VxWorks6.8 操作系统下 NVMe 驱动设计

盘勇军，黄伯铮
（中国航空无线电电子研究所，上海 200241）

[摘 要] 非易失存储器（NVMe）协议是寄存器级的传输协议，用于实现主控软件与非易失存储设备间的数据传输，其在设计时针对固态硬盘（SSD）特性优化了命令发送和完成路径，通过 PCIe 链路传输数据，利用 PCIe 高带宽实现高速数据传输。阐述了 VxWorks 6.8 实时操作系统下 NVMe 驱动设计流程，实现了 NVMe 协议的基本功能，并测试验证其功能和效率；测试结果表明，该系统下 NVMe 驱动发挥了较高的性能，单向数据传输速率约为 700 MB/s（PCIe1.0×4），相较于 SATA 接口 SSD 有极大的性能提升，具有广泛的应用价值。

[关键词] 非易失存储器; VxWorks;PCIe; 固态硬盘
[中图分类号] TP316.2 [文献标识码]A [DOI 编码] 10.3969/j.issn. 1006-141X.2017.04.07
[文章编号]1006-141X（2017）04-0032-006

Design of NVMe Driver Based on VxWorks 6.8

PAN Yong-jun，HUANG Bo-zheng
(China National Aeronautical Radio Electronics Research Institute, Shanghai 200241, China)

Abstract: NVM Express(NVMe) is a register level interface that allows host software to communicate with a non-volatile memory subsystem such as solid state drives(SSD).This interface provides optimized command submission and completion paths for SSD, typically attached to PCI Express interface. Design process of NVMe driver software based on VxWorks 6.8 operating system is described, and NVMe fundamental functions are implemented. Test results show that this NVMe driver has relatively high performance, and the bandwidth is about 700 MB/s(PCIe1.0×4),which creates a significant improvement performance compared with SATA.

Key words: non-volatile memory express (NVMe); VxWorks; PCIe; solid state drive (SSD)

0 引言

固态硬盘（SSD：Solid State Drive）以 NAND flash 为存储介质，具有传输速率高、可靠性高、体积小、功耗低等特点，在航电机载数据存储系统中应用广泛。SSD 一般为 SATA 接口，运行 SATA 协议或 AHCI 协议，SATA 3.0 理论最大传输速率为 600 MB/s，在实际机载环境下，单张 SATA 卡的最大传输速率约为 460 MB/s。随着机载数据总量的爆发式增长和各类型数据传输带宽的显著提高，SATA 接口的单张 SSD 已不能满足 TB（terabyte）级别海量数据的记录要求。运行非易失存储器（NVMe：Non-Volatile Memory express）协议[1-2] 的 SSD 突破了 SATA 接口 SSD 速率传输的瓶颈，极大地提升了数据传输带宽和指令并行处理能力，极其适用于海量机载数据的记录。NVMe 协议专为 SSD 设计，针对 SSD 特点设计多达 65535 个命令队列（单个队列可容纳 64K 条指令），实现

高并发能力和高带宽，释放 SSD 全部潜能。NVMe 物理层采用 PCIe 通道进行数据传输，充分利用了 PCIe 的高带宽和扩展性。PCIe 单向传输带宽理论值分别为 250 MB/s（PCIe1.0）、500 MB/s（PCIe2.0）、1 GB/s（PCIe3.0），4 路绑定的 PCIe 单向传输带宽分别为 1 GB/s、2 GB/s、4 GB/s，远高于 SATA 3.0 的理论带宽。本文详细阐述了 NVMe 协议数据传输实现原理，并基于 VxWorks 6.8 操作系统开发出 NVMe 驱动，经测试其具有较高的性能，可应用于海量机载记录系统。

1　NVMe 命令传输

NVMe 是一个公开的标准和信息集合,在设计时充分考虑了存储介质和传输过程，优化数据传输路径和传输方式，极大提高了数据带宽和每秒的输入输出操作量，并降低了数据访问延时，释放了非易失存储（NVM：Non-Volatile Memory）的最大潜能。NVMe 规范了 PCIe 接口闪存存储标准，定义了寄存器级的接口和操作指令集，主机端软件可通过该接口和一个 NVM 子系统进行通信。NVMe 在命令发射和完成时无需不可缓存的寄存器读操作，命令发射最多需要一个寄存器写操作，且所有用于完成读写请求的信息均包含于 64 字节的命令内，I/O 操作效率高。

NVMe 协议规定了命令传输使用队列机制，每一个命令队列包含一个发送队列和一个完成队列，多个发送队列可共用一个完成队列，最大支持 65535 个命令队列，单个队列最大支持 64K 条指令。主机驱动程序根据上层应用的操作组织对应指令（可能包含多条指令）并放置于发送队列中，通过写控制器端的门铃寄存器启动命令传输；设备接收指令并执行相应操作，指令完成状态由设备端控制器放入对应的完成队列中供主机驱动程序解析。

图 1 为 NVMe 指令执行过程，包含如下步骤：

（1）主机创建 NVMe 命令并提交命令至发送队列中；

（2）主机更新发送队列尾指针并将其写入到发送队列尾指针门铃寄存器，用于通知控制器有新的命令要发送，等待控制器来取；

（3）控制器查询门铃寄存器，获知有新的命令，则从主机端命令发送队列中提取出命令并保存于控制器命令队列中；

（4）控制器执行命令队列中的命令；

（5）控制器执行完命令后，将执行结果封装于完成包中并写入到主机端相应的完成队列；

（6）控制器产生指令完成中断；

（7）主机收到中断，解析完成队列中命令完成包并进行相应的处理；

（8）主机处理完成后，更新完成队列头指针门铃寄存器，用于告知控制器执行完成。

相较于 SATA 或 AHCI 协议，NVMe 协议指令执行交互过程简单，队列数量和深度远胜于前两者，多个队列指令并行操作及简单的指令发送机制保证了指令的高效执行。

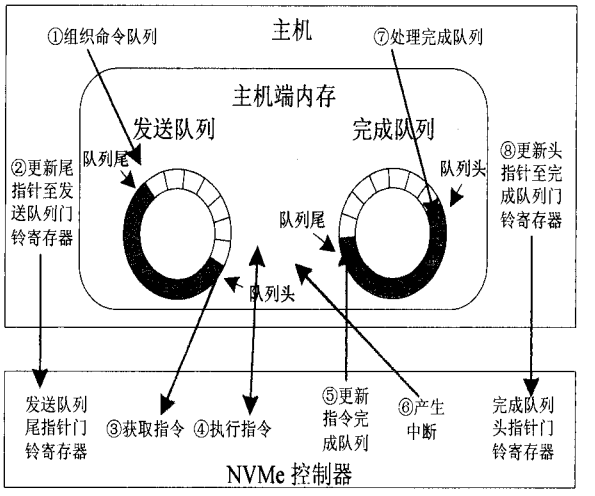

图 1　NVMe 指令执行过程

2　系统组成

本驱动开发硬件环境主要包含开发板和 PCIe 接口 SSD，如图 2 所示。其中开发板包含 MPC8641[3] 主控制器、内存及周围电路，支持串口、网口和 PCIe 接口，运行 VxWorks 6.8 操作系统。PCIe 接口用于连接 PCIe SSD，NVMe 驱动集成于操作系统中，用于实现 NVMe 协议，完成与支持 NVMe 协议的 SSD 设备进行指令交互。

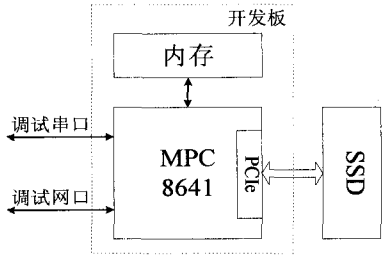

图 2 硬件系统示意图

SSD 为支持 NVMe 协议的固态硬盘，开发板与 SSD 间通过 PCIe 接口连接，逻辑上使用 NVMe 协议进行数据传输。VxWorks 6.8 操作系统中，在块设备之上封装了 XBD（extended block device）层，应用软件的读写请求通过文件系统提供的标准文件存取控制接口访问 XBD 层，并由 XBD 层对块设备层发起读写请求，块设备层根据接收的读写请求调用 NVMe 驱动程序组织相应的 NVMe 指令操作，并通过 PCIe 接口发送指令至 SSD 端 NVMe 控制器，控制器接收到 NVMe 命令后执行对应的操作，从介质（NAND flash）中读取或写入数据。本驱动的软件系统如图 3 所示。

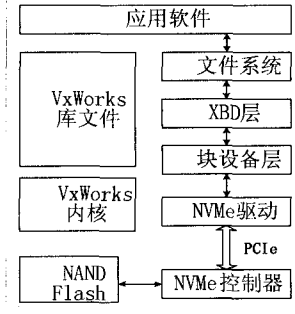

图 3 软件系统示意图

3 驱动设计

3.1 NVMe 驱动模块

NVMe 驱动符合 VxWorks6.8 操作系统中标准驱动程序框架要求[4-5]，满足 NVMe 1.2 协议，支持标准文件系统对 SSD 的存取和控制操作。基于图 3 软件架构示意图，细化各个模块间的相互操作关系以及具体功能子项， NVMe 驱动程序模块设计框图如图 4 所示。

NVMe 驱动程序主要由初始化模块、命令执行模块和 XBD 接口模块三部分组成。初始化模块用于初始化 PCIe 控制器和 NVMe 控制器，创建并初始化 NVMe 管理命令、I/O 命令的发送和接收队列，同时初始化设备，读取相应配置信息，以供 XBD 接口模块使用；命令执行模块为核心模块，它完成主机与设备间命令交互过程，包括命令封装、发送、接收、处理等；XBD 接口模块实现文件系统与块设备间 I/O 访问，将文件系统的 I/O 请求转发至块设备层的接口，进而调用 NVMe 驱动程序完成操作。三个模块间相互关联，共同保证 NVMe 驱动的正常运行。

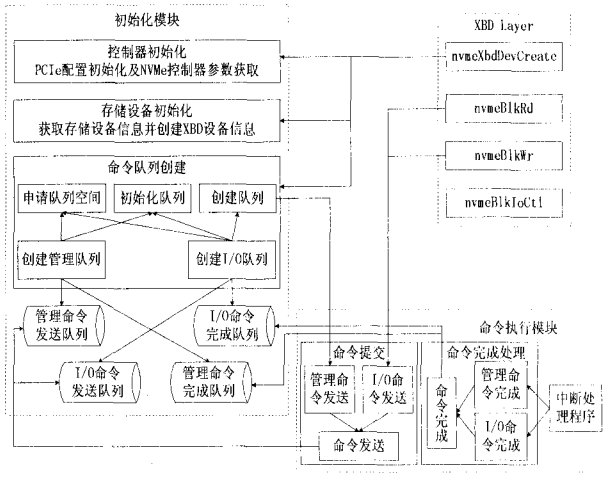

图 4 NVMe 驱动模块示意图

3.2 初始化模块

VxWorks 6.8 操作系统的设备驱动程序基于 VxBus 架构[5]，VxBus 定义了驱动程序访问硬件的机制，能够发现总线上的设备并根据该类型设备驱动提供的函数接口注册驱动，完成初始化工作。结合本开发板特性，在 VxWorks 6.8 系统中配置 PCIe 接口为 PCIe 1.0 ×4，操作系统启动后，总线控制器驱动程序遍历总线上设备，扫描到 PCIe 总线上 SSD 设备，对其 NVMe 控制器进行内存映射等初始化操作，读取对应设备总线信息，进而通过 VxBus 架构注册至操作系统。

NVMe 驱动中定义了如下 3 个数据结构：

```
①LOCAL PCI_DEVVEND vxbNvmeIdList[] =
  {
    {NVME_DEVICE_ID,
NVME_VENDOR_ID},
  }
②LOCAL DRIVER_INITIALIZATION
  vxbNvmeFuncs =
  {
vxbNvmeInstInit,
```

```
vxbNvmeInstInit2,
    vxbNvmeInstConnect
  }
③LOCAL PCI_DRIVER_REGISTRATION
  vxbNvmeStorageRegistration =
  {
    {
    NULL,
    VXB_DEVID_DEVICE,
    VXB_BUSID_PCI,
        VXB_VER_4_0_0,
    "nvme",
    &vxbNvmeFuncs,
    NULL,
    NULL,
    },
    NELEMENTS(vxbNvmeIdList),
    &vxbNvmeIdList[0],
  }
```

vxbNvmeStorageRegistration 描述了 NVMe 驱动向 VxWorks 操作系统中注册的基本信息，其中包括有设备号（VXB_DEVID_DEVICE）、总线类型（VXB_BUSID_PCI）、VxBus 版本（VXB_VER_4_0_0）、设备 ID 和厂商 ID（vxbNvmeIdList）。同时 vxbNvmeFuncs 还定义了本驱动注册及初始化过程中各个阶段依次调用的 3 个函数：vxbNvmeInstInit 在操作系统初始化函数 sysHwInit 中调用，完成 NVMe 驱动实例的检查；vxbNvmeInstInit2 在操作系统初始化函数 sysHwInit2 中调用，完成 PCIe 端状态寄存器初始化及设备映射；vxbNvmeInstConnect 为驱动初始化主体部分，在 sysHwInit2 尾部调用，完成 NVMe 控制器初始化、设备初始化、相关数据结构和信号量创建、命令队列创建、中断使能和块设备创建等功能。

驱动初始化过程中会调用 XBD 接口模块创建 XBD 层，并注册至操作系统，XBD 接口模块所需的设备信息及子函数接口均在 vxbNvmeInstConnect 过程中实现。

3.3　命令执行模块

命令执行模块是 NVMe 驱动的核心模块，用于实现 NVMe 指令封装、交互、中断处理等功能。NVMe 驱动包含多对队列，其中包括一对管理命令队列和多对 I/O 命令队列，每一对队列又包含一个命令提交队列和一个命令完成队列，每一个队列都有相应的属性信息，如队列的虚拟地址、物理地址、长度、头指针、尾指针和相应的门铃寄存器等。本驱动定义了 nvme_dev 和 nvme_queue 两个结构体，逻辑关系如图 5 所示。

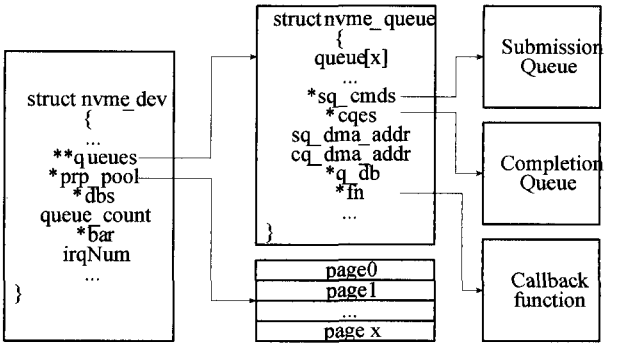

图 5　NVMe 设备结构体逻辑关系

其中，nvme_dev 表示一个 NVMe 设备，在设备初始化的时候为各个成员变量赋值。**queues 指向队列指针数组，数组中的每一项表示一对队列。其中第 0 对队列表示管理命令队列，其他队列表示 I/O 命令队列，每对队列包含命令提交队列和命令完成队列；*prp_pool 指向 PRP(physical region page)地址，用于 DMA 数据传输地址；*dbs 指针表示了门铃寄存器的基地址，每个队列都对应一个 32 位的门铃寄存器；*bar 表示 NVMe 设备的基地址寄存器，在设备初始化的时候映射为主机虚拟地址保存在这个字段中；irqNum 表示设备的 MSI-X 中断号，在系统初始化的时候，系统会给每个 PCI 设备分配中断号，驱动程序可以用这个中断号注册中断处理函数来响应中断。nvme_queue 表示一对队列，其中包含了命令发送队列和完成队列的地址以及队列的相关信息。*sq_cmds 指针指向命令发送队列，表示发送队列的主机虚拟地址，命令发送队列在内核中是一片物理连续的内存空间，驱动程序将命令放在这个队列中并等待 NVMe 控制器读取；*cqes 指针指向命令完成队列，表示完成队列的主机虚拟地址，命令完成队列在内核中是一片连续的内存空间，完成包由 NVMe 控制器放入这个队列中，然后等待主机驱动程序进行处理；sq_dma_addr 和 cq_dma_addr 分别表示命令发送和接收的物理队列地址，传递给 NVMe 控制器使用；*q_db 表示队列门铃寄存器；

*fh 指向指令处理函数。

NVMe 指令均为 64 字节，每个命令可通过队列号和命令 ID 来唯一标识。当驱动程序从内核的 I/O 请求队列中接收到了新的请求后，需要将其转换为 NVMe 规范中规定的 I/O 命令格式，然后放在 I/O 命令发送队列中等待 SSD 控制器预取，NVMe 队列结构如图 1 所示，它在物理上是内核中一片连续的、可用于 DMA 访问的内存空间，逻辑上是一个环型数组缓冲区。

队列写入时使用该队列尾指针来获得下一个命令项可写的位置，每当有新的命令项写入队列时，尾指针加'1'，若尾指针超过了队列的大小，则回转到'0'。因此，队列的尾指针可看作是队列的生产者，只有在队列不满的时候才能向队列中写数据。

每个队列的头指针指向下一个出队列的命令项，每消费一个命令项，头指针加'1'，若头指针超过了队列的大小，则回转到'0'。因此，每个队列的头指针可看作队列的消费者，只有在队列不为空的时候才能消费队列中的命令项。

当队列的头指针和尾指针相等时，表示这是一个空队列。当队列的头指针等于队列的尾指针加'1'时表示这是一个满队列。鉴于此，每个队列中都有一项是不可用的，如一个可以保存 64 个命令的提交队列，最多可以存储 63 个命令。

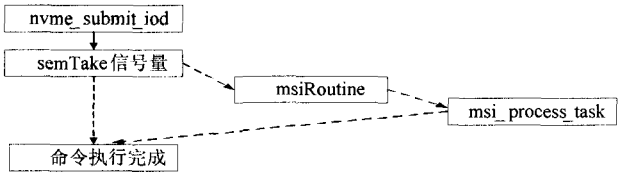

图 6 命令处理机制

本驱动中命令处理机制如图 6 所示，主要流程如下：

(1) 通过 nvme_submit_iod 函数提交读写数据的命令，将命令写入 I/O 命令队列中。nvme_submit_iod 函数先从消息队列中获取一个信号量，然后根据数据长度，分多次提交 I/O 读写命令，最后等待获取（semTake）这个信号量进入阻塞；

(2) NVMe 控制器处理完一个命令后，发送一个 MSI-X 中断到处理器；

(3) 中断处理程序 msiRoutine 开始执行，获取中断号并发送给 msi_process_task 任务。该任务会根据中断号获取对应的完成命令队列，取出完成包，释放命令的标志位，更新完成命令队列的头指针；同时，msi_process_task 任务会检查命令完成总数是否等于提交总数，若相等，则释放步骤(1)中阻塞的信号量；

(4) nvme_submit_iod 函数阻塞结束，将这个信号量放回消息队列中，执行完成。

3.4 XBD 接口模块

XBD 层位于文件系统和块设备之间（如图 3 所示），为两者提供一套标准的接口，将来自上层文件系统的 I/O 请求分发至设备驱动程序。根据 XBD 层接口要求，本驱动主要实现 nvmeXbdIoctl、nvmeXbdStrategy、nvmeXbdDump、nvmeXbdDevCreate、nvmeBlkRd、nvmeBlkWr 和 nvmeBlkIoctl 等接口函数。

nvmeXbdIoctl、nvmeXbdStrategy、nvmeXbdDump 三个函数为向操作系统注册 XBD 设备的函数接口，被操作系统接口函数 xbdAttach 调用并进行 XBD 注册。nvmeXbdIoctl 为功能控制接口，实现控制命令操作；nvmeXbdDump 为卸载 XBD 设备接口；nvmeXbdStrategy 将访问存储驱动的 I/O 任务排成队列，提取并解析文件系统传递过来的数据结构，包括传输方向、传输大小、起始块号信息，将它们作为参数传递给 NVMe 控制器驱动层的数据传输模块，实现指定 I/O 任务。

nvmeXbdDevCreate 通过调用初始化模块中的控制器初始化接口、命令队列创建接口以及存储设备初始化接口完成控制器和存储设备以及驱动程序的初始化，将 nvmeBlkRd、nvmeBlkWr 以及 nvmeBlkIoctl 等接口挂接至 VxWorks 操作系统的标准 XBD 层，并基于 NVMe 存储设备创建块设备。

nvmeBlkRd 和 nvmeBlkWr 函数向上层系统提供基于 NVMe 存储设备的 XBD 层读取和写入操作，向下通过命令执行模块中命令提交模块提交 I/O Command 给 NVMe 控制器执行，并从命令完成处理模块中获取命令执行的结果信息。

nvmeBlkIoctl 函数在 XBD 层内部完成对于 XBD 设备的配置和控制操作。

4 测试结果

系统初始化完成后，使用 vxBusShow 命令查看 VxBus 已例化的驱动状态，可见如下信息：

PCI_BUS @ 0x002e5148 with bridge @ 0x002e9848

Device Instances:

nvme unit 0 on PCI_Bus @ 0x002ea348 with busInfo 0x00000000

输入 devs 命令可见 PCIe SSD 挂载盘符“/nvme0”，说明驱动已初始化并挂载设备成功，如下：

```
－>devs
drv name
0  /null
1  /tyCo/0
1  /tyCo/1
8  host:
9  /vio
3  /nvme0
value = 25 = 0x19
```

调用 nvmeBlkRd 和 nvmeBlkWr 函数分别进行块设备层的读写速率测试，在设置队列数为 30 的情况下，读写速率测试如表 1 所示。

表 1 读写速率测试

单次操作 *扇区数	读速度（MB/s）	写速率（MB/s）
1024	581.82	726.24
2048	637.34	743.83
8192	682.67	758.52
32768	698.18	755.35
65536	702.98	752.94

*注：1 个扇区 =512 字节

从上述测试结果可知，本驱动实现了 NVMe 的基本功能，且在 PCIe 1.0 ×4 接口下实现平均 650 MB/s 的读速度和平均 740 MB/s 的写程度，表现出良好的性能。

5 结论

NVMe 协议充分发挥了固态存储设备（SSD）的性能，相较于 SATA 或 AHCI 协议仍保留机械硬盘的兼容性，NVMe 协议设计目标即专门针对固态存储介质，尽可能发挥其最大潜能。支持 NVMe 协议的存储卡对于需求较高性能的单盘存储方案是最优选择。本文实现了嵌入式系统 VxWorks 6.8 下 NVMe 驱动，测试结果表明，此驱动程序实现了 NVMe 协议的基本功能，支持运行 NVMe 协议的固态存储卡，对于机载领域数据存储有较高的应用价值。同时，受限于硬件开发平台资源，仅测试验证了 PCIe1.0×4 下 SSD 读写性能，同环境在 PCIe 2.0/3.0 下 SSD 读写性能理应有更大的提升，待进一步测试。

参 考 文 献

[1] NVM Express,Inc. NVM express specification, rev 1.2[EB/OL].[2015-2-10]. http://nvmexpress.org.

[2] NVM Express,Inc. NVM express management interface, rev 1.0[EB/OL]. [2015-12-25]. http://nvmexpress.org.

[3] Freescale semiconductor. MPC8641D integrated host processor family reference manual, rev. 2.0[EB/OL]. [2009-10-3]. www.nxp.com

[4] Wind River, Inc. VxWorks device driver developer's guide 6.8 [M]. 3rd ed. Wind River, Inc., 2010.

[5] Wind River, Inc. VxWorks kernel programmer's guide 6.8 [M]. 2nd ed. Wind River, Inc., 2010.

[收稿日期] 2017-08-28

[作者简介] 盘勇军 (1987—) 男，工程师。研究方向：航空电子系统。

黄伯铮 (1991—) 男，助理工程师。研究方向：大容量数据存储。