

微内核架构 与 " 互连互通 "

兼谈 QNX 透明分布式处理

2022-06-05

上个世纪 80 年代中叶，小型工作站一统天下，Sun Microsystems 如日中天。但也就是在那时候，他们就提出了“网络就是计算机”（The Network is the Computer）的理念。在 Internet 都没有普及的时代，这个理念实在有些先进，很多人甚至都不明白具体那是什么意思。

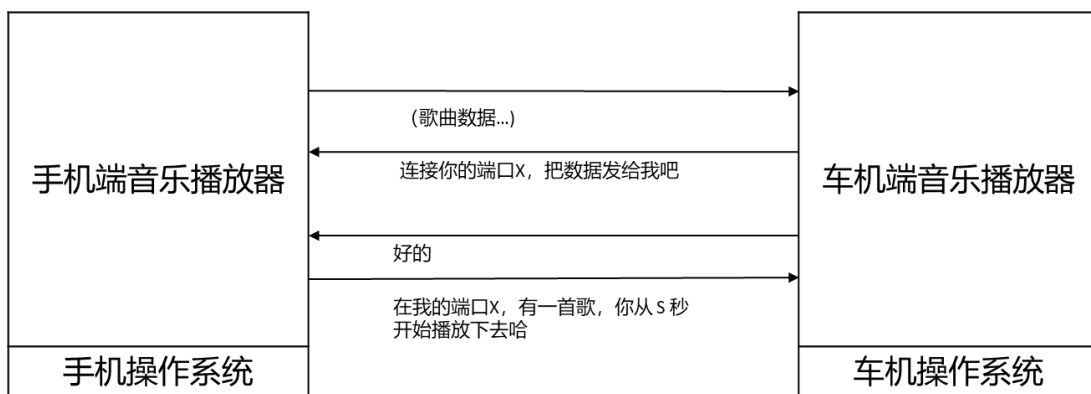
后来 Oracle 并购了 Sun Microsystems，在 Java 和 Database 的路上一路狂奔。网络反而顾得少了。到现在云计算，云平台比比即是，真正实现了“网络就是计算机”。移动互联网跟是把“互联互通”，“万物互联”推到了普通群众的手边。

那么，什么是互联互通？

互联互通

很多人以为，“互联互通”，不就是在手机上通话，打到一半以后可以把通话推送到大屏幕上吗？或者是手机上观看、操纵无人机上的摄像头？这些很难吗。现在的技术这样的 Application 完全都是可以做到的呀。

比如正在手机上播放着音乐，进到汽车座舱以后自动无缝切换到车机播放。这种功能远场实现的话，就是给手机和车机各写一个播放器，在一定的条件下（进了车，用户手势，UI 操作）手机端的播放器，发送一个 URL 给车机，告诉车机“这个歌（URL）从 xxx 秒开始播放”，然后手机停掉就是了。也许有人会问，那如果那首歌是在手机本地怎么办？也很简单，就是手机端的播放器，临时开个流媒体接口，让车机从手机端直接下音乐就可以了。



这个方案，显然不是互联互通的初衷。这里有几个问题。

- 1) 这个方案需要同时在手机端和车机端部署同一个音乐播放器，车机各种各样，不一定是安卓，一一适配工作量就不小
- 2) 车机端也许已经有了自己的播放器，但跟你的协议不一样
- 3) 根本车机就不允许你部署软件
- 4) 这个方案的“互联”是在 Application 部分，意味着如果不是播放音乐，而是播放视频，或者传递摄像头内容，那就必须在别的不同的 Application 里实现

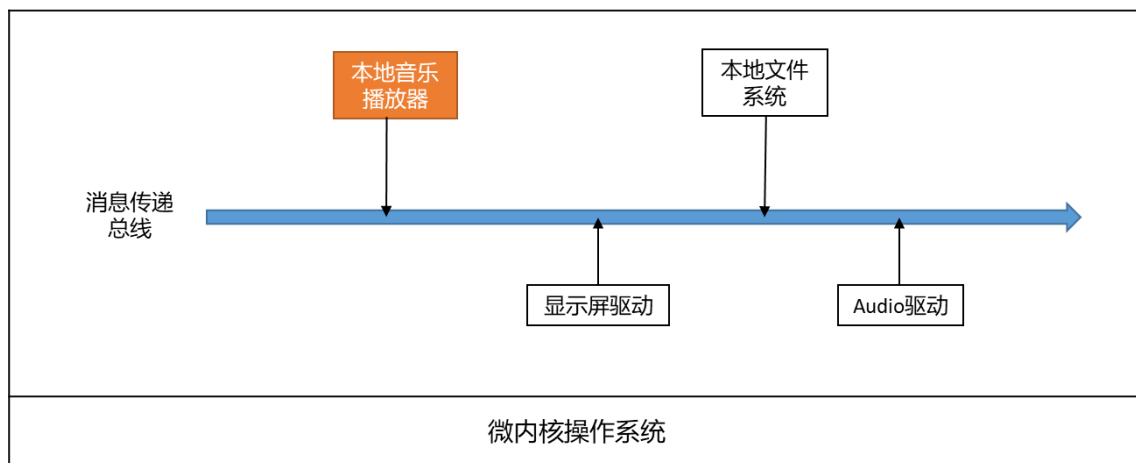
" 互连互通 " 需要解决这些问题：

- 能不能把 " 互连 " 的部份放到比如操作系统里去，这样在写 Application 时就不用特意关心 " 本地 "，" 远程 "，" 网络通信 " 这些问题了
- 有没有办法建立一个比较完整的 " 互通 " 协议，可以支撑比如播放音乐，视频，控制摄像头，开关空调， ... 等等各种应用场景？而不是需要每一个 Application 自己去重新定义通信协议。

答案是有的，一个基于微内核的系统给出了很好的解决基础。

微内核系统

你也许会问，为什么是微内核系统？因为，微内核系统有一个独一无二的优势，叫做消息传递。让我们来想一想，在一个微内核系统里，一个本地音乐播放器是怎么实现的？



在微内核上，当本地某一首歌被选中后，音乐播放器要做什么？

1. 发消息给 Audio 驱动，请求使用 Audio 通道
2. 发消息给 本地文件系统，请求打开被选中的歌（文件）
3. 发消息给 本地文件系统，请求顺序读取被打开的文件
4. 解码读取到的数据
5. 把数据发给 Audio 驱动，由 Audio 驱动去控制 Audio 芯片，播放歌曲
6. 发消息给显示屏，更新显示屏上的信息（播放时间，剩余时间，歌词 ...）

7. 重复上述 3-6，直到文件读完为止

可以看到，在微内核上的一个播放器，其主要工作，就是通过消息传递总线，给各个不同的驱动，子系统发送消息和数据，得到响应。

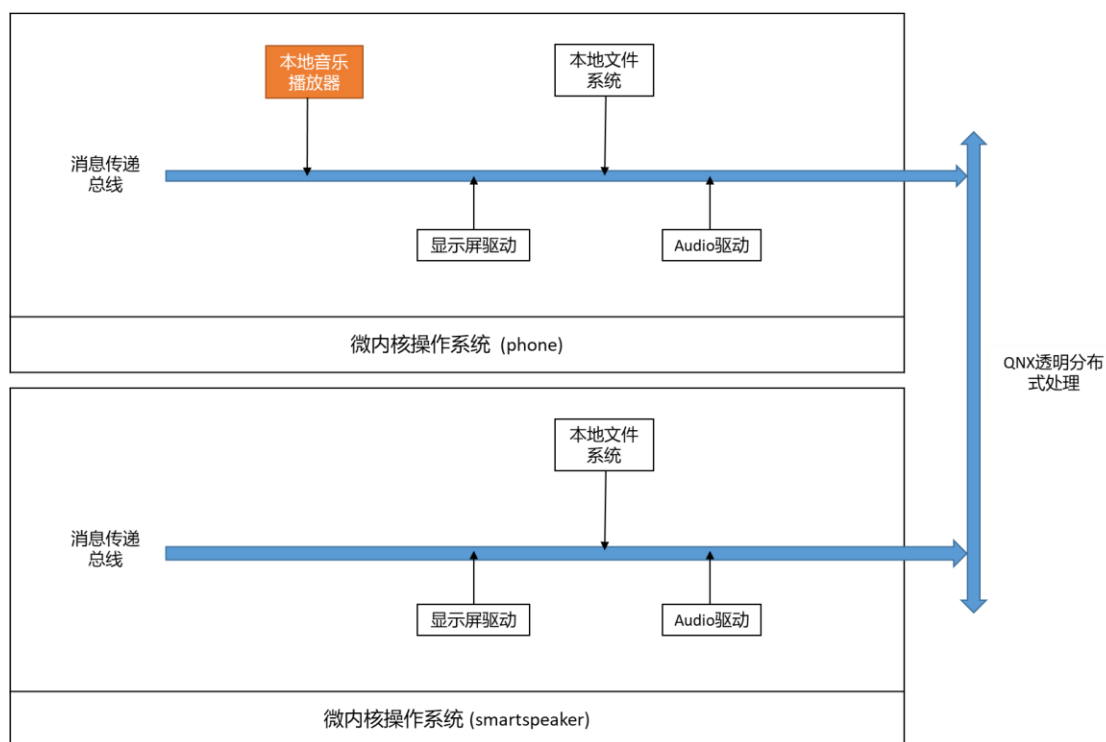
很容易可以想到，如果我们可以扩展消息传递总线，把两台微内核设备的消息传递总线打通的话。一个播放器，就可以给别的设备上的驱动发送消息和数据，这样无需修改播放器代码，就可以轻松实现互联了。

这个扩展消息传递总线的功能，在 QNX 上，就叫做“透明分布式处理（Transparent Distributed Processing）”。

QNX 的透明分布式处理

QNX 透明分布式处理，原理很简单，就是打通两个 QNX 之间的消息传递，让一个 QNX 系统里的进程，可以跟另一个 QNX 系统里的进程，进行“透明”的信息传递。透明的意思是，进程不需要做什么特殊的更改，就可以直接和远程的进程通信了。

我们来看看，在两个 QNX 系统之间的播放器可以怎么工作？



比如上图，我们有一个叫“phone”的 QNX 系统，还有一个叫“smartspeaker”的 QNX 系统，通过透明分布式处理已经互相连上了。这意味着，“phone”里的进程，除了可以通过“phone”里的消息传递总线，跟 phone 里的驱动、子系统通信外；还可以直接跟 smartspeaker 里的驱动、子系统进行通信

我们可以看到，只有 phone 里有本地音乐播放器。这个播放器，可以如前述一样在 phone 里播放本地音乐。

1. 发消息给 phone 的 Audio 驱动，请求使用 Audio 通道
2. 发消息给 phone 的文件系统，请求打开被选中的歌（文件）
3. 发消息给 phone 的文件系统，请求顺序读取被打开的文件
4. 解码读取到的数据
5. 把数据发给 phone 的 Audio 驱动，由 Audio 驱动去控制 Audio 芯片，播放歌曲
6. 发消息给 phone 的显示屏，更新显示屏上的信息（播放时间，剩余时间，歌词 ...）
7. 重复上述 3-6，直到文件读完为止

但如果在 7) 里面，突然决定这首歌需要在 smartspeaker 上播出会怎么样呢？

8. 发消息给 smartspeaker 的 Audio 驱动，请求使用 Audio 通道
9. 发消息给 front 的文件系统，请求顺序读取被打开的文件
10. 解码读取到的数据
11. 把数据发给准备好了的 smartspeaker 的 Audio 驱动，由 Audio 驱动去控制 Audio 芯片，播放歌曲
12. 发消息给 phone 的显示屏，更新显示屏上的信息（播放时间，剩余时间，歌词 ...）
13. 重复 9-12，直到文件读完为止

所以，只要把数据（通过消息传递，经过透明分布总线）发送给另一台机器的 Audio 驱动，就可以在远程播放出声音来了。而播放器程序本身，不需要做任何的改变。甚至不需要专门的网络通信代码。

通信协议

我们在一开始的时候，提到“互连互通”，有两个重要的因素，一是通过系统能够通过网络互连。这样，程序就可以不必额外关心提供服务的服务器是在本地还是在远程了。但是，还有一个问题，怎样才能保证一个音乐播放器，和一个摄像头程序，都能够跟远程互连呢？正常来讲，这两个程序需要用到的通信协议肯定是不一样的。如果远程没有一个跟本地匹配的音乐播放器程序，或是摄像头程序，我怎么才能“互连互通”呢？

这个问题在 QNX 上已经很好地解决了。

大家都知道微内核结构，是一个基于消息传递的系统，但是，具体要传递什么消息才能让一个系统正确地跑起来呢？在 QNX 上，已经为所有的文件处理定义好了消息。也就是说，你要打开一个文件，读写一个文件，seek/state/devctl ... 所有 POSIX 定义好的文件操作，都有相应的标准消息可以用。详细细节，可以参见《QNX 资源管理器》一文。

你可能疑惑，就算有这些消息，那也只是文件操作呀，跟协议有什么关系？可是如果你记得，Unix 设计的一大精要就是所有的设备都当作文件进行操作的。要使用串口，你也是 open/read/write 一个叫 /dev/ser1

的文件；要使用网络 socket 接口，虽然形式上 QNX 为了跟别的系统兼容也叫 socket/send/rcv/...，但是，一个 socket(AF_INET, SOCK_STREAM, 0)，最后其实也就是做了一个 open("/dev/socket/1", ...)的操作，一个 send()，无非最后转换为发送一个 IO_WRITE 请求。所以在 QNX 上的应用程序，很大一部份都是在用标准的文件接口同各种资源管理器进行消息传递啊。

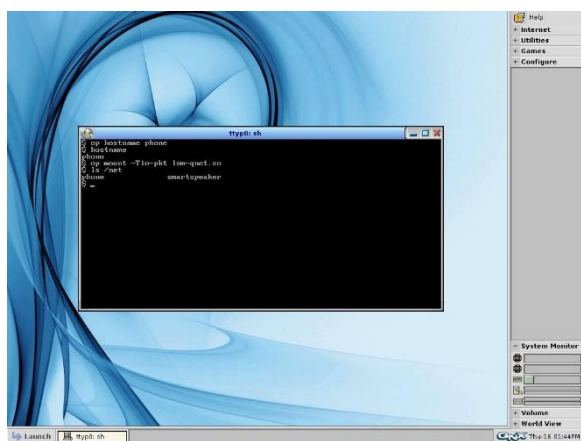
一个音乐播放器，就是给文件子系统发送 open/read/消息来获得要播放的本地歌曲；同时 open/write 解好码的数据到 Audio 资源管理器（声卡驱动）；一个摄像头程序，就是 open()摄像头资源管理器（摄像头驱动），通过 read()来取得摄像头数据，通过 devctl() 来控制摄像头；

这些 QNX 上的应用程序，说到底都是在跟资源管理器进行标准的文件操作消息传递；也就是说，有了 QNX 透明分布式处理，这些程序，就可以直接跟远程的 Audio 资源管理器，或者摄像头管理器通信了。协议就是标准的 QNX 文件处理消息，不需要再有特殊的网络专用协议了。

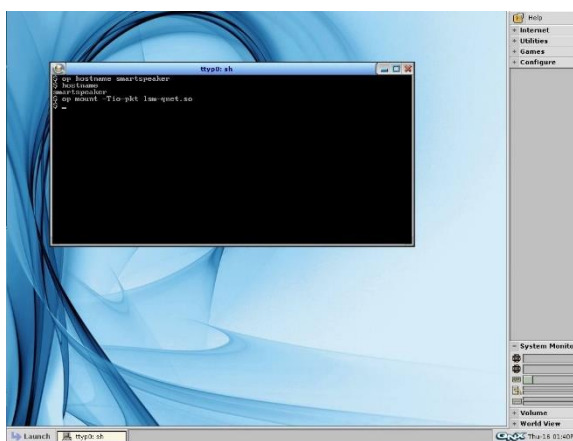
演示与练习

上面这些解释可能依然不太具像，让我们实际操作一下。

首先，我们需要准备两个 QNX 系统。如果你有两个 QNX 机器可用，那最好。不然的话，可以试试装虚拟机。QNX 7.1 在网上可以申请评估版，也可以直接生成 QEMU 或者 VMWare 的虚拟机。透明分布式处理默认是用以太网通信，所以两台 QNX 机器最好能用以太网互联。如果装虚拟机，可以考虑两台虚拟机跑在同一个宿主机上，这样就可以保证两台虚拟机可以通过以太网互联。



QNX 虚拟机 "phone"



QNX 虚拟机 "smartspeaker"

先让我们设一下机器名，然后在两台机器上，我们分别启动 QNX 透明分布处理功能：

```
ttyp0: sh
$ hostname
localhost
$ op hostname phone
$ hostname
phone
$ op mount -Tio-pkt lsm-quot.so
$ ls /net
smartspeaker
$ ls -l /net/smartspeaker/dev/snd/
total 0
-rw-rw-rw- 1 root root 0 Jun 17 10:53 controlC0
-rw-rw-rw- 1 root root 0 Jun 17 10:53 mixerC000
-rw-rw-rw- 1 root root 0 Jun 17 10:53 pcmC000c
-rw-rw-rw- 1 root root 0 Jun 17 10:53 pcmC000p
-rw-rw-rw- 1 root root 0 Jun 17 10:53 pcmC001p
lrwxrwxrwx 1 root root 0 Jun 17 10:53 pcmPreferredc -> pcmC00
0c
lrwxrwxrwx 1 root root 0 Jun 17 10:53 pcmPreferredp -> pcmC00
1p
$ =
```

```
ttyp0: sh
$ hostname
localhost
$ op hostname smartspeaker
$ hostname
smartspeaker
$ op mount -Tio-pkt lsm-quot.so
$ =
```

可以看到，系统里多了 /net 这样一个目录，我们可以能过这个路径名直接访问远程的路径名系统。

我们来看一下播放器的例子，在 QNX 7.1 里，提供了一个 wave 程序，可以直接播放 wave 文件。wave 有一个 -a 参数，可以指定播放的设备。（如果你的版本低于 QNX7，-a 参数不能指定设备名，我放了一个修改过的版本在这里）

我们在系统里放一个可以播放的 wave 文件。让我们试一下：


```
ttyp0: wave
$ hostname
phone
$ ./wave -a /dev/snd/pcmPreferredp starwar.wav
Using device /dev/snd/pcmPreferredp
SampleRate = 16000, Channels = 2, SampleBits = 16
Format Signed 16-bit Little Endian
Frag Size 1364
Rate 16000
Mixer Pcm Group [Wave playback channel]
-
```

播放的命令是：

wave -a /dev/snd/pcmPreferredp starwar.wav

在这里，“-a”后指向的是默认的 Audio 驱动文件名。

接下来，我们用 mix_ctl 程序把 audio 设成静音，再播一遍，你会发现因为本地的喇叭被静音了，所以什么都听不到。



```
ttp0: wave
$ mix_ctl group master volume=0
'Master',0 - Playback Group
  Capabilities - Volume Jointly-Mute
  Channels - Front-Left Front-Right
  Volume Range - minimum=0, maximum=63
  Channel 0 Front-Left - 0 ( 0%)
  Channel 1 Front-Right - 0 ( 0%)
$ ./wave -a /dev/snd/pcmPreferredp starwar.wav
Using device /dev/snd/pcmPreferredp
SampleRate = 16000, Channels = 2, SampleBits = 16
Format Signed 16-bit Little Endian
Frag Size 1364
Rate 16000
Mixer Pcm Group [Wave playback channel]
-
```

那让我们在远程播放一下，只要这样就行了：



```
ttp0: wave
$ mix_ctl group master volume=0
'Master',0 - Playback Group
  Capabilities - Volume Jointly-Mute
  Channels - Front-Left Front-Right
  Volume Range - minimum=0, maximum=63
  Channel 0 Front-Left - 0 ( 0%)
  Channel 1 Front-Right - 0 ( 0%)
$ ./wave -a /dev/snd/pcmPreferredp starwar.wav
Using device /dev/snd/pcmPreferredp
SampleRate = 16000, Channels = 2, SampleBits = 16
Format Signed 16-bit Little Endian
Frag Size 1364
Rate 16000
Mixer Pcm Group [Wave playback channel]
-

$ ./wave -a /net/smartspeaker/dev/snd/pcmPreferredp starwar.wav
Using device /net/smartspeaker/dev/snd/pcmPreferredp
SampleRate = 16000, Channels = 2, SampleBits = 16
Format Signed 16-bit Little Endian
Frag Size 1364
Rate 16000
Mixer Pcm Group [Wave playback channel]
-
```

`./wave -a /net/smartspeaker/dev/snd/pcmPreferredp startwar.wav`

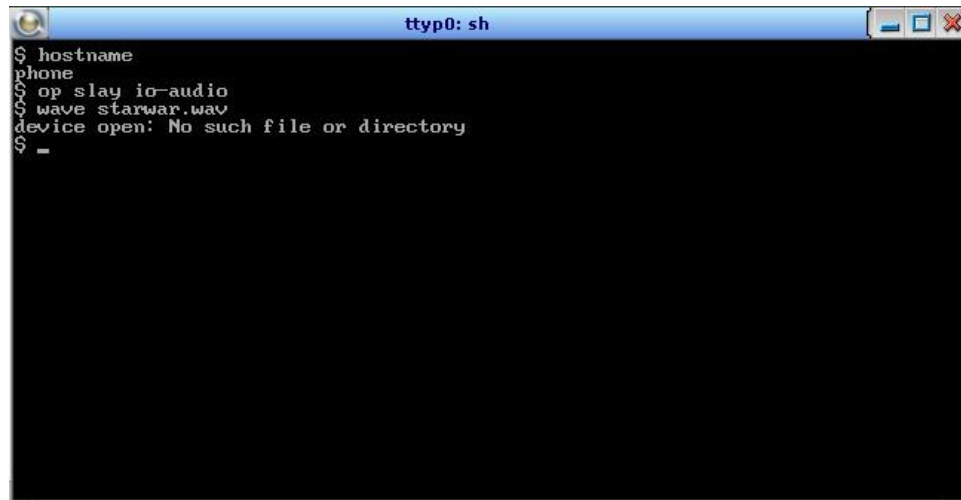
你看，只要把要打开的设备，直接指向远程的机器，声音就能在 smartspeaker 的喇叭里播放出来了。因为两个虚拟机在同一个宿主机上，所以你又能从你的宿主机上听到这首歌了。

wave.c 文件的源码在 QNX 的文档里就有。

<https://www.qnx.com/developers/docs/7.1/index.html#com.qnx.doc.neutrino.audio/topic/wavec.html>

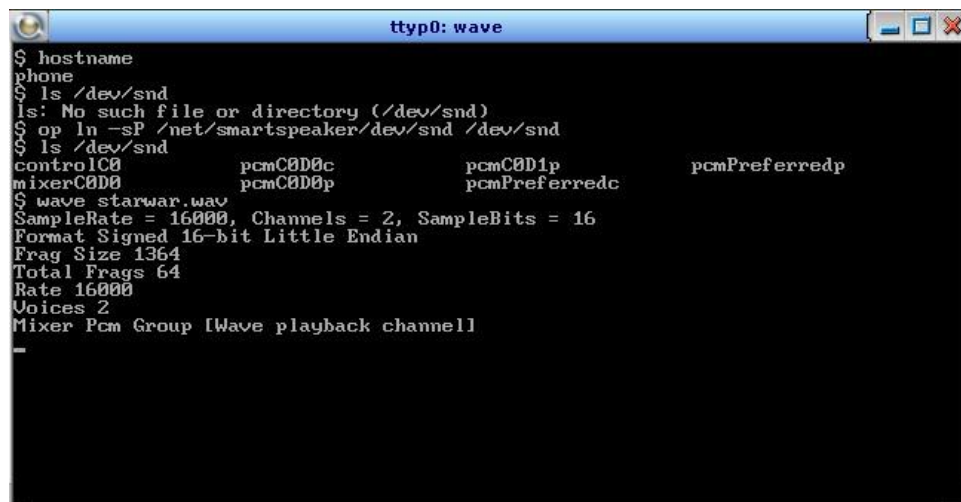
可以看到源码本身完全不需要关心播放设备在不在远程。只是通过 `-a` 指定要播放的设备就可以。

我们再看一个例子，假设 phone 这个虚拟机上根本就没有喇叭（你可以修改虚拟机配置，我这里直接把 io-audio 杀了），可以看到，wave 播放器根本无法工作，因为无法打开本地的驱动。



```
$ hostname
phone
$ op slay io-audio
$ wave starwar.wav
device open: No such file or directory
$ _
```

但是我们知道在网上 smartspeaker 上是有喇叭的，除了在 wave 启动时指向 smartspeaker 上的设备，我们还可以这样做：



```
$ hostname
phone
$ ls /dev/snd
ls: No such file or directory (/dev/snd)
$ op ln -sP /net/smartspeaker/dev/snd /dev/snd
$ ls /dev/snd
controlC0      pcmC0D0c      pcmC0D1p      pcmPreferredp
mixerC0D0      pcmC0D0p      pcmPreferredc
$ wave starwar.wav
SampleRate = 16000, Channels = 2, SampleBits = 16
Format Signed 16-bit Little Endian
Frag Size 1364
Total Frags 64
Rate 16000
Voices 2
Mixer Pcm Group [Wave playback channel]
_
```

`ln -sP /net/smartspeaker/dev/snd /dev/snd`

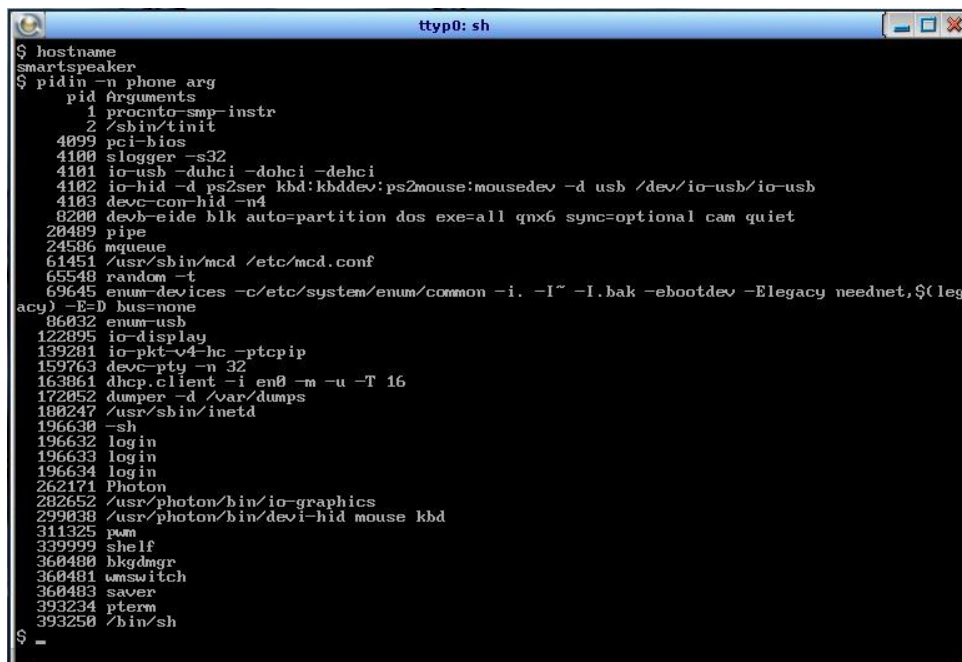
对，我会在 phone 里做一个 " 符号链接 "，这里参数-P 是 QNX 特有的，意思是当解析 "/dev/snd" 这个路径名时，把它解析成 "/net/smartspeaker/dev/snd"，这样，等于直接把 smartspeaker 上的 audio 驱动透出到 phone 上来了。（关于路径名解释，可以参考这篇文章）

这样，当 wave 程序试图使用默认播放器 /dev/snd/pcmPreferredp 里，就会自动地跟远端的 smartspeaker 里的 audio 驱动建立连接，发送消息，从而播放本地歌曲了。

一些透明分布处理的扩展

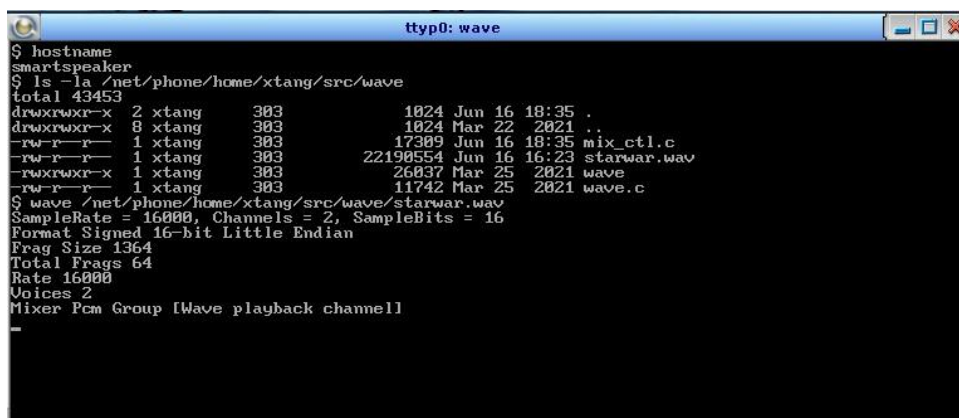
除了上面这个能过 wave 播放器的例子，其实在 QNX 的透明分布处理上，还有很多扩展应用。

比如直接通过指定电脑名字，查看远程电脑里启动的进程：



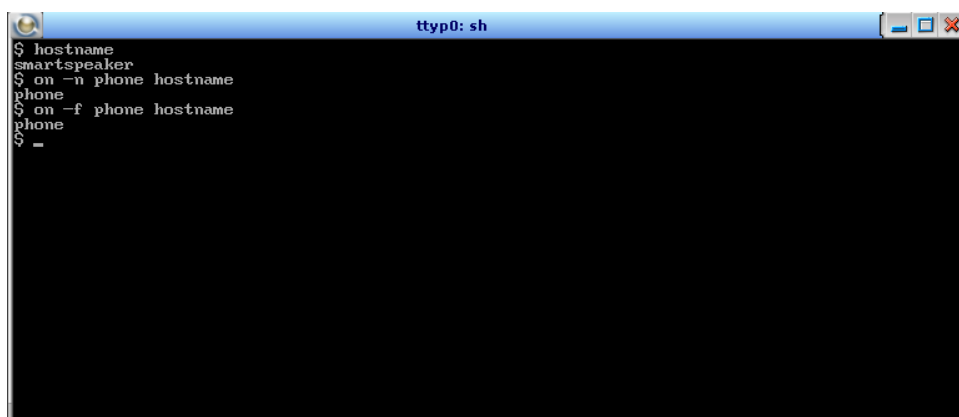
```
$ hostname
smartspeaker
$ ps
  pid Arguments
    1 procinfo-smp-instr
    2 /sbin/tinit
 4099 pci-bios
 4100 slogger -s32
 4101 io-usb -duhci -dohci -dehci
 4102 io-hid -d ps2ser kbd:kbddev:ps2mouse:mousedev -d usb /dev/io-usb/io-usb
 4103 devc-con-hid -n4
 8200 devb-eide blk auto=partition dos exe=all qnx6 sync=optional cam quiet
20489 pipe
24586 mqueue
61451 /usr/sbin/mcd /etc/mcd.conf
65548 random -t
69645 enum-devices -c/etc/system/enum/common -i. -I~ -I.bak -ebootdev -Elegacy neednet,$(leg
acy) -E=D bus=none
86032 enum-usb
122895 io-display
139281 io-pkt-v4-hc -ptcpip
159763 devc-pty -n 32
163861 dhcp.client -i en0 -m -u -T 16
172052 dumper -d /var/dumps
180247 /usr/sbin/inetd
196630 -sh
196632 login
196633 login
196634 login
262171 Photon
282652 /usr/photon/bin/io-graphics
299038 /usr/photon/bin/devi-hid mouse kbd
311325 pwm
339999 shelf
360480 bkgrmgr
360481 wmswitch
360483 saver
393234 pterm
393250 /bin/sh
$ _
```

查看远程的文件系统，播放远程的歌曲：



```
$ hostname
smartspeaker
$ ls -la /net/phone/home/xtang/src/wave
total 43456
drwxrwxr-x 2 xtang 303 1024 Jun 16 18:35 .
drwxrwxr-x 8 xtang 303 1024 Mar 22 2021 ..
-rw-r--r-- 1 xtang 303 17309 Jun 16 18:35 mix_ctl.c
-rw-r--r-- 1 xtang 303 22190554 Jun 16 16:23 starwar.wav
-rwxrwxr-x 1 xtang 303 26037 Mar 25 2021 wave
-rw-r--r-- 1 xtang 303 11742 Mar 25 2021 wave.c
$ wave /net/phone/home/xtang/src/wave/starwar.wav
SampleRate = 16000, Channels = 2, SampleBits = 16
Format Signed 16-bit Little Endian
Frag Size 1364
Total Frags 64
Rate 16000
Voices 2
Mixer Pcm Group [Wave playback channel]
```

在远程电脑里启动进程：



```
$ hostname
smartspeaker
$ on -n phone hostname
phone
$ on -f phone hostname
phone
$ -
```

这里解释一下 on -n 和 on -f 的区别在于，虽然看上去结果是一样的。在 smartspeaker 上，on -n 的时候，用的是在本地机器里的 hostname (/net/smartspeaker/bin/hostname)；进程管理器打开本地/bin/hostname，把它读到远程的/net/phone 的内存里，然后运行。而 on -f 的时候，用的是远程机器里的 hostname (/net/phone/bin/hostname)

衍生讨论

可以看到，微内核的消息传递可以很容易地扩展到跨设备的信息传递，从而达到“互连”的状态。然后 QNX 的资源管理器消息定义，又可以使在不同设备间的通信“互通”。

QNX 的“用文件操作接口来操作硬件”，跟安卓的“硬件抽象层”(Hardware Adaptation Layer)背后的思路，其实是一致的。想一想，如果 HAL 层的所有 API 都有对应的消息传递的话，那是不是所有安卓上的 App，都可以控制远程安卓上的别的设备了？

QNX 的透明分布式处理，本身是一个网络协议，默认是用以太网做数据传输层。但也可以加参数使它用 IP 做数据传输层。甚至还可以自己给 io-pkt 编写传输层驱动（比如通过共享内存）

大部份的 QNX 系统，都是在一个网络里有一些已知的，静态的节点。如果想在有多个不确定设备的情况下，需要考虑的事情就更多。怎样加入退出网络，怎样发布和搜索网络上的服务，怎样认证，以及保证网上传输数据的案全性，这些都需要额外地考虑与处理。

还有就是 QNX 的这个基于以太网调优的分布式处理系统，如果在 Wifi 或别的传输层上传，怎样才能保证网络的稳定，以及达到足够的传输速度，这些都需要加以特别的注意。