路径名空间与搜索

大家已经知道，在 QNX 这样的客户端服务器系统里，服务器（资源管理器）是通过注册一个路径名字来提供服务的。那么，当客户端给出一个名字，操作系统是怎样找到相对应的服务器的？这后面的水其实很深，让我们来一探究竟。

路径名管理器（pathmgr）

首先，所有名字的注册，搜索，注销都不是由内核管理的。是的，你没有看错，不在内核当中。而是由一个叫 pathmgr 的资源管理器来管理的。pathmgr 管理的是"路经名空间"这个资源，道俗点说就是根目录 "/"。

确实 pathmgr 是 procnto 进程的一部份，但它并不是在内核态运行的。

> pronto 是 QNX 的第一个进程，但并不意味着它就是微内核。procnto 进程其实是微内核+内存管理器+进程管理器+路径名管理器的组合。这几个核外的管理器，虽然是普通 QNX 系统一定会用到的。但是在极端的情况下，QNX 也可以做到完全不用这些管理器，只用微内核+定制的进程来实现一个系统，就像早期的 VxWorks 那样。只是那样的系统，等于完全取消了 QNX 的优势，所以虽然技术上可行，但并不流行。

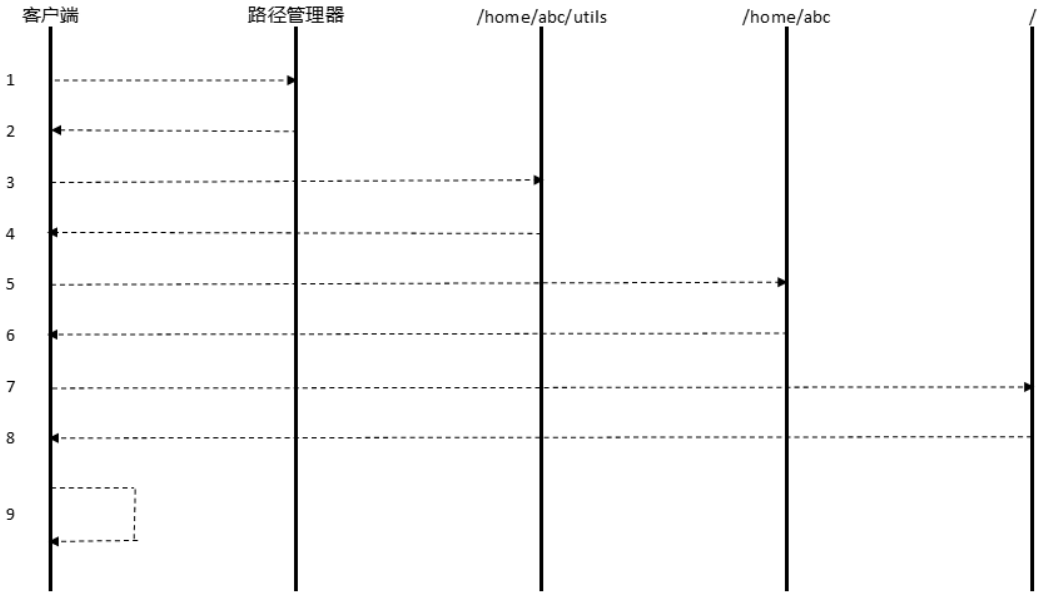
好了，讲回我们的路经名搜索。最简单的就是资源管理器注册"/dev/md5"，然后客户端程序 open("/dev/md5")就好了。你或许会奇怪这有什么复杂的？

可是资源管理器，不光可以注册一个文件，也是可以注册目录的。最容易想到的是文件系统，当你"把某某硬盘分区 mount 到 /home"时，文件系统资源管理器就会注册 "/home/"，意思是说"/home/ 以及至它下面所有的路经名，从此以后都归我管了"。如果你接下来 ls -l /home/user/.bashrc 的话，其实是把整个路径名都发给注册了 /home 的文件管理器，文件管理器就会去实际硬盘分区里找有没有/user/.bashrc 这个文件，然后返回状态。

如果资源管理器可以注册一个子目录，事情突然变得有趣起来。比如，管理器 A 注册了 /home/abc，管理器 B 注册了/home/abc/utils，然后客户端 open ("/home/abc/utils/readme")时，又是哪一个管理器来为客户端服务呢？

这个根据客户端要求的路径名寻找对应的资源管理器的过程，就叫做路径名搜索。基本上就是让客户端跟各个可能的管理器联系，由管理器回复客户端。

我们来看一下时序图



1. 客户端 open("/home/abc/utils/readme")，在 libc 的 open 函数里，首先开始的就是路径名搜索。第一步就是把整个路径名发送给路径名管理器 pathmgr
2. 路径名管理器根据传入的路径名，检索已经注册过的所有资源管理器，返回一个所有可能服务"/home/abc/utils/readme"的资源管理器的列表；在我们的例子里，这个列表包括了 "/home/abc/utils" 管理器，"/home/abc"管理器，还有"/"管理器
3. open() 函数接着会给列表中的"/home/abc/utils"资源管理器发送 io_open 消息，这个消息里想要搜索的相对路径名 ("readme")，意思是说"我要打开 readme 这个文件，是不是你负责？"
4. 资源管理器根据传入的路径名和自己掌握的资源来判断自己是不是应该提供服务，是的话，就可以调用 io_open 回调函数，处理后返回结果，open()就结束了；但如果资源管理器判断 io_open 里的路径名不由自己负责，资源管理器需要回复 ENOENT，POSIX 里这个错误的定义就是"找不到指定的文件"。我们这里假设资源管理器返回的是 ENOENT
5. 客户端的 open()函数，收到第一个资源管理器返回的 ENOENT 后，知道那个资源管理器不管 "/home/abc/utils/readme"，所以它会向第二个资源管理器（/home/abc)发送 io_open 消息，带有相对路径 ("utils/readme")
6. 第二个资源管理器依然回复 ENOENT

7. 客户端向第三个资源管理器发关相对路径进行查询（home/abc/utils/readme)，看看第三个管理器是不是认得这个名字。
8. 第三个管理器依然回复 ENOENT
9. 这时，客户端的 open()函数已经对可能服务”/home/abc/utils/readme” 的所有资源管理器进行了路径查询，找不到一个可以提供服务的管理器，所以最后把 ENOENT 写入 errno，并返回 -1 告知 open()调用失败了。

这个流程看上去很长，但是注意在第 4 步，第 6 步，第 8 步，只有当资源管理器返回 ENOENT 时，路径名搜索才会继续下去；如果资源管理器返回了 EOK，那意思就是”对，是我管的，已经准备好了”，那路径名搜索就结束了。还有一种情况就是资源管理器返回别的错误，比如 EPERM，那意思就是”是的，我管这个名字，但是你没有权限”；这时，路径名搜索也会结束，并将这个错误直接返回给用户。

这个通过向一连串资源管理器发送路径名请求的设计，背后的考虑是在通常情况下，“找得到文件”是大多数情形，“找不到" 是一个错误的情形，应该比较少；而在”找得到”的前提下，怎样更好地排列资源管理器列表的前后次序，让客户端更快地命中正确的资源管理器，从而减少不必要的消息传递，也就很有意思了。

QNX 的路径名管理器有一个"最长匹配优先"的规则。拿上面的例子来说，对于目标路径 "/home/abc/utils/readme"来说，有两个可能的资源管理器，（好吧，三个，因为 pathmgr 同时也是根目录”/”的资源管理器，等于所有路径名都由它兜底），与注册路径匹配长的，就是 "/home/abc/utils/"排在前面，而"/home/abc"的匹配度短一点，所以它排在第二顺位。

如果有两个管理器注册了相同的路径名呢？那么先注册的就排在了前面。（除非后注册的管理器使用了特殊的标记）。

介绍几个路径名管理下 QNX 特有的例子。

虚拟目录名

一个资源管理器可以注册一个目录，然后客户端对这目录下的任何文件操作，都会由这个管理器收到信息，那么完全可以用一个虚拟目录名，来提供一整套服务。这个在别的 Unix 也有，只是 QNX 下的非常灵活，因为名字不再依存于 inode，所以 resmgr_attach()一个调用就可以简单注册。（所以在 QNX 上，mount 不需要预先准备 mountpoint）

比如你可以写一个资源管理器，注册 /car/hvac/ 目录。 而 “ls /car/hvac”无非就是对你的资源管理器进行 opendir()/readdir()/closedir() 操作而已。只要你的服务器回复正确的数据，可以让客户端“看上去”像是你有好多子目录：

/car/hvac/ac/onoff
/car/hvac/ac/temperature
/car/hvac/heat/onoff
/car/hvac/fans/speed
....

对这些文件的读写，都会汇集到你的资源管理器，而根据传入来的相对目录，你也可以正确判断用户是要操作 "fans/speed"，还是 “ac/onoff"。

同样道理，你也可以用一个资源管理器，注册 /proc/sys/，然后提供 Linux 类似的 /proc/sys/ 服务了。

又或者，你可以注册 /mirror/；然后处理 ls -l /mirror/ftp.sjtu.edu.cn/ubuntu-cd/ 这样的请求，实现一个基于 ftp 的文件系统。

同名目录联合（Union）

对目录的“联合”这个概念，也是 QNX 独有的。比如有一个硬盘分区 1 上有 “/b/”, 和 “/d/” 两个目录；而硬盘分区 2 上有 “/a/”, “/c/” 两个目录，通常情况下它们可以分别 mount 到 /home1; /home2； 形成

/home1/b/
/home1/d/
/home2/a/
/home2/c/

但如果把他们同时 mount 到/home 以后，你会获得：

/home/a/
/home/b/
/home/c/
/home/d/

也就是说 opendir()/readdir() 这些操作，不是找到第一个服务器就会停止了的。这其实应该也是符合一般人想像的结果。

问题来了，如果两个硬盘分区，都有一个叫 "/readme" 的文件，当把他们联合到一起时，会发生什么？是不是应该有两个：

/home/readme
/home/readme

理论上确实是这样的，但是在同目录下有重名文件，这个会迷惑用户。所以事实上，当同一目录有重名文件时，QNX 会只显示一个。另一个经常让人迷惑的是，如果你 rm /home/reamde，你再 ls /home 的话，会发现 /home/readme 依然存在，因为 rm 只删除了一个资源管理器里的 readme，而另一个资源管理器里的 readme 还依然存在。

多个资源管理器注册同一路径名

也许你会奇怪为什么两个资源管理器要注册同一个路径名。除了“备份”这样一个原因外，还有可能可以进行“无间断升级”。

比如服务器 v1.0 通过注册 /car/speed 提供车速检索服务；经过一段时间以后，这服务器有了新的 v2.0 版本。一般的升级步骤当然是把前一个 v1.0 服务器停掉，然后启动 v2.0 服务器。但这意味着在哪怕只是短短的一瞬间，/car/speed 这个服务消失了。

更好的做法，是先启动 v2.0 服务器，让它也注册 /car/speed。但是，因为 v2.0 服务器比 v1.0 启动得晚，所有对 /car/speed 的访问依然会去到 v1.0 服务器的。这时候就需要 v2.0 服务器在注册时用到 _RESMGR_FLAG_BEFORE 这个标记了。这个告诉路径名管理器“虽然我注册得晚，但当有人查询 /car/speed 时，请把我放在第一个”。

这样，当 v2.0 启动后，所有 open("/car/speed")的客户端，都会先跟 v2.0 服务器通信了。而 v1.0 暂时不会退出舞台，它还需要为当前已经跟它建立了连接的客户端服务，一直到这些客户端结束连接（一旦 v2.0 启动后，v1.0 就不会有新的连接了，因为新的客户端都会连接到 v2.0）。

当 v1.0 的所有现行客户端都结束连接后，它就可以正式退出。这样我们从 v1.0 到 v2.0 就进行了一次不暂停服务的“无间断升级”。

有心人可能已经注意到了，使用 _RESMGR_FLAG_BEFORE 可以让后注册的服务器挡在先注册的服务器前面，这样就可以“劫持”某一个资源管理器了。是的，确实可以这样，这个基实就是 QNX 的 devc-ditto 服务的基础。

路径名管理器命名传递

QNX 跟别的 Unix 一样，也有一个标准的 ln 程序，通常是用来建立链接文件的，不管是硬链接还是符号链接，在文 件系统里有具体的含义，这个想来大家都知道。但是 QNX 有一个强大的扩展，

-P Create link in process manager prefix tree.

通常我们跟-s 连用，Unix 的文件链接，通常有一定的限制，是不是普通文件，会不会跨越不同分区等，但是 QNX 的 -P，是直接在路径名管理器里做重定向，因为所有的路径名搜索，第一步都是询问路径名管理器，基本上 -P 可以链接任何“路径名”。

 # ln -sP /dev/shmem /tmp

这个用共享内存做临时服务器的做法在没有外部存储的 QNX 系统上很常用。其实际的含义是说在路径管理器里建立一个重定向， 任何人访问 /tmp/... 下的任何文件，都会被 自动重定向到 /dev/shmem/... ，然后再开始路径名解析。

 # ln -sP /net/mediacenter/dev/snd /dev/snd

如果你的机器上没有声卡，但是另一台 mediacenter 上有，这个链接直接在本地机器上建立起了 /dev/snd 目录，本地的播放器程序无须任何修改，当它们试图 open("/dev/snd/...") 时，路径名管理器自动把它置换成 /net/mediacenter/dev/snd/...，从而让本地的播放器与 mediacenter 上的声卡资源管理器之间建立起连接和消息传递。 这就是 QNX 强大的“透明分布式处理”(QNET)，QNX 与 QNX 之间，可以轻松地跨设备协同。而无须为这种协同多写任何代码。

再搞个好玩的

 # /bin/hostname

localhost
 # ln -sP /bin/ls /bin/hostname
 # /bin/hostname
. .lastlogin .profile
.. .ph
 # rm /bin/hostname
 # /bin/hostname
localhost

这里发生了什么？可以看到一开始 /bin/hostname 是正常工作的；但强行将 /bin/ls“链接到”/bin/hostname 上，或者说，强行将 /bin/hostname 这个路径名指向/bin/ls 后，执行 /bin/hostname 就跟执行/bin/ls 一样。而 rm /bin/hostname 则删去了这个重定向，/bin/hostname 就又按硬盘上的 hostname 正确执行了…

结语

综上，这里解释了 QNX 路径名搜索的概念，以及一些细节。也许你会问，知道这些有什么用呢?

知道了路径名是怎么搜索的，就可以在系统设计时尽量避免一些可能降低系统性能的设计。比如，尽量不要交叉注册路径名，由一个资源管理器注册/media/db; 另一个注册/media/control; 会使得路径名搜索时简单明了；而如果一个资源管理器注册/media，另一个注册/media/player 的话，以后每次对 /media/player 下进行读写时，都有可能引起不必要的消息传递。

另一个比较常见的，因为路径名搜索要向不同的资源管理器查询，相对来说整个过程时间会比较长，所以尽量减少 open() 的次数，也可以提高效率。如果你需要频繁地与某个资源管理器通信，每次都 open() / write() / close() 它肯定不是一个好主意。最好是 open()一次以后记住 fd, 后面都通过 fd 来发送命令。