文章编号:1001 - 9081(2005)09 - 2165 - 04

裁剪 Linux 内核屏蔽页面交换机制的研究与实现

王亚军[1,2],刘金刚[2]
(1. 中国人民武装警察部队学院,河北 廊坊 065000;
2. 首都师范大学 & 中国科学院 计算技术研究所
计算机科学联合研究院,北京 100037)
(wangyaj@mails.gscas.ac.cn)

摘 要:页面交换技术是 Linux 存储管理中采用的重要技术之一,但在有实时要求的系统中,是不宜采用页面交换的,因为它使程序的执行在时间上有了较大的不确定性。文中阐述了如何屏蔽 Linux 内核中的页面交换机制,从而满足实时系统的要求。

关键词:页面交换;共享内存;实时性
中图分类号: TP316 **文献标识码**:A

Research and realization of covering page-swapping mechanism by cutting Linux core

WANG Ya-jun[1,2], LIU Jin-gang[2]
(1. *The Chinese People's Armed Police Forces Academy, Langfang Hebei* 065000, *China;*
2. *Join Research Academy for Computer Science,*
Capital Normal University & Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100037, *China*)

Abstract: Page-swapping technology is very important in memory management of Linux. But this technology is not suitable for Real-time system. How to cover page-swapping mechanism in Linux core was expatiated.

Key words: page-swapping; memory-sharing; real time

0 引言

对于 Linux 这样的现代操作系统来说,除 CPU 之外,物理内存页面可以说是最基本、最重要的资源了。在 Linux 系统中,每个进程的用户虚存空间为 3GB,不过,每个进程实际使用的物理内存空间却小得多,一般不超过几 MB,可是,当系统中有成百上千个进程同时存在时,对物理内存空间的需求总量就很大。在这样的情况下,为系统配备足够的物理内存就很困难。所以,Linux 采用了建立在页式虚存管理基础之上的"按需页面交换技术"。为了避免总是在 CPU 忙碌的时候,即在缺页异常发生的时候再来临时搜寻可供换出的内存页面并加以换出,Linux 内核定期检查并且预先将若干页面换出到交换设备或交换文件,从而腾出内存空间,以减轻系统在缺页异常发生时候的负担。对于通用系统来说,这种"以时间换空间"的技术有效的缓解了系统中物理内存不足所带来的矛盾。

但是,当把 Linux 用于嵌入式系统时,一般来说,这种页面交换技术却往往需要加以屏蔽,究其原因主要有二:一方面嵌入式系统一般都不带有磁盘或者磁盘空间很小,因而无法实现页面交换;另一方面嵌入式系统往往都有实时性要求,而页面的换入/换出恰恰是对实时性最大的破坏之一。比如,一个本来很快就可以完成的程序,可能会因为某个子程序或变量所在的页面恰好不在内存而需要从外部设备读入,那么程序的执行在时间上就变得完全不可预测。所以,屏蔽 Linux 内核中页面交换机制,是把 Linux 用于嵌入式系统,特别是有实时性要求的系统的重要工作之一。下面以 Linux2.4.0 内核源代码为例,详细阐述屏蔽 Linux 内核中页面交换机制的研究与实现。

1 Linux 内核中相关源代码的分析

Linux 2.4.0 内核中定义了一个数据结构 swap_info_struct,用以描述和管理用于页面交换的设备或文件。Linux 内核中允许使用多个页面交换设备或文件,所以在内核中建立了一个 swap_info_struct 结构的数组 swap_info,同时还设立了一个队列 swap_list,将各个可以分配物理页面的交换设备或文件的 swap_info_struct 结构按优先级链接在一起。另外,Linux 内核为盘上页面设立了一个专门的数据结构 swp_entry_t,其实只是一个 32 位的无符号整数,其最高的 24 位组成的 offset 字段表示该盘上页面在某个交换设备或文件中的位置,也就是交换设备或文件中的逻辑页面号;而接下去的 7 位组成的 type 字段则表示该盘上页面所在交换设备或文件的序号;最低为永远是 0,对应于页面表项中 P 标志位为 0 的情况,表示该页不在内存中。这样,当一个页面的内容在内存中时,页面表中的响应表项确定了地址映射关系;而当页面内容不在内存时,则指明了页面内容的去向和所在。

在 Linux 系统中,并非所有的内存页面都可以交换出去,只有映射到用户虚存空间的页面才会被换出,而仅仅在系统空间有映射的物理页面内容是不会被换出的。映射到用户虚存空间的页面主要有以下几种:

· 普通的用户空间页面,包括进程的代码段、数据段、堆栈段,以及动态分配的"存储段"。其中有些页面从用户程序即进程的角度看是静态的(如代码段),但从系统的角度看仍

收稿日期:2005 - 03 - 19;**修订日期**:2005 - 05 - 23 **基金项目**:国家 863 计划资助项目(2001AA114201)
作者简介:王亚军,男,讲师,硕士研究生,主要研究方向:嵌入式 Linux 系统设计; 刘金刚,男,教授,博士生导师,主要研究方向:智能接口.

是动态分配的。

· 通过系统调用 mmap( )映射到用户空间的已打开文件的内容。

· 进程间的共享内存区。

而仅仅在系统空间有映射的页面大致分为如下几类:首先,内核代码和内核中的全局量所占的内存页面即不需要分配也不会被释放,这部分空间是静态的。除此之外,内核中使用的内存页面也要经过动态分配,但不会被换出到交换设备或交换文件中去,此类页面根据其内容的性质可以又可以分为两种:

一种是一旦使用完毕便无保存价值的页面,主要有:

· 内核中通过 kmalloc( )或 vmalloc( )分配,用作某些临时性使用和为管理目的而设的数据结构,如 vma_area_struct 数据结构等。

· 内核中通过 alloc_pages( )分配,用作某些临时性使用和管理目的的内存页面,如每个进程的控制块和系统堆栈所在的两个页面等。

另一种是虽然使用完毕,但其内容仍有保存价值的页面,主要有:

· 在文件系统操作中用来缓冲存储一些文件目录项结构 dentry 的空间。

· 在文件系统操作中用来缓冲存储一些文件索引节点结构 inode 的空间。

· 用于文件系统读/写操作的缓冲区。

为了实现页面的换入/换出,Linux 内核中设置了两个全局性的 LRU 队列,即活跃页面队列 active_list 和不活跃"脏"页面队列 inactive_dirty_list,还在每个页面管理区(zone)中设置了一个空闲区队列 free_area 和一个不活跃"干净"页面队列 inactive_clean_list。另外,还通过一个全局的 address_space 数据结构 swapper_space 把所有可交换的内存页面管理起来。这样,物理内存页面的换入/换出要点如下:

1) 空闲。页面的 page 结构通过其队列头结构 list 链入某个页面管理区的空闲区队列 free_area,页面的使用计数 count 为 0。

2) 分配。通过函数__alloc_pages( )或__get_free_page( )从某个空闲区队列中分配内存页面,并将所分配页面的使用计数 count 置成 1,其 page 结构的队列头 list 结构变成空。

3) 活跃状态。页面的 page 结构通过其队列头结构 lru 链入活跃页面队列 active_list,并且至少有一个进程的用户空间页面表项指向该页面。每当为页面建立或恢复映射时,都使页面的使用计数 count 加 1。

4) 不活跃状态(脏)。页面的 page 结构通过其队列头结构 lru 链入不活跃"脏"页面队列 inactive_dirty_list,但是原则上不再有任何进程的页表项指向该页面。每当断开页面映射时,都使页面的使用计数 count 减 1。

5) 将不活跃"脏"页面的内容写入交换设备,并将页面的 page 结构从不活跃"脏"页面队列 inactive_dirty_list 转移到某个页面管理区中的不活跃"干净"页面队列。

6) 不活跃状态(干净)。页面的 page 结构通过其队列头结构 lru 链入某个页面管理区中的不活跃"干净"页面队列 inactive_clean_list。

7) 如果在转入不活跃状态以后的一段时间内页面受到访问,则又转入活跃状态并恢复映射。

8) 当有需要时,就从不活跃"干净"页面队列中回收页面,退回到空闲页面队列,或直接另行分配。

Linux 内核中设置了一个专司定期将页面换出的内核线程 kswapd,这是 Linux 在系统初始化期间调用函数 kswapd_init( )创建的,同时还创建了另一个内核线程 kreclaimd。内核线程 kswapd 既可以由内核在系统中缺少空闲物理页面可供分配时被动地唤醒,也可以由核心交换定时器的定时信号周期性地唤醒。它的主要工作是通过调用函数 do_try_to_free_pages( )完成的,代码如下:

```
static int do_try_to_free_pages(unsigned int gfp_mask, int user)
{ int ret = 0;
  if (free_shortage() || nr_inactive_dirty_pages >
  nr_free_pages() + nr_inactive_clean_pages())
  ret += page_launder(gfp_mask, user);
  if (free_shortage() || inactive_shortage()) {
    shrink_dcache_memory(6, gfp_mask);
    shrink_icache_memory(6, gfp_mask);
    ret += refill_inactive(gfp_mask, user);
  } else {
      kmem_cache_reap(gfp_mask);
      ret = 1;
  }
  return ret;
}
```

该函数是理解 Linux 页面换出机制的关键函数。首先是调用 page_launder( ),目的在于把已经处于不活跃状态的"脏"页面写入交换设备,使之成为不活跃"干净"页面继续缓冲,或进一步回收一些这样的页面变成立即可以分配的页面。根据页面的不同使用目的,例如普通的用户空间页面,或者通过 mmap( )建立的文件映射以及文件系统的读/写缓冲,具体的操作也不一样。因此在屏蔽 Linux 页面交换机制时,要保留 Linux 对通过 mmap( )建立的文件映射以及文件系统的读/写缓冲的支持。另外,共享内存区作为进程间通信的一种手段,是建立在文件映射机制和特殊文件系统"shm"的基础之上的。而特殊文件系统"shm"最终要落实到页面交换盘区中,共享内存区文件的内容总是存储在交换设备上,所以在屏蔽页面交换机制后,Linux 无法继续支持共享内存区机制,该机制应该同页面交换机制一并从内核中裁除。

回到函数 do_try_to_free_pages( )的代码中,经过 page_launder( )以后,如果可分配的物理页面数量仍然不足,就要进一步设法回收页面。由于在打开文件的过程中要分配和使用代表着目录项的 dentry 数据结构和代表着文件索引节点的 inode 数据结构。这些数据结构在文件关闭以后并不立即释放,而是放在 LRU 队列中最为后备,以防在不久将来的文件操作中又要用到。这样,经过一段时间以后,就可能积累起大量的 dentry 和 inode 数据结构,占用数量可观的物理页面。而函数 shrink_dcache_memory( )和 shrink_icache_memory( )的目的就是要回收这些物理页面。另外,内核在运行中也要动态地分配使用很多数据结构,内核中对此采用了一种称为"slab"的管理机制。这种机制也是倾向于分配和保持更多的物理页面,而回收这些页面的任务就交给了函数 kmem_cache_reap( )来完成。这三个函数都是在屏蔽页面交换机制时需要加以保留的。

再回到 do_try_to_free_pages( )的代码中,函数 refill_inactive( )主要作两件事情:一是通过调用函数 refill_inactive_scan( )扫描活跃页面队列,试图从中找到可以转入不活跃状态的页面;二是通过调用函数 swap_out( )找出一个进程,然后扫描其映射表,从中找出可以转入不活跃状态的页面。

另一个内核线程 kreclaimd 的程序结构与 kswapd 相似,

目的是通过调用函数 reclaim_page( )扫描各个页面管理区中的不活跃"干净"页面队列,从中回收页面加以释放。

而从交换设备上将页面换入到内存的操作是由函数 do_swap_page( )完成的。当缺页异常发生时,内核首先检查页表项中的 P 标志位,看看页面内容是否在内存中,如果不在,进而检查表项是否为空,如果为空说明映射尚未建立,就调用函数 do_no_page( )建立映射;反之,如果页表项非空,说明映射已经建立,只是页面内容不在内存中,所以调用函数 do_swap_page( )换入页面。

深入分析 Linux 的内核原理是裁减内核源代码屏蔽页面交换机制的前提。通过上述分析可以看出,Linux 的单内核结构注重的是代码的效率和功能的强大,这是以降低操作系统结构上的清晰为代价的,裁减内核时必须注意内核中相关各部分之间的协调。

2　裁剪 Linux 内核相关源代码

综上分析,为了屏蔽页面交换机制,应该从以下几个方面裁减 Linux 的内核源代码:

首先,由于在屏蔽页面交换机制后,Linux 无法继续支持共享内存区机制,所以需要裁除内核中关于特殊文件系统"shm"和共享内存区的源代码。内核中对特殊文件系统 shm 的定义以及相关的操作都是在文件 mm/shmem. c 中提供的。从该文件外部仅仅引用了该文件中定义的 4 个函数,该文件中所定义的其他函数和变量都在该文件内部调用。分析表明,整个文件 mm/shmem. c 都可以从内核源代码中删除,包括在该文件外部引用的在该文件中定义的 4 个函数:

· 函数 shmem_nopage( )。该函数专门负责处理共享内存区缺页异常,当共享内存区机制从内核中裁除之后,该函数显然失去了意义。该函数除在文件 include/linux/mm. h 中有声明和在文件 ipc/shm. c 中的全局变量 shm_vm_ops 有引用之外,就只在本文件内部引用。为此,先将该函数除在文件 include/linux/mm. h 中的声明去掉,至于全局变量 shm_vm_ops 则在处理文件 ipc/shm. c 时也会裁减掉。

· 函数 shmem_unuse( )。该函数在文件 include/linux/swap. h 中有声明,去掉该声明。另外,该函数在文件 mm/swapfile. c 中有调用,本文后面将说明该文件也将被删除。

· 函数 shmem_file_setup( )。该函数在文件 include/linux/ mm. h 中有声明,去掉该声明。另外,该函数在文件 ipc/shm. c 中的函数 newseg( )有调用,而函数 newseg( )仅用于函数 sys_shmget( )中。从内核中裁除共享内存区机制以后,创建共享内存区的系统调用 shmget( )显然失去了意义,从而其入口函数 sys_shmget( )也将从内核中删除。

· 函数 shmem_zero_setup( )。该函数在文件 include/linux/ mm. h 中有声明,去掉该声明。同时去掉文件 mm/mmap. c 中函数 do_mmap_pgoff( )对该函数的调用,因为从内核中裁除共享内存区机制以后,不再需要为共享内存区建文件映射。同理,还要去掉 drivers/char/mem. c 中对该函数的有条件调用。

从内核源代码目录树中删除文件 mm/shmem. c 后,需要修改目录 mm/下的 makefile 文件,从中删除字符串 swap. o。

裁除了内核中关于特殊文件系统"shm"的源代码以后,还要进一步裁除共享内存区的源代码,有关的源代码集中在文件 ipc/shm. c 中。分析表明,整个文件 ipc/shm. c 都可以从内核源代码中删除,包括在该文件外部引用的在该文件中定义的几个变量和函数:

· 变量 shm_ctlmax、shm_ctlall 和 shm_ctlmni 在文件 kernel/sysctl. c 中有声明,去掉之,同时去掉该文件中全局变量 kern_table[ ]中与之有关的条目。

· 函数 shm_init( )是用来初始化共享内存区机制的,它在文件 ipc/util. h 中声明且在文件 ipc/util. c 中的函数 ipc_init( )调用,这些引用都需要去掉。

· 几个系统调用的入口函数 sys_shmget( )、sys_shmctl( )、sys_shmat( )、sys_shmdt( )都在文件 include/linux/shm. h 中有声明,去掉之;且在文件 arch/I386/kernel/sys_I386. c 中的函数 sys_ipc( )中有条件地调用,也要去掉。同时在 arch/I386/kernel/entry. S 中,将这 4 个系统调用的入口函数修改为默认的出错处理函数 sys_ni_syscall( )。

从内核源代码目录树中删除文件 ipc/shm. c 后,需要修改目录 ipc/下的 makefile 文件,从中删除字符串 shm. o。

裁除了内核中关于特殊文件系统"shm"以及共享内存区的源代码以后,下面开始裁减内核代码屏蔽页面换入/换出机制。相关的源代码主要在 mm/swapfile. c、mm/memory. c、mm/swap_state. c、include/linux/swap. h、mm/page_io. c、include/linux/shmem_fs. h 和 mm/vmscan. c 等几个文件中,其中有些文件需要进行修改,而有些文件可以从内核代码目录树中删除,删除时要注意内核中相关各个文件之间的依赖关系。

分析表明,文件 mm/swapfile. c 可以从内核源代码目录树中删除,删除时需要注意在该文件外部引用的在该文件中定义的几个变量和函数:

· 全局变量 swaplock 是对交换设备队列进行加锁和解锁时使用的,它随着交换设备一同从内核中裁除。在文件 include/linux/swap. h 中 3 处出现了该变量,一处是声明该外部变量,另外两处是定义加锁和解锁操作,这 3 处都要去掉。同样,全局变量 nr_swapfiles、swap_list 和 swap_info 也不再需要,而它们在文件 include/linux/swap. h 中声明为外部变量,去掉之。

· 函数__get_swap_page( )及利用它在文件 include/linux/swap. h 中定义的宏 get_swap_page( )的作用是从交换设备上分配盘上页面,所以都应该去掉。

· 函数__swap_free( )及利用它在文件 include/linux/swap. h 中定义的宏 swap_free( )的作用是释放盘上页面,所以都应该去掉,同时去掉函数__swap_free( )在文件 include/linux/swap. h 中的声明。另外,函数 swap_free( )在文件 mm/memory. c 中有几处调用,除在函数 free_pte( )中的调用需要删除外,由于其他几个调用该函数的函数也将被删除,所以暂时保留。

· 两个系统调用的入口函数 sys_swapoff( )和 sys_swapon( )在文件 include/ linux/swap. h 中的声明需要删除,同时在文件 arch/I386/kernel/ entry. S 中,将这 2 个入口函数修改为默认的出错处理函数 sys_ni_syscall( )。

· 函数 get_swaparea_info( )在文件 fs/proc/proc_misc. c 中有声明,去掉之,并去掉调用该函数的函数 swaps_read_proc( ),并在函数 proc_misc_init( )中定义的局部数组 simple_ones[ ]中去掉含有 swaps_read_proc 的数组项。因为在特殊文件系统 proc 中已经不能再需要显示交换设备信息。

· 去掉函数 is_swap_partition( )在 include/linux/swap. h 中的声明以及在文件 drivers/block/blkpg. c 中函数 del_partition( )中的调用。因为交换分区在系统中将不再存在。

· 去掉函数 si_swapinfo( )在 include/linux/swap. h 中的

声明以及在文件/kernel/info. c、mm/oom_kill. c 和 fs/proc/proc_misc. c 中的 3 处调用。

· 去掉函数 swap_duplicate( )在 include/linux/swap. h 中的声明。函数 swap_duplicate( )在文件 mm/memory. c 中的函数 copy_page_range( )有一处有条件的调用,用于递增盘上页面的共享计数,而当页面交换机制屏蔽以后,这种条件不会具备,所以删除该处调用。另外,该函数在文件 mm/vmscan. c 和 mm/swap_state. c 两个文件中的调用将随着函数 try_to_swap_out( )和 read_swap_cache_async 的删除而删除。

· 去掉函数 swap_count( )在文件 include/linux/swap. h 中的声明和在函数 is_page_shared( )中一处有条件的调用。因为当页面交换机制屏蔽以后,计算页面的共享计数不会再考虑盘上页面的情况。

· 去掉函数 get_swaphandle_info( )在文件 include/linux/swap. h 中的声明。另外,该函数在文件 mm/page_io. c 中的调用将随着该文件的删除而删除。

· 去掉函数 valid_swaphandles( )在文件 include/linux/swap. h 中的声明。另外,该函数在文件 mm/memory. c 中的调用将随着函数 swapin_readahead( )的删除而删除。

从内核源代码目录树中删除文件 mm/swapfile. c 后,需要修改目录 mm/下的 makefile 文件,从中删除字符串 swapfile. o。

分析表明,文件 mm/page_io. c 可以从内核源代码目录树中删除,删除时需要注意在该文件外部引用的在该文件中定义的 2 个函数,2 个函数作用都是通过调用同文件中的另一函数 rw_swap_page_base( )来读写盘上页面。修改如下:

· 去掉函数 rw_swap_page( )在文件 include/linux/swap. h 中的声明。另外,该函数在文件 mm/swap_state. c 中的 2 处调用将随着调用它的两个函数的删除而删除。

· 去掉函数 rw_swap_page_nolock ( )在文件 include/linux/swap. h 中的声明。

从内核源代码目录树中删除文件 mm/ page_io. c 后,需要修改目录 mm/下的 makefile 文件,从中删除字符串page_io. o。

分析表明,文件 mm/swap_state. c 可以从内核源代码目录树中删除,删除时需要注意在该文件外部引用的在该文件中定义的变量、函数:

· 去掉全局变量 swapper_space 和函数 delete_from_swap_cache( )在文件 include/linux/swap. h 中的声明。

· 去掉函数 add_to_swap_cache( )在文件 include/linux/swap. h 中的声明。另外,该函数在文件 mm/vmscan. c 中的调用将随着调用它的函数的删除而删除。

· 去掉函数__delete_from_swap_cache( )在文件 mm/vmscan. c 中有条件调用。

· 去掉函数 free_page_and_swap_cache( )在文件 include/linux/swap. h 中的声明。另外,将文件 mm/memory. c 中函数 free_pte( )中调用该函数的地方改为调用 page_cache_release( )。

· 去掉函数 lookup_swap_cache( )在文件 include/linux/swap. h 中的声明。另外,该函数在文件 mm/memory. c 中的调用将随着调用它的函数的删除而删除。

· 去掉函数 read_swap_cache_async( )在文件 include/linux/swap. h 中的声明以及利用它定义的宏 read_swap_cache( )。而它们在其他文件中的调用将随着调用它的函数的删除而删除。

从内核源代码目录树中删除文件 mm/swap_state. c 后,需要修改目录 mm/下的 makefile 文件,从中删除字符串 swap_state. o。

分析表明,文件 include/linux/shmem_fs. h 可以从内核源代码目录树中删除,删除时需要注意在该文件外部引用的在该文件中定义的结构体类型和变量:

· 结构体类型 shmem_inode_info 在文件 include/linux/fs. h 中定义的结构体类型 inode 中一个共用体(union)中引用,去掉引用条目。

· 结构体类型 shmem_sb_info 在文件 include/linux/fs. h 中定义的结构体类型 super_block 中一个共用体(union)中引用,去掉引用条目。

· 针对全局变量 swp_entry_t,在文件 include/asm-I386/pgtable. h 中定义了几个宏操作,即 SWP_TYPE( )、SWP_OFFSET( )、SWP_ENTRY( )、pte_to_swp_entry( )、swp_entry_to_pte( ),都需要去掉。

分析表明,文件 include/linux/shm. h 可以从内核源代码目录树中删除,而引用该文件中定义的数据结构类型和变量的文件都已经被删除或修改。

针对文件 include/linux/swap. h 所作的修改仅在于去掉不再使用的数据结构 swap_info_struct、swap_header、swap_list_t的定义。

针对文件 mm/memory. c 所作的修改仅在于:

· 删除函数 swapin_readahead( )和 do_swap_page( ),并去掉函数 swapin_readahead( )在文件 include/linux/mm. h 中的声明。

· 修改函数 handle_pte_fault( ),删除对 do_swap_page( )的调用。

针对文件 mm/vmscan. c 所作的修改仅在于:

· 删除函数 try_to_swap_out( )、swap_out_pmd( )、swap_out_pgd( )、swap_out_vma( )、swap_out_mm( )、swap_out( )和宏 SWAP_SHIFT、SWAP_MIN 的定义。

· 去掉函数 refill_inactive( )中调用函数 swap_out( )的一个 while 循环。

3 结语

最后,对裁剪后的源代码重新编译,得到一个屏蔽了共享内存区机制和页而交换机制的新内核。将这个新内核在计算机上运行,可以发现原来内核提供的用来开启和关闭页面交换机制以及支持共享内存区操作的几个系统调用,新内核已经不再提供。更重要的现象是系统中可以并发运行的进程数量明显减少,而应用软件在反应速度和执行时间上的可预测性明显改善,原因就在于原来内核所提供的"以时间换空间"的页面交换机制已经被取缔,内存空间相对变小,而页面换入/换出的时间节省下来。这对于通用计算机系统来说,是一种弊端,但是对于嵌入式系统(尤其是实时嵌入式系统)来说,却具有特殊重要的意义。

参考文献:

[1] 毛德操,胡希明. Lniux 内核源代码情景分析[M]. 杭州:浙江大学出版社,2002. 1 - 190.

[2] (美) ROBERT L. Linux 内核设计与实现[M]. 陈莉君,康华,张波,译. 北京:机械工业出版社,2004. 133 - 150.

[3] 李善平,刘文峰,王焕龙. Linux 与嵌入式系统[M]. 北京:清华大学出版社,2003. 192 - 210.

[4] 王学龙. 嵌入式 Linux 系统设计与应用[M]. 北京:清华大学出版社,2001. 29 - 55.