红旗 Linux 中文操作系统技术白皮书

中文平台技术

中科红旗软件技术有限公司

二〇〇〇年六月

第 1 章 “炎黄”中文平台结构

1.1 设计目标

在 UNIX 系统(Linux 亦如此)中，一般都提供了支持国际化本地化的 NLS(National Language Support)子系统，如图 1-1 所示。NLS 子系统是建筑在基于 ASCII 码的 UNIX 核心上，为世界上不同地域、不同语言环境的应用提供国际化本地化支持。在此基础上，可以建立支持各种不同的语言文化的民族特征数据库(LOCALE)、输入方法(IM)、字体(FONT)和消息机制(MESSAGE)等[1]。

NLS 子系统是 UNIX/Linux 实现国际化本地化的基础，系统中所有支持多国语言的实用程序，包括 X Window 都是建立在这个基础上。当然，并不是没有 NLS 子系统的支持就不能进行在 UNIX/Linux 系统上开发本地语言(如，中文)处理环境，但是，开发的工作量要比有 NLS 支持大得多，而且兼容性和可移植性差，实际使用效果也要差一些。例如，在 X Window 上，利用 NLS 机制实现本地化时只要配置好中文显示字库，在进入中文本地环境之后，绝大多数 X 应用程序就可以显示中文信息，而且不会出现"半个汉字"问题。这时，要开发的只是中文输入方法。但是，如果没有 NLS 支持，还必须开发中文显示部分，而且很难保证不出现"半个汉字"问题。

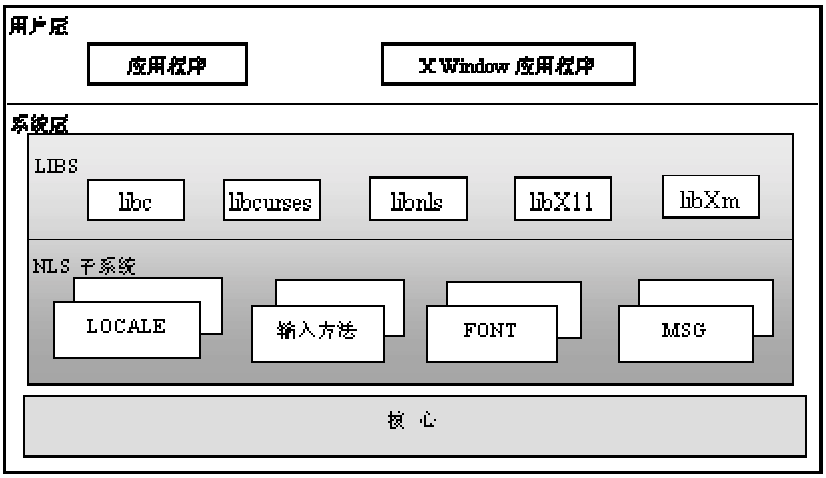

图 1-1. 开放系统体系结构

在整个 NLS 子系统中，与代码集相关的多字节字符与宽字符处理函数则是 UNIX/Linux 实用程序支持国际化本地化的核心，通过这些函数，实用程序把英文与各种本地文字同样处理。这一特性对于中文等多字节文字尤为重要。实用程序(如，vi、sh、awk 等)使用这些函数就可以对汉字进行整字处理(增、删、改、光标移动等)，不会出现"半个汉字"问题。但是，前提是必须在中文 locale 的支持下进行操作，否则，这些函数并不起作用。如果没有 NLS 的支持，要想让系统中的实用程序支持汉字的整字操作，就必须实用程序内部进行处理，工作量大，而且效果也很难保证。在一些 UNIX 中文平台产品中，由于开发人员缺乏对 UNIX 国际化本地化技术的了解，在中文平台上并没有采用 SCO OpenServer

系统已经提供的 NLS 支持，因此，为了能使 vi 能够不出现"半个汉字"问题，不得另行开发了一个 cvi，浪费了人力物力。而且，虽然有些产品为用户提供了中文的消息处理，但是由于没有采用国际化本地化的实现方法，在中英文消息切换时不得不采取复制整个消息目录方法实现，效率很低，并且无法实现消息与平台环境同步改变(即，在中文环境下使用中文消息，在英文环境下使用英文消息)。

对于运行在 PC 和 PC 服务器上的 Linux 系统，还有一个需要特别注意的特点是 PC 或 PC 服务器有一个需要特殊处理的主控台，作为一个完整的中文平台解决方案，必须提供对主控台的中文支持。事实上，这部分也占整个中文平台中工作量较大的部分。

在详细分析了目前市场上的 Linux 中文平台的功能及技术特点基础上，根据 UNIX 系统国际化、本地化技术，以及中文平台技术的现状和发展趋势，我们在"炎黄"中文平台的设计中着重解决了以下几个问题：

1) 标准化问题

UNIX 系统在其几十年的发展中，已经形成了一系列为产业广泛接受并遵循的国际标准和工业标准(如 Posix、XPG/4、SPEC ’95 等)，其中包括了有关国际化、本地化的内容。许多国外公司，如 IBM、SUN、SCO 都遵循这些标准开发了一批支持不同语言(如德文、法文、日文、中文等)的本地化环境，我国台湾地区的一些公司(如大同)也依据这些标准开发了繁体中文环境。而在大陆，由于多数软件开发商对此并不了解，也不够重视，因此，除个别厂商外，绝大多数 Linux 文平台产品都是以解决基本汉字处理(输入、显示、打印)为目标，孤立地对系统中不能处理汉字的地方(如主控台、X Window、vi、sh 等)进行扩充改造，自成体系。这样的中文平台不仅使得原英文系统中支持本地化的功能无法发挥作用，加大了开发的工作量，而且中文平台之间的兼容性差，与其它本地区环境很难同时工作。

针对这种情况，我国制定了《开发系统中文 API 界面规范(GB/T 16681-1996)》[2]，希望能有效地规范各厂商 UNIX 系统中文平台的开发。作为 GB/T 16681-1996 的起草单位之一，我们在开发中遵循了这些标准，在解决中文平台标准化、规范化上下了工夫。

2) 版本无关性问题

由于目前的中文平台是架构应已有的英文系统上的，而在核心内部实现相对比较困难，开发的工作量也较大。而且，由于在核心内部实现与原系统的依赖关系十分紧密，因此一旦原系统版本发生变化，则中文平台往往要在新的英文版本上重新开发，对于人力物力都是浪费。再者，在系统核心内部实现中文平台，其灵活性、稳定性都难于保证。采用"外挂式"技术可以较为有效地解决上述问题。

3) 对于英文软件的兼容性

由于 Linux 系统上的应用软件众多，这些软件中有一些并未按国际化的方法进行设计，这就给支持英文外的语言环境带来了一定的困难，但是一个好的本地平台应该有能力在这样的软件上输入/输出汉字(当然软件本身必须能支持 8 位操作)。

4) 支持 CJK 大字符集

随着计算机应用的不断深入，计算机网络的进一步普及，GB2312-80 标准中的 6763 个汉字已越来越不能满足用户的应用需要。因此，作为新开发的中文平台应能够满足用户对汉字日益增长的需求，不仅应该支持 ISO 10646.1/GB 13000.1 标准中的 CJK 汉字，而且还应为进一步支持更大的汉字字符集预留必要的接口。

5) 多内码功能

近几年来，海峡两岸交流日益频繁及香港回归，有大量的中文电子信息需要在两岸、及港澳地区相互交换使用。由于大陆和台、港、澳地区的汉字标准、内码结构不同，造成了"同文同字不同码"的局面，过去的中文平台一般只能支持一种内码，要使用另一种内码，往往必须重新启动另一个中文平台。这给用户带来很多不便。如何在同一个平台上处理多种不同的内码，如 GB、GBK、UTF-8、Big5 等是在实现中需要研究的问题。

6) 具有良好的用户界面，易于使用与维护

目前，Windows 图形用户界面以其简单明了、易学易用的特点迅速被众多用户所接受，而 Linux 系统除工作站和 PC 主控台上具有 X Window 图形界面外，其它还是以字符界面为主。为了使用户更易于接受、使用与维护，中文平台应提供更好的用户界面。就中文平台的输入法而言，Windows 上已有一批由于全国信标委推荐易学易用的中文输入法，如智能 ABC、表形码、郑码等，在 Linux 中文平台尚无对非专业录入人员十分易学易用的输入法，在我们的平台上结束了这种局面。

1.2 模块结构

由于目前的 Linux 系统核心及各种外部设备是基于 ASCII 的，因此，字符数据在进出系统核心及在设备上进行输入输出时，使用的都是文件码，即多字节字符串。只有在应用程序需要时，才通过 NLS 库中代码集相关的多字节字符与宽字符转换函数文件码将转换成处理码供应用程序使用，其代码流见图 1-2。

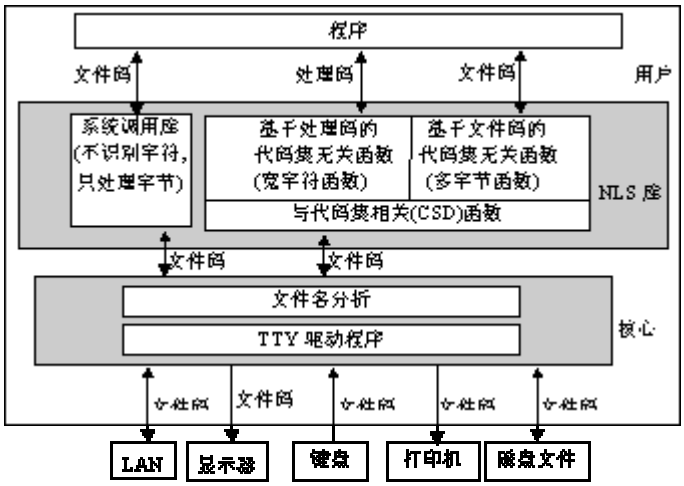

图 1-2. 在 NLS 中的代码流

显然，要在这样的系统上实现支持 ISO 10646 大字符集的中文平台，只能采用在基于字节处理的 Linux 核心基础上用某种内码(文件码)形式，如 UTF-8、GBK 来存储、传输和输入输出 ISO 10646 编码，同时，为应用程序提供一组文件码和处理码互相转换的函数，应用程序可以使用这组函数在其内部对 ISO 10646 编码进行处理。

运行在 PC 及 PC 服务器上的 Linux 系统除了可以配接终端外，在其主控台上还提供了字符和图形(基于 X Window)两套用户界面。因此，PC 及 PC 服务器上的 Linux 系统中文平台至少应包括中文字符界面、中文图形界面、中文打印输出子系统三个相对独立的组成部分。

根据上述分析和平台所要达到的设计目标(见 1.1 节)，我们把"炎黄"中文平台划分为由十余个模块组成的五个子系统，即中文字符界面、中文图形界面、中文打印输出、跨平台输入方法子系统和跨平台高品质字形服务子系统。各子系统和模块关系如图 1-3 所示，其主要功能为：
汉字终端仿真程序在主控台字符界面上提供汉字终端的功能，使得主控台字符界面成为一个功能齐全的汉字终端。

字符界面显示模块为字符界面提供与系统版本无关的汉字显示功能。

字符界面输入模块为字符界面提供汉字输入功能。
图形界面输入服务器与 X Font 服务器作为 X Server 的扩充，为基于 X Window 的应用程序提供汉字输入与显示功能。

中文打印输出程序为整个系统提供与打印机无关的高品质的汉字打印输出服务。

本地化环境包括 locale 数据库和本地化函数库两部分，给整个系统提供了

与本地语言(汉字)文化特性有关的描述信息和函数库。不仅主控台汉字界面可以使用该模块，而且用户还可以通过汉字终端使用该模块。

跨平台输入方法服务器可以为不同的系统平台提供汉字输入服务。该服务器可以与上面各模块在同一系统上，也可以在不同的系统上，通过 TCP/IP 网络来请求服务器提供具体的输入服务。输入法 1~n 模块提供各种具体的输入法(如，拼音、智能 ABC 等)处理。

跨平台 TrueType 字形服务器可以为不同的系统平台提供汉字字形输出服务。该服务器将汉字 TrueType 字形的还原(光栅化)处理，将其还原成汉字点阵信息。它可以与上面各模块在同一系统上，也可以在不同的系统上，通过 TCP/IP 网络来提供服务。

在下一章中，将详细讨论各个模块的具体实现算法与结构。

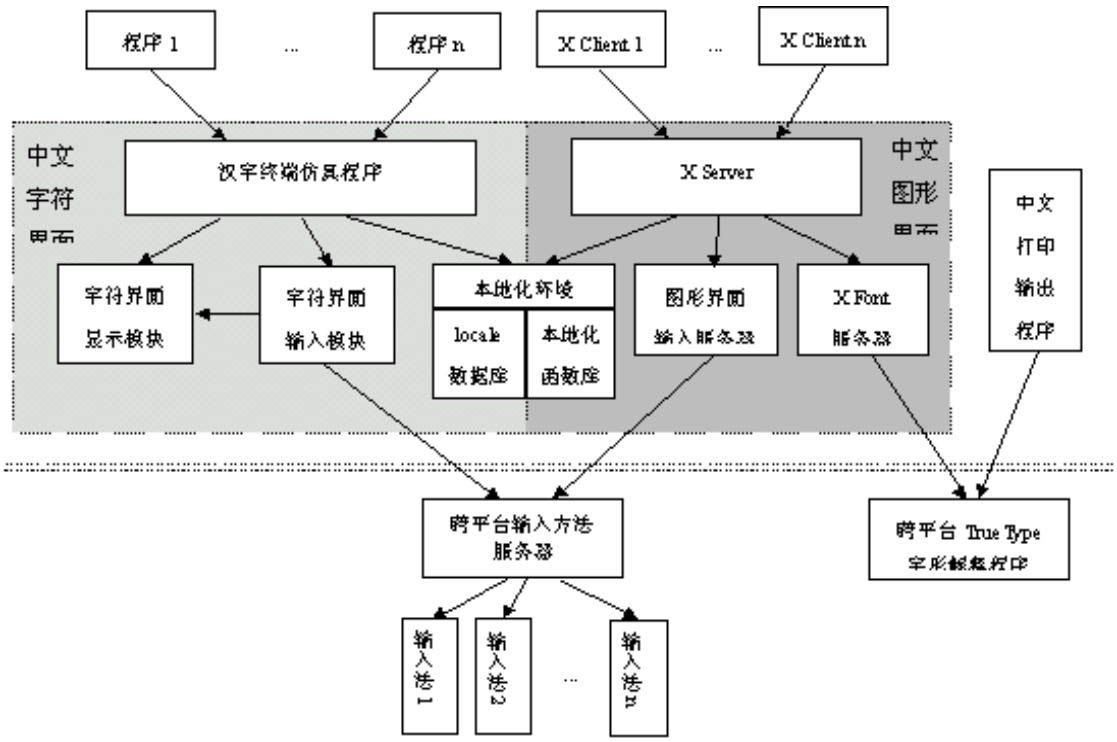

图 1-3. 中文平台模块结构

第 2 章 具体实现

本章将讨论"炎黄"中文平台的各个模块在实现时采用的具体结构和算法。

2.1 内码选择

在"炎黄"中文平台中，支持 ISO 10646 标准 CJK 汉字字符集和汉字多内码(GB、GBK、UTF-8 和 Big5)是两个重要的设计目标。为了达到这一目标，就必须

为该平台选择一个宽阔、简洁、高效、实用的内码作为平台的基本内码，对其它内码的支持则在此基础上进行扩充。显然，这个内码必须能够支持 ISO 10646 标准 CJK 汉字字符集。

从理论上讲，UTF-8 代码是比较好的选择。因为 UTF-8 不仅可以表示 CJK 汉字，也可以表示 ISO 10646 标准中的其它文字与符号，而且有很好的可扩展性，可以表示 CJK Extension A 和 B 中的汉字。但是，从目前实用的角度上看，由于 UTF-8 与现在广泛使用的 GB 和 GBK 内码不兼容，因此在绝大多数应用时，必须不断进行 UTF-8 与 GB/GBK 之间的代码转换，影响整个系统的效率。

根据上述考虑，我们认为使用 GBK 代码作为"炎黄"平台的基本内码是比较实用的，这样既可以与现行的 GB 内码兼容，与其它支持 GBK 内码体系的软硬件产品(如，Windows95 等)兼容，又可以覆盖 ISO 10646 标准 CJK 汉字字符集和 Big5 代码的符号，通过必要的代码映射支持 Big5 和 UTF-8 代码。

在处理码的选择上，考虑到 GBK 是 UCS-4 中 CJK 汉字的一种表示形式，二者之间的转换是基于映射表的一一对应的映射，处理简便直观，因此，我们采用了 UCS-4 代码作为处理码。

2.2 本地化环境

在确定了"炎黄"平台的基本内码后，就要建立基于该内码体系的本地化环境。本地化环境分为 locale 数据库和本地化函数库两部分，是开放系统进行本地化的基础。

2.2.1 本地化函数库的扩充

由于"炎黄"平台采用 GBK 为基本内码(多字节文件码)，而目前 Linux 系统的 NLS 子系统的本地化函数库只能支持 EUC 和 UTF-8 内码体系，并不支持 GBK 内码体系。在这样的系统上开发"炎黄"平台，就必须扩充现有的 NLS 子系统的本地化函数库。要扩充 NLS 子系统可以采用两种不同的方案：一是为支持 GBK 开发一组符合 POSIX、XPG4 和 ANSI C 标准和规范的函数和命令；二是扩充现有 NLS 函数和命令，使之具有支持 GBK 的能力，同时，保持现有 NLS 支持 EUC 和 UTF-8 内码体系的功能和界面。

综合比较以上方案，我们认为，前者存在明显的缺陷。由于系统中有两套本地化函数，用户在使用必须根据自己要用的本地环境使用的是哪种内码体系来更换不同的 NLS 子系统的本地化函数，使用很不方便，也无法充分利用 Linux 系统现有的应用程序。而后者的特点是在现有 NLS 函数和命令的内部进行扩充，使之在保持标准的界面和所支持的内码体系的前提下，具有支持 GBK 的能力。这样，用户在使用时只要设置所使用的本地环境，而无需关心该环境使用的是哪种内码体系。因此，我们在实现时采用了方案二。

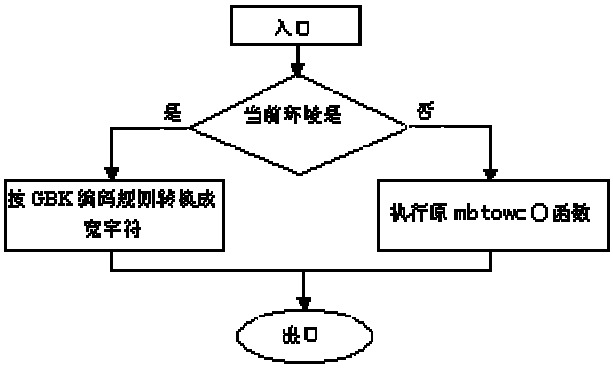

图 2-1. 函数 mbtowc()的处理流程

如果 NLS 子系统的本地化函数是基于动态装入(Dynamic Load)，如 IBM 的 AIX，那么可以把支持不同内码体系的函数设计成可动态装入的对象集合。每个对象都有其宿主函数。当应用程序调用宿主函数时，NLS 子系统将根据给定的路径和 locale，动态地装入和验证相应的本地化函数。但是，Linux 系统的本地化函数都不是基于动态装入的，只支持动态链接(Dynamic Link)功能。针对这一特点，我们在原系统的本地化函数中增加对 GBK 的支持，当应用程序调用该函数时，将根据当前给定 locale 的内码体系确定是使用原函数处理过程，还是使用 GBK 处理过程。例如，扩充了对 GBK 的支持后，函数 mbtowc()的处理流程大致如图 2-1 所示。这样，所有使用该动态链接库的应用程序都可以直接处理 GBK 汉字而无需进行任何改动。

需要扩充的函数主要是 NLS 中涉及文件码(多字节)和处理码(宽字符)转换的函数，包括：

mbtowc()	wctomb()	mbstowsc()	wcstombs()	mblen()
putwchar()	getwchar()	getws()	putws()	

等等。这些函数虽然功能上有所差异，但是需扩充部分的核心算法都是 GBK 与 UCS-2 之间的转换。GBK 编码是从 0x8140 到 0xFEFE 连续的，与 GBK 编码相对应的 UCS-2 码也是分段连续的，具体算法只是基于表驱动的简单运算[3]，本文不再赘述。通过类似的方法，我们正在 Linux 的 NLS 子系统中扩充对 GB 18030-2000 标准内码体系以及 UTF-16 扩充方案的支持。

2.2.2 本地化数据库(localeDB)的建立

语言、字符集和文化习俗等等组合成了一个本地环境 locale。一个 locale 是从本地化特征中选取的执行环境。一个 locale 可以包容人类语言、文化习俗以及其它属性。用户可以在运行时设置整个 locale 或者其一部分。locale 可以通过一组 shell 环境变量(见表 2-1)进行设置，也可以通过 C 语言国际标准中提供 setlocale 函数在应用程序中查询和设置。符合国际化规范的应用程序可以根据用户运行时选择的本地环境来确定输入和输出。通过使用 locale 环境变量，用户可以改变应用软件的行为甚至是不同的应用软件。

为了表示本地环境的众多重要属性，构造了一个 locale 数据库--localeDB，主要的内容包括了消息服务、iconv 代码集转换程序及本地环境服务。表 2-1 是 localeDB 的主要组成。

表 2-1. localeDB 的构成

应用编程界面 API	分　类	Shell 环境变量	说　明
本	Ctype	LC_TYPE	提供字符分类和大小写区分功能
	Charmap		提供字符显示宽度及处理码映射码服务
地	Collation	LC_COLLATE	提供字符串比较和理序服务
	Monetary	LC_MONETARY	提供货币表示格式
环	Numeric	LC_NUMERIC	提供数字表示格式
	Respons		提供肯定/否定回答
境	Time	LC_TIME	提供时间与日期表示格式
代码转换	Conversion		提供字符集之间的转换
消息服务	Message	LC_MESSAGES	提供消息的本地化服务

LocaleDB 的生成主要依赖字符集的分类定义文件 charmap。在这个文件中，对整个字符集的所有字符进行了定义，确定哪些是控制字符、哪些是大写、哪些是小写、哪些是标点符号、哪些是数字，等等。其代码片段如下：

```
LC_CTYPE LC_CTYPE

# 定义单字节字符

isupper 0x41 - 0x5a                                          # 大写字母
islower 0x61 - 0x7a                                          # 小写字母
isdigit 0x30 - 0x39                                           # 数字
isspace 0x20 0x9- 0xd                                         # 空格
ispunct 0x21 - 0x2f 0x3a - 0x40 0x5b - 0x60 0x7b - 0x7e # 标点符号
iscntrl 0x0 - 0x1f 0x7f - 0x9f # 控制字符
isblank 0x20 # 空格
isxdigit 0x30 - 0x39 0x61 - 0x66 0x41 - 0x46 # 十六进制数字
ul <0x41 0x61> <0x42 0x62> <0x43 0x63> <0x44 0x64> <0x45 0x65>
<0x46 0x66> \
```

```
        <0x47 0x67> <0x48 0x68> <0x49 0x69> <0x4a 0x6a> <0x4b 0x6b>
<0x4c 0x6c> \
        <0x4d 0x6d> <0x4e 0x6e> <0x4f 0x6f> <0x50 0x70> <0x51 0x71>
<0x52 0x72> \
        <0x53 0x73> <0x54 0x74> <0x55 0x75> <0x56 0x76> <0x57 0x77>
<0x58 0x78> \
        <0x59 0x79> <0x5a 0x7a>                                  # 大小写映
射

    cswidth 2:2,3:2,2:2

    LC_NUMERIC LC_NUMERIC
    decimal_point.                                               # 十进制小
数点
    thousands_sep,                                               # 千分位符
号

    # 定义辅助代码集 1 (双字节字符)

    LC_CTYPE1
    isupper 0xa3c1 - 0xa3da 0xa6a1 - 0xa6b8 0xa7a1 - 0xa7c1 # 全角大写
字符
    islower 0xa3e1 - 0xa6fa 0xa6c1 - 0xa6d8 0xa7d1 - 0xa7f1 # 全角小写
字符

    # GBK 多字节

    isdigit 0xa3b0 - 0xa3b9                                      # 全角数
字
    isspace 0xa1a1 # 全角空白
    ispunct 0xa1a2- 0xa1fe 0xa2b1 - 0xa2e2 0xa2e5 - 0xa2ee \ # 全角标
点符号
            0xa2f1 - 0xa2fc 0xa3a1 - 0xa3af 0xa3ba - 0xa3c0 \
            0xa3db - 0xa3e0 0xa3fb - 0xa3fe 0xa9a4 - 0xa9ef
    isphonogram 0xa8a1 - 0xa8ba
    isideogram  0x8140 - 0x817e 0x8180 - 0x81fe 0x8240 - 0x827e 0x8280
- 0x82fe \
                0x8340 - 0x837e 0x8380 - 0x83fe 0x8480 - 0x847e 0x8480
- 0x84fe \
                  .
                  .
                  .
                0xfd40 - 0xfd7e 0xfd80 - 0xfdfe 0xfe40 - 0xfe7e 0xfe80
- 0xfefe
```

```
# 全角拼音字母大小写映射表

ul    <0xa3c1 0xa3e1><0xa3c2 0xa3e2><0xa3c3 0xa3e3> \ # 英文字母
          .
          .
          .
      <0xa3d9 0xa3f9><0xa3da 0xa3fa> \

# 全角希腊字母大小写映射表

      <0xa6a1 0xa6c1><0xa6a2 0xa6c2><0xa6a3 0xa6c3> \
          .
          .
          .
      <0xa6b6 0xa6d6><0xa6b7 0xa6d7><0xa6b8 0xa6d8> \

# 全角俄文字母大小写映射表

      <0xa7a1 0xa7d1><0xa7a2 0xa7d2><0xa7a3 0xa7d3> \
          .
          .
          .
      <0xa7bf 0xa7ef><0xa7c0 0xa7f0><0xa7c1 0xa7f1>
```

在具体实现时，我们根据 ISO 639 和 ISO 3160 对本地语言和区域定义了一个特殊的 locale 名字--zh_CN.GBK 来表示采用 GBK 代码体系的 CJK 中文环境。

2.3 与系统无关的中文字符界面

对于 PC 而言，其主控台是一个特殊的设备(包括键盘和由各种 VGA 卡控制的显示器)，通过操作系统内的主控台设备驱动程序、键盘驱动程序、VGA 显示驱动程序和 Linux 终端仿真程序对主控台进行控制。要解决 PC 主控台字符界面的中文处理问题(图形界面中文处理在 2.5 节讨论)，传统的方法是修改系统核心中的 VGA 卡显示控制程序和键盘控制程序，使之具有汉字输入、显示能力。这种方法存在明显的缺陷(见 1.1 节)。

为了使字符界面的中文处理能够真正实现与系统版本无关，在"炎黄"平台中，我们将字符界面中文处理模块设计成一个可以独立运行的程序，当用户运行该程序后，利用 Linux 系统中 VGA 显示驱动程序本身的功能，将屏幕显示设置成图形显示方式，并且负责在主控台上进行汉字显示与输入。

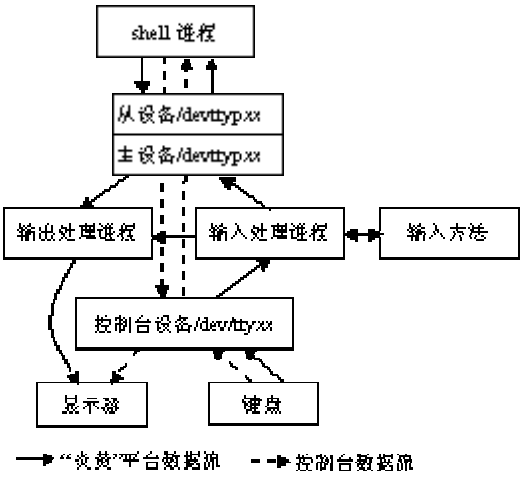

图 2-2. 中文字符界面数据流

由于整个字符中文平台是在用户层运行，所以必须要解决如何使其获得键盘输入和屏幕输出问题。为此，我们将系统主控台的输入输出改向到 Linux 系统的伪终端中，然后在对其进行处理，具体过程如下：中文平台程序启动后，首先将主控台屏幕显示设置成图形显示方式，然后产生三个子进程，一个是由输入模块构成的"输入处理进程"，负责从原用户注册终端/dev/ttyxx 的输入中读取用户的键盘输入，进行必要的中文输入处理后，改向到伪终端主设备/dev/ptyxx[4]或/dev/ptmx[5]中；另一个是新的 shell 进程，该 shell 进程以伪终端从设备/dev/ttypxx 或/dev/ptsxx 为输入输出设备，此后，用户将在这个新的 shell 控制下进行操作(此后，所有在该 shell 控制下的应用程序将继承该 shell 的输入输出设备)；第三个是由显示模块和终端仿真程序组成的"输出处理进程"，负责从伪终端主设备中的输出取出该 shell 的输出，处理终端控制码，最后以图形方式写到屏幕显示设备/dev/vga 中，即完成了在屏幕上的输出。各子进程之间的同步由主进程控制。图 2-2 为中文字符界面的数据流。这种方案的缺点是用户在一个控制台屏幕上使用该中文平台时，至少会多出四个进程，用户误用 kill 命令会退出中文环境。但是，这种方案的优点很多。首先，这种方案无需修改核心，不会影响系统的稳定性与可靠性。在用户使用时，从中文环境与原英文环境的切换无需重构核心与重新引导。其次，由于这种方案利用系统的是本身的 VGA 显示驱动程序功能，因此这种方案是与版本无关的。对于在 PC 机上运行的 UNIX 类系统(如 Linux、SCO UNIX、UnixWare、Sun Solaris)来说都适用。第三，如果采用了基于 Stream 技术的伪终端(/dev/ptmx 和/dev/ptsxx)，还可以在系统中压入一些功能模块[5]，如代码转换(Big5、GBK)模块。这样更容易地在同一中文平台上实现支持多内码的功能。目前"炎黄"平台使用的是传统的伪终端/dev/ptyxx 和/dev/ptsxx。

在下面将逐个讨论各个进程及程序模块的具体实现方法和主要数据结构。

2.3.1 主控进程

主控进程是"炎黄"平台字符界面的入口模块，用户在使用时通过在 shell

提示符下键入 yh 命令启动整个"炎黄"平台字符界面。该进程的功能主要是：对所需的设备(屏幕、键盘、伪终端)、资源进行初始化，生成读、写、shell 子进程，为子进程创建通信管道，对各个子进程进行同步控制。一旦各个子进程开始运行后，主控模块进程挂起，等待某一个子进程退出的信号，当收到该信号时，终止其它子进程运行，返回到用户注册的 shell 提示符下。

主控进程的处理过程如下：

```
main ()
{
        进行初始化设置，保存当前屏幕设置
        申请一个伪终端。
        if(没有可用的伪终端)
        {
          显示出错消息，退出
        }
        创建用于进程间同步与通信的管道 pipeCom 和 pipePrompt
        创建输出处理进程 pidWrite，等待 pidWrite 进程初始化
        if (pidWrite 进程失败)
            退出
        创建输入处理进程 pidRead
        创建输入用户 shell 进程，将申请到的伪终端从设备作为该 shell
进程的标准输入输出设备。此后用户程序将在此 shell 控制下运行：
    {
        Shell 子进程：装载输入法、执行 shell
  }
等待子接收进程终止的信号。一旦收到子进程终止的信号，则终止其它子进程
}
```

2.3.2 输出处理进程

由终端仿真程序和显示模块组成的输出处理进程创建后一直在伪终端主设备/dev/ptyxx 上等待读取 shell 或应用程序的输出。一旦读到输出信息，将其送到 Linux 终端仿真程序模块中对 Linux 终端控制码进行解释，并调用相应的输出程序显示到屏幕上。此外，该进程还负责输入提示区的显示。

输出处理进程的处理过程如下：

```
WriteScr(char *hzfntfile, char *engfntfile)
{
    设置用于控制显示的信号灯
    进行初始化显示
    通过管道 pipeCom 向输入处理进程发消息，通知输出处理进程已经就
绪。
```

```
        while (1)
        {
            在伪终端主设备/dev/ptyxx 上等待读取 shell 或应用程序的输出
            if (读到输出内容)
                调用 linux()对控制码进行解释，并显示到屏幕上
            else if (输入处理进程请求在提示区显示)
            {
                    从管道 pipePrompt 读取要显示的内容
                    if (读到显示内容)
                        调用 linux()在提示区显示输入处理进程的输出内容
            }
          else
              重新显示光标
        }
    }
```

1) 终端仿真程序

终端仿真程序模块的主要功能是解释应用程序的输出流中的 Linux 控制码，并按控制码的要求调用相应的屏幕显示程序将输出系统显示到屏幕上。

2) 屏幕显示程序模块

屏幕显示程序模块负责在显示器屏幕上按图形方式显示要输出的内容，并进行相应的屏幕操作，如，清屏、清行、上滚、下滚等；此外，还要负责汉字输入提示区的显示操作。

3) 字库管理与调度模块

此模块包括两部分功能，其一是对基本字库(GB 或 GBK 字库)的共享管理；其二是对扩充字库(Extension A、Extension B 及 Super CJK 字库)的调度管理。

- **基本字库的共享管理**

由于 PC 机上的 Linux 系统都支持控制台多屏(Multiscreen)功能，用户可以同时在几个不同的屏幕上注册，运行不同或相同的程序，当然也应该可以同时启动几个"炎黄"平台。但是，如果每一个运行的"炎黄"平台都自己装入一个汉字库，特别是 GBK 字库，显然是一种浪费，为此，我们在"炎黄"平台的字符界面程序中以共享内存[6]的方式在多个字符平台进程间实现对内存字库的共享。这样，只有最先启动的"炎黄"平台需要装入显示用的汉字库，此后启动"炎黄"平台只需与已装入内存的字库建立联系即可。采取这种方法，不仅有效地利用了内存资源，而且启动"炎黄"平台的响应速度也大大加快了。

基本字库的共享管理的算法如下：

```
装入基本字库：
{
    if (字库已经装入共享内存)
    {
        使用计数+1
        返回该共享内存区指针，进程使用该共享内存字库
    } else {
        申请共享内存空间
        从磁盘文件中装入基本字库
        使用计数+1
    }
    返回共享内存区指针
    }
    释放基本字库：
    {
        使用计数+1
        if (使用计数为 0)
            释放共享内存空间
    }
```

- **扩充字库的调度管理模块**

为了使"炎黄"平台具有良好的可扩充性，能够支持目前尚在讨论和表决过程中的 CJK Extension A 和 Extension B 以及 Super CJK，在"炎黄"平台中就必须为进一步扩充预留下接口。扩充字库的调度管理模块正是为今后支持 Extension A 和 Extension B 以及 Super CJK 预留的。

由于 Extension A 和 Extension B 以及 Super CJK 中的汉字数量大，使用频度低，而且对于不同的应用只会用到其中极少的汉字(并且通常是只会用到这些字)，因此，没有必要将整个 Extension A、Extension B 或 Super CJK 字库装入内存。但是，将字库全部放在硬盘上，每次从硬盘上读字库点阵速度又太慢(同时运行多个应用时犹为明显)，所以，在实现时我们采用了动态装入字库的方案，即，"炎黄"平台在运行时保留一片共享内存作为活动字库区，当应用程序用到时 Extension A、Extension B 或 Super CJK 字库中的某个汉字，将其点阵装入内存，同时使用"最近最少使用淘汰"算法对活动字库区进行维护。当最后一个中文平台进程退出时，将活动字库区保存到硬盘文件中，下次最先启动的中文平台再将其装入内存。

活动字库区的数据结构如下：

```
struct act_font {
    int a_code; /* 该汉字的内码 */
```

```
    int a_count; /* 该汉字的使用次数 */
    int a_last_use_time; /* 最近使用该汉字的时间 */
    faddr_t *a_addr; /* 该汉字在活动字库区的位置 */
};
struct act_font[1000];
```

2.3.3 输入处理进程

输入处理进程由输入模块组成，创建后一直在用户注册终端设备/dev/ttyxx 上等待读取用户从键盘上打入的每个字符。读到输入信息后，如果是在汉字输入状态，则交给输入方法进程进行处理，处理后的候选字返回给输入处理进程，然后将用户所选的汉字输入串送入伪终端主设备/dev/ptyxx；否则，直接将输入字符送入伪终端主设备中。此外，输入处理进程还要创建一个子进程用于为该平台挂接输入法。

其处理过程如下：

```
ReadKeybrd()
{
    对通信进行初始化：打开管道 pipeCom 和 socket，创建输入法挂接进程，并
    进入监听状态，监听输入法挂接进程传来的输入法信息

    while (1)
    {
        if (要在提示区显示信息)
            通过管道 pipePrompt 将要显示信息送给输出处理进程 pidWrite，
                                    并发 SIGPIPE 信号通知 pidWrite;
        在标准输入上等待读取用户的键盘输出;
        if (接收到用户键盘输入)
        {
            进行输入过滤：汉字输入状态交给输入法处理；否则直接返回;
              if (有信息输入)
                  送入伪终端主设备/dev/ptyxx;
        }
    }
}
```

由输入处理进程创建的输入法挂接进程负责挂接系统中启动的输入法。该进程运行后，在一个端口上等待输入法与之挂接。该端口的端口号与平台相关，以保证在不同的屏幕上运行的中文平台各自使用各自的输入法。当输入法挂接时，输入法进程将其名称(输入法名)和端口号传给输入法挂接进程，输入法挂接进程再将该输入法的名称和端口号传送给输入处理进程。此后，输

入处理进程将直接使用输入法端口与之相连进行汉字输入处理。三者之间的关系如图 2-3 所示。

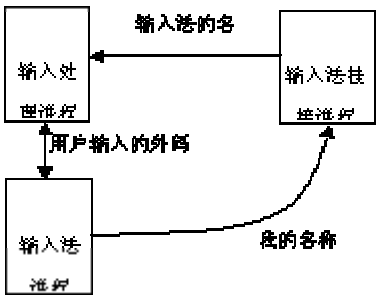

图 2-3. 输入处理进程组间的关系

2.3.4 进程间的同步与通信

由于"炎黄"平台字符界面的中文处理功能是由一组进程共同完成的，因此，进程间的同步与通信就显得尤为重要。在这里，我们采用了 Linux 中的命名管道和信号机制来实现进程间的同步与通信。

下面小节将主要讨论"炎黄"平台字符界面的的同步与通信机制。

1) 主进程与输出处理进程的同步

在"炎黄"平台启动过程中，主控进程创建输出处理进程后需要等待该进程对屏幕设备进行初始化，否则就无法在屏幕上显示要输出的内容。主控进程与输出处理进程的同步控制使用命名管道 pipeCom。

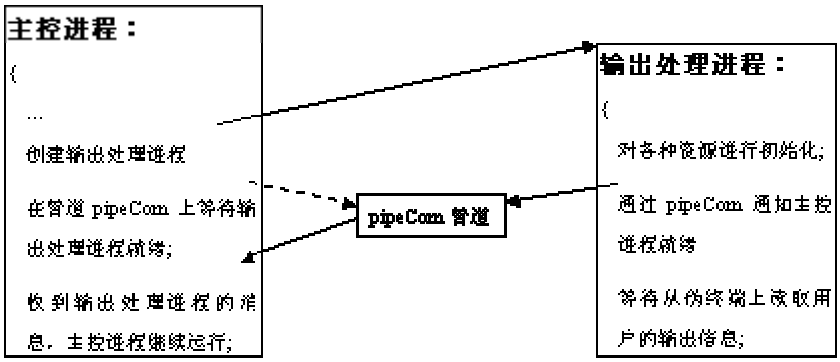

图 2-4. 主控进程与输出处理进程之间的同步关系

2) 主进程与输入法挂接进程的同步

在启动过程中，输入处理进程开始运行后会创建一个输入法挂接进程，如果在输入法挂接进程尚未准备就绪时挂接输入法将导致挂接失败，而主控进程在创建输入处理进程后还要自动为用户挂接预先定义输入法，这时必须进行同步控制。主控进程与输入法挂接进程的同步控制使用命名管道 pipeCom。

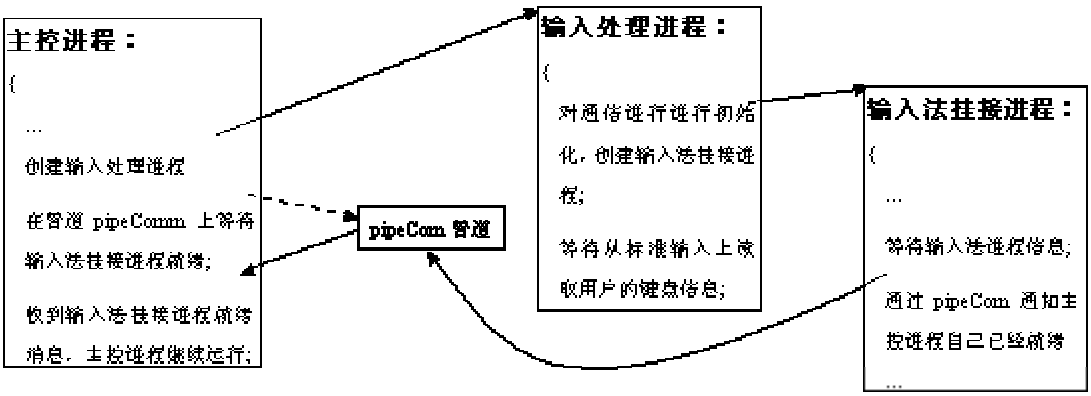

图 2-5. 主控进程与输入法挂接进程之间的同步关系

3) 输出处理进程与输入处理进程间的通信

前面已经提到，输出处理进程总是在伪终端主设备/dev/ptyxx 上等待应用程序得输出，以便将其显示到屏幕上。此外，输入处理进程还要请求该进程为其显示输入提示区。由于输出处理进程一直阻塞在/dev/ptyxx 上，无法知道输入处理进程何时有输出请求，因此，必须提供一个通信机制，使得当输入处理进程要在提示区进行显示时能让输出处理进程从阻塞状态中跳出，来响应输入处理进程的输出请求。

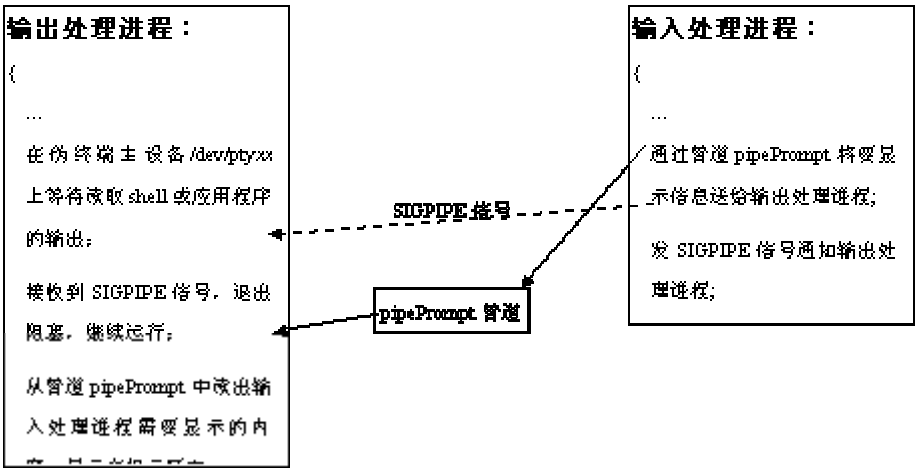

图 2-6. 输出处理进程与输入处理进程间的通信关系

在实现时，我们采用信号和命名管道相配合的方法来建立输出处理进程与输入处理进程间的通信机制：由输入处理进程向处在阻塞状态的输出处理进程发 SIGPIPE 信号，同时将要显示的内容写入管道 pipePrompt 中；输出处理进程在收到该信号后，处在阻塞状态的 read 系统调用出错返回，从而退出阻塞，继续运行。输出处理进程从管道 pipePrompt 中读出要显示的内容，显示到屏幕的输入提示区中。至此，一次通信过程完成。输出处理进程与输入处理进程间的通信关系如图 2-6 所示。

2.4 跨平台输入方法子系统

输入方法子系统是中文平台不可或缺的组成部分，传统的输入方法子系统必须与中文平台在同一台机器上，不能互相利用，造成了人力和物力的浪费；另外，由于一些输入法(特别是一些支持 CJK 汉字的输入法)的程序与数据文件都很大，有数兆字节，如果在每台主机都安装一份是很不经济的，而且对于 NetPC 及瘦型客户机而言也过于庞大。

我们认为，输入方法子系统应该为用户提供了一种透明的通用机制，用户可以简单地利用平台提供的规则定义输入方法界面，通过子系统有效和方便地把用户定义的规则转换成所需的输入方法；该子系统应为应用程序(中文平台)提供统一的界面，并具有与平台无关和网络化的功能，能实现独立于使用地域环境(大陆、台、日、韩)的输入方法的表示体系。基于这种考虑，我们提出并实现了跨平台输入方法子系统[7]，在"炎黄"平台中我们由对该子系统进行了一些简化，使之更加实用。

2.4.1 子系统结构

在"炎黄"中文平台中，我们把平台与输入方法子系统的界面 PIMI 和输入方法子系统与具体输入法的界面 IMPI 简化为两个动态库，分别用于中文平台和输入法程序。在 PIMI 库中，提供一组供中文平台调用的，主要完成挂接输入法函数，如，选择当前输入法，与输入法通信等。IMPI 库中主要为输入法程序提供了一个主程序，负责与中文平台程序的通信。在这个主程序中，调用了一组用来操作输入法的函数，但是这些函数在 IMPI 库中并没有代码，如何实现这些函数，要由输入法的开发者完成。中文平台与输入法之间通过 TCP/IP 协议的通信由 PIMI 和 IMPI 两个动态库中的函数完成。图 2-7 为"炎黄"平台的输入方法子系统结构。

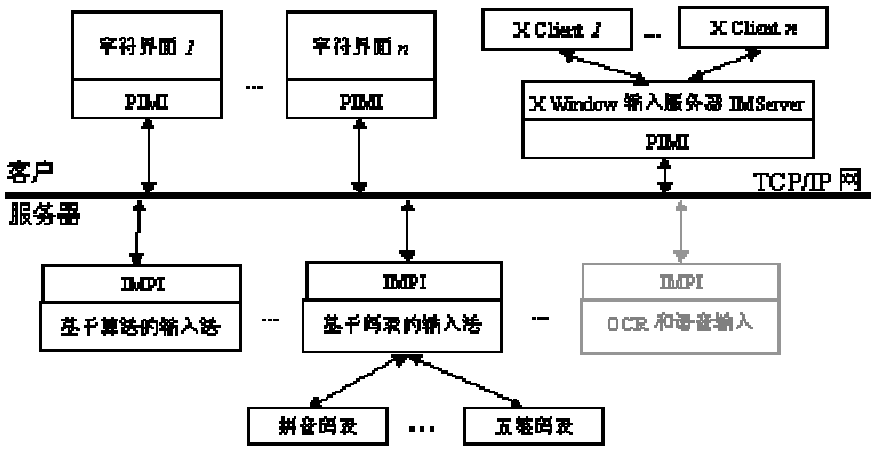

图 2-7."炎黄"平台的输入方法子系统结构

在开发具体输入方法时，只要按 API 的定义开发一组回调函数(CALLBACK)，然后与 IMIP 库连接形成运行代码即可。这样做不仅保持了原子系统的所有特性，而且结构更加简单，减少了连接通信次数，提高了系统效率。同时，该子系统把复杂的进程间的通信过程"封装"在动态库里，开发人员可以直接调用这里提供的函数，为平台挂接输入法，使输入法开发者可以集中精力开发正文平

台和输入法算法，不必考虑它们之间如何连接，从而减少了开发难度与工作量。此外，采用动态链接库使得系统维护与升级更加容易。

2.4.2 输入方法子系统 API

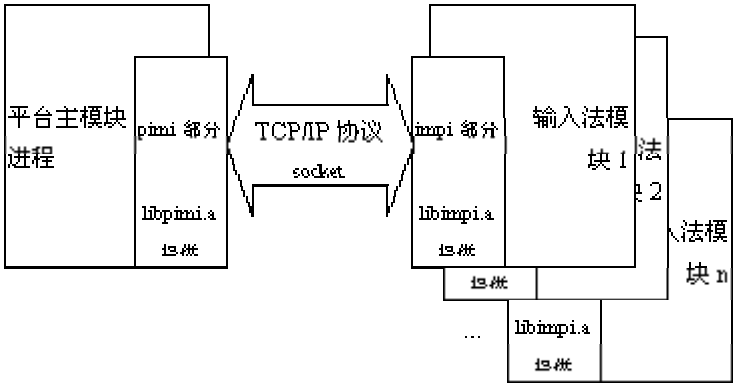

图 2-8. 输入方法子系统界面关系

"炎黄"平台的输入方法子系统[8]API 分为两部分：一是嵌入平台主模块的 libpimi.a(PIMI--Platform Input Method Interface 意为平台到输入法的界面)；另一部分是嵌入各个输入法模块的 libimpi.a(IMPI--Input Method Platform Interface 意为输入法到平台的界面)。两部分的关系如图 2-8 所示，主要流程如图 2-9 所示。

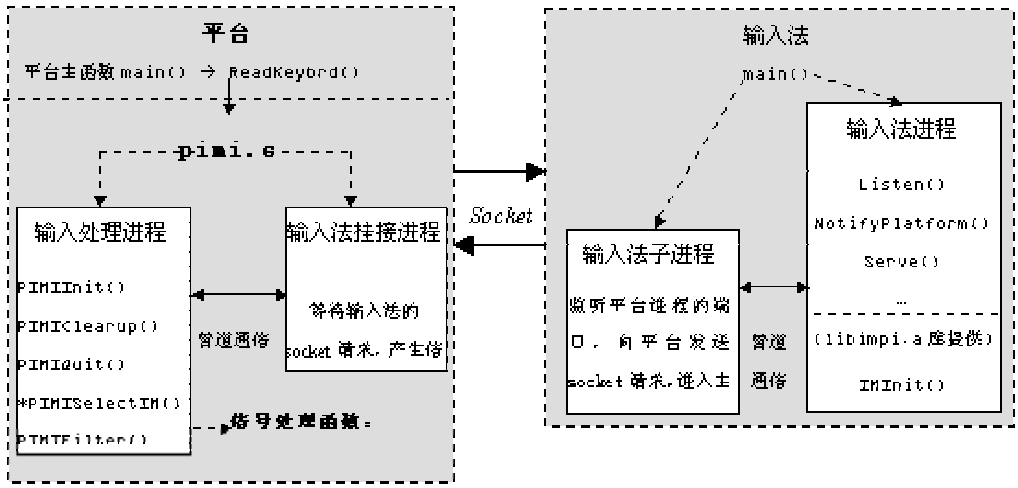

图 2-9. 输入子系统主要流程

1) PIMI

本模块与平台连接后生成平台的独立可执行程序，通过 TCP/IP 协议与 IMPI 通信，并通过它接受输入法的挂接，使用输入法提供的服务等。

PIMI 中主要提供了以下函数：

• 初始化函数 PIMIInit(int port_fd)，它由输入法挂接进程调用，完成对 PIMI 通信的初始化工作，等待连接输入法进程。

• 清除函数 PIMIClearup()，平台应在自己退出之前调用此函数来关闭所有与之相连的输入法，释放资源。

• 函数 PIMIQuit()用来摘除当前输入法。

• 函数 PIMISelectIM(int idx)从已经挂上的输入法中选择一个作为当前输入法并返回被选中的输入法的名字。

• 函数 PIMIFilter(int ch)用当前输入法过滤用户输入的字符。中文平台可以从 PIMI 中的全局变量中得到外码、候选与结果字符串，然后可以按照自己的格式来显示这些信息(既可以按水平方向显示在屏幕底部的提示行中，也可以竖直方式来显示候选等)。

在这些函数中，除 PIMIInit 由输入法挂接进程调用外，其它均由输入处理进程调用。

2) IMPI

该动态库为输入法开发者提供了生成输入法程序模块所需的主函数 main()，以及通过 TCP/IP 协议与 PIMI 连接、通信的函数。输入法开发者无须了解该库的结构，只需要按界面定义的要求为每一个输入法实现下列回调(Callback)函数和其输入法算法，然后与该动态库相链接，就可以生成完整、可以运行的输入法程序。IMPI 界面的回调函数定义如下：

• 初始化函数 IMInit()，应在被调用时做初始化工作。它返回-1 表示初始化失败，其它值表示成功。

• 清除函数 IMClearup()，应在被调用时做清除工作。它返回-1 表示清除失败，其它值表示成功。

• 选择函数 IMSelect(int select)，如果 select = 1，则说明用户切换到本输入法；如果 select = 0，则用户切换出本输入法，输入法应在此函数中做相应的工作。它返回-1 表示选择失败，其它值表示成功。

• 过滤函数 IMFilter(int ch)，ch 为用户输入的字符。此函数通过全局变量返回外码、候选与结果字符串。

IMPI 库中实现的输入法模块主函数和与平台进行连接、通信的函数主要有：

• 主函数 main(argc, argv)，负责建立与平台进程的管道，初始化输入法。

• 监听函数 Listen()，监听服务器端的 socket，返回服务器端的 socket 端口号。

• 建立连接函数 NotifyPlatform(int portListen, char hostPlatform, int portPlatform)按照参数给定的地址、端口，建立、配置 socket，建立连接。把输入法名称和所用的地址、端口发送给平台，最后关闭此 socket。

• 例外处理函数 catch_exception(int signo)接收并处理系统多种"例外"信号：关闭输入法和 socket。

• 输入服务函数 Serve()负责从 socket 中接收从平台进程发来的用户键盘事件，调用 IMFilter()把用户键盘事件交输入法模块进行处理，然后把得到的结果再发送给平台。

采用此方法实现的输入子系统不仅保持了原设计的所有特点，而且更加简洁灵活。我们还可以在 Windows95 上用 PIMI 库实现一个输入法界面，在使用时，则通过 PIMI 库调用在 UNIX/Linux 服务器上的各种输入法，从而实现跨平台的输入操作。另外，该方法还为输入法开发者屏蔽了复杂的输入界面的开发工作，使他们专心于开发输入法算法本身。

2.5 X Window 图形界面

X Window 是 UNIX 类系统的图形用户界面，它采用的是典型的客户机-服务器结构：由 X Server 负责对各种资源的管理、控制与操作，X 应用程序作为 Client 在需要使用资源时请求 X Server 为其服务，服务器与客户程序之间采用 X 协议(X Protocol)进行通信。为了方便应用程序的开发，X 提供了 Xlib 函数库来封装 X 协议，负责 X Client 与 X Server 之间的通信和其他基本操作[9][10]，这样 X Client 就无需考虑的 X 协议而直接调用 Xlib 库函数即可。X Window 的工作机制如图 2-10 所示。

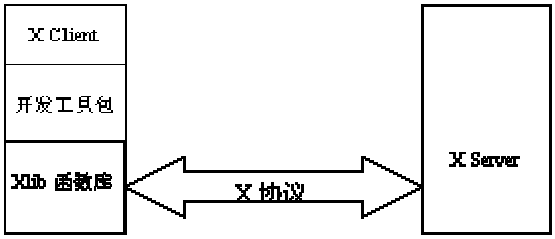

图 2-10. X Window 的工作机制

目前的 X Window 系统已经提供了比较完善的国际化(Internationalization)/本地化(Localization)支持机制，并有一套本地化环境开发规范[11][12][13][14]，这给 X Window 的本地化开发提供了非常的便利条件。但是，完全按照 X Window 的国际化/本地化开发规范实现对中文的支持，也有一些局限性：一是适应性、兼容性较差，只能支持按照国际化规范开发的应用程序。但是，目前 X Window 上还有相当数量的非国际化应用，按此方法实现的本地化环境对于这些应用程序毫无意义；二是可扩充性差。目前的 X Window 系统只能支持 EUC 代码体系(X11R6 可以支持 UTF-8)，很难在其上面构筑支持多内码的中文平台；三是 X Window 没有考虑对多字节、大字符集(如中文、日文)语言处理的特殊性(如，字库容量要远远大于拼音文字)，而是采用相同的方法进行处理，使得处理大字符集语言效率较低。而"炎黄"平台的目标则恰恰是要开发以 GBK 为基础，支持多内码的大字符集中文平台，同时还要能够支持系统中现有的各种应用，因此，在开发中，我们并没有完全遵循 X Window 的国际化/本地化规范，而是根据需要在尽可能遵循规范的前提下，采用了一些

变通的方法。

在本节中，我们将讨论在 X Window 中文处理功能实现的结构。

2.5.1 中文输入服务器

在 X Window 环境中提供中文输入支持比较常用的方法是采用输入服务器(Input Server)。中文输入服务器是一个负责将用户的键盘输入事件转换为汉字的 X 应用程序，它与其它 X Client 通过 Inter-Client Communication Convention[14]进行通信。由于所有与中文输入有关的工作都由中文输入服务器完成，因此，对于应用程序来说，这个过程是完全透明的。
但是，未按国际化/本地化规范开发 X 应用程序在处理输入事件时根本不会向输入服务器发出请求(即，将键码序列传送给后者，要求后者将"过滤"结果返回)。究其原因，是 Xlib 函数库为编程人员提供了两套功能相同的输入输出处理函数，一套支持国际化(多字节处理)，另一套只支持单字节应用。一旦应用程序使用的是只支持单字节的 Xlib 函数，那么该应用程序就不会向输入服务器发出请求。此外，由于现有的 X Window 只能支持 EUC 和 UTF-8 代码体系，对于其它代码，如 GBK、Big5 即使是使用了支持国际化的库函数也不能很好地处理。

为解决这个，我们修改了 Xlib 函数库，"截获"相关的函数，并进行扩充：使只支持单字节的输入函数在我们指定的本地环境下将输入事件送入输入服务器处理；同时对两套输入函数增加对 GBK、Big5 代码的处理。这就相当于在 Xlib 之上插入了一个中间层，使得所有 X Client 都可以与中文输入服务器的交互实现中文输入[16]。

在实现上，中文输入服务器只负责与 X Server 和其它应用程序的通信，以及对输入提示区的管理，但不提供具体输入法的算法[17]。各个不同的输入法算法由 2.4 节中的输入法进程完成，中文输入服务器只需要调用 PIMI 函数库与输入法进行连接即可。这样，系统中无论是字符界面还是图形界面都使用相同的输入法，从而避免了对输入法的重复开发，也减少了对系统资源的占用。X Window 中文输入服务器结构如图 2-11 所示。

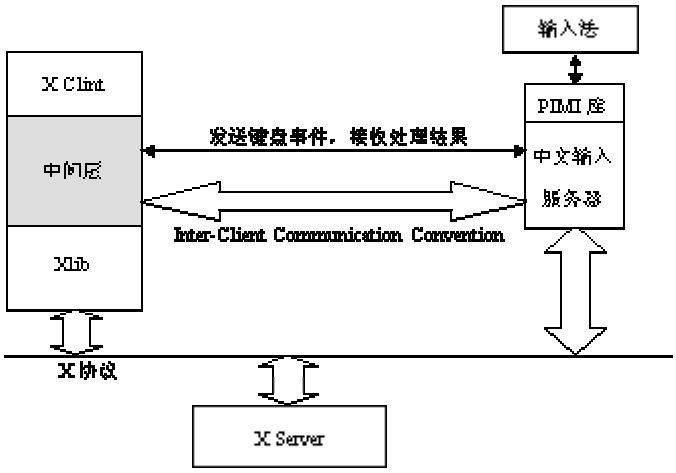

图 2-11. X Window 中文输入服务器

2.5.2 中文字形服务器

虽然 X Window 的国际化/本地化支持在设置本地应用环境后，可以比较容易地通过其自身的输出处理机制以点阵的方式显示汉字，但是仍然存在一些缺陷：一是可操作性差，缩放效率低、效果差，难于满足对字形要求较为复杂的应用(如，字处理)需求，也不能适应显示设备分辨率的提高和图形技术的发展；二是点阵字形庞大，占用资源多，很难实现对 CJK 大字符集的支持。

为了在 X Window 的中文界面中提供对大字符集汉字字形的支持，可以采用两个方案：一是扩充 X Server 来支持汉字字形；二是利用 X Window 的字形服务器机制，开发一个支持大字符集汉字字形的字形服务器[18][19]。由于方案一在扩充 X Server 时需要源代码重新编译生成 X Server，这样会或多或少影响系统的稳定性。因此在"炎黄"平台的实现时，我们采用了开发中文字形服务器的方案。

X Window 字形服务器是在 X11R5 中提出的。它的出现减轻了 X Server 进行字形处理的负担，增加了字形处理的灵活性。通过字形服务器，无需对原系统做任何修改就可以实现对各种字形的支持，实用性强。字形服务器[20]如图 2-12 所示。

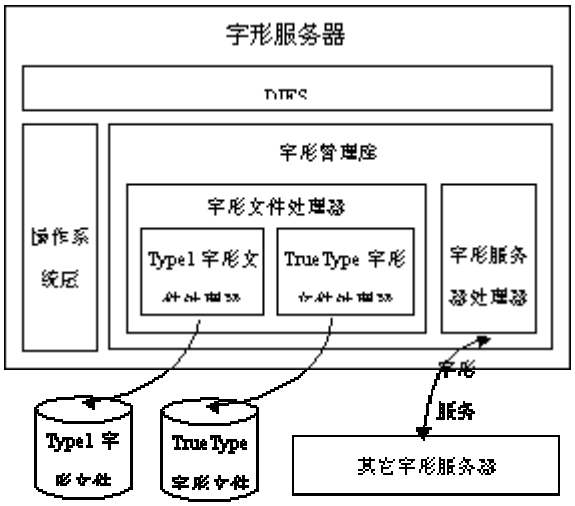

图 2-12. TrueType 字形服务器

在实施时，我们发现当应用程序(X Client)要求装入一个新字形时反应速度较慢，需要经过相当长的时间才能完成操作。经过分析，我们发现这是由于 X Server 在根据 X Client 的请求从字形服务器装入字形数据时，要求一次性从字形服务器返回整个字库中所有字符(glyph)的光栅化数据(包括字符的显示规格和点阵)造成的。我们称这种 X Server 一次性从字形服务器取回所有字符数据的策略为"一次装入"。采用这种策略显然是没有考虑中文等表意文字的特点，因为拼音文字字库一般很小(只有一、二百个字符)，一次性装入所有字符并不会增加太多负担，还可以减少通信开销，提高字形处理速度。但对于有六、七千，乃至数万汉字字形的中文字库(其中许多字符的使用频率是非常低)，这样做费时、费空间。

解决这个问题可以采用修改 X Server 的办法，但是在实际工程中是很困难的(原因同上)。因此，我们采用了与输入服务器相似的方法：

首先，在字形服务器对装入新字形请求的处理时，不返回字形库中的字符光栅化数据。而是在字形服务器内部采用字符缓冲机制：对于 6763 个 GB 汉字，随着需要不断调入缓冲区，但是不淘汰；对于 GB 以外的汉字，则采用与 2.3.2 中的活动字库算法处理。

其次，在为处理汉字输入而扩充的 Xlib 中间层中，增加请求字形服务函数(如，XDrawstring 等)的替代函数。这些函数"截获"所有对字形的请求，判断所要求的字形由谁提供支持；如果是 X Server，则调用 Xlib 中原有函数，否则将直接向字形服务器发送请求以取得光栅化数据并进行相关操作。其结构如图 2-13 所示。

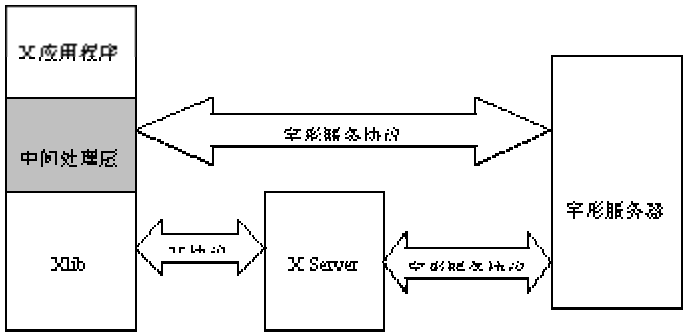

图 2-13. X Window 环境下的字形服务器

2.6 跨平台打印/输出子系统

在通常的中文平台中，显示部分与打印/输出子系统各自有一套字库的还原、处理程序，同时把打印/输出子系统把对打印机的控制操作和对字库的还原(光栅化)程序混成一体，并且必须与中文平台在同一台机器上。这样做虽然效率较高，但是系统间不能互相利用，造成人力和物力的浪费；而且，子系统作为一个整体，升级、更新(如，增加对 Type1 字形支持)都不够灵活。另外，由于曲线字库(如，TrueType 和 Type1)的还原程序复杂，字库文件大(仅支持一个 CJK 汉字的 TrueType 字库，如，宋体，就有 8 兆字节)，在每台主机都安装一份是很不经济的。
我们认为，应该把对字形的还原算法从该子系统中分离出来，形成独立的字形服务器，为网络中所有系统提供字形服务；而中文打印输出程序部分也可以根据应用程序的要求请求提供不同字形还原(光栅化)的字形服务器为其服务。这样做虽然会增加一些通信开销，但是它的优点也是显而易见的，而且独立的字形服务器不仅可以用于支持打印输出，而且可以支持图形界面，乃至字符界面的显示。

基于这种考虑，我们在"炎黄"平台中提出并实现了跨平台打印/输出子系统。该子系统主要是为中文平台提供各种中英文字符的输出与打印支持。随着显示和打印设备分辨率的提高，点阵输出已经不能满足应用的需求，因此，我们在"炎黄"平台中为用户提供了基于 TrueType 字形打印/输出子系统。

2.6.1 中文打印输出程序

该程序为整个系统提供与打印机型号无关、与字形技术无关的高品质汉字打印输出服务，其结构如图 2-14 所示。

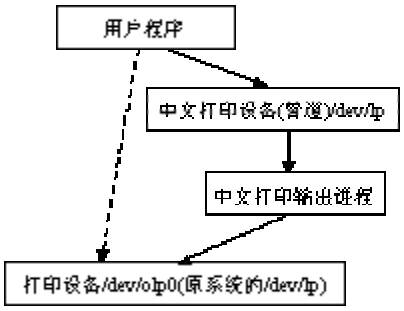

图 2-14. 中文打印服务程序

在实现时，我们将原系统的默认打印机设备/dev/lp 改名为/dev/olp，同时建立一个管道文件命名为/dev/lp。这样，用户程序原来输出到打印机设备/dev/lp 中的输出流都"透明"地进入了新的中文打印设备中。中文打印输出进程则在从该管道设备中读出用户程序要打印的输出流，送入字形服务器中还原为点阵信息，然后从打印机特征文件中取出打印机控制序列，将打印机设置成图形方式，并打印出用户程序输出的内容。

中文打印输出程序分为两个模块：一个负责接收用户程序的输出流和对打印机进行控制、操作；另一个负责与字形服务器进行通信，将要输出的字符串及对字体、字号的要求送给字形服务器，并等待服务器返回相应的点阵信息。

2.6.2 TrueType 字形服务器

TrueType 是苹果公司与微软公司[21]共同开发的一种数字化字形技术。TrueType 技术为屏幕显示和打印输出提供了高质量的字形。它同时还包括了一系列的特性使之易于使用。TrueType 字形技术主要包括两个部分：TrueType 字形文件和 TrueType 光栅化程序。本节只讨论 TrueType 字形服务器的结构，而不对 TrueType 字形光栅化的具体算法与程序[22]进行讨论。

TrueType 字形服务器的作用就是根据客户程序(打印输出程序、X Window 字形服务器等)的请求，对其所需要的汉字 TrueType 字形还原(光栅化)成点阵信息，再返回给客户程序。该服务器与客户程序之间通过 TCP/IP 网络通信，可以为不同的系统平台提供汉字字形输出服务。
与 2.4 节中的输入法程序相似，字形服务器也分为两部分：一部分负责与客户程序通信，接收客户程序发出的请求，调用相应的字形还原程序按客户程序要求的字体、字号将汉字编码串还原成点阵信息，然后送回客户程序。这部分对任何字形(TrueType、Type1 以及矢量字和点阵字形，乃至组合字形)都是相同的；由于处理中文常用的字形种类不外乎以上几种，因此，我们并没有将前一部分构成通用的函数库。另一部分进行具体的字形还原(光栅化)运算，即将传来的汉字编码串按字体、字号的要求还原成点阵信息。这部分根据字形种类(如，TrueType、Type1 等)的不同的而变化。

实现了跨平台打印/输出子系统，特别是提供对 TrueType 字形的支持后，就为进一步开发 WYSIWYG(所见即所得)的应用程序，如字处理软件打下了坚实的

基础。

2.7 多内码处理

至此，已经完成一个基于 GBK 代码体系，支持大字符集，实用化的开放系统中文平台。但是，要使该平台具有支持多种汉字内码的能力，还有一些工作要做。

2.7.1 输入

解决多内码环境下的汉字输入问题，一个简单的方法是按 2.4 节中的结构开发一些用于其它内码，如 Big5 的输入法(如，仓颉、注音等)，当用户进入该内码环境时挂接这些输入法。但是，由于内地用户对这些输入法并不很熟悉，因此，当他要在 Big5 环境下输入汉字时，这些输入法并不能对他提供多少帮助。对于港台用户亦是如此。

为解决这个问题，我们利用"炎黄"平台输入方法子系统的结构特点，在 PIMI 库中增加了相应的内码转换函数，主要包括，GBK 到 Big5、GBK 到 UTF8 等。当用户进入某个内码环境(如，Big5)时，打开 PIMI 库中相应的内码转换函数(GBK 到 Big5)。这时，用户仍然使用其习惯的 GB 或 GBK 输入法输入汉字，PIMI 库中的内码转换函数则把用户输入的 GB 或 GBK 候选字串转换成当前内码，然后显示在提示区供用户选择。

对于从简体汉字到繁体汉字转换中的"一对多"问题，只需要将转换后的多个可选字(词)全部提供给用户，由用户选择，转换函数无需进行处理。

2.7.2 显示与打印

解决多内码环境下的显示问题，采用的方法是在从字库读取字形信息(如，点阵)之前，将当前内码转换成 GBK 代码，然后再去读取字形信息并进行显示。由于 GBK 完全涵盖了其它汉字内码(如，Big5、UTF-8 等)的所有汉字，因此，用 GBK 字库完全可以实现多内码显示功能。只是在显示 Big5 内码时，其字形仍然是内地通用的宋体，而不是港台常用的明体而已。如需按明体显示与打印，只需要在进行内码切换的同时，切换相应的字库。打印的解决方法与显示相似，只是把内码转换加在中文打印输出程序向字形服务器发出字形请求之前即可。

参考文献

[1] Internationalization of AIX Software - A Programmer's Guide, IBM 1992, USA
[2]《信息技术 开放系统中文 API 界面规范(GB/T 16681-1996)》，中国标准出版社，1997，北京
[3] 韦福，《炎黄中文平台内码转换模块详细设计说明书》，技术报告，1998.6，北京
[4] 胡才勇、徐自亮编，《SCO OpenServer 系统管理指南》，清华大学出版社，1999.1，北京
[5] 彭斌、孙昱东等译校，《UNIX 系统 V 第 4 版程序员指南：STREAMS》，电子工业出版社，1992.1，北京

[6]《XENIX 开发系统 C 语言参考手册与库指南》，北京科学技术出版社，1990.4，北京
[7] 吴健、李国华、李祥凯，《开放式跨平台的输入方法子系统》，《软件学报(增刊)》，1999.6，p19~23
[8] 王旭，《炎黄中文平台汉字输入法程序详细设计说明书》，技术报告，1998.8，北京
[9] Robert W. Scheifler，X Window System Protocol，X Consortium Standard，X11R6，X Consortium，1994
[10] James Gettys, Robert W. Scheifler，Xlib--C Language X Interface，MIT X Consortium Standard，X11R6，X Consortium，1994
[11] Katsuhisa Yano and Yoshio Horiuchi，X11R6 Sample Implementation Frame Work，X Consortium，1994
[12] Yoshio Horiuchi，X Locale Database Definition，X Consortium，1994
[13] Takashi Fujiwara，The XIM Transport Specification, Rev0.1, X11R6，X Consortium，1994
[14] Masahiko Marita and Hidaki Hiura，The Input Method Protocol V1.0，X Consortium Standard，X11R6，X Consortium，1994
[15] David Rosenthal，Inter-Client Communication Conventions Manual V2.0，X Consortium Standard，X11R6，X Consortium，1994
[16] 叶以民，《X Window 环境中的通用中文支持系统》，'98 大连-香港国际计算机会议论文集，1998.10，大连，p497~500
[17] 王旭，《炎黄中文平台 X 窗口汉字输入法服务器程序详细设计说明书》，技术报告，1998.11，北京
[18] 叶以民、程波、孙玉芳、胡钦谙，《高品质汉字字形服务器及其实用化》，'98 中文信息处理国际会议论文集，清华大学出版社，1998.11，北京，p48~55
[19] 程波，《基于大字符集的 TrueType 汉字字形服务器》，硕士论文，1999.6，北京
[20] David Lemke，Font Server Implementation Overview，X Consortium，1991，USA
[21] Microsoft Corp.，TrueType Font File，1993，USA
[22] 叶以民、孙玉芳，《中文 TrueType Font Renderer -- X Server 的一个扩展》，信息就是力量--'97 香港-北京国际计算机会议论文集，清华大学出版社，1997.10，北京，p86~90