准则可帮助您消除可能出现的32位和64位移植问题：

1. 更正`int`、`long`和`pointer`数据类型的混用，以便防止数据丢失。
2. 对地址的显式算术操作应替换为等效指针算术表达式。
3. 检查联合和结构字段，查看其是否影响了数据类型大小更改。
4. 检查所有应用程序的内部数据结构中的漏洞。在LP64数据类型模型中，在结构字段之间使用额外的填充可使`long`和`pointer`字段增长为64位。
5. 表达式赋值可能会受到影响，因为标准的C转换规则会受到数据类型大小更改的影响。您应使用显式转换并声明常量类型（对整数常量使用`u`、`U`、`l`、`L`、`f`和`F`后缀），以获得需要的和一致的结果。
6. 当代码转换为64位时，始终使用显式转换以获得想要的结果。这可防止C语言转换规则带来的符号扩展问题。将整数提升为无符号整数的问题就是这种转换问题的一个例子。
7. C编译器假定模块中使用但没有特别定义或声明的所有函数或变量都是整数。LP64数据类型模型中使用的`long`或`pointer`数据类型的任何函数和变量均会导致截断。
8. 在LP64数据类型模型中，`size of()`运行符具有无符号`long`的有效类型。因此，将其传递或转换为`int`数据类型时要特别小心，因为这会导致数据丢失。
9. 应用程序性能的变化可能是由数据大小过大引起的，并可能会需要附加的代码更改以提高性能。
10. 如果数据是发送到或接收自外部接口的，则应确保所发送数据的大小与接收数据的外部接口的数据大小相同，反之亦然。

多线程编程

大多数操作系统使用多线程处理作为一种增加计算粒度，以便更有效地使用可伸缩环境中的资源的方法。内核模式和用户模式的服务均依赖于操作系统实现的基础线程模型。为了

维护各种数据结构之间的完整性以及为系统上运行的所有线程和进程提供受控制的并发性，应使用互斥量和信号量这样的实体。但是，即使是在与UNIX相似的操作系统之间，实现的细节也有很明显的差别；这些细节在移植项目中可能会成为问题。

Solaris内核一直是支持线程的。内核在内部调度线程，以便以最优的方式分享可用的CPU资源。这些内核线程是绑定到轻量级进程（LWP）的。

近些年来的Solaris版本支持了多种线程实现。Solaris线程尤其明显。但是，Solaris操作系统提供的POSIX线程库已成为事实标准，因为它易于在UNIX平台（包括Linux）之间移植应用程序。有关Solaris线程的更多详细说明可在Jim Mauro和Richard McDougall合著的，Sun Microsystems Press出版的Solaris Internals（Solaris内核结构）中找到。使用线程编程需要具有较深层次的技能。Steve Kleiman、Devang Shah和Bart Smaalders的著作Programming With Threads是有关此主题的一本有用的参考书。

在Linux端存在线程包的多个实现，而且所有这些实现近年来都得到了明显的改进。大的实现都是与POSIX兼容的。但是，实现存在多个版本也会带来性能问题。基于内核版本2.2和2.4的Linux发布支持Linux线程。要更好地移植Solaris应用程序，请使用Solaris线程，不过也可以使用另一种开源程序包Solaris兼容线程库（SCTL）。-{}-

Linux 2.6内核捆绑了一个新的线程模型，称为本地POSIX线程库（Native POSIX Thread Library，NPTL），这是一个更加有效的线程库。

除了实现的各种版本，大多数线程库调用的语法都很相似，并且都遵循POSIX线程标准。因此，就移植而言，Linux端线程库调用可以很容易地移植到Solaris 10操作系统。当然，我们可能还要了解makefile的链接器命令中的线程库的顺序和名称。-lpthreads通常就是用于在Solaris平台上访问POSIX线程的链接器选项。

如果由于某种原因而需要在Solaris端使用Solaris线程，则需要额外更改代码，并使用Solaris线程调用替换POSIX线程调用。此外，执行此项操作时还需要分析代码的语义精确度。下表提供了相应的Solaris线程和POSIX线程之间的对应关系，以供读者参考。

编号	POSIX线程函数	Solaris线程函数
1	pthread_create()	thr_create()
2	pthread_exit()	thr_exit()
3	pthread_getschedparam()	thr_getprio ()
4	pthread_getspecific()	thr_getspecific ()
5	pthread_join()	thr_join()
6	pthread_key_create()	thr_keycreate ()
7	pthread_kill()	thr_kill()
8	pthread_self()	thr_self()
9	pthread_setschedparam()	thr_setprio ()
10	pthread_setspecific()	thr_setspecific ()
11	pthread_sigmask ()	thr_sigsetmask ()
12	sched_yield()	thr_yield()
13	pthread_mutex_destroy()	mutex_destroy ()

14	pthread_mutex_init ()	mutex_init ()
15	pthread mutex lock ()	mutex lock ()
16	pthread_mutex_trylock()	Mutex_trylock ()

编号	POSIX线程函数	Solaris线程函数
17	pthread_mutex_unlock()	mutex_unlock ()
18	pthread_cond_broadcast()	cond_broadcast ()
19	pthread_cond_destroy()	cond_destroy ()
20	pthread_cond_init ()	cond_init ()
21	pthread_cond_signal()	cond_signal ()
22	pthread_cond_timedwait()	cond_timedwait ()
23	pthread_cond_wait ()	cond_wait ()

Solaris平台支持资源的进程内互斥锁定以及条件变量同步。除了POSIX规范，Solaris操作系统还提供一种方便的读/写锁定机制，以控制对共享资源的访问。

但是，Solaris和POSIX线程API之间存在一定的区别。例如，下面的Solaris线程API就不受POSIX API的支持：

- thr_continue()
- thr_getconcurrency ()
- thr_main()
- thr_minstack()
- thr_setconcurrency ()
- thr_suspend()

同样，下面的POSIX线程API就不受等效Solaris线程API的支持：

- pthread_once()
- pthread_equal()
- pthread_cancel ()
- pthread_testcancel ()
- pthread_cleanup_push
- pthread_cleanup_pop ()
- pthread_setcanceltype ()
- pthread_setcancelstate ()

但是，上述两种API均可安全地用在Solaris操作系统上的相同程序中。

Linux不支持在不同进程之间共享的可共享同步对象（互斥量、信号量和条件变量）。因此，对具有另一个参数PTHREAD_PROCESS_SHARED的函数调用将无法工作：

- pthread_mutexattr_setpshared
- pthread_condattr_setpshared
- pthread_rwlockattr_setpshared

可兼容Solaris和POSIX API——信号量

对于信号量，Solaris操作系统和Linux均提供了POSIX接口。Solaris实现可与POSIX实现兼容。在Solaris端，下列两组函数中的任何一组均可使用。

编号	POSIX信号量函数	Solaris信号量函数
1	sem_destroy()	sema_destroy()
2	sem_init()	sema_init()
3	sem_post()	sema_post()
4	sem_trywait()	sema_trywait()
5	sem_wait()	sema_wait()

第4章

软件开发工具

编译器差异

一个有趣的地方是Solaris 10产品将GNU编译器和工具作为操作系统产品发布的一个部分提供。Sun Studio 11开发工具还随Solaris操作系统提供，现在还随Linux提供，并且还免费提供许可。因此，GNU和Sun Studio 11工具对使用两种平台的开发人员来说都是可用的。

不过，一般来说Solaris开发人员在Solaris环境中使用Sun Studio编译器，而大多数Linux程序员都使用GNU编译器。这在移植应用程序时就可能导致问题。这些编译器使用不同的命令行选项选择不同的模式。不存在标准格式的命令行选项，但这两种编译器显示的选项的格式和功能都是相同的。

命令行选项	描述
-c	指引编译器进行编译，但不进行链接
-o FILE	指定输出文件
-I DIR	添加包含搜索目录
-L DIR	添加库搜索目录
-lname	搜索和添加名为libname.a（或.so）的库
-Aname[(token)]	定义ISO C断言
-Dname[=val]	定义预处理器宏
-Uname	取消定义预处理器宏
-g	指导编译器发出调试信息
-E	执行预处理，并将结果写入到标准输出或使用-o指定的文件
- S	指示编译器编译而不汇编代码
-w	禁止打印警告

有关编译器命令行选项的差异的列表，请参见附录A中关于编译器选项的部分。有关Solaris链接器和GNU的链接器选项及不同版本的列表，请参见附录B中关于链接器选项的部分。除非特别说明，否则命令行选项对于所有受支持处理器（SPARC、x86和x64）都是有效的。

Sun Studio编译器选项和GNU编译器都是对C语言的扩展。Sun Studio编译器将#pragma用于扩展，而gcc使用后面带有两个左括号的<keyword>__<attribute>__序列在对象的定义或声明中添加信息。

GCC要在使用了-Wall选项时才忽略#pragmas选项；这样，它只在忽略了#pragmas时才发出警告。您要尽量避免使用这个语言扩展约定，因为它会严重影响代码可移植性。有关可兼容的#pragmas和<keyword>__<attribute>__序列的列表，请参见附录C中关于pragmas的部分。附录D则显示了Sun Studio软件中提供的行内程序集工具的各个版本。

源代码与版本控制

在涉及大型团队和大量代码的重要项目中，必须配置有合适的，而不仅是高产的软件配置管理（SCM）。它可帮助在实际的更改代码操作中引入一定的规范，使之具有程度合适的版本控制集成性、可跟踪性和可计量性。移植项目时，同样也需要在项目中使用对平台、语言和脚本透明的SCM工具。Subversion和Ant等新工具就是很好的选择。

但是，在C/C++应用程序中，开发人员仍然倾向于使用SCCS、RCS或CVS等传统UNIX工具。Sun Studio软件提供了类似的工具。我们对您可能会遇到的源代码控制问题进行了介绍。

源代码控制

Linux使用RCS（版本控制系统）进行源代码管理，而Solaris操作系统提供了SCCS（源代码控制系统）。这些程序包提供了兼容的功能，但使用不同的格式对代码更改和命令进行排序，以便对其进行操作。为了将源代码从一个系统移植到另一个系统，您应该从Linux系统中解析出最新的代码，然后以手动方式或使用自定义脚本转化到Solaris系统中。

CVS（并发版本系统）是一种占主导地位的开源、网络透明的版本控制系统，它既可安装在Solaris操作系统上，又可安装在Linux系统上。CVS既可用于单个开发人员，也可用于大型分布式开发团队。附录F提供了常用RCS命令及其对SCCS和CVS的映射的列表。

CVS与RCS

RCS的安装很简单，在小型项目中十分有用。它通过支持文件锁定功能来防止出现多个开发人员修改相同文件的情况。

CVS内置在RCS上部，比RCS具有更好的灵活性。CVS与RCS相比功能更强大且灵活更高，而且CVS能够支持使用脚本，以便提供所有级别的文件锁定功能。CVS的主要目的就在于提供一个允许将一组RCS文件视为单个对象的分组函数集。CVS提供了客户端/服务器支持，团队成员可从网络上的不同位置使用相同的CVS知识库。

CVS能提供其他功能（RCS不能直接支持的）还有：

- 非预留签出，使多个开发人员能够同时处理同一个文件
- 自动从其他源代码控制系统导入“供应商分支”，并合并更改
- 序列化提交

在Linux到Solaris的导入项目中，很可能会碰到要将知识库由RCS迁移到CVS的情况。这个任务相对简单，因为RCS文件是作为CVS知识库直接受到支持的。大多数迁移工作都包括知识库结构化，添加CVS需要的新文件（模块、日志信息，等等），以及将RCS文件物理迁移到新的CVS知识库。要从RCS转换到CVS，可遵循下面的步骤：

1.使用find命令找到所有RCS文件。

2.创建合适的CVS目录。

3.将每个RCS文件复制到CVS知识库中的相应目录。

有关CVS的使用和文档的详细信息，请访问http://www.nongnu.org/cvs/。

Forte™ Developer Code Management Software

这个代码管理系统不是Sun Studio 11软件的正式部分。但是，一般来说倾向于使用Sun提供的工具在Solaris操作系统上进行开发的开发人员都使用Forte Developer Code Management Software（以前名为TeamWare）。这是一个可由开发团队使用的多平台管理工具集，它支持并行开发。此外，它还同时支持个人开发环境以及全集群环境。有关该产品的部署和使用的更多信息，请参见Sun的文档SPARCworks/Team Ware & Pro Works/Team Ware Solutions Guide，地址是：

http://docs.sun.com/app/docs/doc/802-3637/6i7h52fqh。

Forte Developer Code Management Software是为合作开发环境提供的。文件和目录工具代码组织——文件放在特殊的工作区目录中，可同时访问并进行修改。

由于传统的Solaris开发人员熟悉Sun提供的工具，因此可在项目中使用现有的Forte Developer Code Management software，以便迁移到Solaris操作系统中。但是，要注意，这种代码管理软件的服务生命已经完结。

这种代码管理系统的一些必要细节如下所示。

配置管理

常规

- 支持以公共和私有工作区或视图作为配置
- 工作区作为标准UNIX目录进行管理，并在层次结构中进行安排，以使数据的组织操作和管理更简便
- 工作区层次结构可近似地映射到文件系统结构
- 可在任何时候使用工作区快照基线
- 可从工作区基线重新创建旧的文件版本
- 提供描述性工作区标签和历史文件浏览器

版本管理

- 同时管理多个版本
- 对基线以及补丁版本提供统一的工作区模型，以便为版本管理提供一种标准的方法

集成支持

- 自动跟踪和处理重新命名的文件
- 支持管理和解决工作区配置冲突
- 可配置菜单允许改变视图
- 提供了SCCS标记更改功能

版本控制

- 支持ASCII和二进制文件
- 使用图形界面查看颁布历史和执行文件版本比较
- 与SCCS开发实用工具兼容
- 提供了签出、编辑和签入范例

更改控制

- 分布式知识库模型启用了序列化和并行更改
- 分支允许并行开发
- 针对常用基线的两文件版本的三路合并是以图形化方式执行的
- 非覆盖式更改的自动合并
- 通过回放验证检查为工作区所有人启用了自定义更改控制策略强制执行

- 撤销事务命令以使工作区还原为先前状态

生成管理

- · 与Solaris `make`和Forte Developer Code Management Software生成实用工具兼容
- · 在网络上的多个计算机上并行执行生成任务，并显示执行和完成状态

系统管理

安全

- 基于用户或网络组的所有工作区操作同时进行访问控制
- 以工作区为单位指定访问控制

备份/恢复

- 可使用标准Solaris软件管理工具备份或恢复工作区

启动和维护

- 不是系统管理员也可启动和维护
- 可很容易地合并UNIX和SCCS文件
- RCS到SCCS的转换

接口

- 命令行接口
- 直观的Motif GUI，支持拖放功能
- 可容易地从GUI注册和启动其他工具
- 用户指定的文本编辑器

应用程序构建

Make文件

Solaris操作系统和Linux都提供了make实用工具，这种工具可用于从源文件构建应用程序。make实用工具可在描述目标文件及其依赖项之间的关系的make文件上递归工作。

在Linux上，GNU make按如下顺序搜索make文件：

1../GNUmake file

2.. /makefile

3.. /Makefile

此外，GNU make可在必要时从RCS/SCCS目录隐式解析文件。

Solaris make实用工具按如下顺序搜索make文件：

1.非POSIX模式

- ./makefile
- 如果存在./SCCS/s.makefile，它会尝试检索make文件的最新版本。

- ./Makefile
- 如果存在./SCCS/s.Makefile，它会尝试检索make文件的最新版本。

2.POSIX模式

- ./makefile, ./Makefile
- s.makefile, SCCS/s.makefile
- s.Makefile, SCCS/s.Makefile

两种版本的make都具有兼容的命令行选项。有关详细信息，请参见附录D中关于make选项的部分。
或者，您也可使用您的目标平台的GNU make版本3.80。
您可从Sun的免费软件网站http://www. sunfreeware .com获取该软件。

imake实用工具

imake处理包含所有make文件共有的定义的模板文件（默认文件为Imake.tmpl）。此模板包含下面的文件。

文件名	描述
架构定义文件	包含一系列条件块，以标识平台计算机架构。
平台特定的定义文件	在条件块中设置平台特定的生成参数。
本地站点特定的定义文件	跨所有平台设置应用程序特定的生成参数。（本地文件中的定义应放在功能块中，以使得平台特定的文件能够覆盖这些定义）。
项目配置文件	包含项目特定的变量定义。
规则定义文件	包含宏形式的规则定义。
Imake文件（来自当前目录）	包含make变量和宏调用的定义。它使用规则文件定义的规则构建make文件的必要部分。
模板文件	在可用于构建应用程序的make文件底部附加一组通用规则。

有关imake的详细信息，请参见文档《Software Portability with imake（使用imake迁移软件）》（第二版）：http://www.kitebird.com/imake-book/。

应用程序打包

软件打包是一个重要的任务。通过使用合适的打包机制，人们可以很容易地完成软件安装、删除和验证的工作。这些活动中的每一项都需要用户在参与过程中精确地完成，因此应使这些活动的自动化程度达到最大。

Linux一般使用一个名为RPM（Red Hat Package Manager）的工具进行软件的安装、卸载和验证。RPM可能不是最复杂的打包管理程序，在Linux环境中还存在许多更好的选择。但是，基于RPM的打包处理经受了使用测试的考验，并且相当流行。

Solaris软件使用了一种非常复杂的打包格式称为pkgadd格式。pkgadd工具集是UNIX SVR4和Solaris操作系统上默认的程序包安装、卸载和验证工具。这种工具也经受住了多年来使用测试的考验。不幸的是，使用这种打包方法所需完成的脚本编写工作比较复杂。在任何情况下，使用pkgadd与基于RPM的打包脚本相比都更为复杂。

因为打包脚本通常与生成应用程序二进制文件的make脚本放在一起，因此更谨慎的做法是尽量少更改脚本，并选择基于RPM的打包方式，即使是在Solaris 10平台上。

开发工具

Linux工具

Linux许多种开发工具，有些列在下面的表中。有关Linux的可用开发工具的列表，请参见Linux.org网站：http://www.linux.org/apps。

Linux工具	**描述**
Sun™ Studio Compiler Collection	支持多种编程语言（如C、C++、Java和Fortran）的集成开发环境（IDE）。就本文而言，Linux上只有一个早期访问版本可用。
Builder Xcessory PRO	Motif的重要的用户界面生成器
CENV	C和C++的小型开发环境
Portable.Net	工具的可移植套件（包括C#编译器、汇编程序以及运行时引擎）
Mork	Java技术的编译器工具
GTK Object Builder（GOB）	可容易地创建GTK对象的简单预处理器

Sun Studio软件

Sun提供的Sun Studio工具是一种重要的开发环境（参见http://developers.sun.com/sunstudio）。

Sun Studio产品提供了一个高度集成的环境，用于加强Java、C、C++和Fortran应用程序的软件开发速度。这个集成编程环境包含一组用于创建、维护、编译、生成、浏览、编辑、调试和调整应用程序的可视工具。

通过对并行功能（如OpenMP）的支持，这些工具可增加生产率，并帮助生成可靠、高性能和可伸缩的应用程序，包括多线程应用程序。C和C++编译器还能提供通用语言扩展。

当前的Sun Studio 11版本是一个功能强大的IDE，它用于使用C、C++、Fortran和Java编程语言进行开发。此软件可帮助您创建可靠、可伸缩和高性能的32位和64位应用程序，这些应用程序通过调整可在SPARC或x86/x64处理器上运行。Sun Studio工具旨在帮助您容易地开发和调整多线程和多处理应用程序。

图形应用程序

GNOME和KDE是Linux环境中最常用的两种窗口管理器。Linux上大多数基于X11的图形应用程序都已经使用可用的开发框架内置在其中。

在Solaris 10版本中，Sun提供了一个基于GNOME的增强的桌面，称为Sun Java桌面系统（JDS），这是目前最受欢迎的桌面。在早期版本中，Solaris OS支持CDE（公共桌面环境）作为其默认窗口管理器。当然，CDE也是与Solaris 10软件捆绑在一起的。在Solaris 10平台上，用户可选择启用JDS、CDE或KDE等常用桌面管理器中的一个。

可使用Sun Studio工具开发图形应用程序。GNOME的完整的开发工具套件可从`http://www.gnome.org`获取。GTK+库的运行时已经与Solaris 10操作系统捆绑在一起。同样，也可以从的开发和工具帮助。

硬件中立的GNOME或KDE应用程序的移植相对容易。

调试与代码优化

在任何涉及大量代码的软件项目中，开发人员都需要使用合适的开发工具调试代码。在Linux端，GDB的GNU调试器或基于它的变体正是最为流行的。GDB的派生和扩展有DDD（数据显示调试器）和XDB。Solaris 10操作系统为开发人员提供了所有这些工具。

但是，在需要涉及Solaris平台的更深层次的分析的情况下，谨慎的做法是考虑使用由Solaris技术创建的专业工具。Solaris 10操作系统提供了系统调用跟踪工具`truss`、动态跟踪（DTrace），以及许多专业化工具作为优秀的替代工具。

Sun Studio dbx通过有效调试多次线程应用程序极大地提高编程效率，而Sun Studio Performance Analyzer提供了一个高级图形工具来测量应用程序性能和发现瓶颈与优化问题。

使用LAMP移植应用程序

Internet的爆炸式增长导致对Web应用程序需求的增长。有很大一部分的应用程序开发人员已经使用技术组合LAMP来开发Web应用程序。LAMP表示Linux、Apache、MySQL和Perl/PHP。Linux提供了低成本的承载平台。开源Apache Web服务器以及许多有用的扩展提供了启用Web的技术。MySQL这种免费的RDBMS为在很大程度上依赖于查询的数据库提供了一种非常有效的数据存储。它们之间的Perl和PHP为Web页面、动态内容和业务逻辑等提供了脚本编辑选项。

为了获得真正可伸缩的解决方案，这些应用程序可能需要一个可伸缩的和安全的平台，如Solaris 10操作系统。将此类型的应用程序迁移到Solaris 10平台相当简单，因为操作系统在应用程序堆栈中提供了所有的组件。Apache、MySQL、Perl和PHP的预编译二进制文件在Solaris发布本身中就是可用的。所有这些组件的性能、安全和调整准则也在包括Sun在内的各种资源中可用。为了为RDBMS提供更多选择，Sun已宣布要将PostgreSQL捆绑在Solaris 10发布中。

除了使用这些技术支持新应用程序开发以及Web服务外，Sun Studio还提供了一组非常有效率的工具，以便将这些应用程序移植到Solaris 10操作系统。

内存管理

在需要获得可能的最佳性能的产品和应用程序中，您可能需要考虑应该让操作系统如何管理相对稀缺的资源，例如内存。这就使得您需要深刻了解承载操作系统上的一些问题，例如内存管理。许多操作系统问题与将大量数据和文件计划、锁定和映射为内存映射的文件相关；这些问题可能会对性能产生相当大的影响。
因为每个操作系统版本都在实现中提供了许多详细信息，所以最好以应用程序为单位考虑对应用程序进行内存管理相关的更改，只有这些情况下才对性能敏感。由于没有确定这些策略是否可用的通用方法，所以在大多数情况下最好在本机可用的内存管理系统中运行应用程序。

第5章

API差异

系统调用与库调用

Linux和Solaris OS都是遵循POSIX标准的UNIX系统，并提供定义良好的系统调用接口访问内核工具。尽管内部实现不同，两个系统提供的系统调用接口却非常相似。根据标准UNIX约定，大多数接口函数都返回-1表示出现了错误，设置全局变量*errno*提供完整的错误描述。

除了系统调用，两个系统上都提供了函数调用分类。当在两个平台上都可用时，虽然它们的大多数行为都非常相似，但是它们在参数类型、返回类型和异常处理方面则经常有差别。

对于仅在一个平台上可用的调用，开发人员需要花费更多的精力；移植时这些调用会导致更多的问题。从Linux移植到Solaris 10操作系统时，我们显然应该更加关注那些仅在Linux中可用，并且Solaris 10操作系统上没有直接等效项的函数调用。

虽然有许多文章列出了Linux和Solaris操作系统在API集方面的差异，但是使用手工方式跟踪这些差异并不是最有效的解决方案。这种方法也很容易出错。为了有效地找出由于API不兼容或丢失而导致的问题，Sun强烈推荐使用一种基于工具的方法。Sun的LinCAT工具就是一种有用的工具。

LinCAT

LinCAT（Linux Compatibility Assurance Toolkit）是一种代码扫描工具，用于分析C/C++源文件以及标识可能与代码和函数兼容性问题有关的问题。

此工具基于Java技术，可用于标识C/C++源代码文件中的移植问题。此实用工具可分析在将源代码文件由Linux移植到Solaris操作系统时可能带来的大多数问题。它还可帮助估计进行移植所需要完成的总工作量。LinCAT提供关于以下内容的信息：

- 函数调用的名称、位置以及使用频率
- 具有潜在移植问题的源代码

- 解决所选择的移植问题的解决方案
- 不兼容调用——该工具通过查找函数调用数据库提供有关解决方案的建议

LinCAT在内部使用从名为JScore的比较引擎获得的结果。这个引擎使用一个参考数据库，并指向与兼容性有关的问题或者Linux代码中的函数中的问题。
因此，在大型项目中，使用LinCAT可为开发人员节省大量的时间和精力。

LinCAT生成的报告还可通过提供有关需要更改的信息评估所需完成的工作量。一旦标识了所需的更改，实际更改代码、编译代码以及生成应用程序二进制文件的工作就相对简单了。遵循软件开发的常规步骤（如生成、测试和发布），然后将应用程序用于Solaris平台时其功能不会受到任何影响。Sun为找出性能问题（如果存在）提供了动态跟踪（DTrace）等高质量工具。

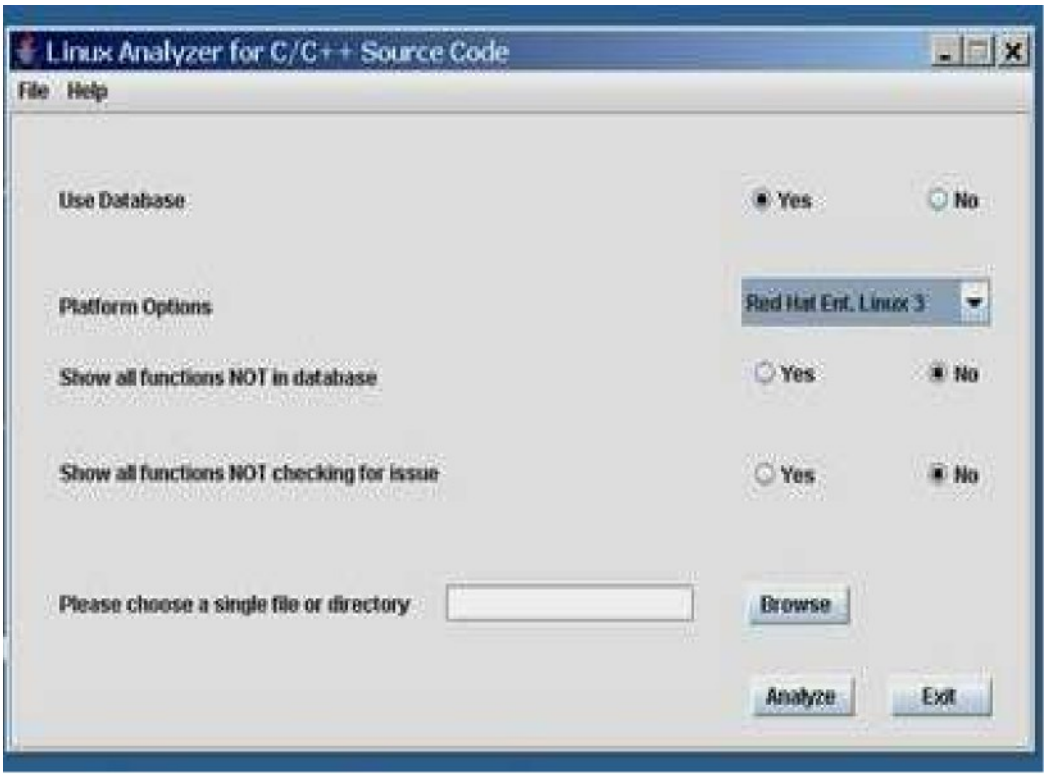
LinCAT的最新版本可从

http://developers.sun.com/prodtech/solaris/downloads/lincat/index.html下载。

LinCAT试用版

LinCAT工具可运行在任何支持Java运行时环境的平台上；它可运行在Solaris、Linux和Windows操作系统上。如下所示的开放屏幕允许开发人员选择要扫描的源文件。

此屏幕还为支持的原始源文件提供了一些选择。就目前而言，它支持来自Red Hat Enterprise Linux 3和4的源文件。分析的结果以HTML文档的格式提供，其中包括将代码移植到Solaris 10操作系统的详细信息。仔细分析此报告后，开发人员就可以评估此项工作的性质以及需要完成的工作量。



结束语

此指南的目标在于帮助开发人员将在Linux上运行的应用程序成功地迁移到Solaris 10操作系统。我们就两个平台的操作系统功能、架构差异、API和开发环境进行了比较。此信息能够帮助开发人员计划和执行从Linux到Solaris操作系统的平滑迁移。

参考资料

下面的Internet资源和学习资料可能能够提供帮助。

- *In a Class By Itself -- The Solaris 10 Operating System*
- 这本白皮书信息量大，内容具体详细，其中包括Solaris 10操作系统的如下功能：
  http://www.sun.com/software/whitepapers/solaris10/classbyitself.pdf
- Solaris免费软件：Solaris平台的所有版本的最新更新均可从下列站点下载：
  http://www.sun.com/freeware
- Solaris 64-bit Developer's Guide，来自Sun提供的用于在Solaris操作

系统上开发64位应用程序的准则Solaris 10 Software Developer Collection。http://docs.sun.com/app/docs/doc/816-5138

- Jim Mauro和Richard McDougall的Solaris Internals（Solaris内核结构），由Sun Microsystems Press出版。关于Solaris线程和Solaris内核的讨论；是关于Solaris操作系统的一个不错的参考资料，但此版本稍嫌过时。
- Steve Kleiman、Devang Shah和Bart Smaalders的Programming With Threads，由SunSoft Press出版。这是一本关于多线程编程和应用程序开发（尤其是在Solaris操作系统上）的好书。
- LinCAT工具：http://developers.sun.com/prodtech/solaris/downloads/lincat/index.html
- Sun Developer Network站点上的Sun Studio hub。提供有关Sun Studio编译器和工具的完整信息，包括完整的文档、技术资料和用户论坛：http://developers.sun.com/sunstudio/
- Andrew Koenig的C Traps and Pit Falls，由Addison-Wesley出版。在编写可移植程序时有用。
- 有关Red Hat Linux的信息：http://www.redhat.com

附录A

编译器选项

有关Sun Studio 11 C编译器选项的详细信息，请参见http://docs.sun.com/doc/819-3688中的用户指南。

有关完整的Sun Studio 11文档，请访问：http://developers.sun.com/prodtech/cc/documentation/ss11/index.html。

Linux GCC选项	Sun Studio C编译器选项	描述
-v	-#, -V	在gcc中，-v显示编译器版本信息。 在Sun Studio C编译器中，-#打开verbose模式，该模式会显示命令选项的扩展情况以及调用的每个组件。-V选项提供编译器的版本信息。
--help -v	-flags, -xhelp	显示所有可用选项的总体介绍。使用gcc时，使用--help -v选项获取所有可用编译器（包括C++编译器、基于Java技术的编译器、Fortran编译器以及其他可用编译器）的总体介绍。
	-# # #	显示可能调用的每个组件，但不实际执行。同时还显示命令选项的扩展情况。
-p/-profile	-P (Capital P) -p (Small p)	-P（大写P）准备收集配置数据的目标代码。 -p（小写P）通过预处理器运行源文件。
-mcpu=i 386 / -march=i386	-x3 86	为80386处理器进行优化。现已被弃用。目前改用-xchip=generic。
-mcpu=i 486 / -march=i4 86	-x 486	为80486处理器进行优化。现已被弃用。目前改用-xchip=generic。
-fP IC - fpic	-KP IC -Kpi c	生成与位置无关的代码。-Kpic和-KPIC在x86/x64下相同。-KPIC和- kpic在SPARC平台上不受支持。现在应该使用-xcode = pic32代替-KPIC，使用-xcode = pic13代替-Kpic。

Linux GCC选项	Sun Studio C编译器选项	描述
-O	- fast	选择最优的编译选项组合以便提高可执行代码的速度。

	-xalias_level	使编译器能够执行基于类型的别名分析和优化。
	-xbui ltin	改进调用标准库函数的代码的优化。
	-xcros s file	启用跨源文件优化和内嵌。gcc中不存在等效选项。
-ffunction-sections, -fdata-sections	-xF[=[no%] func, [no%] gbldata, % all, %none]	进行函数和变量的优化排序，方法是将函数和/或数据放入分开的区段中，以便由链接器进行进一步的重新排序。
-fno-inline functions	-f inline-controlled -xinline= { list. . . }	尝试内嵌指定的函数。gcc中不存在直接等效的选项。实际上，gcc中的内嵌使用的是-finline-functions 和-finline-limit=N。
	-xipo	通过调用程序内分析组件对整个程序进行优化。
- f fast-math	-xl ibmi l	内嵌一些库，以便更快速地执行。
	-xlic_lib=sunpe rf	Sun提供的性能库的链接。
	-xmaxopt=off, 1, 2, 3, 4, 5	此代码将#pragma opt限制为指定级别。默认值为-xmaxopt=off，该值会导致忽略pragmaopt。
Default option	-xnolibmil	不内嵌数学库例程。这是gcc的默认设置。
-O1/-O2 / -O3	-x01 to -x05	优化的上升级别
	-xpre fetch	启用prefetch指令。依赖项。对于-xprefetch=yes，改用-xprefetch=auto,explicit。对于-xprefetch=no，改用-xprefetch=no%auto, no%explicit。
	- xp re fe t ch_l eve l	控制-xprefetch=auto设置的pre fetch指令的自动断言的侵略性。
- fpro file -generate, -fprofile-use	-xpro fi le= [collect |use]	生成代码以便收集配置信息。这可在以后由配置工具使用。
	-xsafe	允许编译器假定不存在基于内存的陷阱。gcc中不存在等效选项。
-Os	- x space	不优化或并行化可增加代码大小的循环。

Linux GCC选项	Sun Studio C编译器选项	描述
-funroll-loops/-funroll-all-loops, --param max-unroll-times=N	-xunroll=n	建议优化器解环*n*次。
	-xchar_byte_order r	通过在以指定字节顺序放置多字节字符常数生成整数常数。
implicit	-xmemal ign	指定最大的假定内存对齐以及对齐错误的访问的行为。
	-fnon s td	导致浮点算法硬件的非标准初始化。gcc中不存在等效选项。
	-fn s	为基于SPARC的处理器打开非标准浮点模式。
	-fpre ci s ion	初始化浮点控制字中四舍五入模式的位。
	-f round	在程序初始化期间设置运行时建立的IEEE 754四舍五入模式。这是gcc中的默认值，不能设置为其他值。
-ffast-math	-fsimple	允许优化器对浮点算法作出简化假设。
	-f single	导致编译器将浮点表达式作为单精度数而不是双精度数评估。gcc中不存在等效选项。
-f float-store	- fs tore	导致编译器将浮点表达式或函数的值转化为赋值表达式的左边的类型。在x86上也可用。
	-ftrap	在启动时有效设置IEEE 754陷阱模式。gcc中不存在等效选项。
	-no f store	不将浮点表达式或函数的值转化为赋值表达式的左边的类型。gcc中不存在等效选项。x86上也可用。
Default option	-xlibmieee	强制IEEE 754样式在出现异常的情况下为数学例程返回值。
-f single -precision-constant	- x s fp cons t	以单精度数表示无后缀浮点常数。
	-xvector	自动生成对矢量库函数的调用。

Linux GCC选项	Sun Studio C编译器选项	描述
-threads 使用Solaris线程的多线程处理 -pthreads 使用Solaris线程的多线程处理	-mt	展开到-D_REENTRANT -lthread的宏选项。
- ftl s -model=MODEL	-xthreadvar [=dynamic |no%dynamic]	指定此选项控制本地线程变量的实现。
	-xautopar	为多个处理器打开自动并行化处理。gcc中不存在等效选项。
-fstack-check	-xcheck	为堆栈过流添加运行时检查。
	-xdepend[=ye s | no]	分析内部迭代数据依赖项的循环，并不重新构建循环。gcc中不存在等效选项。
	-xexplicitpar	（仅适用于SPARC。）根据#pragma MP指令规范生成并行化代码。gcc中不存在等效选项。
	-xloopinfo	显示那些循环是并行化循环，哪些不是。gcc中不存在等效选项。
	-xopenmp	为显式并行化支持OpenMP接口，包括一系列源代码指令、运行时库例程和环境变量。
	-xparallel	对于x86/x64，自动并行化循环。对于SPARC，既由编译器自动并行化，也由程序员显式指定循环并行化。
	-xreduction	在自动并行化期间打开降低识别功能。gcc中不存在等效选项。
	- x restrict	将指针函数参数视为受限指针。Gcc通过-fargument-noalias和argument-noalias -global支持此选项。 此外，受限指针通常在代码中有__restrict__指示器时用于gcc。
	-xvpara	关于指定了#pragma MP指令的循环的警告，但可能无法在并行化时正常运行。gcc不支持。
	-Zll	为lock_lint工具创建程序数据库，但不生成可执行代码。gcc不支持。
-A name=token s	-Aname [(tokens)]	将name作为谓词与指定*tokens*关联，就像#assert预处理指令所做的一样。

Linux GCC选项	Sun Studio C编译器选项	描述
-C	-C	防止与处理器删除注释，但预处理指令行上的注释除外。
-D name[=tokens]	-Dname[=tokens]	将name与指定*tokens*关联，就像#define预处理指令所做的一样。
-E	-E	仅通过预处理器运行源文件，并将输出发送到stdout。
-Ws tr i ct-prototypes	- fd	报告K&R样式的函数定义和声明。
-M/-MM	-H	在当前编译期间显示标准错误，每行显示一个错误，并包含每个文件的路径名称。
-I	-I	对具有相对文件名的#include文件，将目录添加到搜索的列表中。
-EP	-P	仅通过C预处理器运行源文件。然后将输出放在具有.i后缀的文件中。与-E不同，此选项中不提供行号信息。在gcc中，-EP禁止行号生成，但到stdout的输出中可以生成。
-U name	-Uname	删除预处理信号***name***的所有初始定义。
	-xCC	接受C++样式的注释。Gcc在默认情况下接受C++样式的注释，但这可使用-std禁用。
- s td= c 99	-xc 99	控制受支持的C99功能的编译器识别。
-funsigned-char, -fsigned- char	-xchar[=signed | unsigned]	有助于从char定义为无符号字符的系统迁移出去。
- f input - chars et=CHARSET	- xc si	允许C编译器接受本地编写的不符合ISO C源字符代码要求的源代码。
- fexe c-charset=UTF-16	-xu s tr= { ascii_utf16_short |no}	此选项允许编译器识别国际化应用程序中的ISO10646 UTF-16字符串。
-M	-xM	要仅对命名的C程序上运行预处理器，则需要生成make文件依赖项，并将结果发送到标准输出。
-MM	-xM1	收集-xM等依赖项，但/usr/include文件除外。
-aux-info FILENAME (not exactly equivalent)	-xP	显示此模块中定义的所有K&R C函数的原型。 gcc选项-aux-info严格意义上说并不是等效项，但它确实可以输出翻译单元中所有函数的原型化声明。

Linux GCC选项	Sun Studio C编译器选项	描述
-pg	-xpg	准备目标代码，以便为gprof(1)配置收集数据。
	-xsb	为源代码浏览器生成额外的符号表信息。gcc中不存在等效选项。
	-x sb fast	为源浏览器创建数据库。gcc中不存在等效选项。
-trig raph s	- x t r i graphs [=yes |no]	确定对三字符序列的识别。 在 gcc 中， -trigraphs 是 -xtrigraphs=yes 的等效项。
-c	-c	让编译器禁止与ld(1)的链接，并为每个源文件生成一个.o文件。
-o	-o	为输出文件命名。
-S	- S	使编译器生成汇编源文件，但不汇编程序。
	-W	将参数传递到C编译系统组件。两种编译器不支持相同的组件。
-pedantic	-X	-X选项为ISO C标准指定可变的编译程度。在gcc中，-pedantic选项与此选项最匹配。
TMPDIR Environment variable	-xtemp=dir, TMPDIR Environment Variable	将cc使用的临时文件的目录更改为dir。对于gcc，设置环境变量TMPDIR。
-Q	-xtime	报告每个编译组件使用的时间和资源。
-B	-Y	为C编译系统组件的位置指定新目录。
Combination of -nostdinc and -idirafter DIR	-YA	更改默认目录以便搜索组件。
	-YI	更改默认目录以便搜索include文件。
	-YP	更改默认目录以便查找库文件。
	-YS	更改默认目录以便启动目标文件。
	-errfmt	为错误消息添加error:字符串作为前缀，以便更容易地区分错误消息和警告消息。
-w	-w, -e rro f f	禁止编译器警告消息。
	-errshort	在编译器发现类型不匹配的错误时，控制编译器产生的错误消息的详细程度。

Linux GCC选项	Sun Studio C编译器选项	描述
	-errtags	显示每条警告消息的消息标记。gcc中不存在等效选项。
-Werrors	-errwarn	如果发出了警告消息，cc就会在错误状态下退出。
-Wall	-v	指导编译器执行更严格的予以检查，并启用其他类似lint的检查。
-fsyntax-only	-xe	仅针对源文件执行语法和语义检查，但不产生任何目标代码或可执行代码。
	-x transition	在K&R C和Sun ISO C存在差异时发出警告。Gcc不具有等效选项，但它会对违反ISO C规则的情况发出警告。
-g	-g	为调试器产生附加的符号表信息。
-WI, - s	- s	从输出目标文件中删除所有符号调试信息。
	-xs	允许在不具有目标文件（.o）的情况下进行调试。
-static	-B	指定用于链接的库德绑定是静态地还是动态的。默认情况下是动态的。
implicit	-d	在链接编辑器中指定动态或静态链接。
-shared	-G	将该选项传递到链接编辑器以产生共享目标而不是i动态链接的可执行文件。
-WI, name	-h name	为共享动态文件指定一个名称，从而获得不同版本的库。
	-i	将选项传递到链接器以忽略所有LD_LIBRARY_PATH设置。
-L	-L	在列表中添加链接器搜索库的目录。
-l	-l	与目标库*libname.so*或lib*name*.a链接。
	-mc	从目标文件的comment区域删除重复的字符串。
	-mr	从.comment区域删除所有字符串。也可以在目标文件的该区域插入一个***string***。gcc中不存在等效选项。
	-Q	向输出文件发出或不发出标识信息。
-Wl, - rpath	-R	将用于指定库搜索目录的一个逗号分隔的目录列表传递到运行时链接器。
	-xMerge	将数据段合并到文本段中。
	-xcode	指定代码地址空间。

Linux GCC选项	Sun Studio C编译器选项	描述
默认选项	-xstrconst	在文本段而不是默认数据段的只读数据区域中插入字符串。默认情况下这是在gcc中进行的。 在gcc中，可通过使用-fwritable-strings获取字符传递可编写副本。
	- xildoff	关闭附加链接器并强制使用ld。gcc中不存在等效选项。
	-xildon	打开附加链接器并在附加模式中强制使用ild。gcc中不存在等效选项。
	-xnativeconnect	在目标文件和后面的共享库中提供接口信息，以便从Java代码调用共享库。
-gstabs, -gdwarf-2	-xdebugformat=[stabs |dwarf]	设置调试信息格式。
-nostdlib	-xnolib	默认情况下不链接任何库。
-march	-xarch	指定指令集架构。
	-xcache	定义缓存集以供优化器使用。gcc中不存在等效选项。
	-xcg	指定-xarch、-xchip和-xcache的值。依赖项。改用-O以便利用-xarch、-xchip和-xcache的默认值。
SPARC: -mcpu=NAME x86: -march=NAME	-xchip	指定目标处理器以供优化器使用。
-ffixed<reg> / -fcall-used-<reg> / -fcall-saved-<reg>	-xregs	指定生成的代码的注册器的用法。在gcc中，显示的三个选项之一可用于指定注册器。
-b	-xtarget	指定指令集和优化的目标系统。已弃用。对于-xtarget=386 和 xtarget=486 ，改用-xtarget=generic。
	-xpch={ auto | auto first. . }	此编译器选项可激活预编译的头功能。编译器自动维护预编译的头文件。 gcc还支持预编译的头，但它需要对头进行显式编译，就像编译C程序文件一样。预编译文件的自动生成功能不可用。

¤

Linux GCC选项	Sun Studio C编译器选项	描述
平台特定的多媒体及流指令用法。 Intel 和 AMD 处理器家族：-mmx, -msse, -msse2, -msse3, -m3 dnow SPARC 处理器家族：-mvis	平台特定的多媒体及流指令用法。 Intel 和 AMD 处理器家族：-xarch=sse, -xarch=sse2 SPARC 处理器家族：-xvis	这些选项启用了ISA特定的特殊优化多媒体和流式指令的用法。
-save-temps	-keeptmp	临时文件未删除。
-malign-double	-dalign	使用双字加载和存储汇编指令。这个SPARC技术特定的选项是专为SPARC指令集设计的。已弃用。 改用-xmemalign=8s。
-mcmodel	-xmodel=<model>	用于只在-xarch=amd64 d64a之下的x64。支持基于大小的不同数据访问模式。<model>可以是中型、小型或内核。
-fomit-frame-pointer	-xregs=frameptr	允许将框指针注册器（x86上的%ebp，x64上的%rbp）用作未分配调用对象的保存的注册器。
<none>	-xthreadvar	控制线程局部变量的实现。

¤

附录B

链接器选项

gld选项	Solaris链接器选项	描述
-static	-a	仅用于静态模式，生成一个可执行目标文件；对于未定义的引用给出错误。这是静态模式的默认行为。
	-b	仅用于动态模式，不对引用共享对象中符号的再分配做专门处理。如果没有-b选项，链接编辑器为引用共享对象中定义的函数创建了专门的与位置无关的再分配。
	-B direct	通过记录各个符号引用和提供定义的依赖性之间的关系，建立直接绑定信息。
	-B dynamic | static	管理库包含的选项。如果指定了-B static选项，将不会接受共享对象，除非遇到-B dynamic。
	-B eliminate	从符号表中去除不是为任何版本定义指定的符号。在GNU链接器中没有等价选项。
	-B group	作为一个组来建立一个共享对象及其依赖关系。在GNU链接器中没有等价选项。
	-B local	致使不为一个版本定义指定的全局符号减少到本地。这可利用GNU链接器用版本定义文件中匹配的统配符来实现。
	-B reduce	在生成可再分配对象时根据版本规范降低符号信息。在GNU链接器中没有等价选项。
-symbolic	-B symbolic	仅用于动态模式。构建一个共享对象时，该选项在该对象内将对全局符号的引用绑定到它们的定义上。
-c	-c name	为运行时使用记录配置文件名。

gld选项	Solaris链接器选项	描述
	-d y | n	指定了动态或静态链接：y表示“是”，n表示“否”。
	-D *token* [,token]	根据令牌环的规定打印调试信息。在GNU链接器中没有等价选项。
-f name	- f name	仅用于共享对象，规定了共享对象的符号表用作由名称指定的共享对象符号表上的辅助过滤器。
-soname *name* -h name	-h *name*	仅用于动态模式，在构建共享对象时，该选项在对象的动态节中记下了name。在GNU链接器中必须使用-s oname选项。
-shared	-G	生成共享对象。
	-i	忽略LD_LIBRARY_PATH环境变量。在GNU链接器中没有等价选项。
--dynamic-linker *name*	-I name	在构建可执行文件时，使用name作为要往程序头中写入的解释器的路径名。在GNU链接器中没有等价选项。
-L path	-L path	将路径添加到库搜索目录。
(-M)	-m	在标准输出上生成输入或输出节的内存映射或列表，以及任何非致命的多重定义符号。
--version-script map *file*	-M map *file*	将mapfile作为指令的文本文件读到ld中。
	-N *string*	该选项在正在构建的对象的.dynamic节中添加一个DT_NEEDED项。GNU链接器中没有等价选项。
	-p	标识一个用于在运行时审计对象的审计库。GNU链接器中没有等价选项。
	-r	组合可再分配对象文件，以生成一个可再分配对象文件。GNU链接器中没有等价选项。
-rpath -R *path*	-R *path*	为运行时链接器指定了以冒号分隔的搜索目录列表。GNU链接器接受-rpath和-R path具有同样的含义。
-S / - s	- s	从输出文件中剥离出符号信息。任何调试信息，也就是.Debug、.line和.stab节，以及它们的相关再分配项都会被删除。
-t	- t	关闭关于具有不同大小的多重定义符号的警告。GUN链接器中的-t意味着在链接器处理文件名时打印它们。

gld选项	Solaris链接器选项	描述
-WI, -whole-archive/ --no-whole-archive	-z allextract/-z defaultextract /-z weakextract	改变从档案中的提取标准。
	-z abs e xe c	规定了对外部绝对符号的引用应该立即解析，而非留待运行时解析。GNU链接器中没有等价选项。
	-z comb re lo c	组合了多个再分配节。GNU链接器中没有等价选项。
	-z endfiltee	标记一个filtee，这样在被过滤器处理时，它会终止过滤器进行的进一步filtee搜索。GNU链接器中没有等价选项。
	-z ignore | record	忽略或记录没有被作为链接-编辑的一部分而引用的动态依赖性。GNU链接器中没有等价选项。
	-z la z yload/-z no lazyload	分别启用或禁用动态依赖性的惰性加载。GNU链接器中没有等价选项。
	-z mulde f s	支持多个符号定义。GNU链接器中没有等价选项。
	-z node f s	支持未定义符号。这是构建共享对象时的默认选择。GNU链接器中没有等价选项。
	-z redloc s ym	去除除来自SHT_SYMTAB符号表的SECT符号之外的所有本地符号。GNU链接器中没有等价选项。

附录C

兼容的#pragmas

注意：所有属性在gcc中的声明如以下格式：

```
type _attribute_ ((<attribute-name>))
```

attribute-name可以是下面其中一个属性。

变量属性

aligned (alignmode)

规定了变量的最小对齐字节数。例如：

```
int x _attribute_ ((aligned (16))) = 0;
```

在Sun Studio编译器中，这可通过下面语句实现：

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

packed

规定了一个结构将具有最小的可能的对齐。可使用aligned属性指定一个更大的值。在Sun Studio编译器中，这可通过下面语句实现：

```
#pragma pack(n)
```

section

```
section ("section-name")
```

规定了变量存在于名为section – name的特定节中，而不是像data和bss这样的默认节中。Sun Studio编译器中没有等价的声明。

unused

这意味着告诉gcc变量可能未被使用。gcc不会产生警告。Sun Studio编译器中没有等价的声明。

函数属性

noinline and always-inline

控制函数的内联。no inline防止函数被看作内联函数，always -inline内联被声明为内联函数的函数，即使没有指定优化。

在Sun Studio编译器中，您可以利用下面语句将函数声明为内联和非内联：

```
#pragma [no_] inline (funcname[ , funcname])
```

Sun Studio编译器仅支持全局内联控制。

constructor

致使gcc在调用main()之前调用函数。Sun Studio编译器中的等价声明为：

```
#pragma init (f1[ , f2.. .,fn])
```

destructor

使gcc在main()函数之后调用函数。这被期望是void类型的，并且不接受参数。Sun Studio编译器中的等价声明为：

```
#pragma fini (f1[ , f2.. .,fn])
```

pure

该选项在gcc中的用法为：

```
int memcmp (const void *, const void *, size_t ) _attribute_ ((pure));
```

这告诉gcc，这个函数如果比程序所称的那样少调用几次会很安全。依赖于易变内存或其它系统资源的函数不能被声明为pure。

Sun Studio编译器中没有等价的声明。但是，可利用下面语句来部分实现：

```
#pragma no_side_effect (funcname);
```

这声明了该函数没有任何副作用。

extern int old_extname (int) _asm_ ("new_extname");

这允许gcc利用new_extname替换old_extname的每个外部定义。结果，链接器在链接时只看到new_extname。在Sun Studio编译器中，这可通过下面语句实现：

```
#pragma redefine_extname old_extname new_extname
```

#ident string

将string放到可执行文件的.Comment节中。在Sun Studio编译器中，这可通过下面语句实现：

```
#pragma ident string
```

weak

使声明被定义为弱符号而不是全局符号。在Sun Studio编译器中，这可通过下面语句实现：

```
#pragma weak name
```

有用的Sun Studio编译器选项（gcc上不可用）

#pragma opt level (funcname [, funcname])

这个值规定了funcname子程序的优化水平。您可以指定优化水平为0、1、2、3、4和5。设置为0会关闭优化。Funcname子程序必须在pragma之前原型化。

#pragma rarely_called(funcname [, funcname])

这个pragma向编译器后端暗示了指定的函数不常被调用。这允许编译器对这样的例程的调用点进行配置文件反馈风格的优化，并且没有配置文件收集阶段的开销。由于这个pragma选项只是一个建议，编译器的优化器可能不会根据这个选项进行任何优化。

#pragma returns_new_memory (funcname[, funcname])

这个pragma断言了指定函数的返回值不与该调用点上的任何内存混叠。实际上，该调用返回了一个新的内存位置。该信息允许优化器更好地跟踪指针值和澄清内存位置。这导致了改善的调度、管道化和环路并行化。然而，如果如果断言失败，程序的行为就变得不明确。

#pragma unroll (unroll_factor)

该pragma为unroll_factor参数接受一个正的整数值。该选项应用于当前块内的下一个环。对于除1之外的解开因子，该指令用作对编译器的建议，建议指定环应该由指定的因子解开。编译器就会在可能的时候使用该解开因子。

当解开因子为1时，该指令用作一个命令，向编译器规定了环不能被解开。编译器充分利用了3级或以上优化级别上的信息。

内联汇编

Sun Studio软件具有一个不同的内联汇编处理机制，它基于.Il内联模板文件和__asm ("string")关键字。在内联模板文件中，用户可以按以下形式编写模板：

```
.inline <name>,0

  <assembly code ...>

.end
```

参数传递和返回值基于底层架构的调用约定。编译器可以优化被提供代码的性能。详细信息请参见inline手册。

gcc有一个扩展的内联汇编机制，它陈述了输入和输出操作数，但是没有优化。

make选项

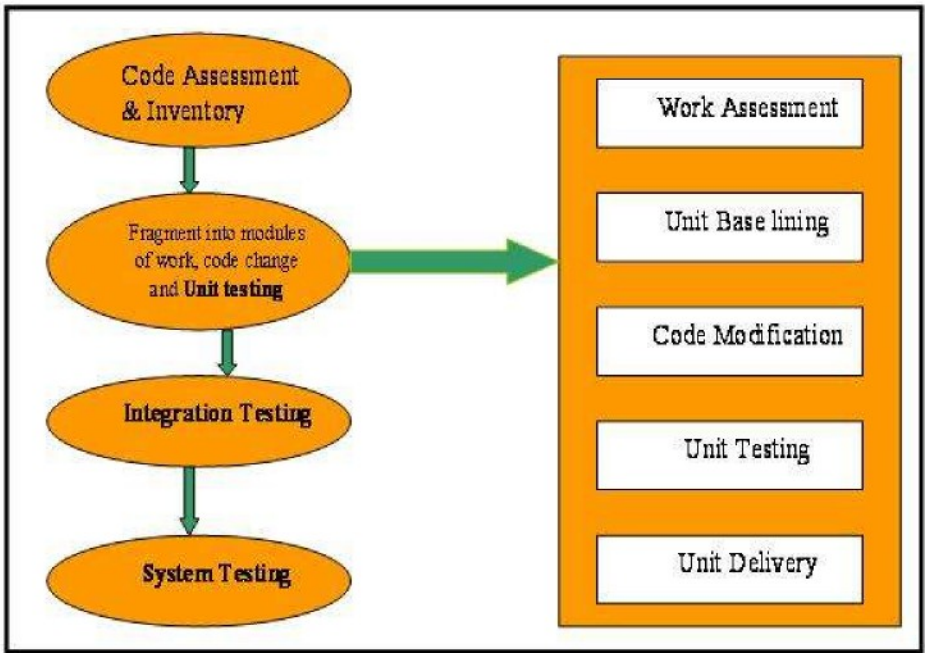
Linux GNU	Solaris OS	描述
-C dir, --directory=dir		在读取makefiles前更改到目录dir。如果指定了多个-C选项，每个都相对于前一个被解释：-C / -C usr等价于-C /usr。Solaris OS中没有等价的选项。
-d, --debug	-d, -dd	打印调试信息。
-e, --environment overrides	-e	环境变量覆盖了makefiles内的对齐。
-f file	-f file	将file用作makefile。
-h, - -help		显示帮助信息。Solaris OS中不存在等价选项。
-i	-i	忽略为remake文件执行的命令中的错误。
-I DIR		搜索DIR目录，查找包含的makefile。Solaris OS中没有等价的选项。
-k	- k	在当前目标上出现错误时，继续其他依赖性分支。
-n	-n	实际上不运行命令；只是打印它们。
-p	-p	打印出完整的宏定义集和目标描述。
-q	-q	问题模式。退出状态规定了目标文件是否是最新的。
-r	- r	不在隐式的规则文件中读取。
-R		禁用内置的变量设置。Solaris OS中没有等价的选项。
-s	- s	安静模式。不响应命令。
-S	- S	在命令返回一个非零的退出状态时，取消-k选项的效果，并停止处理。
-t	- t	接触目标文件，而不是重新制作它们。
-w		打印当前目录。Solaris OS中不存在等价选项。
-W file		假装目标文件刚被修改。等价于使用touch。

RCS、SCCS和CVS映射

RCS命令	SCCS命令	CVS命令	描述
ci file	sccs admin -i -nfile file	cvs add *filename* cvs commit *filename [1]*	首次登出文件，并创建修订历史文件。
co file	sccs get file	cvs -r checkout file	登出一个文件，以便读取。
co -l file	sccs edit file	cvs checkout file	登出一个文件，以便编辑。
ci file	sccs delta file		登入一个之前锁定的文件。
ident file	what file		打印关键字信息。
rlog file	sccs prs file	cvs history [-report] [-flags] [-options args] [files...]	打印文件历史。
rcsdiff -rx -ry file	sccs sccsdiff -rx -ry file		比较两次修订。
rcsdiff file	sccs diffs file	cvs diff [-lR] [format_options] [[-r rev1 | -D date1] [-r rev2 | -D date2]] [files...]	比较当前和上次修订。
r c sme rge -rx-y file	scc s edit -ix-y file		将两个版本间的变化合并到文件中。
rcs -l file			锁定最新修订。
rcs -u file			解除对最新版本的修订。由于它可能打破其他人的锁定，所以会向其他用户发送邮件解释原因。

建议的方法学

应用程序移植项目通常按照阶段性方式执行。下图显示了具有很多行代码和数据的项目的典型的主要阶段。



移植评估

这是一个很重要的阶段。首先，要准备Linux上应用程序的所有部分的清单。这有助于整体理解系统。在对总体系统的研究和设计的基础上，平台相关特性上的依赖性被标识出来。与可能在Solaris平台上发现的比较之后，会研究出一种合适的策略，使移植过程尽可能有效。于是工具（如果有的话）就可以确定了。像LinCAT和JScore这样的工具可能有帮助。从管理的角度来看，在开始工作之前需要合理估计出工作量。从项目里程碑和规定时限的角度来实现它也很方便。

Module-wise移植

这是代码中实际修改的地方。可通过适当地将项目分割成小的单元来完成。通常这被称为模块。每个模块一般有一个五步骤的方法。这些阶段如下所述：

1.评估：在该阶段，整个模块根据它已知的复杂度和依赖性来访问。与该模块相关的具体观察被记录下来。根据涉及工作量的不同，工作被分配给一个或多个工程师。

2.单元基线测试：也可在Solaris平台上考虑使用适当的测试套件（用在Linux上的，如果可用的话）。在该阶段，可利用参考安装对现有模块进行一些基线测试。

3.代码修改：在此阶段，可手工或通过IDE内适当的自动化工具对代码（C/C++代码，makefile、脚本、参考资料等）进行了适当的修改。假定有合适的源代码控制机制来帮助跟踪所作的修改。

4.单元测试：在代码修改完成并且模块准备好进行单元测试之后，就可利用Linux上使用的同一测试套件来进行测试。大部分情况下，这些测试套件在使用之前需要移植到Solaris OS上。

5.单元交付：在该阶段，模块可能作为共享库或者部分参与总体系统的组件而被适当打包。如果这些都不可能，或者它与手头上的任务无关，更新过的模块就被放到代码知识库中，直到集成测试时。

集成和功能测试

在所有模块被移植并且单元测试后，整个产品就脱离了源代码知识库并被适当打包。一旦完成，二进制文件就可安装在Solaris平台上并进行配置。利用可用的集成测试套件，测试结果被按照一种预先批准的标准格式记录下来。如果期望应用程序使用已存在数据，那么这些数据也需要考虑。

系统测试

项目的这部分对于业务理由很关键，不能轻易开始，除非所有模块都已参与了之前的所有阶段。再造代码应该满足新平台上声称的所有目标。该阶段中，特性、性能和其他期望行为的任何偏差都应该小心应对。根据该问题严重性的不同，找出受影响的模块或程序，并且进行适当的代码修改，直到问题解决。

部署

已移植应用程序在Solaris平台上的部署是在系统测试成功之后完成的。任何不明确的部署后行为都要仔细考虑和应对。