

Table of Contents

1. IDE Device Driver (<i>RTLide</i>).....	1
Summary	1
Description	1
Layer.....	2
API / Compatibility.....	2
Dependencies.....	2
Status.....	3
Implementation issues.....	3
Configuration	4
Mode of operation	4
Validation Criteria	5
Tests	5

Chapter 1. IDE Device Driver (*RTLide*)

Summary

Name	IDE Device Driver
Description	Device driver for IDE hard disks.
Author/s	Alejandro Lucero
Reviewer	Ismael Ripoll
Layer	High level RTLinux.
Version	0.1
Status	Testing
Dependencies	Requires Linux kernel to initialise PCI and DMA devices.
Release Date	M3

Description

RTLinux does not support direct access to any kind of permanent massive storage systems, and in particular IDE hard disks. When a RTLinux thread has to store data on the hard disk, it has to use the Linux services. The usual way of doing the transfer is by means of RTFifos: a rtl-task sends the data to Linux through an RTfifo, and then, a Linux process writes this data on the hard disk.

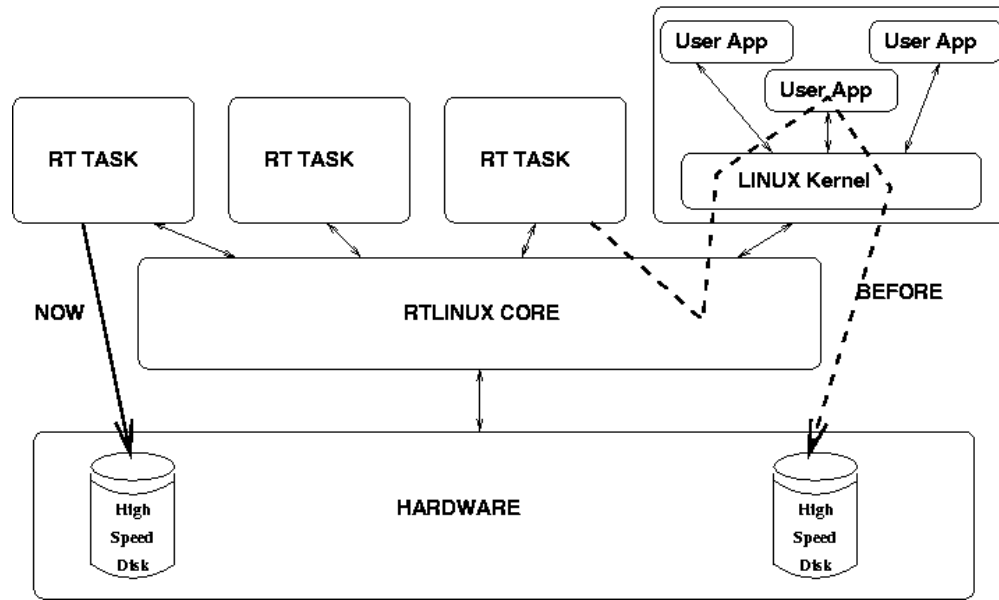


Figure 1-1. Access to the hard disk.

The existing method to access hard disks is slow, unpredictable and inefficient, since it is executed by the Linux (background). Moreover, in some applications, like continuous media applications, it is required a certain degree of predictability. Therefore, for multimedia applications is more convenient that rtl-tasks have access to the hard drive, so that disk access can be done with real-time guarantees. To achieve this goal it is necessary to: 1) port the IDE driver to RTLinux, 2) implement a real-time filesystem, and 3) implement a real-time disk scheduler.

This component provides the porting of the Linux IDE driver to RTLinux. The other two blocks of the file subsystem (disk scheduler, and filesystem) are provided in a separate component (see rtlfs documentation).

We have used the Linux device driver for IDE disks code and ported it to work in RTLinux. The part of the driver that deals with the hardware has been used as-is. Most of the porting effort was dedicated to remove the interface with Linux and the facilities and data structures provided by Linux.

Layer

Current version is a high level RTLinux component since the code do not modify RTLinux code.

API / Compatibility

Generally, IDE devices are not user oriented. Although they are implemented (inside UNIX systems) with the standar file operations as `open`, `read`, `write`, ..., users don't work with them directly, but using a file-system layer. However, some special system commands as `fdisk`, and (of course) the file system use these operations directly.

RTLinux has a standard file system interface to access devices as fifos or shared memory. For this RTLinux uses a `rtl_file_operations` structure with the `open`, `read`, `write`, `lseek`, `mmap`, `unmap`, `close` functions. The ide driver implements these functions to achieve the POSIX approach.

Dependencies

The current version of the component depends on initialisation functions of Linux kernel for PCI, IDE buses, and DMA (Direct Memory Access) device. As the goal is to offer a high performance, is necessary to use DMA functionality which is present in modern disks. Old drivers without DMA functionality could be supported easily, but it is not clear what advantages could have.

Status

Second internal version IDE Driver is working with *DMA functionality*. The first driver version implemented at the earliest stage of this component with only had single sector operations functionality (which was used to study how Linux and RT Linux could share the same IDE disk).

Some decisions has been taken to simplify the code: First, Linux supports a wide range of IDE drivers including some weird and screwy (literal from Linux docs) ones. We support the default standard types since usually IDE disks have the same standard behaviour, but we don't support the weird and screwy devices. Second, DMA devices provides the possibility to merge several memory blocks not contiguous in memory in one single request. It's usual that file systems have a block size of 1K or 4K, and that these blocks that are contiguous on disk aren't when are located in the host memory. With this functionality disk manufacturers are aware of operating system necessities. As we explain bottom, this advanced DMA functionality is not supported in this version.

If this component gains acceptance, the weird cases and special DMA features could be implemented in next releases without excessive effort.

ATAPI devices are not supported. ATAPI is a interface originally developed to support CD-ROM devices.

In this version is not implemented the IDE sharing by Linux and RTLinux. A dedicated disk is needed for RT tasks. Therefore, the system will require at least two physical disks.

Implementation issues

Since the way the IDE device will work with RTLinux is the same as works with Linux, we could use the Linux implementation code. However, Linux IDE device driver implementation is very related to the Linux block layer which includes buffer and page cache structures. Linux requests are processed inside the IDE device driver using the `do_rw_disk()` function. All of this functionality depends on Linux buffer heads, a critical component in Linux Block Layer design.

The Linux block buffer was designed taking into account that the more frequent use exhibits: files are relatively small, read and write access are mostly sequential (spatial locality) , and data is usually accessed again in a short period of time (temporal locality). These general access characteristics are not longer valid for real-time systems: the size of the files will be much bigger (multimedia streams), spatial locality, and a few temporal locality.

Among other heuristics, the Linux block buffer reads ahead disk blocks (to improve sequential access), and keeps the disk data some time before it is finally written into the disk. Although these heuristics improves the overall performance of the system, they are not adequate for real-time systems. The block buffer has been removed in the porting.

Master DMA is a very important disk functionality. DMA is useful to avoid the CPU overhead when processing I/O requests. If DMA is not activated, the CPU must read/write data to disk using I/O instructions, with the limitation of the number of

bytes a CPU can move in one single operation. On the other hand, when DMA is activated, CPU releases the bus to allow another device (IDE Disk) to do I/O requests, meanwhile the CPU do other tasks. Other advantage of using DMA is that 64K can be moved in each operation, avoiding a high number of interrupts when is CPU who takes care of the request.

Since the RTLinux file system and disk scheduler are designed to support efficiently the allocation of media streams, DMA becomes in one of the most important points in the system. Last DMA modes support 133MB/s, which is a bandwidth unreachable with PIO modes (CPU Programmed Input/Output). As DMA in Linux IDE driver works with buffer heads, a new implementation of DMA routines has been necessary.

DMA is useful to avoid CPU overhead processing I/O requests. If DMA is not activated, is the CPU that must read/write data to disk using I/O instructions, with the limitation of the number of bytes a CPU can move in one single operation. On the other hand, when DMA is activated CPU only releases the bus to let another device to do I/O requests (Master DMA), then CPU can do other tasks. Other advantage of using DMA is that can move 64K in each operation avoiding a high number of interruptions that occurs when is CPU who takes care of the request.

Configuration

Linux makes a conservative DMA configuration: if the device is not known or is in the black list¹, DMA is not activated. Obviously the successful of our design is based on DMA functionality, so a more aggressive configuration is required. The checks to do during initialization are:

1. Is the IDE disk present?
2. Is a Master DMA and LBA capable disk?
3. Is disk into the devices black list?
4. What DMA mode is supported?

Initialization of IDE disks is done by Linux kernel at boot time. RTLinux only needs the structures that Linux has initialized, so no special initialization routines are necessary. Therefore, this component can not be directly used in stand-alone RTLinux unless buses (PCI, ISA) and devices (DMA) initialization has also been ported.

Mode of operation

DMA takes care of operating system necessities. Most file systems use to use block sizes of 1K-4K to manage data, also, data that has to be stored contiguously on the disk is scattered in several of these blocks. Some DMA devices can be programed with a list of non-contiguous physical blocks so that the DMA feeds the disk device as it where only a single large data block. This way, the number of disk operations is reduced. Although this is a powerful functionality for general purpose operating systems, it has not so importance in our design, since memory is managed more restrictively (at initialization time) within the RTLinux approach. When developers ask for memory at inicialization they can use the `__get_free_pages` linux kernel function that returns a pointer to a contiguous memory block. The steps to use the

driver are showed in the next graphic:

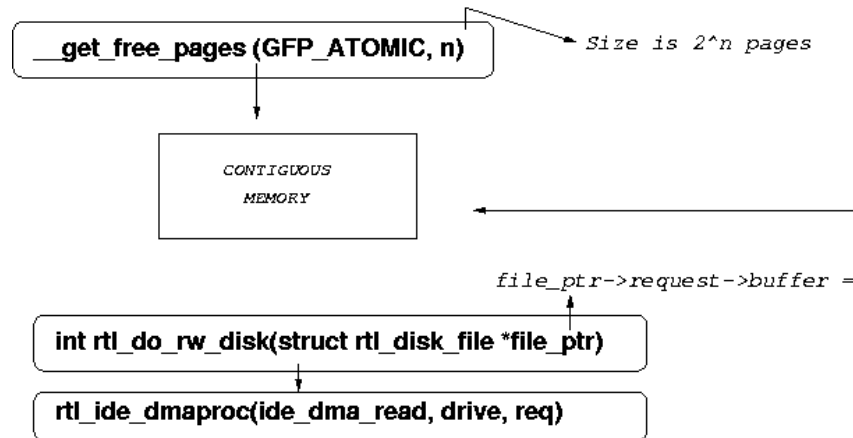


Figure 1-2. Contiguous memory allocation

The entry point to ide disk driver is the `rtl_do_rw_disk` function. A pointer to a `rtl_disk_file` structure is received, which is an object for an open file (related with one inode). The `rtl_disk_file` structure has a `request` field, a pointer to a `rtl_request` structure that will be used for the driver for access the disk. And one field of the `rtl_request` structure is a pointer to a buffer. This buffer must be allocated by users at initialization time and must be contiguous in memory, it means the `__get_free_pages` function or similar must be used. After that, the `rtl_ide_dmaproc` will be called. This approach is distinct from the Linux one, since Linux must take care of scattered memory blocks, using additional functions as `buid_dma_table` and `build_sg_list`. The scattered blocks functionality makes sense in a dynamic memory enviroment, when memory is a resource very demanded for short spaces of time.

Validation Criteria

Since this component has not utility without the file system and disk scheduler, validation criteria would focus on how the complete block layer works. The interface with the RTLlinux tasks is located in the file system, so validation criteria for IDE driver will be same than the validation criteria for the file system.

Another validation criteria would be how many devices has been tested with the driver. As we comment above, some special devices with a weird behaviour are not supported.

Tests

See tests section of the file system and disk scheduler component.

Notes

1. Black list: list of chip sets that are not fully compatible, or that may cause problems with the Master DMA functionality.

