Linux内核多语言文件子系统的设计与实现*)

芮建武 谢 谦 吴 健 孙玉芳

(中国科学院软件研究所开放系统与中文信息处理中心 北京100080)

摘 要 操作系统的多语言支持是网络环境下计算机软件发展的必然结果。由于POSIX标准的国际化体系结构对多语言和分布式应用需求的支持有其局限性,导致遵循POSIX标准的Linux文件子系统在支持多语言文本时可能造成数据丢失。本文从多语言角度考察了Linux文件子系统,重新构造了一个能够支持Unicode编码的逻辑文件系统EXT2U,改进了文件子系统,同时提供了基于Unicode编码的系统调用接口。通过新文件系统与系统调用接口,为操作系统多语言处理提供了更好的基础。

关键词 文件子系统,多语言化,Unicode,EXT2U

Design and Implementation of Linux File System Supporting Multilingualism

RUI Jian-Wu XIE Qian WU Jian SUN Yu-Fang

(Open System & Chinese Information Processing Center, Institute of Software, Chinese Academy of Sciences, Beijing 100080)

Abstract Multilingualization of modern operating system is necessitated on the Internet. Since internationalization architecture complying to POSIX has less support for multilingual and distributed applications, Linux file system dirived from POSIX may lose data in storing some multilingual text. Based on Linux file system, a new File System called EXT2U is presented, which supports metadata encoded by UTF-16. Meanwhile, virtual filesystem switch in Linux kernel is improved and a set of system calls using a data type special to UTF-16 are proposed. by means of EXT2U and new system calls, the ability to process multilingual information can be achieved easier in Linux.

Keywords File system,Multilingualization,Unicode,EXT2U

1 问题的提出

1.1 多语言环境

现代操作系统应为用户提供一个多语言环境。多语言环境要求:(1)为用户提供多语言用户界面,除用户可以选择自己使用的语言外。2)为开发人员提供多语言编程界面,使用应用程序具有输入与输出多种语言文本的支持。3)提供多语言信息的存储能力。其中,系统对多语言信息的存储是最基础的系统支持,由文件系统来实现。

1.2 文件系统与编码方案

通常所说的文件系统可能包含以下的方面:(1)已格式化的存储介质,例如硬盘的一个已格式化分区;(2)存储在物理文件系统中的所有信息;(3)特定的文件组织格式,例如EXT2、FAT32等文件系统格式,可以把它看作是逻辑文件系统;(4)操作系统中对物理文件系统中的信息进行管理和操作的模块。

首先,不同的文件组织格式形成不同的逻辑文件系统,其实现是不相同的。

其次,文件系统中存储的信息也与文件子系统的实现有关。我们把存储在物理文件系统上的信息分为两个类别:1)用户实际存放的信息;2)标识某一信息的信息,称为元信息。作为文件系统来说,最关键的是那些标识信息的信息。至于文件系统中文件的内容根据用户的需要存储,因此与文件系统结构本身无特别的关联。元信息对文件系统的组织结构至关重要。例如,如果用户需要把某些信息存储到文件名为"Linux系统混合文本示例"的文件中,则文件系统将创建文件结构,把此文件结构的"名称"字段标记为"Linux系统混合文本示例"。当用户需要再次访问此文件时,文件系统会根据路径找到"Linux系统混合文本示例"文件的文件结构。查找过程中使用比较操作:把文件系统中的有关文件结构的"名称"字段信息与字符串"Linux系统混合文本示例"进行比较,如果找到匹配的文件结构记录,则此文件结构所指定的信息就是所要找的内容。可以看出,在进行字符串比较时,文件系统中文件结构的数据与要查找的字符串应该在编码上一致。

计算机发展历程出现过许多的编码方案,相互之间未必兼容。这些编码方案包括:英文的ASCII编码,欧洲国家的ISO 8859系列编码,中国使用基于GB 2312、GBK、GB 18030等标准的编码,还有HZ、Big5编码等。鉴于不同字符集编码之间的不兼容现象造成信息处理与共享的极大困难,国际标准化组织制定了ISO/IEC 10646标准[1]与Unicode联盟制定了Unicode标准[2],以统一的代码点结构收录了世界上大多数语言,二者相互对齐。这使得计算机软件实现多语言支持有了更好的基础。

可以看出,如果文件系统同时采用不同的编码方案,将造

*)基金项目:国家863计划软件重大专项《民族语言版本Linux操作系统及办公套件研发》(项目编号:2003AA1Z2110)和中国科学院知识创新工程方向性项目《基于Linux的跨平台藏文信息处理系统》(项目编号:KGCX2-SW-504)联合资助。芮建武 博士研究生,主要研究方向为操作系统与中文信息处理。谢 谦 博士研究生,主要研究方向为操作系统中与中文信息处理。吴 健 副研究员,主要研究方向为系统软件与中文信息处理。孙玉芳 研究员,博士生导师,主要研究方向为系统软件与中文信息处理。

成文件系统上信息的混乱。当我们使用编码方案 A 来表示字符串"Linux 系统混合文本示例"时，系统正确地找到了此文件；当使用编码方案 B 时，将无法找到此文件。文件系统本身提供的系统接口实际上隐含了这一事实：它使用编码方案 A 而不是编码方案 B。因此，对文件系统来说需要约定：使用某种编码方案来表示文件系统内部的元信息。

2 目前 Linux 文件子系统体系结构

POSIX 标准中把一个字符定义为"代表单个图形符号或控制代码的一个或多个字节的序列"，把一个字符串定义为"以值为零的字节(代表 NULL)结尾的一个连续字符序列或仅包含一个值为零的字节的空字符串"。同时，基于 POSIX 标准的具可移植性的编程接口中使用指向字符的指针(char *)作为参数来传递文件名和路径名等文件系统元信息。这样，实际上约定：在表示这些元信息的字符串编码中不应该存在包含值为零的字节，否则，信息将被从值为零的字节处截断。

这种约定与广泛使用的 ISO/IEC 2022标准[3]的编码体系一致。这种编码体系以 ASCII 编码为基础，扩充后支持世界上其他大多数语言。基于此编码体系的所有编码方案中，其代表的字符串编码的中间都不会包含值为零的字节。Linux 操作系统完全遵循 POSIX 标准，在文件子系统提供的接口中，使用字符串类型(char *)来传递有关文件系统元信息的参数。因此 Linux 文件系统可支持那些与 ISO/IEC 2022标准相兼容的字符编码方案。采用这样的编码方案有以下的缺点：(1)它们仅覆盖某些字符集，如 GB 2312等。(2)同时支持多语言时需要进行编码转换。编码之间的转换复杂且系统开销很大。克服这些缺点可使用 Unicode 标准所指定的编码形式，它可覆盖世界上所有语言并且通常无需编码转换。

Linux 操作系统中本地环境(Local Environment，简写为 Locale)属于国际化与本地化机制的基础部分。用户使用计算机时，总是基于一个本地环境，所有的输入与默认的输出都由 Locale 环境所定义的编码字符集为基础，我们称基于 Locale 定义的字符集所使用的编码为本地编码。然而，用户可能会改变应用程序所使用的本地环境。例如，用户在应用程序 A 中使用本地环境为 zh_CN.GB2312(简体中文，GB 2312编码方案)，他创建了一些文件或目录。用户也可能基于本地环境 zh_TW.Big5(繁体中文，Big5编码方案)使用应用程序 B 进行了一些有关文件的操作。所有这些信息都会在文件系统中保存下来。其结果是：文件系统中既有 GB 2312编码，又有 Big5编码的信息。事实上，它可以存在任何与 ISO/IEC 2022标准一致的编码方案所指定的信息。在这样的情况下，文件系统中存储的信息呈现混乱现象，使用我们难以确认其中信息的具体内容。文件所有者可能会知道这一串数据采用什么编码，代表什么样的内容，但是，如果遗忘了其编码方案，也许就难以正确理解这些信息了。

Linux 文件子系统分层结构使用了一个抽象层——虚拟文件系统切换(Virtual Filesystem Switch，VFS)[4,5]。VFS 屏蔽了所有逻辑文件系统的特殊属性，提供了一个统一的系统接口。用户使用统一的系统接口对文件子系统进行操作。Linux 文件子系统把文件分为普通文件、目录文件、符号链接文件、设备文件、管道和套接字(Socket)等，VFS 对不同类型的文件根据其各自特性来实施相应的操作。当用户对文件进行操作时，文件子系统首先根据用户指定的文件位置，确定文件所属的逻辑文件系统；其次建立相应的 VFS 对象，把逻辑文件系统对应的函数入口点加入到这些 VFS 对象的操作集合中。这些函数入口点实际上对应了相应逻辑文件系统的接口及其驱动程序。这样，用户就可透明地实现对文件系统相应信息的存取。Linux 文件子系统支持 VFAT、EXT2等多种逻辑文件系统。通过对 Linux 内核文件子系统源码进行分析，我们得出如图1(a)所示的 Linux 文件子系统结构图。

3 系统设计

3.1 系统目标

通过对 Linux 内核[6]进行分析后可以看出：Linux 内核文件子系统处理本地编码的文件系统元数据。这种策略导致文件子系统所存储的信息混乱。因此，支持多语言的文件子系统需要：(1)明确指定文件子系统元数据使用的编码方案，根据前面的分析，使用 Unnicode 编码才能无二义地表示用户信息；(2)提供基于相应编码方案的系统调用接口。

Unicode 标准[2]定义了如下的标准编码形式：UTF-8，UTF-16，UTF-32。这些编码形式各有优缺点：(1)UTF-8编码形式。优点：(a)可实现对 POSIX 类应用程序编码接口的兼容；(b)动态扩充 Unicode 标准所支持的语言文字；(c)存储 ASCII 字符集的字符时 UTF-8编码不需要额外的空间。缺点：(a)尽管 POSIX 类编程函数在语法上仍适用于 UTF-8编码，然而语义上并不相同。这同样会造成信息存储的混乱，因此在某些应用中使用 UTF-8编码来支持多语言会付出较大代价；(b)与操作系统其他部分兼容经常需要转换编码，影响系统效率；(c)与上层系统库(如 Glibc、Qt、ICU 等)编码方案的不一致性将潜在性地影响系统可扩充性；(d)汉字字符增加50%的空间。(2)UTF-16编码形式。特点：(a)存储 ASCII 字符集的字符时增加了50%的空间，但是不增加存储汉字的空间；(b)它属于变长编码方案；(c)它与现存的类 POSIX 操作系统有一定程度的不兼容。(3)UTF-32编码形式。特点：(a)覆盖所有的 ISO/IEC 10646的编码空间，在可扩充性方面没有任何障碍；(b)对每个字符使用4个字节，导致耗费更多的内存与外存空间。

根据 POSIX 可移植性标准，使用 UTF-16编码会导致编码接口语法的不兼容问题。如果我们选择 UTF-8编码，会导致语义不兼容问题。本文经过分析比较后认为：语义的不兼容问题会造成系统维护或可移植性方面的更大代价。因此，在全面考察了目前各种软件系统使用 Unicode 编码来支持多语言的现状及对各编码形式进行分析之后，我们选择 UTF-16编码形式。

3.2 改进的文件子系统体系结构

通过改进现有的文件子系统，可得出支持多语言的文件子系统结构图，如图1(b)所示。图中细线双箭头表示两模块之间交换数据信息时使用本地编码；粗双箭头表示使用 Unicode 编码；虚线双箭头表示模块之间存在本地编码与 Unicode 编码之间的编码转换。

从图1(b)可以看出，支持多语言的文件子系统包括以下部分：(1)逻辑文件系统 EXT2U；(2)支持 Unicode 编码的虚拟文件系统及基于 Unicode 编码的系统调用接口；(3)系统完整性设计，包括对 VFAT、EXT2及 PROC 逻辑文件系统的支持。

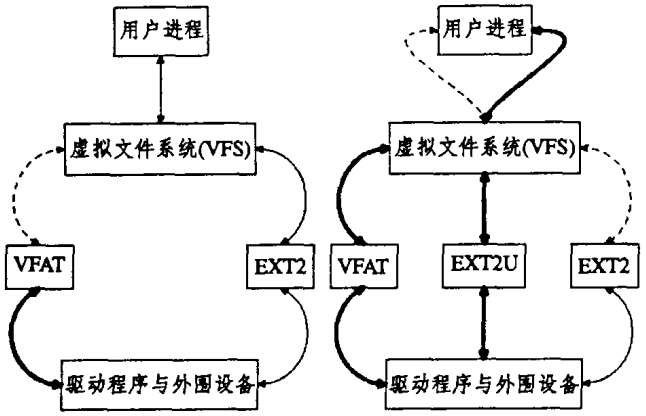

图1 Linux 文件子系统结构图

3.3 逻辑文件系统 EXT2U

本文以 EXT2文件系统为基础，建立一种新的逻辑文件系统 EXT2U(Second Extended File System for Unicode)。EXT2U 与 EXT2有不同的操作语义。所谓操作语义表现在两个方面：(1)EXT2U 内部的元数据将以 Unicode 编码形式存储，而不使用本地编码；(2)EXT2、EXT2U 与虚拟文件系统的接口不同。由于虚拟文件系统内部使用 Unicode 编码，因而它与 EXT2、EXT2U 之间的接口不一致，并且存在编码转换。除此之外，EXT2U 在存储结构上与 EXT2完全相同。

3.4 虚拟文件系统

支持多语言的文件子系统内部默认使用 Unicode 编码。作为逻辑文件系统的抽象层，虚拟文件系统也基于 Unicode 编码。它主要实现：(1)提供各种逻辑文件系统的抽象，主要包括目录项结构(struct dentry)、i 结点(i-node)、文件结构(struct file)等。其中用于表示文件系统元数据的数据结构应该使用支持 Unicode 编码的数据类型。(2)提供文件子系统的基于 Unicode 编码的系统调用(system call)。这些系统调用在接口上与原先的系统调用不同，对于文件系统的元数据也应该使用支持 Unicode 编码的数据类型。这些系统调用包括：sys_wmkdir()，sys_wchdir()，sys_wrmdir()，sys_wcreat()，sys_wmknod()，sys_wgetcwd()，sys_wopen()，sys_wlink()，sys_wunlink()，sys_wrename()，sys_wsymlink()，sys_wreadlink()，sys_wchmod()，sys_wchown16()，sys_wlchown()，sys_wlchown16()，sys_wlchown()，sys_wstat()，sys_wlstat()，sys_wstatfs()，sys_wnewstat()，sys_wnewlstat()，sys_wstat64()，sys_wlstat64()，sys_wtruncate()，sys_wtrncate64()，sys_wutime()，sys_wutimes()，sys_wmount()，sys_wumount()，sys_waccess()，sys_wacct()，sys_wuselib()，sys_wswapon()，sys_wswapoff()，sys_wquotactl()，sys_wchroot()，sys_wpivot_root()

3.5 系统完整性设计

为了与现有系统兼容，新的文件子系统提供两类系统调用接口：(1)新增接口使用基于 Unicode 编码的接口参数与语义；(2)原有的接口保持原有参数与语义，以兼容原来的系统。

在 Linux 文件子系统内部，为了使虚拟文件系统与逻辑文件系统 EXT2、VFAT 和 EXT2U 完整结合在一起，我们需要改进以下部分：1)兼容 EXT2逻辑文件系统；2)支持 VFAT 逻辑文件系统；3)基于 Unicode 编码的 PROC 文件系统。

事实上，兼容原先的 EXT2逻辑文件系统仅仅是一种临时策略。最终可以完全抛弃 EXT2逻辑文件系统，直接使用基于 Unicode 编码的 EXT2U 文件系统。

3.5.1 *编码转换* 图中虚线双箭头表示模块之间存在本地编码与 Unicode 编码之间的编码转换。这通常由 Linux 内核中的本地语言支持(Native Language Support，NLS)子系统来完成。NLS 子系统实现了基于各种代码页(codepage)的字符编码与 Unicode 编码的相互转换。代码页实际上指定一种字符集及其编码方案，通常支持英语和其他某种语言，因此，不同语言其代码页不同。当指定一个代码页后，某字符串可被明确地编码。

Linux 内核在系统初始化时形成一个单向循环链表 tables 来管理它所支持的所有代码页。当系统动态加载代码页模块时，系统将为此代码页分配链表节点，并提供了相应的编码转换函数。这样，通过它们就可以实现基于代码页的各种字符编码与 UNICODE 编码之间的相互转换。NLS 子系统内部提供的编码转换函数：(1)uni2char()把 Unicode 编码字符转换为所属代码页的字符；(2)char2uni()把所属代码页的字符转换为 Unicode 编码字符。

3.5.2 *兼容 EXT2逻辑文件系统* 考虑与 EXT2兼容仅是一个完整性设计方案，实际上如果 EXT2U 完成后，支持 EXT2就没有价值了。

从图1(b)中可以看出：虚拟文件系统使用 Unicode 编码的数据处理后，EXT2逻辑文件系统的处理函数语义上与文件子系统不一致，因为所用的编码方式并不相同。使用 NLS 子系统可以实现二者的相互转换。

3.5.3 *支持 VFAT 逻辑文件系统* FAT32/FAT16分区格式的文件系统使用 Unicode 编码。在原来的 Linux 内核文件子系统中，VFAT 逻辑文件系统使用 NLS 子系统来实现 Unicode 编码与本地编码之间的转换。在完成虚拟文件系统的 Unicode 编码技术后，无需再进行编码转换，二者自然地结合在一起。

3.5.4 *基于 Unicode 编码的 PROC 文件系统* Linux 内核中 PROC 文件系统是一个特殊的文件系统，它并不物理地存在于磁盘上，但在系统初始化时动态地建立起来。使用 PROC 文件系统的目的在于：提供一个除系统调用之外的接口，让用户可以在内核外部读写内核映像和内核中各个数据结构以及堆栈，从而可查看系统运行状况并可修改某些系统配置信息。PROC 文件系统实际是内核的一部分，用户可通过下述方式使用它：1)与其他文件系统一样，使用系统调用可存取它的信息。2)PROC 文件系统提供了特殊的函数让外部模块使用它。3)用户使用外部命令直接存取文件，从而修改了内核中的某些数据信息。与 EXT2一样，需要把 PROC 与虚拟文件系统完整统一起来，因此需要修改二者之间的接口。

4 系统实现

系统实现时，首先建立新的逻辑文件系统 EXT2U；然后改造虚拟文件系统，实现基于 Unicode 编码的系统调用接口；最后完成对 EXT2和 VFAT 逻辑文件系统的支持。

4.1 EXT2U 的建立

EXT2U 在内部结构和操作算法上与 EXT2有了改进。相对虚拟文件系统来说，EXT2U 写入或读出 Unicode 编码的文件系统元数据，而 EXT2读写基于本地编码的元数据。其中涉及的数据结构和操作主要有(细节描述及用法见文[5,6])：

(下转第248页)

报,2002,32(1):6～10
2 谭顶良.学习风格论[M].江苏教育出版社,1999
3 张向葵,关文信.学习策略的理论和操作[M],吉林大学出版社,2002
4 Shen R,Wang Q. A Distance-Learning Model Based on Web Mining [J]. Shanghai Jiaotong University Computer Dept. Of Science and Technology
5 Wu Feng,Shi Pengfei. Client-Transaction-Behavior Analysis Using Conceptual Clustering[J],Microcomputer Application, 2000 (5)

(上接第236页)

```
struct ext2u_dir_entry_2 {
    __u32 inode; /* Inode number */
    __u16 rec_len; // Directory entry length
    __u8 name_len; /* Name length */
    __u8 file_type;
    wchar_t name[EXT2U_NAME_LEN]; /* File name */
};
static struct super_operations ext2u_sops={
    read_inode:ext2u_read_inode,
    write_inode: ext2u_write_inode,
    put_inode: ext2u_put_inode,
    delete_inode: ext2u_delete_inode,
    put_super: ext2u_put_super,
    write_super: ext2u_write_super,
    statfs: ext2u_statfs,
    remount_fs: ext2u_remount,
};
struct inode_operations
    ext2u_file_inode_operations={
    truncate: ext2u_truncate,
};
struct inode_operations
    ext2u_fast_symlink_inode_operations={
    readlink: ext2u_readlink,
    follow_link: ext2u_follow_link,
};
struct file_operations ext2u_file_operations = {
    llseek: generic_file_llseek,
    read: generic_file_read,
    write: generic_file_write,
    ioctl: ext2u_ioctl,
    mmap: generic_file_mmap,
    open:generic_file_open,
    release: ext2u_release_file,
    fsync: ext2u_sync_file,
};
struct file_operations ext2u_dir_operations = {
    read: generic_read_dir,
    readdir: ext2u_readdir,
    ioctl: ext2u_ioctl,
    fsync: ext2u_sync_file,
};
```

4.2 虚拟文件系统

虚拟文件系统应该提供合适的系统调用。在兼容原有系统调用的情况下,VFS需要提供如3.4节所列的系统调用。

虚拟文件系统支持对Unicode编码的元数据进行处理,从而文件子系统支持多语言。系统存取文件时,通过逐层解析一个文件的路径名来找到这个文件在磁盘上的索引结点,然后根据此索引结点操作文件。系统在解析路径名的过程中,要使用一个操作:在子目录的所有目录项(包括文件项)中找出适合的项。这是通过字符串比较来完成的。核心中对路径名(包括文件与目录)的表示采用字符数组指针。如前所述,字符数组指针遵循POSIX标准关于字符串的定义,因此支持多语言需要使用其他的数据类型。我们使用宽字符类型wchar_t来表示Unicode编码的数据。

在虚拟文件系统中,文件系统主要完成从指定文件路径名的字符串到相应文件或目录的搜索过程,其中最主要的是路径名解析算法,具体算法请参见文[4]。

4.3 对EXT2和VFAT的支持

对EXT2和VFAT的支持主要是实现其逻辑文件系统各种操作的适当语义,这些操作包括超级块操作(struct super_operations)、文件i结点操作(struct inode_operations)、文件与目录操作(struct file_operations)等。

5 测试

针对文件子系统的测试主要是正确性测试,测试方法是:(1)使用专门的程序来测试系统调用,并通过直接查看磁盘数据的方法来测试系统调用。(2)通过设计EXT2U、EXT2和VFAT使用边界条件来测试文件系统元数据的正确性。测试结果显示Linux内核运行正常,对EXT2U文件系统的相关对象的操作是正确的。

结束语 目前有些系统使用UTF-8来实现对现有系统的兼容和对Unicode标准的支持仅是一种折衷方案,带来许多潜在的隐患。虽然本文提出的Linux文件子系统与现有的POSIX标准不完全兼容,但是,本文认为操作系统对Unicode标准的完全支持是操作系统发展的必然方向,这样的尝试是值得的。

多语言文件子系统是实现全球化操作系统的基础。建立Linux多语言文件子系统的意义在于:(1)根本性地解决了文件系统无二义地存储多语言文本的问题;(2)为上层应用提供了一组基于Unicode编码的系统调用。这样,使用上层系统(包括支持库及应用程序)进行系统开发时,减少了许多不必要的字符集处理的复杂性。对一个全球化操作系统来说,仅文件系统支持多语言是不够的,系统支持库、上层图形库的国际化与本地化机制的改进将是下一步工作的方向。

致谢 本文在此感谢课题组张立强、倪剑、陈萍等同志的努力与合作。

参考文献

1 ISO/IEC. ISO/IEC 10646-1:2000 Information technology - Universal Multiple-Octet Coded Character Set (UCS)-Part 1: Architecture and Basic Multilingual Plane. 2000
2 Aliprand J,et al. The Unicode Standard Version 4.0. Addison-Wesley, Aug. 2003
3 ECMA(European Computer Manufacturers Association). Standard ECMA-35: Character Code Structure and Extension Techniques. Dec. 1994
4 Bovet D P,Cesati M 著,陈莉君,冯锐,牛欣源译.深入理解LINUX内核.中国电力出版社,2001
5 毛德操,胡希明.LINUX内核源代码情景分析.浙江大学出版社,2001
6 陈莉君.LINUX操作系统内核分析.人民邮电出版社,2000