跨平台系统迁移

高 巍
黑龙江省大庆市萨尔图区发展路 6 号，北京 100600

[摘要]随着信息技术的发展和应用范围的扩大，企业和组织在不同的平台和操作系统上开发和运行应用程序的需求越来越多，为了满足业务需要或者降低成本，许多企业和个人也会将自己的应用程序做跨平台迁移，跨平台系统迁移可以帮助企业和组织降低 IT 系统的复杂性，提高应用程序的灵活性和可移植性，并减少 IT 投资的成本和风险。然而，不同平台和操作系统之间存在着巨大的差异，这些差异涉及到硬件架构、操作系统接口、编程语言和编译器等方面，因此跨平台系统迁移是一项复杂的工作。

[关键词]跨平台系统迁移；方案设计；实施过程
DOI：10.33142/sca.v7i1.10908 中图分类号：TP315 文献标识码：A

Cross Platform System Migration

GAO Wei
No.6 Development Road, Sartu District, Daqing City, Heilongjiang Province, Beijing, 100600, China

Abstract: With the development of information technology and the expansion of its application scope, there is an increasing demand for enterprises and organizations to develop and run applications on different platforms and operating systems. In order to meet business needs or reduce costs, many enterprises and individuals also migrate their applications across platforms. Cross platform system migration can help enterprises and organizations reduce the complexity of IT systems, improve the flexibility and portability of applications, and reduce the cost and risk of IT investment. However, there are huge differences between different platforms and operating systems, which involve hardware architecture, operating system interfaces, programming languages, and compilers. Therefore, cross platform system migration is a complex task.

Keywords: cross platform system migration; scheme design; implementation process

引言

跨平台迁移是将一个系统从一个操作系统或硬件平台迁移到另一个操作系统或硬件平台的过程。这种迁移通常需要将应用程序和数据从一个平台转移到另一个平台，并确保它们在新平台上能够正常运行。跨平台系统迁移是一项复杂的技术工作，在跨平台系统迁移过程中，需要对原有系统进行分析、评估、规划、实施等一系列步骤，此外，还需要进行测试和验证，以确保迁移后的系统能够正常运行并满足业务需求。

不同的操作系统具有不同的特点和优势，根据自己的需求选择适当的操作系统，在迁移时，首先需要检查应用程序是否兼容目标操作系统，如果应用程序无法在目标操作系统上运行，则需要寻找替代品或考虑修改应用程序的一部分，虽然可能会有一些挑战和风险，但正确的方法和计划可以使迁移过程变得容易和顺利，

为了让迁移后的系统能够在新平台上运行，有时需要重写程序代码，并且需要重新编译、测试和部署。对于 c/c++程序，从 Windows 系统往 Linux 系统迁移的情况，IBM 总结了一些常用函数的迁移案，除此之外的网络设定，Tcp/Ip 通信和串口通信的迁移方法等在网络上也有记载，

本文作者根据自己参与过的系统迁移项目，参考的各方资料，以从需要费用的 Windows 系统迁移到免费开源的 Linux 系统，c/c++开发出来的程序为例，在框架不变的基础上，从修改部分代码的角度，介绍再迁移的经验，比如方案设计和实施过程，问题的调查方法和解决办法，以及仍然存在的课题等，希望可以为跨平台系统迁移提供一定的参考和帮助，达到降本增效的目的。

2 将应用程序从 Windows 迁移到 Linux 系统的基本步骤

（1）分析现有系统：首先需要评估当前系统的应用程序和数据，了解现有系统的架构、技术栈、数据模型、功能需求和性能指标等，以确定哪些模块和数据需要迁移，并检查它们是否兼容 Linux 操作系统。

（2）数据迁移：将数据从 Windows 系统转移到 Linux 系统。可以使用工具如 rsync、scp、ftp 等进行文件传输，也可以使用专业的数据迁移工具进行迁移。

（3）应用程序迁移：将 Windows 系统上运行的应用程序迁移到 Linux 系统上。如果应用程序没有 Linux 版本，则需要寻找替代品或重写应用程序。本文重点说明重写应用程序时的一些做法，本次迁移的程序是一个服务器程序，不是从零开始重新设计，而是对既有的程序尽可能地保留使用的原则上进行改造。

（4）网络设置：需要检查和更改网络设置，以确保程序在新系统能够正常连接到网络。

（5）测试和验证：在迁移完成后，需要进行测试和验证，以确保系统能够正常运行，并且满足业务需求。搭建两种系统的测试环境，进行比较测试。

3 跨平台系统迁移修改程序时的准备

（1）明确目标系统的基本信息，比如下面的信息。

表 1　目标系统的基本信息

System 版本	要确认是 64-bit 还是 32-bit。 例如：Ubuntu　16.04
编译器版本	例如：GCC 4.4.6
CPU	例如：Intel(R) Pentium(R) 4 CPU 3.40GHz
Shell 版本	例如：bash 4.1.2
endian	通信时 big-endian 还是 little-endian
库版本	例如：glibc 2.12

（2）共通确认点

代码以外，可以从以下几个方面一一进行确认，没问题的直接跳过，需要做改造的按照改造方针去做即可。

最大网络接续数：通过 FD_SETSIZE 定义的，Linux 比 Windows 的值大，不需要特别注意

最大路径长度：Linux 比 Windows 的长度值大，不需要特别注意

批处理：Windows 是 Bat 文件，根据实现的功能改造成 Linux 的 shell 文件

服务类：比如 Web 服务，Windows 的是 IIS，Linux 是 Apache。定时处理的话 Windows 是 task，要变换成 Linux 的 crontab。

（3）程序代码的确认点

大概有以下几个方面需要考虑。

API

字符/字符串处理，文件/设备处理，数据库处理，线程进程处理，共享内存，信号量，内存处理，时间处理（包括 sleep），Signal，环境变量，Section，Socket（阻塞非阻塞模式），事件选择等几个方面考虑，罗列出需要改造的函数，这里需要注意的是 Linux 上没有对应的 API，需要自己实现一个等价函数时只需要追加一个函数的定义，函数名跟 Windows 的函数名相同，把实现的功能追加进去，不需要把所有调用 Windows 的函数的地方做替换。

取得路径名的方法

Windows 和 Linux 的方法不同的，一是路径分隔符不同：在 Windows 系统中，使用 `\` 作为路径分隔符；而在 Linux 系统中，使用 `/` 作为路径分隔符。二是根路径不同：在 Windows 系统中，根路径为盘符加上 `\`，例如 `C:\`；而在 Linux 系统中，根路径为 `/`。三是文件路径表示方式不同：在 Windows 系统中，文件路径可以使用驱动器号加上冒号作为开头，例如 `C:\Windows\System32\cmd.exe`；而在 Linux 系统中，文件路径以 `/` 开头，例如 `/usr/bin/bash`。

数据类型

充分利用宏定义去重构变量的数据类型，不要再使用的地方全部去改。有一些是 Windows 特有的类型，比如 HINSTANCE 是 Windows 系统中的一个句柄类型，在原有的代码中用这个类型定义的变量也比较多，在重构代码的时候较为普遍的做法是不去一一修改定义变量的地方，而是用 typedef 替换成目标系统的相应类型，比如创建一个数据类型的头文件，在头文件中如下定义，这样在所有使用 HINSTANCE 的地方都可以在 Linux 系统下变得有效了。

```
Windows:
HINSTANCE        testHinstance = NULL;
Linux:
typedef void *HINSTANCE;
HINSTANCE        testHinstance = NULL;
```

图 1　Windows 程序和 Linux 程序的比较

注册表

跟 Windows 在注册表中记录产品的所有信息不同，Linux 中一切都是文件

环境变量的切分字符

Windows 用“;”’，Linux 用“:”

路径的切分字符

Windows 的路径中切分字符是“\”，Linux 则是“/”

文件名字的大小写

Windows 的文件名或者路径不区别英文字母的大小写的，Linux 系统是区别的。因此，在改造头文件和使用其他文件的时候需要留意。

头文件

每个文件中使用到的头文件要一一确认，比如：#include <winbase.h>明显是 Windows 使用的，Linux 中不存在这样的头文件，针对头文件定义声明的内容，查找有没有 Linux 的头文件中有定义，如果没有，需要新做成一个头文件。有些头文件比如<stdio.h>，虽然 Windows 和 Linux 系统都有这个文件，但是里面的内容是有差别的，缺少的部分需要补充上。

换行符

Windows：\r\n、Linux：\n

扩展名

Windows 的可执行程序，批处理程序的扩展名有固定的名字，Linux 中可以没有扩展名，当然为了方便理解通常我们会将 shell 文件定义为 XXX.sh

编译

原有 Windows 程序的编译选项，在 Linux 程序编译时要有对应的选项。

另外，对于程序中既有 c 文件又有 c++文件时，存在

调用的时候，需要对供外部使用的接口额外追加以下的内容，才能编译链接通过[1]。

```
#ifdef __cplusplus
extern "C" {
#endif
//c program
#ifdef __cplusplus
}
#endif
```

图 2 Windows 程序和 Linux 程序的比较

4 需要注意的事项以及可能遇到的课题

4.1 静态代码分析

使用静态代码分析工具，查找编程中的常见错误，比如变量未初始化，内存泄漏等。

4.2 单步调试

代码总量太大用自动化测试工具的情况下，为了方便统计测试情况通常加上覆盖率工具 GCOV，LCOV 等，比如规定未变动的部分无视，只有重构部分达到 100%即可的话可以及时掌握测试的进度。

4.3 线程锁

线程锁是一种用于多线程编程中同步访问共享资源的机制。线程锁可以确保在同一时刻只有一个线程可以访问共享资源，从而避免了多个线程同时对共享资源进行修改的情况，从而导致数据的不一致性和错误。

线程锁问题包括以下几个方面：

死锁问题：当多个线程同时请求锁时，如果它们之间的依赖关系形成了一个环路，就会出现死锁现象，导致程序无法继续执行。

锁竞争问题：当多个线程同时请求同一个锁时，就会出现锁竞争问题，如果锁竞争过于激烈，就会导致系统性能下降，甚至出现死锁。

优先级反转问题：当一个低优先级的线程持有了锁，而高优先级的线程需要等待锁时，就会出现优先级反转问题，导致高优先级的线程无法及时执行，从而影响系统的响应能力。

Windows 系统中对于同一个线程用 EnterCriticalSection 多次加锁是没有问题的，但是 Linux 系统像如下例子对于同一个线程多次加锁会出现死锁的现象。

```
Window(OK):
Function(){
        EnterCriticalSection(&cs);
        funTest();
        LeaveCriticalSection(&cs);}
funTest(){
        EnterCriticalSection(&cs);
        LeaveCriticalSection(&cs);
}

Linux(Deadlock):
Function(){
        pthread_mutex_lock(&mutex);
        funTest();
        pthread_mutex_unlock(&mutex);
funTest(){
        pthread_mutex_lock(&mutex);
        pthread_mutex_unlock(&mutex);
```

图 3 Windows 程序和 Linux 程序的比较

为了避免线程锁问题，可以采用以下几个方法：

（1）尽量减少锁的使用，尽量避免多个线程同时访问同一个资源。

（2）尽量避免持有锁的时间过长，尽快释放锁，让其他线程有机会访问共享资源。

（3）采用不同的锁策略，如读写锁、乐观锁等，根据具体情况选择最合适的锁策略。

（4）使用死锁检测工具，及时检测并解决死锁问题，保证程序的正常运行。

4.4 内存

内存问题是指在程序运行过程中，因为内存使用不当而导致程序出现异常，如内存泄漏、内存溢出等。

内存泄漏是指程序在运行过程中分配的内存没有被及时释放，导致内存空间被占满，从而影响程序的正常运行。内存泄漏可能是由于程序设计缺陷、资源管理不当或代码错误等原因导致的。

内存溢出是指程序在申请内存时，超出了系统可用的内存空间，导致程序异常或崩溃。内存溢出可能是由于程序设计缺陷、数据结构错误、算法不当或输入数据异常等原因导致的。

为了避免内存问题，可以采用以下几个方法：

（1）合理使用内存管理函数，如 malloc、free 等，确保内存分配和释放的正确性和及时性。

（2）使用内存分析工具，及时检测和解决内存问题，如 Valgrind 等。

（3）自己写个内存打印的工具，通过 gdb 的方式查看内容使用情况，查找泄露处。

4.5 Socket

Socket（套接字）是一种通信协议，用于在网络中实现进程间的通信。Socket 问题通常指与 Socket 相关的程序运行时出现的异常或错误。

常见的 Socket 问题包括：

连接超时：当客户端向服务器发起连接请求时，如果服务器没有及时响应或者网络延迟较大，就可能导致连接超时。

连接中断：当客户端和服务器之间的连接中断时，可能是由于网络异常、服务器故障或客户端程序错误等原因导致的。

端口占用：当一个程序使用了某个端口，另一个程序再次尝试使用该端口时，就会导致端口占用异常。

数据丢失：在数据传输过程中，由于网络延迟、网络拥塞等原因，可能导致数据丢失或者传输不完整。

为了避免 Socket 问题，可以采用以下几个方法：

（1）合理设计程序架构，确保 Socket 的正确使用。

（2）避免程序中出现资源泄漏等错误，及时释放资源。

（3）采用可靠的网络协议，如 TCP 等，确保数据传输的可靠性和完整性。

（4）使用心跳机制等技术，检测连接状态，及时处理连接中断等异常。

（5）避免在网络高峰期发送大量数据，减少网络拥塞的发生。

4.6 epoll 问题

epoll 是一种高效的 I/O 多路复用机制，可用于在单个线程中同时处理多个文件描述符的 I/O 操作。epoll 问题通常指与 epoll 相关的程序运行时出现的异常或错误。

使用 epoll 时，要注意根据实际情况设置触发模式，是边缘触发（Edge Triggered）模式还是水平触发（Level Triggered）。

常见的 epoll 问题包括：

文件描述符泄漏：在使用 epoll 机制时，需要及时关闭不再使用的文件描述符，否则可能导致文件描述符泄漏，占用系统资源。

事件处理错误：在使用 epoll_wait 函数时，需要正确处理返回的事件类型和事件数据，否则可能导致程序逻辑错误或者内存访问错误。

内存泄漏：在使用 epoll 机制时，需要正确管理内存资源，避免内存泄漏，导致系统性能下降或者程序崩溃。

程序死锁：在使用 epoll 机制时，需要避免出现死锁情况，否则可能导致程序无法正常运行。

为了避免 epoll 问题，可以采用以下几个方法：

（1）合理设计程序架构，确保 epoll 的正确使用。

（2）避免程序中出现资源泄漏等错误，及时释放资源。

（3）使用合适的数据结构，如红黑树等，提高 epoll 的性能和效率。

（4）使用非阻塞 I/O 技术，避免阻塞等待 I/O 操作，提高程序的并发能力。

（5）使用线程池等技术，提高程序的并发处理能力，避免出现死锁等问题。

4.7 共享内存

共享内存是一种进程间通信（IPC）方式，它允许多个进程访问同一块物理内存区域，从而实现数据共享。

共享内存通常分为两个步骤：

（1）创建共享内存区：由一个进程创建并初始化一个共享内存区，然后允许其他进程访问该内存区。

（2）访问共享内存区：其他进程可以通过映射该共享内存区到自己的地址空间来访问该内存区。

共享内存的优点是速度快，因为它避免了数据的复制，多个进程可以同时访问同一块内存，这样可以减少数据传输的时间和开销。但是共享内存也有一些缺点，例如需要进行同步和互斥处理，以避免多个进程同时修改同一块内存造成的竞态条件，还需要注意内存的管理和释放问题。

在使用共享内存时，需要注意以下几点：

（1）需要使用特殊的系统调用来创建和访问共享内存区。

（2）多个进程需要协调访问共享内存区，以避免数据冲突和竞态条件。

（3）需要注意共享内存区的大小和管理，避免内存泄漏和其他内存问题。

（4）在使用共享内存时需要保证数据的正确性和一致性，避免出现数据错误或不一致的情况。

（5）共享内存通常需要搭配其他 IPC 方式一起使用，例如信号量、管道等，以实现更完整的进程间通信机制。

5 结论

本文中提到的编程语言和操作系统已经被使用多年，并不是最新出现的语言和操作系统，他们对开发人员来说很熟悉，但由于移植所涉及的工作不仅数量大，而且细节很多，作者根据自己的项目经验进行了部分总结，日后仍需不断的实践以积累更多更全面的经验[3][4]。

[参考文献]

[1]兰雨晴,洪雪玉.从 Windows 到 linux 的应用移植实现-平台技术与接口篇[M].北京:国防工业出版社,2013.

[2]Fall K R.Advanced Programming in the UNIX Environment[M].Addison-Wesley:Professional America,2013.

[3]Pradillo M A , Himyr N , Sanmillan P L.Migrating Engineering Windows HPC applications to Linux HTCondor and Slurm Clusters[J].The European Physical Journal Conferences,2020,245(1):09016.

[4]Shao S,Tian S,Guo S.Container-Based Internet of Vehicles Edge Application Migration Mechanism[J]. 计算机、材料和连续体(英文),2023(6):4867-4891.

作者简介：（1985.12—），毕业院校：北京化工大学，所学专业：电子信息工程，2008 年本科毕业，学士学位。