# An Optimized Memory Management Algorithm for Real-time Simulation on Linux Operation System

**Miao Zhang**
College of Information System and Management, National University of Defense Technology
137 Yanwachi, Changsha, Hunan, 410073, P .R .China
+8615073134072
zmustc11@163.com

**Zhiwen Jiang**
College of Information System and Management, National University of Defense Technology
137 Yanwachi, Changsha, Hunan, 410073, P .R .China
+8618684996886
jiangzhiwen@nudt.edu.cn

**Yiping Yao**
College of Information System and Management, National University of Defense Technology
137 Yanwachi, Changsha, Hunan, 410073, P .R .China
+8613508489219
ypyao@nudt.edu.cn

**Tianlin Li**
College of Information System and Management, National University of Defense Technology
137 Yanwachi, Changsha, Hunan, 410073, P .R .China
+8613667372541
ltl@mail.ustc.edu.cn

## Abstract

As an important branch of simulation technology, real-time hardware-in-the-loop simulation has been widely used in industrial design and equipment testing. The current real-time simulation software is mostly based on proprietary systems or real-time transformation of the Windows system and cannot run in the general Linux system with a large number of users. At the same time, the existing memory management algorithm has low utilization rate and cannot adapt to the needs for a long time simulation. In view of this problem, this paper combines the design method of the existing real-time software to design the real-time simulation framework in Linux environment, and improves the memory management method to meet the needs of large-scale and long-term simulation. The experimental results show that the real-time simulation program can be executed correctly, and the memory consumption is greatly reduced.



SIMUTOOLS '17, September 11–13, 2017, Hong Kong, China

ACM ISBN 978-1-4503-6388-4/17/09…$15.00
https://doi.org/10.1145/3173519.3173542

***CCS Concepts***

• **Computing methodologies ~ Optimization algorithms** • **Computing methodologies ~ Motion path planning** • **Software and its engineering ~ Parallel programming languages** • *Software and its engineering ~ Extensible Markup Language (XML)*

***Keywords***

Kylin operating system, Real-time simulation, Memory optimized algorithm, Inter process communication

## 1 Introduction

Hardware-in-the-loop simulation plays an important role in product design, styling and testing. Compared with the physical simulation, we can greatly reduce the cost of the experiment in the premise of achieving similar results. Compared with the pure mathematical simulation, results are more real. To ensure the correction of the simulation result, there are two major factors essential. One is real-time operating system, and the second is well working simulation software. There are many real-time modeling and simulation software available on the market, such as Simulink from MathWorks, dSPACE real-time simulation system developed by German dSPACE Company and RT-Lab

real-time simulation platform developed by Opal-RT Canada. However, they are specific to the specific system or with a specific programming environment closely, and the hardware and software platform is less common. In the current research, the development of real-time simulation software platform for Linux system is still insufficient. But in some important areas, especially in the industrial control, server and other fields, Linux occupies a larger share, which calls for a well working real-time simulation software platform based on Linux operating system.

However, the Linux system itself is a general time-sharing operating system[1], which cannot meet the real-time requirements. Therefore, in this paper, the real-time version of Kylin System developed by Galaxy Kylin Company is used to develop the simulation platform. This real-time version is based on the standard version, using a series of techniques to improve the real-time performance of the system, including the addition of real-time patches, replacing normal kernel with real-time kernel, using interrupt threads, adding preemption points, replacing the spin lock and remove the big kernel lock and so on. The paper is organized as follow: The second section is the related work. The section third is the simulation framework. The fourth section is the memory optimization algorithm. The fifth section is results and analysis of the experiments. The last section is the summary of this paper and the work in the next step is also given.

## 2 Related Work

In the present study, some achievements have been made in the development of real-time software platform for Linux [3] [5]. KD-DRT real-time target model development platform is a commercial development kit developed by Wind River Company in the United States of America. This platform has a wide range of applicability, and is especially suitable for complex multi object system. The whole platform adopts the running mode of host / target machine. The host uses the windows system to accomplish the project construction, code writing and debugging work. The target machine uses Linux system to provide the platform for the simulation program. Host and target machine communicate with

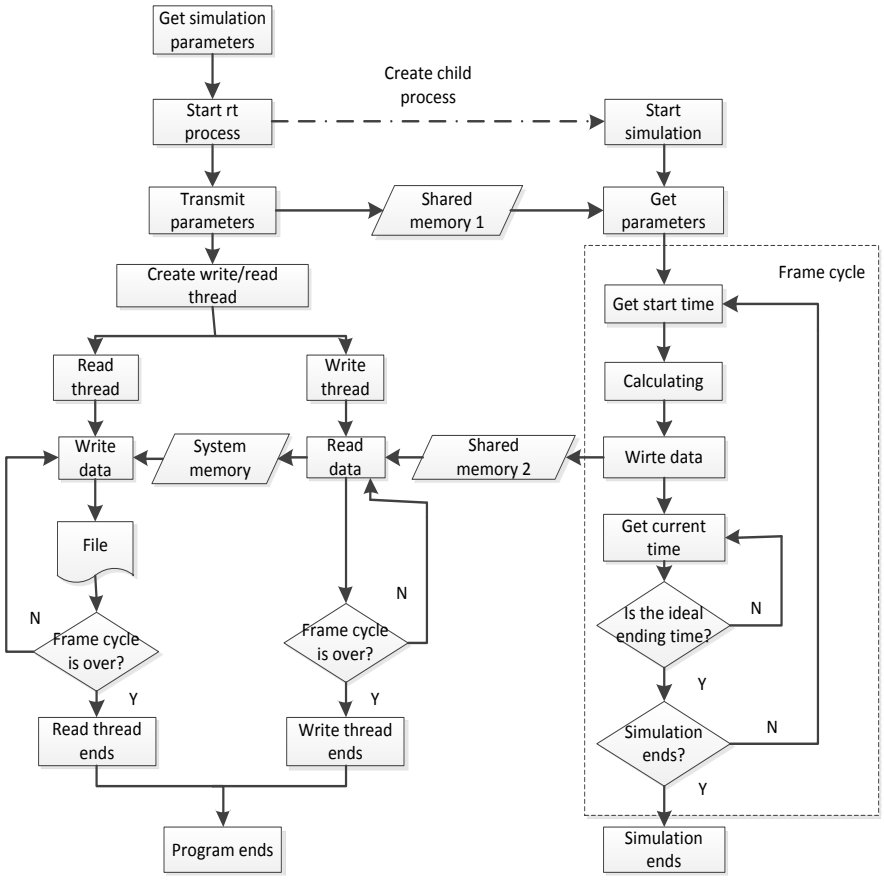


**Figure1: Simulation flow chart**

each other through the serial port or TCP/IP network protocol. Although this method can make full use of the advantages of windows system and Linux system, the hardware requirements are very rigorous and cannot accomplish the simulation process in the pure Linux system.

YHSIM[2] real-time simulation software platform is another available platform developed by NUDT (National University of Defense Technology). It uses RTX package to extend the Windows system to make Windows meet real-time requirements. In order to ensure the real-time performance of the simulation not being affected by the time-consuming I/O operation, YHSIM

stores all the frame data in system memory and not writes them into a file until simulation is over. This kind of memory management method is very effective in the short time simulation, but it is prone to get trapped in the memory breakdown situation when it refers to a long time simulation.

## 3 Design of Real-time Simulation Framework

In the process of real-time simulation, it is necessary to complete the core work of human-computer interaction, model calculation, data display and storage [6]. In all of these tasks, not all tasks have harsh real-time requirements. If all the processes are set up to the real-time priorities, it is bound to lead to the competition between real-time processes upon the CPU resources, which may cause the phenomenon of frame dropping. In this paper, the simulation task is decomposed, and the corresponding work is done by the real-time process and the non-real-time process. Specific simulation flow chart is shown as **Fig. 1**.

### 3.1 Non-real-time Process

The non-real-time process mainly completes the task of man-machine interaction, data display and data storage. It has two child processes: man-machine interface process and the simulation control process. When the real-time simulation task is to be started, the parameters of the simulation, such as the frame time and the total simulation time, are transferred to non-real-time process through the man-machine interface. After the man-machine interface process acquired the relevant parameters, the parameters are transmitted to the simulation control process using the shared memory mechanism. The main task of the simulation control process is to start the real-time simulation process and keep the communication with the real-time process. Linux system supports a variety of inter process communication mechanisms, such as signals, pipes, semaphores shared memory and so on. Because of the high demand for time and the huge amount of data to be transmitted between processes, the use of shared memory is a more appropriate method. As the system overhead is large when creating shared memory in a process, YHSIM adopt the method of circular shared memory. This method uses the read and write pointer to control the usage of shared memory. In this paper we adopt this method.

In order to start the real-time process, we use *fork ()* system call to create a child process and replace the remaining execution image of the child process with the mirror image of the real-time task. In Linux system, we can use the *execl ()* system call to accomplish this work. In the simulation flow, the simulation control process is created by the interface process, and then the real-time simulation process is created by the simulation control process. At this point in time, the process will have the same user permission as the parent process, that is, ordinary user permission. However, some operations in real-time sub process, such as modifying the process priority, locking the memory page and other operations, require root permission to perform. In order to solve this problem, in the use of *execl ()* system call, we firstly call "sudo" command, and pass the executable file corresponding to the real-time task to "sudo" command as a parameter.

The specific implementation of the code is shown as follows:

```
execl("/usr/bin/sudo","/usr/bin/sudo","/home/zm/rtSim/Optimized/rt
",(char*)0);
```

In the code above, the first parameter is the path of "sudo" command file. The second parameter is the same with the first as agreed, and the third is the path of the executable file corresponding to the real-time task. After executing this statement, we need to enter the user password in the terminal so that real-time process can be correctly executed.

### 3.2 Real-time Process

Real-time process is mainly to complete the model calculation, I/O communication, frame time control and other operations. In order to make the process run in real time, it is necessary to set the process parameters in the program. The factors that affect the execution time of the process include the priority of the process and the task scheduling strategy of CPU. In kylin real-time operating system, the value of the process priority is defined by the MAX_PRIO macro, which ranges from 0 to 139. The priority ranging from 100 to 139 represents the ordinary user processes, and the priority ranging from 0 to 99 is reserved for the real-time processes. The smaller the value is, the higher the corresponding priority is. Each priority has a task queue corresponding to it, and only when the CPU resource is not requested by the real-time task, the ordinary processes can be scheduled. There are four scheduling strategies, SCHED_OTHER, SCHED_FIFO, SCHED_IDLE and SCHED_RR. The scheduling strategy for real-time process is SCHED_FIFO and SCHED_RR. In this paper, we use *sched_setscheduler ()* system call to change the priority of the process to the highest and the scheduling strategy to SCHED_FIFO.

In order to ensure the accuracy of the frame time, high precision clock and proper frame time control algorithm are also needed. Linux kernel provides a variety of time related system calls and kernel variables [4]. The Jiffies record the number of system timers triggered since the system was started. Generally in milliseconds, the accuracy is poor. *Gettimeofday ()* system call can read data stored in the real-time clock RTC and get the wall time in the accuracy of microseconds. For real-time applications, the accuracy is still insufficient. We can also convert the number of beats of CPU into time and get high precision by CPU clock frequency. However, because of the current computers mostly using multi-core structure, the clock frequency between different cores is difficult to achieve synchronization. At the same time, for the sake of energy saving, the clock frequency of some CPU models will be changed according to the number of tasks. In this paper, we use *clock_gettime ()* system call [7] to achieve nanosecond time accuracy. To use this function, we need to include the <time.h> header file in the program and add the -lrt compiler options to connect with the real-time library. In order to accurately control the frame time, after the completion of each task in a frame, the *clock_gettime()* is called to read the current time, and the difference between the ideal ending time and the actual ending time is calculated. If the difference is greater than 100 microseconds, using *usleep()* function to suspend the process for 80 microseconds, or wait until the difference between current time and the ideal ending time is less than 10 nanoseconds. At this time, old frame is over and new frame is starting.

## 4 Memory Optimization Algorithm

In the work to be done in the non-real-time process, human-computer interaction and data display can execute very fast because of few I/O operations. Relatively, the data storage procedure needs to accomplish a large number of hard disk read and write operations. Especially when the model parameters are numerous, the time required to store a frame of data may be greater than the ideal frame time. In order to prevent the large number of I/O operations from affecting the real-time performance of simulation, in current memory management algorithm, the frame data will not be written in a file stored in the disk until all the calculation results are stored in system memory. In this way, the time performance of the simulation is guaranteed at the expense of the space performance. However, when the simulation is carried out for a long time, the memory consumption caused by this method is unbearable, and it is likely to cause other tasks to collapse due to lack of memory. The memory optimization algorithm proposed in this paper is aimed to solve this problem.

```
typedef struct memory_block
{
      time_struct time_block[N];
      data_struct data_block[N];
      enum status flag;
}memory_block;
```

**Figure2：Structure of memory block**

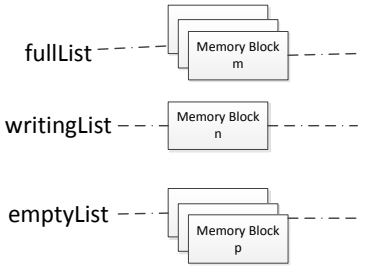


**Figure 3: Schematic diagram of the memory block list**

**ALGORITHM 1:** Optimized Memory Management Algorithm

Create *a new memory block M1.*
Create *three memory block lists*, including *emptyList, writingList and fullList.*
Initialize *current_write_block as M1.*
**While** *simulation* ***is continuing*****, do**
**Write thread:**
**If** *the space of current_write_block* ***has been used up***
Look over *the emptyList.*
**If** *the list* ***is empty***
Create *a new memory block.*
**Else**
Take *the last element of emptyList* as *the current_write_block.*
**End**
**End**
Get *the frame data from the shared memory* and write *it in curren_write_ block at the position of block_writeIndex.*
Update *position.*
**Read thread:**
**If** *the data in current_read_block* ***has all been read***
Look over *the fullList.*
**If** *the list* ***is empty***
Wait *until the writingList* is *not empty.*
Take *the element of writingList* as *current_read_block*
**Else**
Take *the last element of the fullList as the current_read_block.*
**End**
**End**
Get *the frame data at position of block_readIndex in current_read_block* and write *the data to the file stored in the disk..*
Update *position.*
**End**

In the algorithm above, we do not store all the frame data in memory, but allocate memory dynamically when necessary to reduce memory consuming. Firstly, a memory block M1 is allocated in the system, which contains N frame data and a variable used as flag to indicate the status of current memory block (see **Fig. 2**). The block status is represented by an enumeration type, including *data_empty*, *data_writing* and *data_full*. Among them, *data_empty* indicates the current block has not been used or all the data has been read and can be used again; *data_writing* indicates the current block is in use and the space has not been used up; *data_full* indicates the current block has been written and is ready to be used by read thread. Each state corresponds to a pointer list of memory block, including *emptyList*, *writingList* and *fullList* (see **Fig. 3**). When the program starts, the main thread of simulation control process continuously reads data from the shared memory and writes the data to a memory block. At the same time, we create a new thread in addition to read data from the memory block and write data to the file. Combining with OpenMP programming technology, this method can make full use of muli-core processor and reduce the total executing time of the program.

## 4.1 Write Thread

The main task of write thread is to read frame data from shared memory and write it to the memory block. This thread needs two variables to control its behavior: *current_write_block* and *block_writeIndex*. *Current_write_block* is a pointer of memory block type, used to save the address of the memory block being written. *Block_writeIndex* is an integer variable that holds the next available location to save frame data in current memory block, ranging from 0 to N-1. *Current_write_block* is initialized as the pointer of memory block M1.
Before writing data to memory block, we need to get the proper value of *current_write_block*. If the space of current block has been used up, that is, the state variable of the memory block is *data_full*, we first attempt to get an available memory block from *emptyList*. If the list is empty, allocate a new memory block *Mi* and modify its state variable as *data_writing*. At the same time, assign the address of block *Mi* to variable *current_write_block* and add this value to the rear of *writingList*. At this point, the value of *block_writeIndex* is set to 0. After the current frame data is successfully written to the memory block, it is necessary to

check for the value of *block_writeIndex*. If it has reached the last position of the memory block, that is, the variable value is N-1, the status of current block is set to *data_full*, and the pointer of current block is moved from the *writingList* to the *fullList*. Otherwise, the value is increased by 1.

### 4.2 Read Thread

The task of read thread is to read the data from memory block and write it to the file. Similar to write thread, it also needs two variables to control its behavior: *current_read_block* and *block_readIndex*. The meaning of the two variables is alike.

Similar to the write thread, we need to get the correct value of the *current_read_block* before reading the data. If the pointer of current block is null or the data of current block has all been read, that is, the status variable is *data_empty*, a new readable memory block is required. Read thread first looks over the *fullList*. If the list is not empty, take the tail element of *fullList* and assign its value to *current_read_block*. Otherwise, wait until the *writingList* is not empty, and then the element of *writingList* is assigned to *current_read_block*. When the current frame data is successfully written to the file, view the value of *block_readIndex*. If the value is N-1, which means that the data of current block has all been read, the block state is set to *data_empty*, and the pointer of current block is moved from the *fullList* to the *emptyList*. Otherwise the value is increased by 1.

## 5 Test

### 5.1 Test Environment

In this paper, we use the following test environment: DELL desktop, 3.2GHz Intel Quad-core processor, 8 Gb of memory, real-time version of the Galaxy Kylin operating system and so on.

### 5.2 Test of Real-time Performance

The real-time performance of the simulation program is mainly reflected in two aspects: the response time and the stability of the frame time. Response time reflects whether the operating system is able to respond to the requests of real-time application in time. Generally, the real-time system can reduce the response time by the way that making the kernel preemptive and increasing the preemption points. The stability of the frame time is mainly reflected whether the real-time process can avoid being preempted by other processes. In order to prevent some processes from starving to death, operating system will conduct periodic task scheduling according to the priority of process and select the highest priority process from ready queue to acquire the CPU time slice. If the real-time process is preempted in a task scheduling period, it is likely to cause the phenomenon of frame losing and frame timeout, which is unbearable in a real-time simulation program. In the Kylin real-time operating system, we set priority of the process to rt priority to ensure the stability of the frame time. For real-time simulation, more attention is paid to the stability of the frame time, so this paper mainly tests the frame time.

In the test program, the frame time is set to 1ms, and the total time is set to 100s. The simulation model includes 2000 trigonometric functions. In each frame, the starting time and the ending time is read by *clock_gettime()* system call and the time data is recorded in a file. The real-time performance curve of the system can be obtained by calculating the difference between the actual frame time and the ideal frame time. The result is as follows.

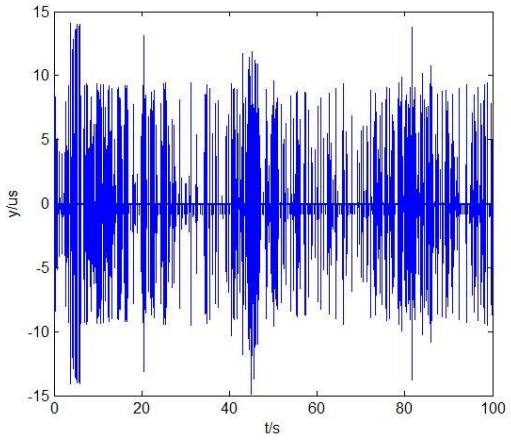


**Figure 4: Jitter of the frame time**

It can be seen from **Fig. 4** that the real-time performance meets the requirements. The maximum frame time error is less than 15 microseconds, and there is no frame leakage phenomenon.

### 5.3 Test of Memory Optimization Algorithm

In the original memory management algorithm, in order to prevent the time-consuming I/O operations from affecting the real-time process, all the frame data will be stored in system memory firstly. The data will not be written into the file until simulation is over. But the obvious drawback of this approach is that it cannot support a long time complex simulation, otherwise the system memory is prone to excessive consumption. In the original algorithm, when the simulation program calculates 2000 trigonometric functions and save 200 parameters with double type in each frame, the memory allocation operation will be failed once the total simulation time is set to more than 16000 seconds. When it comes to the optimized algorithm proposed above, the simulation can execute successfully for 12 hours. Theoretically, it can execute for unlimited time as long as the space of the hard disk is huge enough to store all the frame data. To compare the total running time between two memory management algorithms, the total simulation is set to 10000 seconds. Results of two algorithms are shown in **Fig. 5** and **Tab. 1**.

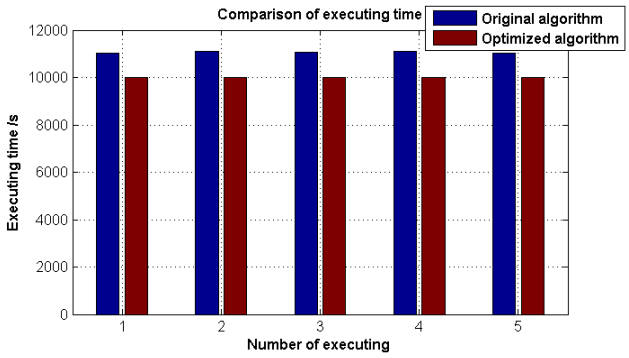


**Figure 5: Comparison of total executing time between original algorithm and optimized algorithm**

**Table 1: Comparison of memory consuming between original algorithm and optimized algorithm when the total simulation time is set to 10000s.**

| Time /s | MSORA | MSOPA |
|---|---|---|
| 1 | $10^3$ | $2 \times 10^3$ |
| 10 | $10^4$ | $2 \times 10^3$ |
| 100 | $10^5$ | $2 \times 10^3$ |
| 1000 | $10^6$ | $2 \times 10^3$ |
| 5000 | $5 \times 10^6$ | $2 \times 10^3$ |
| 10000 | $10^7$ | $2 \times 10^3$ |
| 50000 | / | $2 \times 10^3$ |

In the table above, 'Time' represents the total simulation time. 'MSORA' is the abbreviation of memory consuming of original algorithm and 'MSOPA' is the abbreviation of memory consuming of optimized algorithm. When the simulation time is continuingly increasing, memory consuming with optimized algorithm contains invariable with the number of $2 \times 10^3$ times memory consuming of one frame data. In contrast, memory consuming with original algorithm is increasing linearly along with the simulation time. When the simulation time is longer than 16000s(acquired by testing), program with original algorithm cannot run successfully.

At the same time, with the help of the OpenMP programming language for the parallel operation of the two threads, the total execution time of the simulation is reduced by ten percent when total simulation time is 10000 seconds (see **Fig. 5**).

## 6 Conclusion

In this paper, based on the design experience of the existing real-time simulation software, we construct a software framework for real-time simulation working on Galaxy Kylin operating system. At the same time, the existing memory management algorithm is improved, which makes the memory consumed during the simulation reduces greatly under the premise of meeting the real-time performance requirements, and slightly reduces the total running time of the simulation program. However, in the current framework of real-time simulation software, we have not considered the introduction of graphical modeling. In the next step, we can try to combine the Matlab graphical modeling technology to simplify the modeling process.

## References

[1] Sousa P B, Pereira N, Tovar E. *Enhancing the real-time capabilities of the Linux kernel*[J]. Acm Sigbed Review, 2012, 9(4):45-48.

[2] Jiang Z W. *Design of modeling and real-time simulating software YH-RTSIM based on RTX*[J]. Journal of Computer Applications, 2010, 30(6):1635-1634.

[3] Deepti V, Mahender K, Venkatram N. *Develepment of Real Time Application Platform for Linux Kernel with Preempt-RT*[J]. International Journal of Computer Applications, 2014, 95(22):21-25.

[4] Zhou P, Zhou M T. *Design and Implementation of a High-Resolution Timer in Linux Kernel*[J]. Computer Technology & Development, 2006.

[5] AIAA. *Real-time simulation using Linux*[C]// AIAA Modeling and Simulation Technologies Conference and Exhibit. 2001.

[6] Gurtov A. *Technical Issues of Real-Time Simulation on Linux*[C]// Finnish Data Processing Week. 1999.

[7] Jiang Z, Yao X, Chong W U. *Time Accuracy Analysis of Real-time Simulation Program in Linux OS*[J]. Computer Simulation, 2006, 23(10):76-77.