(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 105242986 A
(43)申请公布日 2016.01.13

(21)申请号 201510576270.1

(22)申请日 2015.09.11

(71)申请人 金步国
地址 223001 江苏省淮安市开发区杨圩街28号

(72)发明人 金步国

(74)专利代理机构 东莞市神州众达专利商标事务所（普通合伙） 44251
代理人 皮发泉

(51)Int.Cl.
G06F 11/14(2006.01)
G06F 17/30(2006.01)

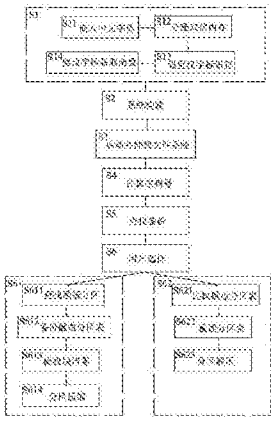
权利要求书3页 说明书11页 附图1页

(54)发明名称

备份与还原Windows操作系统的方法

(57)摘要

本发明公开一种备份与还原Windows操作系统的方法。首先，向FreeBSD内核中添加中文字体，同时修改内核的字体获取函数，从而实现内核级的中文显示支持；然后，将同时支持BIOS与UEFI规范的GRUB2引导管理器、经过中文化修改的FreeBSD内核、实现备份与还原功能的用户空间程序、以及保存备份文件的储存空间组合，形成该备份与还原系统。再通过本机硬盘、外置存储、PXE网络三种方式之一，启动该备份与还原系统，并根据启动方式的不同，分别挂载ZFS或NFS文件系统；通过序列号模块，计算出本机的“主机序列号”以及“硬盘序列号”，并将其作为独一序列号，查找是否存在对应的备份文件；最后，根据用户的选择，进入“备份”或“还原”流程。



CN 105242986 A

1.一种备份与还原Windows操作系统的方法,其特征在于,包括以下步骤:

S1、FreeBSD内核中文化:向FreeBSD内核中添加中文字体,并修改FreeBSD内核中的字体获取函数,以实现内核级的中文显示支持;

S2、系统构建:将同时支持BIOS与UEFI规范的GRUB2引导管理器、经过中文化修改的FreeBSD内核、实现备份与还原功能的用户空间程序、以及保存备份文件的储存空间组合在一起,形成该备份与还原系统;

S3、启动并挂载文件系统:通过本机硬盘、外置存储、PXE网络三种方式之一,启动该备份与还原系统,并根据启动方式的不同,分别挂载用于存放备份文件的ZFS或NFS文件系统;

S4、计算序列号:通过序列号模块,计算出本机电脑的“主机序列号”及“硬盘序列号”,并将其组合为用于识别本机及内置硬盘的独一序列号;

S5、查找备份:查找是否存在与本机匹配的“主机序列号-*.mbr”的文件,若存在,则显示“备份”与“还原”两个选项供用户选择,否则只显示一个单独的“备份”选项;

S6、用户选择:根据用户的选择,进入“备份”流程,以备份本机的磁盘分区表以及所有Windows系统分区;或进入“还原”流程,以从备份文件中还原本机的磁盘分区表以及所有Windows系统分区。

2.根据权利要求1所述的备份与还原Windows操作系统的方法,其特征在于,S1的FreeBSD内核中文化处理具体包括以下步骤:

S11、嵌入中文字体:向内核中添加中文点阵字体,将一个汉字劈为左右两个8×16像素的二维点阵,从而让每半个汉字等价于一个西文字符;

S12、分配双倍内存:保持内核中原有的所有数据结构与变量定义不变,特别是用于保存字符代码及字符属性的变量的定义不变,但是使用动态内存分配函数为其额外多分配一倍的动态内存,并将多分配出来的内存完全用于容纳中文字符的代码;

S13、设置汉字标识符:首先将代码为0x00与0xff的两个字符统一替换为0x20,然后不再将0x00与0xff视为普通的字符代码,而是将其处理为汉字左右部分的标识符;

S14、修改字体获取函数:修改FreeBSD内核中原有的获得字体数据的函数,添加一个判断"字符代码"是否为0x00或0xff的逻辑,以正确获取中文的字符代码,然后到已在S11步骤嵌入内核的中文字体中提取相应的字体数据;最后,直接调用原有的字形绘制函数,即可将汉字显示在屏幕上。

3.根据权利要求2所述的备份与还原Windows操作系统的方法,其特征在于,S11中的中文字体具体制作步骤如下:

S111、选取任意一款等宽中文字体,要求其中每个半角字符的宽度严格等于全角字符的一半;

S112、对每个代码小于256的半角字符按照8x16点阵进行采样,将点阵中有笔画的点标记为"1",没有笔画的点标记为"0",这样每个半角字符就转化成了一个8x16的二维矩阵;

S113、对每个代码大于255的全角字符按照16x16点阵进行采样,并将采样结果从中间劈为左右两半,得到左右两个8x16点阵,同样将点阵中有笔画的点标记为"1",没有笔画的点标记为"0",这样每个全角字符就转化成了左右两个8x16的二维矩阵;

S114、将每个 8x16 的二维矩阵的每一行的 8 个 bit 转化为一个 16 进制数，这样每个矩阵就会得到 16 个 0x00 ～ 0xFF 之间的无符号单字节整数，这样就完成了字符的数字化；

S115、将每个字符按照其字符代码的顺序，依次简单的连接起来，封装到字体数组中即可作为嵌入内核的字体使用。

4. 根据权利要求 1 所述的备份与还原 Windows 操作系统的方法，其特征在于，S4 中的计算序列号的方法为 MD5 单向散列算法，具体算法如下：

主机序列号 =MD5（主板制造商名称 + 主板硬件序列号）；

硬盘序列号 =MD5（硬盘制造商名称 + 硬盘硬件序列号）。

5. 根据权利要求 1 所述的备份与还原 Windows 操作系统的方法，其特征在于，S6 中所述“备份”流程具体包括下述步骤：

S611、查找系统分区：通过 Windows 系统分区查找模块查找 Windows 系统所在的硬盘及分区；

S612、备份磁盘分区表：通过磁盘分区表备份模块将每个 Windows 系统所在硬盘的分区表与引导区备份为对应的“主机序列号 - 硬盘序列号 .mbr”文件；

S613、磁盘块清零：通过磁盘清理模块将每个 Windows 系统分区上的空闲磁盘块清零；

S614、分区压缩：通过分区压缩模块将每个 Windows 系统分区压缩为对应的“主机序列号 - 硬盘序列号 - 分区号 .xz”文件。

6. 根据权利要求 5 所述的备份与还原 Windows 操作系统的方法，其特征在于，S613 中磁盘块清零的具体做法为：挂载指定的磁盘分区，然后在该分区上创建一个临时文件，接着向这个临时文件中持续写入无限长的 "00000....." 二进制零串，直到因为磁盘满而出错，最后再删除这个临时文件。

7. 根据权利要求 5 所述的备份与还原 Windows 操作系统的方法，其特征在于，S614 中分区压缩模块的具体步骤为：

S6141、载入开源的 liblzma 库，其默认使用单线程压缩；

S6142、检测 CPU 的总核数，若大于 1 则开启 liblzma 库的多线程压缩功能，并且将线程数设为总核数；

S6143、打开指定的磁盘分区所对应的设备文件，将磁盘分区视为一串 "0" 与 "1" 组成的字节流，并将其作为 liblzma 库的压缩输入，同时将 liblzma 库的压缩输出保存到对应的“主机序列号 - 硬盘序列号 - 分区号 .xz”文件中，压缩完成后，关闭打开的设备文件与输出文件。

8. 根据权利要求 1 所述的备份与还原 Windows 操作系统的方法，其特征在于，S6 中所述“还原”流程具体包括下述步骤：

S621、还原磁盘分区表：通过磁盘分区表还原模块将每个匹配的“主机序列号 - 硬盘序列号 .mbr”文件还原到对应的本机硬盘上；

S622、重读分区表：重新读取所有硬盘的分区表；

S623、分区解压：通过分区解压模块将每个匹配的“主机序列号 - 硬盘序列号 - 分区号 .xz”文件解压到对应硬盘的对应分区上。

9. 根据权利要求 6 所述的备份与还原 Windows 操作系统的方法，其特征在于，S623 中分区解压模块的具体步骤为：

S6231、载入开源的 liblzma 库，其默认使用单线程解压；

S6232、检测 CPU 的总核数，若大于 1 则开启 liblzma 库的多线程解压功能，并且将线程数设为总核数；

S6233、打开指定的“主机序列号 - 硬盘序列号 - 分区号 .xz”文件，并将其作为 liblzma 库的解压输入，同时打开对应的磁盘分区所对应的设备文件，并将其作为 liblzma 库的解压输出；

S6234、解压完成后，关闭打开的备份文件与设备文件。

备份与还原Windows操作系统的方法

技术领域

[0001] 本发明涉及Windows操作系统的备份与还原技术，特别涉及一种利用FreeBSD技术备份与还原Windows操作系统的方法。

背景技术

[0002] (一)Windows操作系统的备份与还原

常见的Windows操作系统的备份与还原技术主要有两种：一种是Windows操作系统自带的“系统还原”技术，另一种是基于Ghost的各种“一键Ghost”技术。

[0003] Windows自带的“系统还原”技术有严重的先天缺陷。因为使用“系统还原”的前提是能够正常启动Windows系统，所以一旦Windows系统自身无法正常启动（例如某些关键文件被破坏），这项技术也就失效了。

[0004] 所有的“一键Ghost”在本质上都由如下三个部分组成：

1. 一个虚拟光盘镜像（iso文件）或虚拟软盘镜像（img文件）。其中包含了独立的Windows或DOS环境以及Ghost工具。

[0005] 2."GRUB4DOS"引导管理器。由于"GRUB4DOS"引导管理器支持一种独特的"磁盘仿真"功能，可以将iso文件或img文件作为虚拟光盘或虚拟软盘启动，从而为“一键Ghost”提供了至关重要的独立Windows或DOS环境，而其他引导管理器都不具备"磁盘仿真"功能，因此所有的“一键Ghost”都无一例外的使用了"GRUB4DOS"引导管理器。

[0006] 3. 一个用来保存备份文件的NTFS或FAT文件系统。通常是C盘之外的其他磁盘分区（一般默认D盘）。

[0007] “一键Ghost”通过在一个独立的Windows或DOS环境中运行Ghost程序，不再依赖于电脑上原有的Windows系统，巧妙的避开了“系统还原”技术的先天性缺陷。因此，“一键Ghost”在实践中得到了广泛的应用。但是这种技术仍然存在一些无法克服的缺点：

1. 安全性不佳。一方面，由于“一键Ghost”必须在Windows环境中安装，所以其自身(iso或img文件）只能安装在NTFS或FAT文件系统上（一般默认C盘），另一方面，由于Ghost程序也需要在Windows或DOS环境中运行，所以备份文件也只能存储在NTFS或FAT文件系统上（一般默认D盘）。因此在Windows正常使用的过程中，“一键Ghost”自身（iso或img文件）与备份文件（gho文件）都有可能被误删除或者被恶意程序破坏。虽然在实践中，各种“一键Ghost”都会将这些文件加上隐藏属性，但是依然不能从根本上杜绝这种风险。从根本上说，所有基于Windows或DOS系统的备份与还原技术，都存在这个缺点。

[0008] 2. 数据压缩技术不理想。Ghost在备份时使用了1993年发明的deflate压缩算法，这种压缩算法重在节约CPU资源，但是压缩效果比较差。随着CPU性能的突飞猛进，应该使用一种压缩率更高、更能充分利用现今强大CPU性能的压缩算法。同时由于Ghost仅使用单线程压缩，因此也不能充分利用现今多核CPU的并行计算能力。

[0009] 3. 智能化程度低，不适合大规模场景下的使用。“一键Ghost”是面向个人用户设计的产品，并未考虑规模较大的企业环境，主要问题在于备份文件与目标机器之间的匹配

问题。假设某公司有1000台电脑，维护人员在给这1000台电脑进行Ghost备份时必须明确的区分每台电脑的备份文件（通常是使用不同的文件名）。而在还原时又要根据每台电脑的标识（例如资产编号），找到与这台电脑匹配的备份文件，然后将它们还原到对应的电脑上。由于数量巨大，靠人工手动匹配，是很容易出错的，所以缺乏自动化的匹配机制使得维护人员的工作量与错误率大大增加。

[0010] 4. 不支持最新的UEFI技术规范。这个问题的根源并不在Ghost本身，而是在于各种“一键Ghost”所使用的"GRUB4DOS"引导管理器。由于"GRUB4DOS"引导管理器自身并不支持UEFI技术规范，所以无法在UEFI平台上启动，而各种“一键Ghost”又必须依赖于"GRUB4DOS"所独有的"磁盘仿真"功能，从而间接造成了各种“一键Ghost”在事实上无法在纯UEFI平台上工作的囧境。进一步深入来说，"GRUB4DOS"所独有的"磁盘仿真"功能其实依赖的是16位x86实模式BIOS所提供的直接中断调用功能，而UEFI规范并不支持古老的16位x86实模式，所以自然也就不支持直接中断调用功能。因此"GRUB4DOS"所独有的"磁盘仿真"功能从根本上说，是不可能移植到UEFI平台的。由于BIOS正逐渐被UEFI取代，并且很多最新的主板已经不再兼容传统的BIOS模式，所以各种“一键Ghost”的适应性正在不断降低。

[0011] 5.Ghost原生英文界面，对中文用户不友好。

[0012] (二)FreeBSD操作系统的中文显示

首先简要介绍一下FreeBSD内核在控制台上显示字符的原理。

[0013] FreeBSD控制台有两种：文本模式控制台与图形模式控制台。

[0014] 文本模式控制台只能显示VGA显卡内嵌的256个字符，且屏幕大小固定为80x25个字符。当内核想要显示一个字符的时候，只需将字符的代码（1字节）和属性（1字节）写入字符缓冲区的相应地址即可。换句话说，在文本模式下，FreeBSD内核只能告诉显卡"以xx属性（例如黑底白字）显示代码为xx的字符（例如'A'）"，而不能决定字符在屏幕上的形状（例如是Fixed字体还是Courier字体），更不能显示VGA显卡内嵌的256个字符以外的其他字符。

[0015] 图形模式控制台在FreeBSD内核中是通过帧缓冲模块实现的。通过帧缓冲模块，FreeBSD内核可以直接操作帧缓冲设备（也就是屏幕的抽象），进而在屏幕上显示任意形状的图像。换句话说，在图形模式下，屏幕被抽象为一段内存（帧缓冲区），FreeBSD内核通过控制帧缓冲区的内容，即可精确的控制屏幕上的每一个像素。因此，为了在图形模式控制台上显示字符，FreeBSD内核需要首先将字符转换为位图，然后再将位图写入帧缓冲区，即可将字符（位图）呈现于屏幕。

[0016] 为了将字符转化为位图，FreeBSD内核内嵌了点阵字体，而字体实际上是一个二维点阵的数组。具体说来，每个字符都是一个8x16像素的二维点阵，有笔画的像素对应的bit为"1"，无笔画的像素对应的bit为"0"，这样就将字符（也就是位图）数字化了。

[0017] 显然，想要显示复杂的中文，只能使用图形模式控制台。

[0018] 然而由于FreeBSD内核的图形模式控制台存在设计上的缺陷（在具体实施方式部分有详细的说明），现实的实际情况是，FreeBSD操作系统的内核并不支持直接在控制台上显示中文。如果想要在控制台上显示中文，必须额外使用中文外挂程序，例如zhcon或fbterm。但使用外挂程序有如下缺点：

1. 需要额外的用户空间程序，且依赖关系复杂，不适合用于嵌入式系统。嵌入式系统希望软件结构越简单越好，最好是除操作系统内核外，再外加一个用户空间程序即可。但是为了显示中文，不但需要额外加上中文外挂程序，还需要加上许多外挂程序运行时所需要的函数库以及中文字体。这显然增加了系统的复杂性。

[0019] 2. 可能会与其他用户空间程序冲突，兼容性不佳。因为中文外挂程序需要占用帧缓冲设备，因此会与同样使用帧缓冲设备的其他程序（例如 w3m）发生冲突，从而造成兼容性故障。

[0020] 要想从根本上解决上述两个缺点，必须将中文的显示支持移入到内核空间中。

发明内容

[0021] 针对上述技术中存在的不足之处，本发明提供一种结构简单、操作方便的备份与还原 Windows 操作系统的方法。

[0022] 为了达到上述目的，本发明一种备份与还原 Windows 操作系统的方法，包括以下控制步骤：

S1、FreeBSD 内核中文化：向 FreeBSD 内核中添加中文字体，并修改 FreeBSD 内核中的字体获取函数，以实现内核级的中文显示支持；

其中，S1 的 FreeBSD 内核中文化处理具体包括以下步骤：

S11、嵌入中文字体：向内核中添加中文点阵字体，将一个汉字劈为左右两个 8×16 像素的二维点阵，从而让每半个汉字等价于一个西文字符；

S12、分配双倍内存：保持内核中原有的所有数据结构与变量定义不变，特别是用于保存字符代码及字符属性的变量的定义不变，但是使用动态内存分配函数为其额外多分配一倍的动态内存，并将多分配出来的内存完全用于容纳中文字符的代码；

S13、设置汉字标识符：首先将代码为 0x00 与 0xff 的两个字符统一替换为 0x20，然后不再将 0x00 与 0xff 视为普通的字符代码，而是将其处理为汉字左右部分的标识符；

S14、修改字体获取函数：修改 FreeBSD 内核中原有获得字体数据的函数，添加一个判断 " 字符代码 " 是否为 0x00 或 0xff 的逻辑，以正确获取中文的字符代码，然后到已在 S11 步骤嵌入内核的中文字体中提取相应的字体数据；最后，直接调用原有的字形绘制函数，即可将汉字显示在屏幕上。

[0023] 其中，S11 的中文字体制作步骤如下：

S111、选取任意一款等宽中文字体，要求其中每个半角字符的宽度严格等于全角字符的一半；

S112、对每个代码小于 256 的半角字符按照 8x16 点阵进行采样，将点阵中有笔画的点标记为 "1"，没有笔画的点标记为 "0"，这样每个半角字符就转化成了一个 8x16 的二维矩阵；

S113、对每个代码大于 255 的全角字符按照 16x16 点阵进行采样，并将采样结果从中间劈为左右两半，得到左右两个 8x16 点阵，同样将点阵中有笔画的点标记为 "1"，没有笔画的点标记为 "0"，这样每个全角字符就转化成了左右两个 8x16 的二维矩阵；

S114、将每个 8x16 的二维矩阵的每一行的 8 个 bit 转化为一个 16 进制数，这样每个矩阵就会得到 16 个 0x00 ～ 0xFF 之间的无符号单字节整数，这样就完成了字符的数字化；

S115、将每个字符按照其字符代码的顺序，依次简单的连接起来，封装到字体数组中即可作为嵌入内核的字体使用。

[0024] S2、系统构建：将同时支持 BIOS 与 UEFI 规范的 GRUB2 引导管理器、经过中文化修改的 FreeBSD 内核、实现备份与还原功能的用户空间程序、以及保存备份文件的储存空间组合在一起，形成该备份与还原系统；

S3、启动并挂载文件系统：通过本机硬盘、外置存储、PXE 网络三种方式之一，启动该备份与还原系统，并根据启动方式的不同，分别挂载用于存放备份文件的 ZFS 或 NFS 文件系统；

S4、计算序列号：通过序列号模块，计算出本机电脑的“主机序列号”及“硬盘序列号”，并将其组合为用于识别本机及内置硬盘的独一序列号；

其中，S4 的计算序列号的方法为 MD5 单向散列算法，具体算法如下：

主机序列号 =MD5（主板制造商名称 + 主板硬件序列号）；

硬盘序列号 =MD5（硬盘制造商名称 + 硬盘硬件序列号）；

S5、查找备份：查找是否存在与本机匹配的“主机序列号 -*.mbr”的文件，若存在，则显示“备份”与“还原”两个选项供用户选择，否则只显示一个单独的“备份”选项；

S6、用户选择：根据用户的选择，进入“备份”流程，以备份本机的磁盘分区表以及所有 Windows 系统分区；或进入“还原”流程，以从备份文件中还原本机的磁盘分区表以及所有 Windows 系统分区。

[0025] 其中，S6 中所述“备份”流程具体包括下述步骤：

S611、查找系统分区：通过 Windows 系统分区查找模块查找 Windows 系统所在的硬盘及分区；

S612、备份磁盘分区表：通过磁盘分区表备份模块将每个 Windows 系统所在硬盘的分区表与引导区备份为对应的“主机序列号 - 硬盘序列号 .mbr”文件；

S613、磁盘块清零：通过磁盘清理模块将每个 Windows 系统分区上的空闲磁盘块清零；

S614、分区压缩：通过分区压缩模块将每个 Windows 系统分区压缩为对应的“主机序列号 - 硬盘序列号 - 分区号 .xz”文件。

[0026] 其中，S613 中磁盘块清零的具体做法为：挂载指定的磁盘分区，然后在该分区上创建一个临时文件，接着向这个临时文件中持续写入无限长的 "00000....." 二进制零串，直到因为磁盘满而出错，最后再删除这个临时文件。

[0027] 其中，S614 中分区压缩模块的具体步骤为：

S6141、载入开源的 liblzma 库，其默认使用单线程压缩；

S6142、检测 CPU 的总核数，若大于 1 则开启 liblzma 库的多线程压缩功能，并且将线程数设为总核数；

S6143、打开指定的磁盘分区所对应的设备文件，将磁盘分区视为一串 "0" 与 "1" 组成的字节流，并将其作为 liblzma 库的压缩输入，同时将 liblzma 库的压缩输出保存到对应的“主机序列号 - 硬盘序列号 - 分区号 .xz”文件中，压缩完成后，关闭打开的设备文件与输出文件。

[0028] 其中，S6 中所述“还原”流程具体包括下述步骤：

S621、还原磁盘分区表：通过磁盘分区表还原模块将每个匹配的“主机序列号 - 硬盘序

列号.mbr”文件还原到对应的本机硬盘上；

S622、重读分区表：重新读取所有硬盘的分区表；

S623、分区解压：通过分区解压模块将每个匹配的“主机序列号-硬盘序列号-分区号.xz”文件解压到对应硬盘的对应分区上。

[0029] 其中，S623中分区解压模块的具体步骤为：

S6231、载入开源的liblzma库，其默认使用单线程解压；

S6232、检测CPU的总核数，若大于1则开启liblzma库的多线程解压功能，并且将线程数设为总核数；

S6233、打开指定的“主机序列号-硬盘序列号-分区号.xz”文件，并将其作为liblzma库的解压输入，同时打开对应的磁盘分区所对应的设备文件，并将其作为liblzma库的解压输出；

S6234、解压完成后，关闭打开的备份文件与设备文件。

[0030] 本实用新型的有益效果是：

与现有技术相比，本发明的备份与还原Windows操作系统的方法具有以下优势：

1. 安全性更高。因为本发明的安装与运行都不依赖于Windows系统，而是依赖于一个独立的FreeBSD系统，所以本发明的自身组件“GRUB2引导管理器+FreeBSD内核+用户空间程序”以及备份文件全部存储在Windows无法识别的ZFS文件系统或NFS文件系统上，从而彻底杜绝了备份文件与本发明自身在Windows环境下被误删除或者被恶意程序破坏的风险；

2. 高度智能化，适应各种不同的需求场景，且用户界面非常简洁。由于本发明专门针对Windows的备份与还原进行设计，因此尽量使用自动检测与智能匹配技术，用户无需设置任何参数，也不必担心备份文件的匹配问题。

[0031] 3. 支持最新的UEFI技术规范。由于FreeBSD内核与GRUB2引导管理器都支持最新的UEFI技术规范，同时也支持传统的BIOS技术规范，因此本发明可以在纯UEFI平台上工作。

[0032] 4. 彻底的中文界面。通过改造FreeBSD内核，使其可以直接在控制台上显示中文，从而实现了FreeBSD内核级的中文显示支持，彻底解决了在FreeBSD控制台上显示中文需要依靠外挂程序的麻烦。

附图说明

[0033] 图1为本发明备份与还原Windows操作系统的方法的流程示意图。

具体实施方式

[0034] 为了更清楚地表述本发明，下面结合附图对本发明作进一步地描述。

[0035] 参阅图1，本发明一种备份与还原Windows操作系统的方法，包括以下步骤：

S1、FreeBSD内核中文化：向FreeBSD内核中添加中文字体，并修改FreeBSD内核中的字体获取函数，以实现内核级的中文显示支持；

S2、系统构建：将同时支持BIOS与UEFI规范的GRUB2引导管理器、经过中文化修改的FreeBSD内核、实现备份与还原功能的用户空间程序、以及保存备份文件的储存空间组合在

一起，形成该备份与还原系统；

S3、启动并挂载文件系统：通过本机硬盘、外置存储、PXE 网络三种方式之一，启动该备份与还原系统，并根据启动方式的不同，分别挂载用于存放备份文件的 ZFS 或 NFS 文件系统；

S4、计算序列号：通过序列号模块，计算出本机电脑的“主机序列号”及“硬盘序列号”，并将其组合为用于识别本机及内置硬盘的独一序列号；

S5、查找备份：查找是否存在与本机匹配的“主机序列号 -*.mbr”的文件，若存在，则显示“备份”与“还原”两个选项供用户选择，否则只显示一个单独的“备份”选项；

S6、用户选择：根据用户的选择，进入“备份”流程，以备份本机的磁盘分区表以及所有 Windows 系统分区；或进入“还原”流程，以从备份文件中还原本机的磁盘分区表以及所有 Windows 系统分区。

[0036] 为了更好的帮助理解下文对 S1 中“FreeBSD 内核中文化”的具体过程的描述，首先简要介绍一下现有 FreeBSD 内核的图形模式控制台的设计及其缺陷：

为了和 VGA 文本模式兼容，FreeBSD 内核将图形模式控制台的字符缓冲区（后文以 "strbuf" 表示）设计为每字符 2 字节大小（1 字节代码 +1 字节属性），其格式与文本模式时的字符缓冲区格式完全相同。但这样一来，图形模式的字符缓冲区就变得和文本模式的字符缓冲区一样，最多只能表示 256 个字符了。

[0037] 针对上述设计缺陷，常规的改进方案一般是修改 "strbuf" 的定义，将原有的 "unsigned short *" 修改为 "unsigned int *"，这样字符缓冲区的大小将变为每字符 4 字节（2 字节代码 +2 字节属性），从而可以容纳 65536 个字符。但是 FreeBSD 内核中大量的代码都是基于 "strbuf" 是 "unsigned short *" 这样的假定。因此，一旦修改了 "strbuf" 的定义，就需要对内核中大量晦涩难懂的代码进行修改。不但如此，这样修改之后，还需要专门针对汉字再编写一套字形绘制函数，因为每个汉字都是一个 16x16 像素的二维点阵，不同于西文字符 8x16 像素的二维点阵，无法直接使用原有的字形绘制函数。

[0038] 为了尽可能少修改原有的代码，需要转变常规的思维方式。本发明利用 "strbuf" 本质上是一个指针，其所指向的内存是通过 kmalloc() 函数动态分配的特点。以此为突破口，设计了本发明的改进方案。

[0039] 在本实施例中，S1 中“FreeBSD 内核中文化”的具体过程为：

S11、向内核中添加中文字体，不再将汉字作为单个字符处理，而是将每个汉字劈为左右两半。也就是将每个汉字视为左右两个 8x16 像素的二维点阵，每半个汉字等价于一个西文字符。具体步骤如下：

S111、选取任意一款等宽中文字体（例如开源的文泉驿 Unibit 字体），要求其中每个半角字符的宽度严格等于全角字符的一半。

[0040] S112、对每个半角字符（代码小于 256）按照 8x16 点阵进行采样，将点阵中有笔画的点标记为 "1"，没有笔画标记的点标记为 "0"。这样每个半角字符就转化成了一个 8x16 的二维矩阵。

[0041] S113、对每个全角字符（代码大于 255）按照 16x16 点阵进行采样，并将采样结果从中间劈为左右两半，得到左右两个 8x16 点阵，同样将点阵中有笔画的点标记为 "1"，没有笔画标记的点标记为 "0"。这样每个全角字符就转化成了左右两个 8x16 的二维矩阵。

[0042] S114、将每个 8x16 的二维矩阵的每一行的 8 个 bit（正好一个字节）转化为一个 16 进制数，这样每个矩阵就会得到 16 个 0x00 ～ 0xFF 之间的无符号单字节整数。这样就完成了字符（也就是位图）的数字化。

[0043] S115、将每个字符按照其代码顺序，依次简单的连接起来，封装到字体数组中即可作为嵌入内核的字体使用。

[0044] 通过将每个全角字符（汉字）劈成左右两个半角字符，在逻辑上取消了 " 全角字符 " 的概念，这样就为接下来在代码中将所有字符统一视为 " 半角字符 " 提供了字体支持。

[0045] S12、保持内核中原有的所有数据结构与变量定义不变，特别是用于保存字符代码及字符属性的变量 "strbuf" 的定义不变，但是使用动态内存分配函数为其额外多分配一倍的动态内存（后文将这部分多出来的部分称为 "unibuf"），用伪代码表示就是将原有的

strbuf=kmalloc(strbuf_length);

修改为

strbuf=kmalloc(strbuf_length*2);

并将多分配出来的空间完全用于容纳中文的字符代码（不含属性）。这样可以避免破坏内核原来的数据结构，进而避免了对内核代码的大规模修改。

[0046] 举例来说，假设要在屏幕上显示黑底白字的 "ABC" 三个字符，按照 FreeBSD 原来的做法，只需要给 "strbuf" 分配 6 字节内存，并在其中写入 "0x410f,0x420f,0x430f" 即可。但是按照修改之后的做法，则需要分配 12 字节内存，其中前 6 字节 ("strbuf") 与原来的方案完全相同 (1 字节代码 +1 字节属性)，因此可以完全兼容原有的内核代码。而多出来的后 6 字节 ("unibuf")，则用于存储以 2 字节表示的字符代码，也就是 "0x0041,0x0042,0x0043"（虽然这部分内容在本例全英文字符的情况下并无实际用途）。

[0047] S13、考虑到 0x00 与 0xff 以及 0x20 都显示为一个空格（也就是一个 8x16 的空白点阵），因此牺牲代码为 0x00 与 0xff 的两个字符（用 0x20 代替），不再将这两个代码视为字符代码，而是将其视为汉字的左右部分的标识符。具体说来就是，根据不同的字符代码分别按如下规则进行处理：

S131、如果代码等于 0x00 或 0xff ，则直接将其改写为 0x20 ；

S132、对于小于 0xff 的代码（西文），在向 "strbuf" 写入字符代码的同时，再写入一份到 "unibuf" 中。而对于大于 0xff 的代码（中文），首先向 "strbuf" 中固定写入 0x00 和 0xff 两个代码（因为一个中文等于两个英文），然后再写入两份到 "unibuf" 中。其中，"0x00" 表示该字符的代码要到 "unibuf" 中提取，并需要绘制左半部分；而 "0xff" 则表示该字符的代码要到 unibuf 中提取，并需要绘制右半部分。

[0048] 举例来说，如果要在屏幕上显示黑底白字的 " 牛 B"，那么 "strbuf" 中的内容就是 "0x000f,0xff0f,0x420f"，而 "unibuf" 的内容则是 "0x725b,0x725b,0x0042"。

[0049] 对于 "strbuf" 部分来说，"0x00","0xff" 分别表示 " 牛 " 的左右半边，"0x42" 则是字母 "B" 的字符代码；而 "0x0f" 则是属性，表示黑底白字。因为一个汉字为两倍的英文字母宽度，所以在屏幕文字缓冲区上也必须占用两个字符的位置。又因为必须有一种机制能知道应该绘制左半部分和右半部分，所以这里使用的就是 0x00 和 0xff 两个魔数。对于 "unibuf" 部分来说，"0x725b" 是汉字 " 牛 " 的字符代码 (2 字节)，"0x0042" 是字母 "B" 的字符代码 (2 字节)。

[0050] S14、修改获得字体数据的代码。因为已经在字体中将每个汉字拆为了左右两半，所以不需要修改原有仅适用于 8x16 点阵的字形绘制函数。只需要修改获得字体数据的代码，添加一个判断 " 字符代码 " 是否为 0x00 或 0xff 的逻辑，具体如下：

S141、如果为假（不等于 0x00 或 0xff），则直接使用此 " 字符代码 " 作为索引，到字体数组中去提取对应的二维点阵 (8x16)。

[0051] S142、如果为真（等于 0x00 或 0xff），则从 "unibuf" 提取真正的 " 字符代码 "，并依据是 0x00 还是 0xff，到字体数组中去提取对应的左或右半部分的二维点阵 (8x16)。取得字体数据之后，继续按照原有的流程，调用原有的字形绘制函数，即可将汉字显示在屏幕上。

[0052] 在本实施例中，S4 的计算序列号步骤中的序列号模块是根据电脑的硬件信息计算出“主机序列号”与“硬盘序列号”。而这两个序列号是实现备份文件与目标电脑之间自动匹配的关键。每台电脑都对应着两种备份文件，一种是分区表与引导区的备份文件 (.mbr)，另一种是 Windows 系统分区的备份文件 (.xz)。这两种备份文件的文件名格式如下：

主机序列号 - 硬盘序列号 .mbr ；

主机序列号 - 硬盘序列号 - 分区号 .xz ；

在本实施例中，“主机序列号”与“硬盘序列号”的计算方法如下：

主机序列号 =MD5（主板制造商名称 + 主板硬件序列号）；

硬盘序列号 =MD5（硬盘制造商名称 + 硬盘硬件序列号）；

因为所有主板与硬盘制造商都会给他们的每件产品赋予一个唯一的硬件序列号，同时，操作系统也可以方便的通过固件 (BIOS/UEFI) 获取这些信息，所以使用“造商名称 + 硬件序列号”的组合，可以可靠的区分每块主板与硬盘。另一方面，MD5 也是一种可靠的单向散列算法，把两者相结合，即可算出长度固定的唯一序列号，从而可靠的实现备份文件与目标电脑之间的自动匹配。需要注意的是，如果目标电脑安装有多块硬盘，则会计算出多个“硬盘序列号”，但是每台电脑都只有一块主板，因此只会计算出一个“主机序列号”。

[0053] 在本实施例中，S6 的“备份”流程具体包括下述步骤：

S611、查找系统分区 ：通过 Windows 系统分区查找模块查找 Windows 系统所在的硬盘及分区 ；

S612、备份磁盘分区表 ：通过磁盘分区表备份模块将每个 Windows 系统所在硬盘的分区表与引导区备份为对应的“主机序列号 - 硬盘序列号 .mbr”文件 ；

S613、磁盘块清零 ：通过磁盘清理模块将每个 Windows 系统分区上的空闲磁盘块清零 ；

S614、分区压缩 ：通过分区压缩模块将每个 Windows 系统分区压缩为对应的“主机序列号 - 硬盘序列号 - 分区号 .xz”文件。

[0054] 在本实施例中，S613 磁盘清理模块的作用是将指定的磁盘分区上的所有空闲磁盘块清零。目的是降低压缩文件的大小，节约备份文件所需要的存储空间，并提升压缩速度。在 Windows 操作系统中，文件被删除并清空回收站之后，只是释放了已删除文件所占用的磁盘块（也就是将这些磁盘块重新标记为可用状态），但是磁盘块中的内容依然存在。随着文件的不断写入与删除，最终，大量的磁盘块中都将充斥着没有意义的垃圾字节。注意，这里的“磁盘块”是 Unix 术语，等价于 Windows 术语“族”的意思，也就是磁盘空间的最小使用单位，大小通常是 4KB。而本发明的“分区压缩模块”是基于字节流的压缩，也就是说，本

发明的“分区压缩模块”将每个磁盘分区视为一串"0"与"1"组成的字节流，而不关心这串字节流的实际含义，因此也就无法感知字节流之上的扇区、磁盘块（族）、文件系统、文件分配表、文件属性这样的高级概念。所以，前面所说的垃圾字节也会被无差别的对待，从而无意义的增加压缩文件的大小。本模块就是为了解决这个问题而设置的。

[0055] 具体做法是挂载指定的磁盘分区（若是NTFS分区则需要借助开源的NTFS-3G工具，若是FAT分区则FreeBSD本身就支持），然后在该分区上创建一个临时文件，接着向这个临时文件中持续写入无限长的"00000000....."零字节串，直到因为磁盘满而出错，最后再删除这个临时文件。这样所有空闲的磁盘块就都被清零了。经过这样的清零处理之后，所有空闲的磁盘块就都变成了统一的"00000000....."零字节串，这样的字节流可以非常快的压缩，并且经过压缩之后几乎不占用存储空间，从而大大降低了压缩文件的大小，同时也大大提升了压缩速度。

[0056] 在本实施例中，S614分区压缩模块的作用是将指定的磁盘分区备份（压缩）为对应的“主机序列号-硬盘序列号-分区号.xz”文件。按照直观且简单的想法，备份Windows系统，可以像使用WinRAR或7z那样，直接把系统盘（也就是C盘）上的所有文件打成一个压缩包，然后在还原的时候直接解压回去就好了。但实际上这个想法是行不通的，尤其是对于主流的NTFS分区更加行不通。因为各种压缩工具（包括Windows平台上的WinRAR/7z/WinZIP以及FreeBSD平台上的Bzip2/Xz/Gzip等等）都只能将文件的内容压缩保存(WinRAR可以保存少部分文件属性)，但是在Windows系统盘（也就是C盘）上除了简单的文件内容之外，还有可能存在：

1. 分区引导信息，位于分区的开头，具体占用的扇区数量取决于分区格式(FAT还是NTFS)；

2. 与文件关联的各种元数据。包括常规的：系统、隐藏、存档、只读、创建时间、修改时间、访问时间；以及高级的：索引、属主、属组、审核项目、权限与访问控制、权限与访问控制的继承关系、关联数据流……

3. 特殊的文件形式，例如：硬连接与软连接；

4. 某些文件可能是加密存放或者压缩存放的；

5. 磁盘限额信息。

[0057] 要想完全精确的备份与还原上述这些数据，是非常复杂的，甚至是不可能的，因为NTFS是一个封闭的文件系统，其文件系统的细节是不公开的。正是因为这个原因，本发明才使用了简单的、基于字节流的压缩技术，将磁盘分区视为一串"0"与"1"组成的字节流，而不关心这串字节流的实际含义。通过在更低的字节流层次上对磁盘分区进行压缩备份，完全避开了解析上述复杂数据的麻烦，同时又能做到完全精确的还原分区数据。事实上，Ghost程序也不是在文件层次对分区进行备份，而是在磁盘块层次（比本发明的字节流层次更高一些），原因应该也是相同的。

[0058] 就具体的压缩算法而言，本模块使用了2009年发明的LZMA2压缩算法，原因有三：

1. 开源，有现成的liblzma库可用，因此可以方便的进行二次开发。

[0059] 2. 压缩率更高，几乎是目前压缩率最高的算法，通常可比DEFLATE算法节省30%的空间。

[0060] 3. 支持多线程压缩，可以充分利用多核 CPU 的计算能力，成倍的提高压缩与解压速度。

[0061] 具体实施步骤也非常简单：

1. 载入开源的 liblzma 库，其默认使用单线程压缩。

[0062] 2. 检测 CPU 的总核数，若大于 1 则开启 liblzma 库的多线程压缩功能，并且将线程数设为总核数。

[0063] 3. 打开指定的磁盘分区所对应的设备文件（也就是将磁盘分区视为一串"0"与"1"组成的字节流），并将其作为 liblzma 库的压缩输入，同时将 liblzma 库的压缩输出保存到对应的“主机序列号 - 硬盘序列号 - 分区号 .xz”文件中。

[0064] 4. 压缩完成后，关闭打开的设备文件与输出文件。

[0065] 在本实施例中，S6 的“还原”流程具体包括下述步骤：

S621、还原磁盘分区表：通过磁盘分区表还原模块将每个匹配的“主机序列号 - 硬盘序列号 .mbr”文件还原到对应的本机硬盘上；

S622、重读分区表：重新读取所有硬盘的分区表；

S623、分区解压：通过分区解压模块将每个匹配的“主机序列号 - 硬盘序列号 - 分区号 .xz”文件解压到对应硬盘的对应分区上。

[0066] 在本实施例中，S623 中分区解压模块的具体步骤为：

S6231、载入开源的 liblzma 库，其默认使用单线程解压；

S6232、检测 CPU 的总核数，若大于 1 则开启 liblzma 库的多线程解压功能，并且将线程数设为总核数；

S6233、打开指定的“主机序列号 - 硬盘序列号 - 分区号 .xz”文件，并将其作为 liblzma 库的解压输入，同时打开对应的磁盘分区所对应的设备文件，并将其作为 liblzma 库的解压输出；

S6234、解压完成后，关闭打开的备份文件与设备文件。

[0067] 本发明的优势在于：

1. 安全性更高，因为本发明的安装与运行都不依赖于 Windows 系统，而是依赖于一个独立的 FreeBSD 系统，所以本发明的自身组件 (GRUB2 引导管理器 +FreeBSD 内核 + 用户空间程序）以及备份文件全部存储在 Windows 无法识别的 ZFS 文件系统或 NFS 文件系统（仅针对网络启动）上，从而彻底杜绝了备份文件与本发明自身在 Windows 环境下被误删除或者被恶意程序破坏的风险。

[0068] 2. 优秀的数据压缩技术，使用 2009 年发明的 LZMA2 数据压缩算法，不但可以获得比 deflate 算法更优秀的压缩率（可节省 30% 左右的存储空间），配合使用多线程技术，在多核 CPU 上还能成倍提升压缩与解压速度。

[0069] 3. 高度智能化，适应各种不同的需求场景，且用户界面非常简洁。由于本发明专门针对 Windows 的备份与还原进行设计，因此尽量使用自动检测与智能匹配技术，用户无需设置任何参数，也不必担心备份文件的匹配问题。利用 FreeBSD 与 GRUB2 的灵活性，可以从内置硬盘启动运行，也可以从外置存储设备（例如移动硬盘或者 U 盘）启动运行，还可以通过 PXE 从网络启动运行。相应的，备份文件可以存储在内置硬盘上，也可以存储在外置存储设备（例如移动硬盘或者 U 盘）上，还可以存储在网络硬盘上。从而很好的满足了个人用

户、小型企业、中大型企业的各种不同场景。而且无论以何种方式运行，其用户界面都完全相同。用户只需选择是要＂备份＂还是要＂还原＂即可，其他一切动作都是自动的。可谓简洁至极。

[0070] 4. 支持最新的UEFI技术规范，由于FreeBSD内核与GRUB2引导管理器都支持最新的UEFI技术规范（同时也支持传统的BIOS技术规范），因此本发明可以在纯UEFI平台上工作。

[0071] 5. 彻底的中文界面，通过改造FreeBSD内核，使其可以直接在控制台上显示中文，从而实现了FreeBSD内核级别的中文显示支持。彻底解决了在FreeBSD控制台上显示中文需要依靠外挂程序的麻烦。

[0072] 以上公开的仅为本发明的几个具体实施例，但是本发明并非局限于此，任何本领域的技术人员能思之的变化都应落入本发明的保护范围。

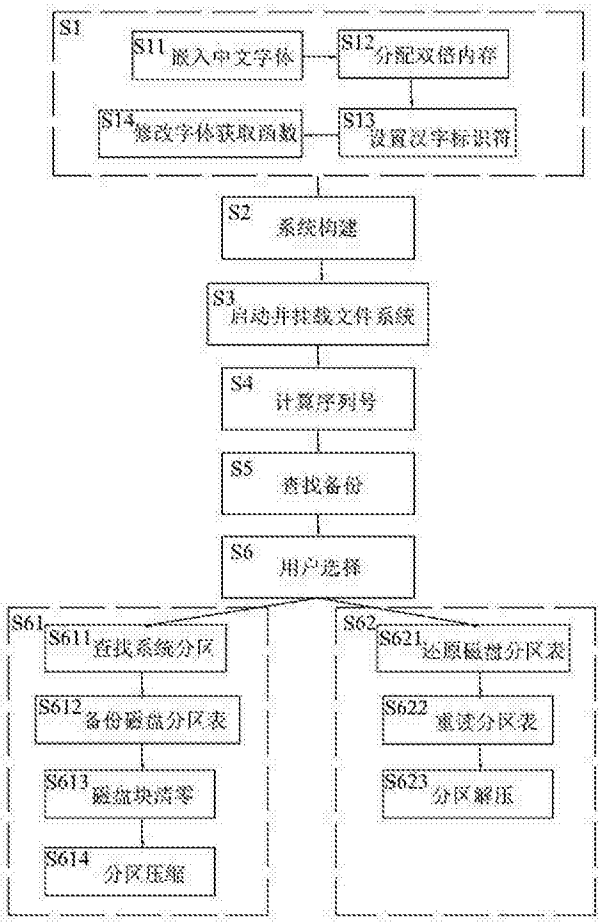

S1
S11 嵌入中文字体
S12 分配双倍内存
S14 修改字体获取函数
S13 设置汉字标识符
S2 系统构建
S3 启动并挂载文件系统
S4 计算序列号
S5 查找备份
S6 用户选择
S61
S611 查找系统分区
S612 备份磁盘分区表
S613 磁盘块清零
S614 分区压缩
S62
S621 还原磁盘分区表
S622 重读分区表
S623 分区解压

图 1