ISSN:0975-9646

A Modified O (1) Algorithm for Real Time Task in Operating System

Rohan R. Kabugade[#1], S. S Dhotre [#2], S H Patil [#3]

[#1]M.Tech Computer Department, Bharati Vidyapeeth University College of Engineering Pune,India
[#2]Associate Professor Computer Department, Bharati Vidyapeeth University College of Engineering Pune,India
[#3] Professor Computer Department, Bharati Vidyapeeth University College of Engineering Pune,India

Abstract - The main goal of the proposed architecture is to provide a fast scheduling algorithm, which makes a perfect balance between fairness and quick response. In this project, we presents a modified algorithm named MOFRT (Modify O (1) For Real-Time) based on the Linux kernel 3.2 to improve the Queue Management for Real time Tasks. Though, some of these algorithms have not been implemented since it is very hard to support new scheduling algorithms on nearly every operating system. To solve this problem, we improve the scheduling mechanism in Linux to provide a elastic scheduling framework, we select the kernel in 3.2 to improve, because the O(1) schedule algorithm is very high powered and fair. We reserve I/O waiting queue to reduce the response time, eliminate the expired queue to enhance the stability of real time tasks, and use dynamic calculation methods to distribute time slice and priority.

Keywords — kernel, real-time OS, run queue, schedule.

I. INTRODUCTION

Real-time computing is required in many application domains, such as avionics systems, traffic control system, automated factory systems. Each application has peculiar characteristics in terms of timing constraints and computational requirements s (such as periodicity, criticality of the deadlines, response time, etc). Some mission-critical real-time systems may suffer irreparable damages if a deadline is missed. It is the system builder's responsibility to choose an operating system that can support and schedule these jobs according to their timing specifications so that no deadline will be missed.

The basic data structure in the scheduler is the run queue. The run queue is the list of runnable processes on a given processor; there is one run queue per processor. Each runnable process is on exactly single run queue.
On the other hand, some soft real-time applications such as streaming audio/video and multiplayer games also have timing constraints and require performance guarantees from the underlying operating system. The application output provided to users is optimized by meeting the maximum number of real-time constraints (e.g., deadlines). But unlike hard real-time applications, occasional violations of these constraints may not result in a useless execution of the application or catastrophic consequences.

The use of computers for control and monitoring of industrial processes has expanded greatly in recently, and will probably expand even more dramatically in the near prospect.. In other installations, however, an efficient use of the computer can only be achieved by a careful scheduling of the time-critical control and monitor functions themselves. Two scheduling algorithms for this type of programming are studied; both are priority driven and pre-emptive; meaning that the processing of any task is interrupted by a request for any higher priority task. The first algorithm used to study a fixed priority assignment and can achieve processor utilization. The second scheduling algorithm can achieve full processor utilization by assigning priorities dynamically.

A priority-based scheduling is a common type of scheduling algorithm. The thought is to rank processes based on their worth and need for processor time. A higher priority processes run before the lower priority processes, Linux builds on this idea and provides dynamic priority-based scheduling. To fulfil scheduling objectives we begin with an initial base priority and then enable the scheduler to increase or decrease the priority dynamically. Linux is pre-emptive, when a process enters the task running state, the kernel checks whether its priority is higher than the priority of the currently executing process. If it is, the scheduler is invoked to anticipate the currently executing process and run the newly runnable process. In addition, when time slice of a process reaches zero, then pre-emption process is done.

Advances in computer technology have also dramatically changed the design of many real-time controller devices that are being used on a daily basis. Many traditional mechanical controllers have been gradually replaced by digital chips that are much cheaper and more powerful. In fact, we believe that the computing power of future embedded digital controllers will be at the same level as that in today's big system servers. As a result, future embedded devices must be able to handle complex application requirements, real-time or otherwise. How we can design real-time operating systems (RTOSs) to support applications with mixed real-time and non real-time performance requirements will be an important issue.
These three types of timing requirements (hard real-time, soft real-time, and non real-time) are all important for many real-time systems. It is the goal of our research to make MOFRT to satisfy these different requirements.

II. RELATED WORK

This section gives the overview of the research work carried out related to the Queue Management. This overview mainly focuses on the Improvement of queue management and Improvement of process analysis.

Wang Chi Zhou Huaibei, Ma Chao Chen Nian. in [3] has proposed an approach to Modified O(1) Scheduling Algorithm for Real-Time Tasks. In this author presents a modified algorithm named MOFRT(Modify O(1) For Real-Time) based on the Linux kernel. Researchers in the real-time system community have intended and studied many advanced scheduling algorithms. On the other hand, most of these algorithms have not been implemented since it is very difficult to support new scheduling algorithms on most operating systems. To solve this problem, they enhance the scheduling mechanism in Linux to provide a flexible scheduling framework, they choose the kernel in 2.6.11 edition to improve, because the O(1) schedule algorithm is very high-powered and fair. The main goal of this architecture they present is that with the help of fast prototyping scheduling algorithms, that makes a perfect balance between quick response and fairness. With the help of reserve I/O waiting queue to diminish the response time, remove the expired queue to improve the steadiness of real-time tasks, and use dynamic calculation methods to distribute time slice and priority.

The design of an Operating System (OS) scheduler is meant to allocate its resources to all applications. Wong C.S., Tan I.K.T. , Kumari R.D., Lam J.W., and Fun W. [2] has introduced the scheduling techniques used by two Linux schedulers first is O(1) and second is Completely Fair Scheduler (CFS). The CFS is the Linux kernel scheduler that replaces the O(1) scheduler in the 2.6.23 kernel. The goal of design for CFS is to provide fair CPU resource allocation among executing tasks without degrading the interactive performance. To prevent the starvation it is necessary to achieve good fairness in distributing CPU resource among tasks. Though there are many conventional operating system benchmarks that are geared towards measuring systems performance in terms of throughput these design goals have never been scientifically evaluated despite. Therefore they scientifically evaluate the design goals of CFS by empirical evaluation. Also by using fairness and interactivity benchmarks, they measure the fairness and interactivity performance. Comparisons of CFS kernel and O(1) schedulers of the open source Linux OS are used to provide a meaningful representation of results. So the experience indicated that the CFS does achieve its design goals.

Liu CL, Layland JW in [5] has discussed about Scheduling algorithms for multiprogramming in a hard real-time environment. In this authors explain that the problem of multi-program scheduling on a single processor is studied from the viewpoint of the characteristics peculiar to the program functions that need guaranteed ser- vice. It is shown that an optimum fixed priority scheduler possesses an upper bound to processor operation which may be as low as 70 percent for large task sets. It is also shown that full processor use can be achieved by dynamically assigning priorities on the basis of their recent deadlines. They also discussed about the combination of these two scheduling techniques.

CPU scheduler is a very important subsystem which affects fairness and interactivity. Development of Linux kernel is comparatively fast-paced. In many CPU schedulers have been designed by kernel hackers and researchers. It is necessary to accurately analyse and compare different characteristics among these schedulers, so as to design and understand better CPU schedulers for various applications. However, to compare and analyse these CPU schedulers precisely, researchers lack a straight-forward method. Shen Wang, Yu Chen, Wei Jiang, Peng Li, Ting Dai and Yan Cui [4] has systematically analyse and measure interactivity, fairness and multi-processors performance of three schedulers: O(1), CFS and RSDL, by using micro, synthesis and real application benchmarks. In Linux kernel-2.6.29, all these schedulers have been ported in a single scheduler framework. Experimental results show that there minor differences in synthesis and real applications and a notable differences in fairness and interactivity under micro benchmarks. The impact of implementations of schedulers on fairness and interactivity of applications also has been analysed. Also it discusses the challenges in estimating application resource requirements in different environments. They also present some ideas for developing future CPU schedulers.

The process scheduler is an important part of the kernel because running processes is the point of using the computer in the first place. Juggling the demand of process scheduling are nontrivial. However, a large number of runnable processes scalability concerns tradeoffs between latency and throughput, and the demand of various work load make a one size fits all algorithm hard to find. The Linux kernel new process scheduler, however, comes very close to appeasing all parties and providing an optimal solution for all case with perfect scalability. The 2.6 Linux kernel introduces a completely new scheduler that's commonly referred to as the O(1) scheduler. The scheduler can perform the scheduling of task in constant time. M A Wei-feng and WANG jai-hai [1, 6] describes how a task is executed on single CPU. They also mention data structures like runqueues, priority array and process descriptor.

In [6] author explains that the Linux kernel is one of the most interesting yet least understood open-source projects. It is also a basis for developing new kernel code. That is why Sams is excited to bring you the latest Linux kernel development information from a Novell insider in the second edition of Linux Kernel Development. This authoritative, practical guide will help you better understand the Linux kernel through updated coverage of all the major subsystems, new features associated with Linux 2.6 kernel and insider information on not-yet-released developments. You'll be able to take an in-depth look at Linux kernel from both a theoretical and an applied perspective as you cover a wide range of topics, including algorithms, system call interface, paging strategies and kernel synchronization. Get the top information right from the source in Linux Kernel Development.

III. PROPOSED SYSTEM

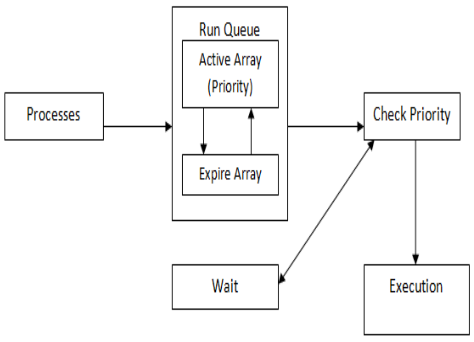

Fig. 1 System Architecture

i. **Processes**- It is a small task in execution. Processes are given as an input to the system.

ii. **Run Queue:** Run queue contains all runnable process. This run queue contains the two arrays:

a. **Active Array:** All the tasks in the associated run queue that have time slice left are contained in the active array

b. **Expire Array:** The expired array contains all the tasks in the associated run queue that have exhausted their time slice.

iii. **Check Priority:** Each priority array contains one queue of runnable processors per priority level. For discovering the highest priority runnable task in the system, we use a priority bitmap, which contained in priority array. This will improve the performance of the system by different scheduling techniques.

iv. **Wait:** It consists of the processes in waiting state.

v. **Execution:** It consists of the processes in execution state.

IV. CONCLUSIONS

This paper concentrates on Scheduling techniques for real time system. Process scheduling is a frequently overlooked determinant of real-time performance. This paper presents a modified algorithm base on Linux kernel for real-time system. It is the mixture of normal operation system and real-time operation system. We modify Linux to satisfy multi functionality of the real-time system. We can deal with real-time tasks rapidly and accurately by using scheduling techniques. We also improve scheduling algorithms compatibility with other Real-time System.

REFERENCES

[1] MA Wei-feng, WANG Jai-hai "Analysis of the Linux 2.6 kernel scheduler". 2010 International Conference on Computer Design and Application (ICCDA 2010).

[2] Wong C.S., Tan I.K.T. , Kumari R.D., Lam J.W., Fun W "Fairness and Interactive Performance of O(1) and CFS Linux Kernel Scheduler". IEEE 2008.

[3] Wang Chi Zhou Huaibei, Ma Chao Chen Nian. "A Modified O(1) Scheduling Algorithm for Real-Time Tasks". In: Proc. of the IEEE, 2006.

[4] Shen Wang, Yu Chen, Wei Jiang, Peng Li, Ting Dai and Yan Cui "Fairness and Interactivity of Three CPU Schedulers in Linux".2009 15th IEEE international Conference on Embedded and Real-Time Computing Systems and Applications

[5] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM, 1973.

[6] Robert Love Linux Kernel Development Second Edition Jan 2006.