Available online at www.sciencedirect.com

SciVerse ScienceDirect

Energy Procedia 16 (2012) 883 – 888

Energy Procedia

2012 International Conference on Future Energy, Environment, and Materials

A High Performance Data Storage Method for Embedded Linux Real-time Database in Power Systems

Lu Jian-feng, Wang Chun-yi, Hu Jie

Innovation and Technology Building, 19F, Block A, Xin Mo-fan Road, No. 5, Nanjing, 210003, China

Abstract

In power systems, extraordinary amounts of data collected from smart grid application systems need to be stored efficiently and effectively in real time. A new data storage method was proposed to deal with the huge data storage problem in embedded Linux operation system. The new method combined the advantages of Linux kernel system customization feature and operation system file management characteristics to fully utilize the hardware resources. The simulation results demonstrated that the proposed method can improve the real-time database system's storage and query performance.

Keywords: power grid monitor; real-time database; embedded Linux system; high performance data storage

1. Introduction

Real-time database is a database management system with the time series properties. It is for the high-frequency real-time acquisition data, with very high storage speed, query retrieval efficiency and data compression ratio[1], mainly used to store vast amounts of real-time acquisition data in the industrial process, providing the powerful support to the enterprise's production management, data analysis and intelligent decision-making from the data perspective.

In the power system, with the continuing development of computer monitoring system and the increasing levels of automation of power grid management, the amount of acquisition data from the power grid operation monitoring system is extremely large. Embedded system is the main carrier of the secondary equipment in the power system. Due to a specific operating environment and application size constraints, the contradiction between the limited hardware resources and unlimited application requirements has become increasingly prominent. With the increasing storage and query performance

 doi:10.1016/j.egypro.2012.01.141

requirements of the mass real-time data, the storage efficiency of the data file is facing increasing challenges, making the study of the efficient file storage technology of the real-time database becoming a research hotspot [2].

Located in the specific application scenarios of the power system, the existing embedded application system is becoming unable to meet the mass data storage needs. As a fundamentally support software, the embedded real-time database system needs excavate the potentiality of the modern processor and operating system thoroughly, increase the throughput rate of single event substantially. At present, along with the steady improvement of hardware speed, to improve processing performance, the international mainstream research focuses on improving the operating system architecture. How to improve the operating system itself to ensure high-speed file storage becomes the key of efficiency breakthrough. [3]

Taking advantage of revisability and customizability of an embedded Linux operation system kernel source code, for network monitoring applications, this paper puts forward a high-performance file storage technology in real-time database system. It makes the throughput rate of database reach ten million per second, to meet the real-time storage requirement of mass data.

2. Introduction to operating system calls

The system calls use a series of kernel functions provided by the operating system kernel, to provide services for the applications through the system call interface (API).

Embedded Linux operating system realizes the system calls through the interrupt handling: the user processes call the operating system API, do routine on-site protection, find the corresponding kernel function address according to the system call number, and turn to the function to complete the tasks given by the user program.

A user process makes the switch-off between the user mode and the kernel mode - two privilege level during the program execution, thus the implementation of the highest privilege level (0) kernel code. Code running in kernel mode operates kernel data structures and executes kernel code directly, has an unrestricted access to system storage and the external equipment with the high efficiency. [4]

3. High-performance file storage technology real-time database

3.1. Data processing in real-time database

In power system applications, the real-time data source has two main characteristics: one is the large quantity of data; the other is the high speed of data submitting. For example, in PMU steady-state acquisition, the amount of data reaches ten million per second.

Facing the mass data of power grid, real-time database divide the data collected from different acquisition front ends into two types: discrete data and continuous data. All data become to be different memory buffer data blocks after loss compression and lossless compression. When you need to archive real-time data blocks to the hard disk, you can segment the data block by archiving algorithm, then call the operating system file storage system call API, store to the archive data files and the archive index files. The data processing flow of the real-time database is as follows:

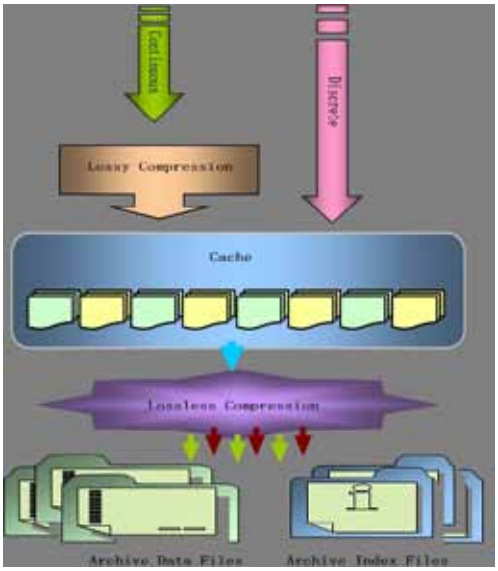

Fig. 1. Data processing flow of the real-time database

Because of the massive data storage and processing requirements, real-time database files need to repeatedly call the operating system storage interface, write the real-time data in the memory onto the hard disk. In embedded systems with limited hardware resources and in the face of such a huge amount of data, using the normal file handling will be a very serious performance bottlenecks: the mass data cannot be handled in time which result in the filling up of the memory and the interrupt of the program; the real-time data throughput rate decrease due to the hard disk read and write bottlenecks. [5]

Needless to say, optimizing the critical repeated calling code to improve the efficiency of file storage operation is the key factor of realizing a high-performance file storage.

3.2. Design and implementation of the high-performance file storage technology

In order to resolve the conflict between the mass real-time data persistence and slow hard disk physical operation, drawing on the concept of operating system file management, this paper presents an innovative embedded real-time database high-performance file storage technology, that we have its own intellectual property rights. The core code of file operation is packed into the kernel source code of operating system by the custom modified kernel source code of the embedded Linux operating system, to fully exploit the high processing capacity to achieve the high-speed storage of mass data.

The real-time database management system based on embedded Linux realizes the high-speed of the hard disk persistence of the memory real-time data through the operating system file storage interface. These memory blocks have the following characteristics in practical applications: (1) a single block is bulky in memory real-time data; (2) the allocation and reallocation are very frequent; (3) uses mode and kernel mode switching frequently; (4) repeated cycles persisted to disk. On the high performance requirements of real-time database, frequent calls to the system API are unbearable. It makes a great decline in the database performance due to the frequent switching between the user space and the kernel space, which is time consuming.

For this condition, in order to improve the real-time property of the database, based on the customization feature of embedded Linux system kernel, this paper presents a concept of logical kernelization of the real-time database file storage. By the package of kernel API, it changes the frequently-operating file storage code in the real-time database from user mode to kernel mode, reduces the program switching between them, and fully improves the file storage efficiency. Mainly take the following measures: (1) a big memory block is submitted one time; (2) pack the file storage code into the system calls API; (3) complete the file storage in a system call. As shown below in figure 2:

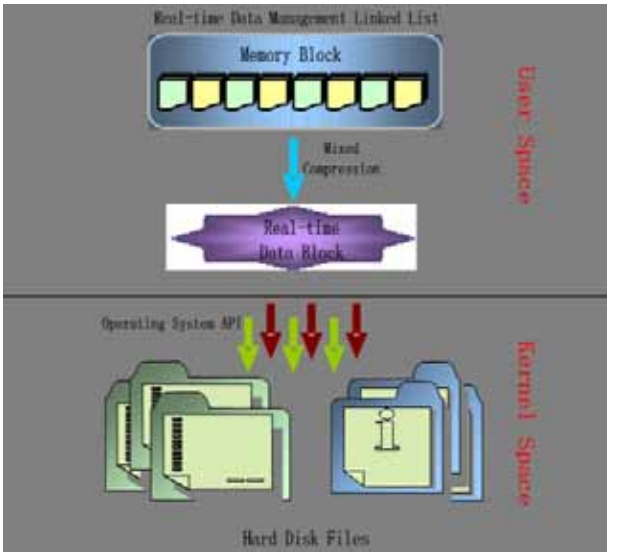

Fig. 2. File storage mechanism of real-time database

The key step of high-performance file storage technology is finding the repeatedly calling file operation code, and packing it to the kernel processing logic. The specific steps are as follows:

(1) Writing custom kernel-level system calls:

Write a kernel-level real-time database file storage function, implement the file storage code in the kernel level instead of the user level, and add it into the kernel file "kernel/ sys.c":

```
asmlingkage bool *sys_RealizeFileCall(ACE_HANDLE file, ACE_UINT32 offset, bool bRead, bool bSys)
{
void* addr = sys_mmap(0, length, PROT_RDWR, ACE_MAP_SHARED, file, offset);
}
```

(2) Compiling user defined system call back function and linking them to the kernel

Preparation of the new system call code, the follow-up is to make it known to the rest of the kernel, and then rebuild the kernel including the new system call. In order to link a new function to the existing kernel, the following two files need to be edited:

1) "inculde/asm/unistd.h"

Including a line in this file:

```
#define_NR_RealizeFileCall 191
```

2) "are/i386/kernel/entry.s"

This file is used to initialize the pointer arrays. Add a line in it:

```
Long SYMBOL_NAME(_sys_RealizeFileCall)
```

At the same time, change “NR_syscalls-190” to “191”, compile and run the new kernel.

(3) Using custom system calls in service program

In the kernel program of real-time database, call the custom system calls, realize the disk storage of data. The key code as follows:

```
#inculde<linux/unistd.h>
_syscall0(int, RealizeFileCall)
main()
{
.....
RealizeFileCall();
.....
}
```

As mentioned above, the core principle of the proposed high-performance file storage technology is to transform the operating system kernel code, which adds the custom kernel-level system calls for the real-time processing demand, fully exploits the capacity of the operating system to improve the processing capacity of the real-time database remarkably.

Needless to say, because of the physical operating characteristics of the hard disk, the file storage efficiency of the data block has been a real-time database performance bottleneck. [6]The file management of the database is an important part of the real-time database. Any improvement in the file storage efficiency can have a significant influence on overall performance. At the same time, as the limited hardware resources of the embedded system, the way of using memory and hard disk is high demanding.

The advantage of the technology proposed in this paper is the ability of reasonable allocation of memory for specific application scenarios. It atomized stores big real-time data block, decreases the program switching between the user space and the system space through the kernel mechanism of the operating system, so that the overall real-time database storage efficiency is greatly improved.

4. Test methods and results

By writing the interface test program, called the batch interpolation of the real-time database, inserted 100 real-time values to 500 measuring points, 5000 measuring points and 50000 measuring points respectively, to test its data throughput rate. The key test code as follows:

```
nRet= DB_InsertBlockValueByID(connectionId, id, values, count);
if(nRet)
{
    printf(" fail");
    return nRet;
}
printf("test Ok");
```

By systematic tests, compared to the original performance, the data processing ability improved remarkably. The test results are as follows:

Table 1. Performance of the original real-time database

Performance items	Time-consuming	CPU utilization	Numbers of events
50,000 (500 points*100 frames)	49.095 us	5.0%	10,184,336
500,000 (5,000 points*100 frames)	174.48 us	5.2%	28,655,101
500,000 (500,000 points*1 frames)	27.351 ms	4.4%	3,530,624

After the introduction of this technology, improved real-time database performance indicators as follows:

Table2. Performance of the modified real-time database

Performance items	Time-consuming	CPU utilization	Numbers of events
50,000 (500 points*100 frames)	16.365 us	1.0%	30,553,009
500,000 (5,000 points*100 frames)	58.163 us	2.1%	85,965,304
500,000 (500,000 points*1 frames)	9.117 ms	1.4%	10,591,873

Thus, the implementation of the high-performance storage technology of power system mass real-time database based on embedded Linux significantly reduced the CPU utilization, enhance the throughput as high as nearly 300%, to make the embedded real-time database performance indicators reach the international advanced level.

5. Conclusion

For the specific application scenarios in the power system, this paper gives a detailed analysis of the file storage mechanism of the embedded Linux real-time database system, uses the customized character of open source operating system, proposes and implements an innovative file storage technology, improves the mass data storage efficiency of the real-time database system greatly.

References

[1] McDONALD J D. Substation Automation:IED Integration and Availability of Information. *IEEE Power and Energy Magazine*, 2003, 1(2): 22—31.

[2] House Sean B., Niehaus Douglas.KURT. Linux Support for Synchronous Fine-grain Distributed Computations. *Proceedings of Real-time Technology and Applications*, 2000:78—87.

[3] Zhang Jian-min, Jiang Jian-ning, Zhao Fang et al. Application of IEC 61850 on Substation Relay Protection Fault Information Subsystem. *Automation of Electric Power Systems*, 2003, 27(13): 61—63.

[4] Yuan Yi et al. A Remote Radio Surveillance System Based on Embedded Web Server Technology. *Power System Technology*, 2000, 24(5): 71—73.

[5] Zhu Bin-quan, Jiang Jian-ning, Zhang Jian-min et al. Power Grid Fault Information System Based on B/S/C Architecture and IEC 61850 MMS. *Automation of Electric Power Systems*, 2004, 28(24): 41—45.

[6] Lu Guang, Zhang Bo-ming, Sun Hong-bing et al. The Embedded Real-Time Linux and ITS Application in the Automation System of the Power Network. *Automation of Electric Power Systems*, 2002, 26(7): 62—65.