Concurrency and Operating Systems, Processors, and Programming Languages

Bojin Sun [1, †, *], Minze Sun [2, †]

[1] Software engineering, Taiyuan University of Technology, Taiyuan, 030000, China

[2] Software engineering, Taiyuan University of Technology, Linfen, 041000, China

* Corresponding Author Email: sunbojin5383@link.tyut.edu.cn

†These authors contributed equally to this article.

Abstract. The main content of our research is to optimize and improve the concurrency of the system from the three aspects of operating system, processor and programming language. Previous solutions have focused on hardware techniques to improve concurrency. Our goal is to solve the problem of resource occupancy and contention through algorithms, data structures, optimization interrupts, and optimization of process and coroutine scheduling. In the final ideal case, the finite-state machine switching code generated by the compiler can be used to switch the coroutines in the same process. For threads, the thread code provided by the kernel does the switching. The asynchronous system call is optimized to automatically generate the corresponding system call request code according to the execution compiler and maintain the coroutine control block data structure. This is also the most difficult part of this research.

Keywords: CPU; Process; Thread; Concurrency; Operating Systems; Interrupts.

1. Introduction

This paper is mainly based on doing concurrency with the operating system processor and programming language. This is a broad topic, but in fact the main description is that operating systems are designed to handle concurrency, and in general most applications are executed serially, but not always. When a programmer writes a program. If it's C, it's going to have a main function, and other languages have a main function. When a program starts to execute, it usually follows the main function, there may be function calls and returns in between, but during the execution of a program it's basically serial, meaning that the CPU can only execute one instruction at any one time. For this reason, the computer system is actually very inefficient. If the interactive program is editing text, which is the most common way to use a computer, even though the human typing speed is very fast, the computer is actually idle a lot of the time. So, operating system CPU and programming language these three things from their own Angle to improve the use of computer efficiency. This is the concurrency of operating systems. In the second part, some knowledge and theories related to x86 system will be introduced. X86 can be called 32bit, and x86 is a 32-bit system, where bit stands for 32-bit version of the system. X86 is a standard numbering abbreviation of Intel general computer series, and also identifies a set of general computer instructions. X has nothing to do with processors and is a simple wildcard definition for all x86 systems. X86 stands for 32-bit system, so x86 stands for 32-bit system. X86 memory is more addressable and runs faster than 64-bit.

An operating system provides an environment for its upper-layer applications, where concurrency allows each application program to think that it has only one application on the computer, but only one machine. The way to achieve this state is to set up multiple processes in which the operating system does a process switch, so that one application is running while the other is not running, and the frequency of this switch is very high, so that the user does not feel out, then can achieve the effect of concurrency. This extends down to the fact that the CPU can increase concurrency by providing multiple cores and hyperthreads, in effect taking advantage of the CPU's arithmetic logic unit bus and storing these resources. The reason for improving the degree of concurrency from three aspects is that, from the historical development point of view, CPU was the first to appear and directly used

machine code to write, and then a large number of programming languages appeared and a perfect operating system was developed. Therefore, these three aspects are most directly related to CPU. Both the operating system processor and the programming language will improve it from their own perspective, and the efficiency and final effect will be much greater if several aspects are applied simultaneously. One of the things that we're learning and looking at right now, putting these things together, is that concurrency is actually stop-and-go, which is also called asynchrony, so what we want to look at is to build an operating system based on asynchrony with the support of the operating system language and CPU to achieve the ultimate concurrency performance. There are deadlocks in the process.Methods and systems are provided for locking data objects. According to one implementation, a method is provided that comprises automatically selecting, with a lock service module whether a) logical locks for the data objects are set when changes of the data objects are requested, or b) logical locks for the data objects are set when the first or each further processing module is started or c) no logical locks for the data objects are set when processing the data objects [1].

2. Method

The following chapters mainly introduce interrupt when CPU is abnormal, time slice rotation algorithm, interrupt instruction in x86, some related introduction of threads in the kernel, advantages and disadvantages of kernel mode for operating system users, and coroutine switching.

2.1 CPU Interruption When a CPU Exception Occurs

The most important part of handling concurrency problems for the CPU is the interrupt mechanism. This is shown in Figure 1. Interrupt means that during the execution of a program, a stop or jump occurs due to an exception that is not included in the program. When on a computer to join the external input and output devices at this time of the program cannot be completely run according to the logic of the program design, in this case the CPU calls are executable program will be interrupt interrupted, and then transferred to a program that handle exceptions and special situation, and then wait for the abnormal return event processing is completed, then the original program for processing. this is shown in Figure 2. There is a problem of saving the field when interrupts occur and these operations are mainly dependent on the registers in the CPU. Depending on the purpose of handling, interrupts or exceptions can be divided into three categories: First: exceptions and hardware interrupts generated by all processors. Hardware interrupTS, INCLUDING those from PROCESSOR external INTERRUPT PINS, ARE generated by INTERRUPT or exception HANDLERS in PROTECTED mode. Category 2: Hardware interrupts that can be masked under special circumstances when virtual mode extension is enabled. Category 3: All software interrupts, that is, interrupts that occur when an instruction is executed [2].

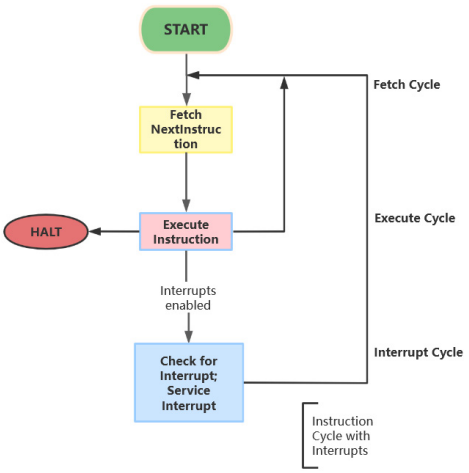


Figure 1. Schematic diagram of interrupt flow

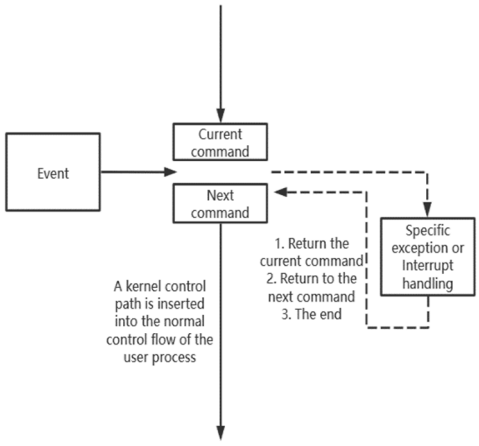


Figure 2. Instruction Cycle with Interrupts

2.2 Time Slice Rotation Algorithm

There is a time slice rotation algorithm in the operating system. When two processes exist, one process executes a time slice and switches to the other process to execute a time slice. This switch is based on clock interrupts. For clock interrupts, it is closely related to the CPU. This can be achieved by concatenating several CPU clock interrupts together, but more often each CPU may have its own speed, so each CPU will have its own clock interrupt. A good time slice rotation algorithm improved method based on time slice rotation scheduling algorithm in the process of the last execution time slice allocation optimization, an improved time slice rotation scheduling algorithm, this algorithm has better real-time performance, while reducing the number of task scheduling and process switching times, reduces the system cost, improve the operation efficiency of the CPU, the operating system The performance has been improved to some extent [3].The idea of progression is also mentioned in several papers. By easy it means that the programmer can communicate with the computer in a language that is easy to understand, as well as far removed from the details of the machine. This decreases the number of errors that occur because of human mistakes. The system is also made more efficient by performing certain tasks on behalf of the programmer or user, as well as distributing the resources in such a manner that they are used as often as possible [4].

2.3 x86 Interrupt Instructions

Intel x86 is a 16-bit microprocessor introduced by Intel Corporation in 1978. X86 refers to a family of Intel 8086-based backward compatible CPU instruction set architectures. In the early days of Intel, processors were named after a number format like 80x86, including Intel 8086, 80186, 80286, 80386, and 80486. Because of the "86" ending, the architecture was called "x86".

On x86 systems, there's an int interrupt, there's an ireturn that's the instruction that's returned from the interrupt and then if you have a user mode interrupt, then you're going to be provided with another instruction called ureturn, and when you process it, you need to know where it's going, so that's UEPC, that is, the current execution to a certain position, stop, after processing the interrupt, then return back to interrupt position, need to break down what causes this information, detailed view at this time may require multiple register for storage, for example a missing page of the instruction execution process, then need to know this instruction is not only due to lack of pages, The second thing we need to know is that the missing page may not be the page where the instruction corresponds to the memory unit, but a memory unit of the data segment accessed by the instruction. In this case, we need a few more registers to indicate that the memory unit accessed is faulty, etc. About the x86 concept in multicore processes. Intel x86 is a 16-bit microprocessor introduced by Intel Corporation in 1978. X86 is a backward compatible CPU instruction set architecture based on Intel 8086. In Intel's early days, processors were named after 80x86-like digital formats, including Intel 8086, 80186, 80286, 80386, and 80486. The architecture is called "x86" because it ends in "86".

On x86 systems, you have an int interrupt, you have an ireturn return instruction from the interrupt, and then if you have a user mode interrupt, then you're going to provide another instruction called ureturn, and when you process it, all you need to know is that this is UEPC, that is, the current execution goes to a certain point, stops, interrupts after processing, and then returns Back to the location of the interrupt, what needs to be broken led to these information, detailed view may require multiple register for storage at this moment, a page missing in the process of instruction execution, for example, you will need to know this instruction not only because of the lack of a page, the second thing is that we need to know the missing pages may not be a instruction memory cells in the corresponding page, It is the memory unit of the data segment that is accessed by the instruction. In this case, we need more registers to indicate that the accessed memory unit is wrong, and so on. In heterogeneous parallel architecture, the traditional core CPU and the computing core GPU work together, which is consistent with the separation of data communication and computing tasks in the data stream programming model. It can fully expose the functional modules that can be parallelized in the data stream program, and rationally allocate hardware resources. However, the processing methods such as task division and scheduling, software pipeline construction and data communication under the traditional CPU cluster platform cannot be simply copied to the heterogeneous cluster platform. Aiming at the multilevel parallel structure of heterogeneous cluster hardware platform, a data stream programming model for heterogeneous cluster of multi-core CPU/ multi-core GPU should be designed based on the data stream application and CPU/GPU heterogeneity characteristics. The data flow programming model mainly includes four modules: two level task division scheduling to realize task unit mapping, hierarchical stage assignment construction software pipeline, MPI/OpenCL hybrid programming model to complete data communication by combining distributed storage and shared storage mode, and C++ and OpenCL hybrid object code generation. In this way, the characteristics of large-scale computing and the potential parallelism of data stream applications in multimedia domain can be maximized. The hardware configuration of heterogeneous cluster architecture is given full play to make CPU and GPU work together, and the two-level task pipeline is constructed through hierarchical division and scheduling to improve the overall execution performance of the program [5].

2.4 Threads in the Kernel

Threads in the operating system, one is the user thread, one is the kernel thread, the user thread is actually equivalent to a process has several things to do, need to coordinate these several things, that can actually be used to coordinate this situation. In the early days, the operating system only supports multiple processes, do not support multithreading, there are some user programs, will write their own logic, can let a number of tasks communicative execution, then need two sub-tasks, the first sub-task cannot progress, to wait for a certain condition, cut to another task. Previous solutions have focused on hardware techniques and software page colouring to alleviate this problem. Our goal is to investigate how and to what extent shared resource contention can be alleviated through thread scheduling. Scheduling is an attractive tool because it requires no additional hardware and is relatively easy to integrate into a system. This is the first comprehensive analysis of scheduling only dispute mitigation techniques. The goal is to find a classification scheme for threads that determines how they affect each other when competing for shared resources [6]. Keep doing it, wait for that condition to be satisfied and then go back and do it. You do a system call, which is one of the things that you have to do, use the services of the operating system, and then you go into the kernel, and all the concurrent stuff in user mode is masked, because you're going into the kernel. In a traditional operating system, this is something that stops the process. The kernel finishes executing and then comes back. Since each user program does this on its own in a user thread, even if a library is provided, it requires a library to be shared in each process's technical space. A new improved method of multi-thread quicksort algorithm is also used to develop the optimization algorithm through C++ multi-thread programming. Secondly, the performance of the optimization algorithm is improved by comparing

the algorithm performance. Finally, the optimization algorithm is theoretically analyzed, and the upper limit of performance improvement is obtained under the optimal condition [7].

2.5 Advantages and Disadvantages of Operating System User Mode Kernel

Security is a problem in user mode, and the operating system kernel supports it so that when a system call is made, other parts of the same process can also run. This gives you kernel threads, the equivalent of having the operating system support to create a new thread. To perform a specified function, use this API, you can create a thread to create, if there are a multi-threaded, multiple threads are carried out in system calls, that this time will not affect the concurrency, so it is of advantages, but there is a problem, if you want to for switching between threads, in a process, the equivalent of all the staff in the bank and the same. In the coordination between employees, there is no window like the customer, only through the window, to carry out orders, customers can carry out more free and flexible coordination. Now, with more Windows, the conversion from user mode to kernel mode will save money, so it will cost resources. In user mode, when the code is executed, there is a function call stack. After cutting to the kernel, the kernel will process the services required by my process. In the kernel, there is also a function call stack. For security purposes, user-mode function stacks and kernel-mode function stacks, the kernel stack is the process that pushes the stack when the kernel executes a system call. When the kernel completes the orbit, nothing is left in the user-mode kernel state. Because when the kernel stack is empty, the threads created basically have only the stack resources. Less than 50 percent of those resources are being utilized synthetically, in the kernel, transactions in the user stack need to be saved, but most of you are executing code in user mode. During this time, the kernel stack must be empty, and resource utilization is inefficient. From a memory efficiency point of view, this is not possible. For thread, there is no unified address space, but each has a stack, so to switch from the thread, to save and restore the whole content of the stack, then the kernel into a separate process, the kernel stack can be eliminated, because the two processes communicate, does not need to set up in another process to communicate with the stack. In view of the characteristics of workflow in complex application domain, a method to ensure the correctness of workflow concurrent execution is proposed. Firstly, the concepts of task conflict, conflict category, shared lock and mutex of the same name are defined, and then a workflow concurrency control mechanism based on these concepts is proposed. In this concurrency scheduling mechanism, due to the full consideration of the concurrency control of different workflows and the concurrency control between different instances of the same workflow, the concurrency mechanism can ensure the correctness of the concurrent execution of multiple workflows and multiple instances in complex application environments [8]. Linux operating system, embedded system and communication system are generally written in C/C++ language. Because C language has the characteristics of low - level hardware, strong portability and high execution efficiency. But with the advent of multi-core parallel machines, many languages have begun to support multithreaded programming. Because C language itself has the problem of not checking the memory boundary when accessing the memory, it causes the reliability and security problems of the software system. For multithreaded C programs, it is more difficult to verify the memory security of multithreaded C programs at runtime because of the uncertainty of multithreaded programs. Through the use of improved pointer runtime verification technology, multi-core and multi-thread technology, parallel computing, lock-free data structure technology, source code pile technology, and combined with the open-source tool Clang compiler to achieve the prototype tool Movec to support multi-thread C program. This tool realizes the runtime verification of the memory security of multithreaded C programs. Then, Mibench and SARD test cases are used to verify the effectiveness of the tool for runtime verification of multithreaded C programs [9].

2.6 Switch of Coroutines

A coroutine is equivalent to a state machine. In process scheduling, the state machine is maintained by the operating system. In coroutine, the state machine is run by the compiler. The code maintained by this state machine can be thought of as a library embedded in the compiler. That is, library subcode

that adds thread coroutines to the compiler. However, thread support cannot be used across operating systems because stack resources are allocated and recycled in an operating system dependent manner. Coroutines do not have separate stacks, whereas threads can have separate stacks. When the CPU's scheduling object is the compiler, the operating system and the hardware together, just switch the coroutines. In this model, an object is able to change its behav- ior dynamically to adapt to the current environment by modifying its internal structure [10].

3. Result

3.1 Ideal Coroutine Switching Process (Scheduling of Coroutines, Threads, and Processes)

Coroutine switching can be represented as the selection of the next ready coroutine in different processes and different threads.

1. Coroutine switching: switching between coroutines in the same process;

Finite-state machine switching code automatically generated by the compiler completes the coroutine switching;

2. Thread switching: switching between preempted coroutines in the same process;

Switching requires user stack saving and switching;

Thread switching is accomplished by thread switching code provided by the kernel;

3. Process switching: switching between different processes due to preemption;

Save the current user stack, switch to kernel state, complete the process address space.

3.2 Ideal for Asynchronous System Calls

1. User mode asynchronous system call will execute the compiler to automatically generate the corresponding system call request code, maintain the coroutine control block data structure;

2. The kernel is entered when the first system call is requested and when the last system call is completed.

3. The intermediate system calls do only the request submission and result query of the system call, and do process, thread, or coroutine switching.

4. Coroutine switching is done only when the system call of the current coroutine has not returned a result and there are no new executable user tasks.

4. Conclusion

Through this research on the concurrency and cooperation of the operating system kernel, we found that the open-source concurrency and cooperation will be a very potential development strategy for the operating system. The languages using the operating system kernel, such as RISC-V and RUST, have been widely concerned and used, and have been widely recognized.

At the same time, the collaboration of operating system with CPU instruction set and programming language may lead to innovative ideas, which can improve the performance of operating system, reduce the difficulty of operating system development, and reduce the vulnerability of operating system.

Under the cooperation of hardware technology and compilation technology, asynchronous programming technology may deeply influence the development of operating system, which provides more help for the further development and utilization of operating system.

References

[1] Rapp R. Methods, computer systems and software applications for providing a lock service: US07206794B2[P]. 2007.

[2] Pei Ya-li. Research and Implementation of CPU Interrupt Mechanism in X86 Architecture [D]. Beijing Jiaotong University, 2012. in Chinese.

[3] Xiao Jianming, Zhang Xiangli. An Improved time slice rotation scheduling algorithm [J]. Computer Applications, 2005, 25(B12):2. in Chinese.

[4] Seidl H. Programming Languages and Systems: Proceedings[J]. Handbook of Statistics, 1993, 9(4):141-168.

[5] Yang Rui-Rui. Research on Data Stream Programming Model for Multi-core CPU/ Multi-core GPU Heterogeneous Cluster [D]. Huazhong University of Science and Technology, 2019.in Chinese.

[6] Zhuravlev S, Blagodurov S, Fedorova A. Addressing Shared Resource Contention in Multicore Processors via Scheduling[C]// Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2010, Pittsburgh, Pennsylvania, USA, March 13-17, 2010. 2010: p.129-141.

[7] Li Tianrui, Cao Qingnian, Meng Kaiyuan. Design and optimization of multi-thread Quicksort algorithm [J]. Wireless Internet Technology, 2022, 19(7):3.in Chinese.

[8] Song Baoyan, Yu Ge, Ge Wei, et al. Workflow Concurrency Control Mechanism for Complex Applications [J]. Journal of Northeastern University: Natural Science Edition, 2002, 23(1):4. in Chinese.

[9] Chen Tao, wang mingming. Multi-threaded programs-oriented memory security runtime validation [J]. Journal of computing technology and automation, 2019, 38 (02): 102-107. The DOI: 10.16339 / j.carol carroll nki jsjsyzdh. 201902019.in Chinese.

[10] Yamamoto S. Introduction of Coroutines for the Self-Reconfigurable Objects. 2013.