UbuntuNativePOSIX线程库Linux认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/623/2021_2022_UbuntuNati_c103_623583.htm 在2.6版本以前发布的Linux内核中，Linux线程库叫做LinuxThreads，为glibc2.0以后的GNU C库所支持。该库虽然使用了POSIX API，但是并不真正遵循POSIX标准。从2.6内核开始，Linux引入了NPTL。它比LinuxThreads在性能上有了很大的提高，也更遵循 POSIX标准。但是，仅仅使用2.6内核并不等于使用了NPTL。尽管有些发行版会同时携带NPTL和LinuxThreads, 但所有现代的Linux发行版都缺省携带NPTL。 用下面的命令可以查看你的系统上正在使用的POSIX实现： 编辑请注意，该代码中需要翻译的内容为：(1)“ This was returned from SUSE 9.1 installation ”翻译成“ 这是SUSE 9.1返回的结果 ”(2)“ This was returned from Fedora 2.6.9-1.667 Instatllation ”翻译成“ 翻译成“ 这是Fedora 2.6.9-1.667返回的结果 ”(3)“ This was returned from an old RedHat installation ”“ 翻译成“ 这是一个老版本的RedHat返回的结果 ”$ getconf GNU_LIBPATHREAD_VERSION $ getconf GNU_LIBPATHREAD_VERSION $ getconf GNU_LIBPATHREAD_VERSION 用下面的方法可以查看正在使用的Linux发行版是用什么编译工具编译链接的。 要找到/bin/ls链接的libpthreads库，如下： (代码)(P81倒数第14行) $ ldd /bin/ls |grep libc.so.6 从上面的输出内容可以看到，libc.so.6是和“ Native POSIX Threads Library by Ulrich Drepper ”一起链接的。 NPTL实现了一对一的线程模型.也就是说，一个用户线程对应一个内核线程。NPTL也实现了POSIX进程间的同步

原语，而且线程选项PTHREAD_PROCESS_SHARED也被明确支持了。 1 最大线程数 在Linux上一个应用程序能够创建的线程最大数量在不同的发行版上是不同的。运行在2个CPU 2GB内存上的SUSE9.1在pthread_create返回EAGAIN错误前允许创建16317个线程。EAGAIN错误的意思是说应用程序可能超过了系统的某些限制。该应用程序创建线程的栈大小为16384，这是创建线程时允许的最大栈大小。在把栈大小设置为16384前，该应用程序会在创建第 1021个线程时失败，并返回错误码ENOMEM。任何情况下，8000到16000个线程已经足以满足任何应用程序的需求。 要在Linux上创建大数量的线程，需要先做下面的事情： 1. 创建正确的栈大小(注释17). 2. 检查ulimit(可以输出内存、栈大小限制等的工具)，修改对应项或编辑/etc/security/limits.conf文件。 第4、5、6章更详细的讲述了POSIX线程的内容，以及它与NPTL的区别。 2 国际化(I18N)(注释18)和本地化 I18N对项目移植会有多大的影响在很大程度上取决于待移植的应用程序。如果一个应用程序仅需要一些简单的消息目录(message catalogue)转换、时期和时间显示，或使用正则表达式进行简单的文本串查找，那么把这些功能从UNIX平台移植到Linux上还是比较容易的。但是，如果该应用程序进行了复杂的文本分析，就像有时在文本编辑器理使用的那样，移植这些内容将会是比较困难的，见下一段的分析。 Linux遵循ISO对标准locale名称的命名规范，[locale]_[territory].[codeset]。其中，locale 由两个字符组成，代表言语.territory由两字符组成，代表国别。例如en_US.iso885915和zh_CN.gb18030。但是，每个系统上可用的locale和locale的内容是各不相同的。移植使用了locale复杂

应用的应用程序可能需要学习具体的语言规范和翻译规则，甚至需要修改Linux上已有的locale才能使移植后的应用程序像在源系统上那样运行。 表3-4列出了支持的GNU libc国际化函数。linux.ctocio.com.cn/imagelist/2009/168/5bb91ws241x2.pdf" target=_blanklt. input-file 4 如何创建消息目录(message catalog)(注释19) 消息目录是一个文件，用来把应用程序语言相关的输出内容转换成系统locale设置的语言。程序3-5给出了一个在Linux上如何用GNU xgettext和msgfmt工具创建消息目录的示例。 (代码)(P88第3行) 该例中，我们要以西班牙语输出本书的书名和作者。首先，我们对示例程序运行xgettext： (代码)(P88第22行) $ xgettext hello.c 这会生成一个叫message.po的文件： (代码)(P88第24行) $ cat message.po 编辑文件message.po，对每个要翻译的消息修改msgstr，并编辑charset(如果在运行xgettext前没有设置的话)。然后对message.po文件运行msgfmt命令。文件my_messages.mo指向的是bindtextdomain()设置的域。 (代码)(P89第16行) $ msgfmt v o my_message.mo message.po 为运行示例程序，我们在本地目录中为西班牙语(哥斯达黎加)创建一个目录(不使用缺省的/usr/lib/locale目录，因为我们没有root权限)： (代码)(P89第20行) $ mkdir p locale/es_CR/LC_MESSAGES 然后把my_message.mo移到该目录下： (代码)(P89第22行) $ mv my_message.mo locale/es_CR/LC_MESSAGES 为环境变量LC_MESSAGES输出正确的值，并运行示例程序： (代码)(P89第25行) $ export LC_MESSAGES=es_CR $ ./hello 至此，我们成功地在Linux上创建了一个消息目录文件。 5 大小端(Big/Little-Endian，也叫字节序)环境 Linux最初是在Intel平

台上开发的，而Intel平台主要是一个小端(little-endian)环境，但是，现在Linux已经被移植到了很多支持大端(big-endian)的硬件平台上。大小端，或者字节序，指的是一个数据元素及其每个单独的字节是如何存放的。在big- endian环境中，最低地址放在多字节的最高位(或者最左侧位).在little-endian环境中，最低地址放在多字节的最低位(或者最右侧位)。通常来说，第0位在big-endian环境中是最高位，但是在little-endian环境中是最低位。 程序3-6给出了一个打印数据字节内容的示例。分别在big-endian和little-endian环境中编译运行时会有不同的输出。 (代码)(P90第13行) 在Intel服务器(LE环境)上编译运行时，输出的内容是： (代码)(P90倒数第6行) 在IBM Power服务器(BE环境)上编译运行时，输出的内容是： (代码)(P90倒数第2行) 需要注意的是，前面的示例是使用gcc编译的，缺省情况下生成的是32位的应用程序。 大多数基于RISC的计算机(包括IBM PowerPC服务器等)和网络协议(Internet Protocol，IP)使用的都是BE结构，但是Intel和Alpha体系使用的是LE结构。移植软件时，需要特别小心大小端(字节序)问题，因为这些问题通常不易发现，而且一旦发生又很难定位。 移植后的软件通常会出现的问题包括： - 不统一的数据引用(注释20) - 在BE和LE之间共享数据 - 在网络设备(例如，IP和PCI)(注释21)间交换数据 不统一的数据引用较多发生在用户空间的应用程序中，而后两种问题常常在底层代码中遇到(例如，设备驱动程序)。不统一的数据引用往往是因为不正确的引用与大小端相关的数据类型，通常是在处理联合或指针类型时。大小端处理得当的代码应该包含一些定义来判断平台是BE或LE。一个好的编程习惯是，不要把指针强制转换成int，并且需

要时在转换过程中明确地引用数据类型和各字节的值。 6 从32位移植到64位 64位Linux平台正在逐步取代32位系统。64位的程序环境能够非常明显地提高内存寻址的性能和操作超大数据结构时应用程序的吞吐量。运行在IBM PowerPC 和AMD 的64位体系结构上的Linux可以同时运行32位和64位应用程序，而且没有任何性能损失。当32位环境不能为应用程序提供足够的内存地址空间时可以把应用程序编译成64位的，从而有效地利用Linux的上述优点。 用-m64标志可以让gcc生成64位的目标文件，如下例： (代码)(P92第8行) $ gcc m64 sample.c o sample.o 需要注意的是，在有些平台上，例如IBM PowerPC，gcc编译器缺省生成的是32位目标文件，即使运行的是64位Linux。而对于运行在AMD 64位体系上的64位Linux，gcc缺省生成的是64位目标文件。和UNIX平台类似，64位的目标代码只能和其它64位的目标代码一起运行。因为地址冲突，32位的代码不能和64位的代码在同一个应用程序空间中运行。 32位的数据类型模型和64位的数据类型模型是不同的。32位应用程序的C语言数据类型模型是ILP32模型，其中，I代表int，L代表 long，P代表指针，32表示这些数据类型都是32位的。64位应用程序的数据类型模型是LP64模型。除了int类型外，long(L)和指针(P) 类型都变成了64位的。C语言的int和符点类型在两种数据类型模型中是相同的。 7 常见的移植错误 在代码不兼容的问题中，数据类型不匹配是较为常见的。这常常是因为大小端和32位到64位的问题。我们通常会遇到，32位的应用程序会假设 int、long和指针类型具有同样的字节大小。但是，long和指针类型的字节大小在LP64数据模型中变成了64位的，这个变化本身是导致ILP32 到LP64问

题的主因。在分析阶段，要留出时间尽早找到这些不兼容的代码。 8 假设LP64中int和指针具有同样的字节大小 LP64中，指针类型(ptr)是64位的。如果没有注意到这个区别至少会导致编译器警告(或者更坏的情况，导致应用程序出现未定义的行为)。 来看下面的例子：(代码)(P93第2行) 要解决这个问题，可以把int改称long，或者更好的方法，使用stdint.h中定义的uintptr_t。 9 忽略了int和long类型字节大小的不同 在ILP32环境中，int和long具有同样的字节大小，这也很容易让编程人员错误地以为在LP64中int和long也是同样大小。 来看示例3-7。(代码)(p93倒数第15行) 编译成32位应用程序运行：(代码)(P93倒数第7行) gcc bad_1.c o foo $ ./foo 80000000 编译成64位应用程序运行：(代码)(P93倒数第3行) gcc m64 bad_1.c o foo $ ./foo ffffffff80000000 调用sizeof返回一个size_t类型的整数。因为size_t类型在LP64中变成了64位的，所以注意不要把sizeof的返回值传给期望int类型参数的函数。否则，会被截短。 10 忽略符号位的扩展 示例3-7同时也演示了转换到LP64时符号位的扩展问题。ISO C整型进位(promotion)规则表明，字符、短整数或整数位，所有有符号的或无符号的，或枚举类型的对象，都可能和整数一起出现在某个表达式中。这种情况下，如果整形能够表示上述所有源类型的值，则这些类型的值都会转换成整数.否则，转换成无符号整数。 要解决该问题，可以把1 lt. 31 改成 1Llt.31。 11 字符串转换时缺少必要的检查 字符串函数，例如printf，sprintf，scanf，以及sscanf，用的是格式化的字符串，这些字符串需要遵循long类型规范，pencentl用于long类型参数，percentp用于指针参数。在LP64环境中不使用这些规范将导致不可预测的格式化结果。 12 最优方法 一个

移植32位应用程序到64位环境的最优方案建议把移植工作分两个步进行。第一步先把应用程序从源系统(AIX、Solaris,或HP-UX)移植到Linux上.第二步再把移植后的32位程序改成64位的。 13 小结 在真正的移植开始之前，对应用程序的分析可以说是最重要的工作。如果分析做得好的话，可以发现一些隐藏的陷阱，并以此来进一步完善整个项目计划。极少数情况下，待移植的应用程序可能使用了一些平台相关的特性，而这些特性又是Linux所不支持的。此时，需要移植人员来找到一个绕开或替代该特性的方法。幸运的是，现在的各种Linux版本都支持最常用的API标准，例如POSIX线程、大页面、异步I/O、消息队列、64位结构等。在Linux 上找到源系统的一些替代方法从来没像现在这么容易。 下面列出了本章讲述的一些重点内容： - 文档“ Conflicts between ISO/IEC 9945(POSIX) AND THE Linux Standard Base ”详细描述了Linux支持的标准。(注释22) - Linux提供了支持库版本化的三种方法：内部版本化、外部版本化，以及符号版本化。 - 通过Native POSIX线程库，Linux现在更完整地实现了对POSIX线程的支持，这使得移植多线程应用程序到Linux变得更加容易了。 - Linux对UNIX平台上使用大页面支持的应用程序也提供了大页面支持功能。 - 针对具体情况，大小端环境可能对待移植的应用程序产生影响。大小端问题只有在待移植的应用程序所在的源平台和目标Linux平台使用的字节序不同时才会有影响。 - 从32位到64位的移植应该当成一个完全独立的移植过程。如果待移植的应用程序是32位的而且要移植成64位的程序在Linux上运行，那么应该把该过程当成两个独立的移植项目：第一个是把32位程序移植到Linux上，第二个是把32位

程序移植成64位的。 本章对Linux2.6的功能只介绍了一个大致的轮廓，具体的移植章节(移植Solaris、AIX，和HP-UX应用程序)通过列举Linux和各UNIX平台之间的区别及相似性更详细地讲述了这些技术特性。接下来我们就进入各移植章节。