基于嵌入式 Linux 的媒体播放插件实现①

Media Plug-in Based on Embedded Linux

刘 巍 陈蜀宇 吴晓烁 (重庆大学 软件学院 重庆 400030)

摘 要: 针对目前嵌入式设备在网络媒体处理过程中浏览器与播放器无法集成、控制效率较低和缓冲区设置不合理等问题，以嵌入式 Linux 系统中广泛采用的浏览器为基础平台，描述了一个网络媒体播放插件的实现方法。实践证明该插件不但为用户提供了一个高度集成的浏览播放一体化平台，而且在设备资源的占用率与媒体播放效果之间做出了良好的平衡。

关键词: 嵌入式 Linux 播放插件 缓冲区控制策略

1 引言

1.1 文章结构

本文第 2 节介绍基于 gecko 核心的浏览器插件体系结构。第 3 节给出媒体播放插件的具体实现方法。第 4 节分析插件中两个重要的技术难点。第 5 节对前述的工作进行总结并说明该插件当前的应用情况。

1.2 基本介绍

随着泛嵌入式时代的到来，传统 PC 机与嵌入式设备之间的界限日渐模糊。许多原本基于 PC 机的功能和应用模式均被移植到了嵌入式设备之上。与此同时嵌入式 Linux 操作系统凭借其开放原代码、定制方便等优点正在嵌入式设备中得到越来越广泛的应用。伴随着无线网络技术的逐渐成熟和各种网络媒体服务（如在线电视、在线直播等）种类的不断增加，嵌入式终端设在提供传统信息处理功能的基础上也需要具有强大的媒体资源处理能力。虽然目前高端嵌入式设备大都配有浏览器和播放器等媒体信息处理软件，然而受制于嵌入式设备显示区域、内存容量等因素，这两个软件大都各自为阵分开工作，无法给用户提供一个统一的网络媒体处理平台。所以如何协调系统中播放器与浏览器处这两个网络流媒体处理软件间的关系，以最小的资源消耗代价提供给用户一个集成的、高效的、统一的网络媒体处理平台也就成为高端嵌入式设备中迫切需要解决的问题。本文描述了一个基于嵌入式 Linux 浏览器媒体播放插件的实现原理和关键技术。该插件对嵌入式 Linux 系统中的浏览器与播放器进行整合，以浏览器为载体形成了一个集成的、高效的、统一的、功能强大的网络媒体处理平台。

2 插件接口描述

目前 Linux 系统上流行的浏览器大都源自 Netscape 系列，主要包括 Mozilla 和 Firefox。该系列的浏览器支持一套名为 NPAPI 的标准插件接口[1]，插件开发者通过实现这套接口完成插件功能的编写(以下我们对插件的讨论均针对这套接口而言)。从物理形式上看，插件可被视为一个独立于浏览器之外的一个动态链接库[2]。浏览器运行时，会将符合要求的插件加载到内存中进行相关调用。从程序设计的角度来看，插件可以被视为是一个“类”，它既包含了用于描述插件内部状态的一组属性，又包括了一组供外部(浏览器程序)调用的接口。当一个插件被载入内存后，可以认为其是一个特定类的实例。此时插件对象的属性可以依据初始化对象的不同分为两种:标准数据成员和自定义数据成员。标准数据成员由调用插件的浏览器程序负责实例化(如插件工作窗口、工作模式、插件 URL 等)，而自定义数据成员则是指由用户负责初始化的数据成员(如自定义属性、选项等)。同样的，插件对象的方法也可以分为两种不同的类型：插件自定

① 基金项目:教育部新世纪优秀人才支持计划项目(NCET-04-0843);重庆市信息产业发展基金项目(200611009)
收稿时间:2008-08-28

义方法和浏览器定义方法[3]。浏览器定义方法是浏览器提供的方法，不需用户编写。该类方法主要用于程序设计者获取浏览器相关信息，其函数名以 NPN 为前缀。主要功能包括：在浏览器的窗口中创建工作窗口、设置浏览器头信息、请求 URL 数据流、向 URL 发送数据、确定浏览器版本、确定浏览器图形库版本等。插件自定义方法是插件开发者必须自己实现的方法，函数名大都以 NPP 为前缀。浏览器将按照一定的顺序和模式调用这些方法。可以说，正是这一套方法实现方式的多样性导致了不同插件功能的多样化。其主要功能包括：初始化插件实例、创建新数据流、接收数据流、接收浏览器分配窗口句柄等。

在明确了有关插件的一些基本概念之后，接下来我们来看看插件整个生命周期的活动过程。图 1 表示了一个数据流插件从加载到删除的整个过程：

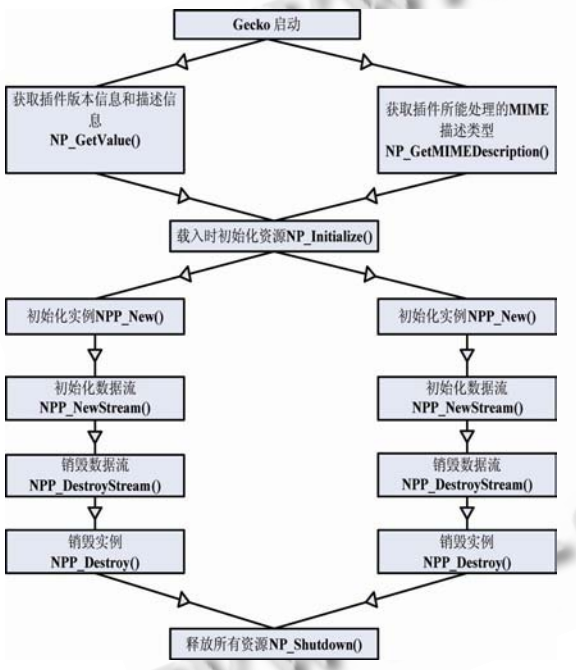

图 1　数据流插件生命周期图

①启动阶段：浏览器会从指定的文件夹获取插件描述信息和对应的 MIME 类型，并根据获取的信息在浏览器中对插件进行注册。

②初始化资源：当浏览器碰到自身无法解析的资源名称时，会查询启动阶段注册的插件信息，并将合适的插件链接库载入内存。当一个插件被载入后，它将首先调用函数 NP_ Initialize[4],这个函数负责分配内存和初始化以后各个实例将要使用的共 享资源。该函数在插件生命周期中仅被调用一次。

③初始化实例：在初始化资源完成以后,浏览器调用 NPP_New[4]创建实例。不同于 NP_Initialize[4]函数，NPP_New 会被每个插件实例调用。用户主要在这个函数中接收浏览器传递的参数来初始化实例私有的相关数据，主要参数包括插件 MIME 类型、实例数据、工作模式和 Html 参数等。

④初始化数据流：当插件初始化完成之后，浏览器会调用 NPP_NewStream[4]函数，通知插件存在一个待处理的数据流。于是在 NPP_NewStream 函数中，插件就可以为接收数据流做一些准备工作。准备工作完成之后浏览器会通过 NPP_WriteReady[4] 和 NPP_Write[4]函数传递数据流到插件中。

⑤销毁数据流：当浏览器完成数据传输或者数据传输被异常终止时 NPP_DestroyStream[4]函数会被浏览器调用。该函数主要会根据不同的数据流终止原因进行相应的处理，如重新请求数据流或者释放数据流占用的相关资源等。

⑥销毁实例和释放资源：当插件实体被删除时，浏览器会调用 NPP_Destroy[4]和 NP_Shutdown[4]两个函数。其中 NPP_Destroy 在每个插件实例被删除时都将被调用，主要用于释放插件实例的私有资源。而 NP_Shutdown 则将在所有该类型插件实例均被删除后调用，主要用于释放插件的共享资源。

3　媒体播放插件设计与实现

本文主要讨论的是基于浏览器的播放插件的实现，并不过多涉及浏览器与插件交互关系的初始化建立过程，所以下面将以分模块的形式介绍网络流媒体播放插件的具体实现，主要涉及上部分提及插件整个生命周期的 3、4、5、6 四个步骤。

3.1 插件的初始化

插件的初始化主要包括插件实体的初始化、插件窗口的初始化。其中插件实体初始化工作均在函数 NPP_New 调用中完成，主要实现了读取 html 页面参数(如窗口大小、媒体文件名等)初始化插件对象的相关属性，初始化线程控制参数等功能。当完成该项工作之后，浏览器调用插件的 NPP_SetWindow 完成窗口初始化工作。插件在该函数中接收来自浏览器的一个窗口句柄，并保存在插件对象的属性当中，该窗口将作为媒体播放窗口。在得到以上属性的取值之后，插

件会根据 Html 页面指定参数、控制台图片大小等因素调整播放窗口的大小，并在窗口中绘制播放器控制界面。

3.2 媒体数据流的处理

媒体数据流的初始化主要处理网络流媒体数据由浏览器传递到插件的过程，主要涉及 NPP_NewStream、NPP_WriteReady、NPP_Write 三个插件接口函数。浏览器发现网页中嵌入的媒体资源之后，将调用 NPP_Newstream 函数，该函数负责将有关数据流的信息传递给插件，并通知插件相应数据流可以接收。此时插件完成以下工作：

①对初始化过程中的 html 页面获取的媒体 URL 参数进行处理：去除浏览器 URL 中冗余信息(如%20 等)、将浏览器参数进行字符集转换（中文十六进制编码到中文 GBK 编码，仅对本地文件进行）。同时在这个过程中，插件会根据媒体数据流的传输速度确定合适的数据缓冲区大小。缓冲区大小的确定方法在下一个部分有较为详细的描述。

②播放器的建立：当插件处理媒体播放时，插件的实时控制功能与浏览器和插件的交互过程是并行的。我们将负责播放器控制的模块独立出来实现为一个独立的线程，这样即使在浏览器与插件的交互过程中，用户也可以通过插件流畅的控制播放器各种操作。当参数预处理工作完成之后，插件开启一个独立的播放器控制线程，该线程负责完成插件参数的预处理工作，并以数组的形式对相关的参数进行存储。完成以上处理后，由于该线程并不确定期望的媒体数据是否达到最小数据缓冲量，于是阻塞于一个条件信号量上。媒体数据通过 NPP_Write 函数从浏览器传送到插件，该函数由插件交互线程(原线程)负责控制。当传送的数据量达到缓冲数据量时，交互线程唤醒条件信号量，即唤醒被阻塞的播放器控制线程进行媒体播放工作。此时插件中线程的关系可见图 2。

③媒体播放：该部分由播放器控制线程控制，首先完成剩余媒体播放器参数的填写。然后通过 fork 系统调用生成一个新的进程运行播放器应用程序，完成媒体的播放功能。一个统一的媒体处理平台要求用户能够在浏览器页面中控制播放器的行为。然而此时插件和播放器分属不同的进程，如何在这两个进程之间建立一个高效的通信机制，成为必须要解决的一个难点问题。为了解决该问题，插件实现过程中采用了一种称为“双工管道”的进程控制方案，该方案将在下部分予以详细描述。

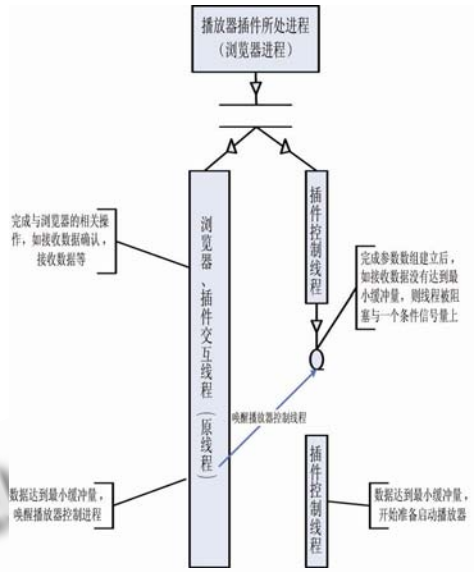

图 2　插件线程关系图

④数据流析构：该部分主要是 NPP_DestroyStream 函数中实现，完成了相关资源如线程、缓冲区的回收工作。

3.3 插件的析构

该部分完成的功能比较简单，主要功能实现在 NPP_Destroy 与 NP_Shutdown 函数中。在插件完成媒体播放之后，该部分负责释放插件对象占用的相关资源。主要包括播放器进程回收、窗口资源释放、插件对象参数释放等。

4 关键技术描述

4.1 “双工管道”机制

在前一部分提到，当浏览器开始播放网络媒体资源时，整个系统存在两个相关的进程：插件进程和播放器进程。为了使插件更好的模拟普通播放器的功能，插件必须要能够实时的控制播放器的播放过程，相关的功能操作包括暂停播放、停止播放、调节声音等。同时为了让插件能够向用户报告播放器的工作状态，提示用户进行相应的操作，插件也必须及时地捕捉播放器所产生的相关事件，如播放完成、播放错误等。如何在这两个进程之间提供高效的通信机制也就成为插件开发中必须解决的一个关键问题。我们针对此问题采用了一种名为“双工管道”的解决方案。“双工管道“解决方案的原理如图 3 所示。

在插件进程(父进程)开启子进程之前打开两个用于通信的管道。播放器进程(子进程)被打开后将自动继

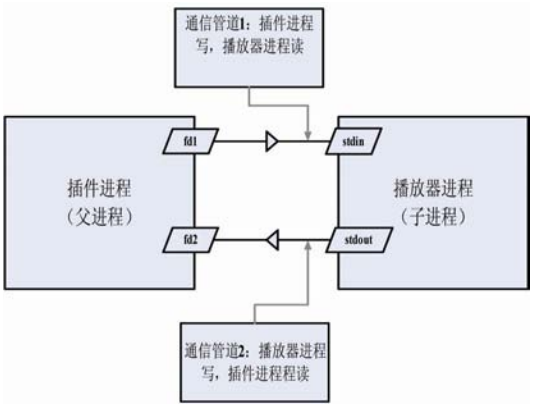

图 3　双工管道原理图

承这两个管道。此时这两个进程之间就具有了两个可用于通信的管道。由于 Linux 系统一般只支持单向通信管道使用[5]，此时我们需要在一个管道上关闭插件进程的读功能和播放器进程的写功能，形成一个插件进程写入播放器进程读出的单功能通信管道。对另一个管道进行相反的操作，形成一个由播放器进程写入插件进程读出的单功能通信管道。待管道建立后，再将播放器进程的标准输入端和标准输出端重定向到这两个管道上。此时插件进程的输入即被当作播放器的标准输入用以控制播放器的工作，而插件进程也可以通过读取播放器的标准输出来判断播放器工作状态。由于该方案在系统内核中只通过对内存页的单向读写实现，并不涉及其他复杂的数据处理流程（如网络封装，消息队列），所以这种解决方案具有极高的通信效率，能够较好的满足插件功能的需要。

4.2 客户端缓冲控制策略

由于该插件主要应用于资源受限的嵌入式设备中，出于平衡资源占用率和媒体播放效果的考虑，插件必须在每次处理媒体数据时根据当时网络状况动态地设定缓冲区的大小。如果缓冲区设置过小，网络传输速度也比较慢，那么媒体文件播放的流畅性将会受到很大的影响，出现跳帧等情况。相反如果缓冲区设置过大，而网络传输速度又比较快，那么该插件应用不但占用了过多的内存导致了整个嵌入式宿主设备的性能的降低，也造成了对网络传输速度的浪费。已有的缓冲区长度计算方法比较复杂，不容易在嵌入式设备上实现。为了更好的解决这个问题，我们在文献[6]的基础上设计实现了一种新的“客户端缓冲控制策略”的方法。该方法在每次确定缓存大小之前，先打开一个新的探测线程，该探测线程执行一段简单的媒体数据抓取代码，连续五次抓取服务器上的媒体数据，并将结果返回给插件。接下来插件首先根据返回结果计算每次抓取中的数据传输率(即数据量与时间的比值)，接着运用概率理论进行分析。我们将数据传输率看作一个可重复采样的独立样本，采用了样本均值与样本方差方差这两个样本特征值来进行分析。所以可以得到如下这两个计算公式:

样本均值:　$\bar{X} = \sum_{i=1}^{5} V_i / 5$

样本方差:　$S^2 = \sum (V_i - \bar{X})^2 / 5$

对应网络流媒体传输的参数，我们可以看到，样本均值反映了流媒体传输的平均速率 $\bar{V}$，而样本方差则较好的反映了流媒体传输的延时抖 T_{jitter}。接下来采用下面这个公式估算缓冲区的长度:

缓冲区长度:　$L_{buffer} = \beta \bar{V} T_{jitter}$

其中 β 为一个辅助系数，可通过经验尝试获得。

5　结束语

本文在阐明浏览器插件开发原理的基础上，提出了一个完整的嵌入式 Linux 下网络媒体播放插件的实现方案。该插件可完全屏蔽用户对播放器的依赖，使浏览器成为嵌入式设备上一个集各种媒体处理功能于一身的浏览播放一体化平台。该插件不但使嵌入式 Linux 平台的多媒体功能更加丰富灵活，也在一定程度上提高了资源受限的嵌入式设备对大容量流媒体资源的处理效果。目前该插件及相关项目成果已经成功应用于重庆网舟网络技术公司生产的 e-boat 网络计算机中，并取得较好的应用效果。

参考文献

1 Oliphant Z. Programming Netscape Plug-ins. 2007(2002-12-20).http://docs.rinet.ru/Plugi/.

2 Boswell King D. Creating Applications With Mozilla.[S.l.]: OREILLY & ASSOCIATES INC,2005.

3 蒋勇,杜中军,鞠飞.基于 RTP 协议的浏览器通用视频插件的实现.计算机应用研究, 2005,22(4):153－154.

4 Firefox Community. Gecko Plugin API Reference. 2007(2007-9-18)http://developer.mozilla.org/en/docs/Gecko_Plugin_API_Reference

5 Stevens WR, Rago Stephen A. Advanced Programming in the UNIX Environment. 2nded.北京:人民邮电出版社, 2006.

6 文远保,林建明.嵌入式流媒体客户端缓冲控制策略研究.华中科技大学学报(自然科学版), 2005,33(10): 84－86.