Computers and Electrical Engineering 37 (2011) 1090–1105

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

On Linux starvation of CPU-bound processes in the presence of network I/O ☆

K. Salah [a,*], A. Manea [b], S. Zeadally [c], Jose M. Alcaraz Calero [d,e]

[a] Computer Engineering Department, Khalifa University of Science Technology and Research (KUSTAR), Sharjah, United Arab Emirates
[b] Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia
[c] Department of Computer Science and Information Technology, University of the District of Columbia, Washington, DC 20008, USA
[d] Department of Communications and Information Engineering, University of Murcia, Murcia 30011, Spain
[e] Hewlett-Packard Labs, Cloud and Security Lab, Bristol BS34 8QZ, UK

ARTICLE INFO

Article history:
Received 7 November 2010
Received in revised form 3 July 2011
Accepted 4 July 2011
Available online 29 July 2011

ABSTRACT

Process starvation is a critical and challenging design problem in operating systems. A slight starvation of processes can lead to undesirable response times. In this paper, we experimentally demonstrate that Linux can starve CPU-bound processes in the presence of network I/O-bound processes. Surprisingly, the starvation of CPU-bound processes can be encountered at only a particular range of traffic rates being received by network processes. Lower or higher traffic rates do not exhibit starvation. We have analyzed it under different network applications, system settings and network configurations. We show that such starvation may exist for the two Linux scheduler, namely the 2.6 O(1) scheduler and the more recent 2.6 Completely Fair Scheduler (CFS). We instrumented and profiled the Linux kernel to investigate the underlying root causes of such starvation. In addition, we suggest possible mitigation solutions for both schedulers.

1. Introduction

Nowadays, many computer systems are using the Linux operating system around the world. Linux is being used in aerospace, military, government, surgery, and definitively, in many critical services. The Linux kernel is the core of this operating system. The kernel provides some basic services such as memory management, processes scheduling, input/output management, and others. Any problem in the kernel may drastically affect the proper execution of applications, which in turn, may impact services.

The design of operating systems entails many challenges, one of which is to ensure that processes do not encounter starvation while running. The starvation problem is defined as a system state where a process is perpetually denied to use system resources, and the program cannot finish its task without those resources. There are numerous causes for starvation. For example, starvation may arise when the operating system receives many interruption requests (Input/Output (I/O) requests). In this case, the running processes may never receive Central Processing Unit (CPU) time since the kernel uses all the CPU time processing these interruptions. The process scheduler is a module of the kernel in charge of managing the CPU resources fairly among the different processes running in the system without affecting the inherent behavior of any process [1].

The main objective of this paper is to demonstrate that CPU starvation can occur in the current Linux kernel 2.6. Such starvation is exhibited by CPU-bound processes which are non-interactive processes that require intensive CPU computation. The problem particularly occurs when network I/O-bound processes run in the system at the same time. In contrast to

☆ Reviews processed and approved for publication to the Editor-in-Chief Dr. Manu Malek.

* Corresponding author.
E-mail addresses: khaled.salah@kustar.ac.ae (K. Salah), manea@kfupm.edu.sa (A. Manea), szeadally@udc.edu (S. Zeadally), jmalcaraz@um.es (J.M. Alcaraz Calero).

0045-7906/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compeleceng.2011.07.001

CPU-bound processes, network-I/O processes are non-interactive processes that use network I/O intensively. Examples of CPU-bound processes may include password cracking, simulation, and mathematical computation such as that of PI and matrix manipulation. Examples of network I/O bound processes may include Web and FTP servers, online game servers and applications, and multimedia streaming servers and applications. In order to establish the scope of this problem, this paper takes into account many experimental conditions covering different types of network I/O-bound processes, network interface cards, single and multiple processor mainboard architectures, and two Linux schedulers in order to demonstrate the occurrence of the starvation problem under all these conditions. Moreover, an in-depth analysis of the measurement results have been done to identify the root causes of such starvation. Finally, proposed solutions that can mitigate such starvation are given for both Linux Schedulers, namely the *2.6 O(1)* scheduler and the more recent *2.6 Completely Fair Scheduler* (*CFS*), henceforth referred as *2.6 CFS*.

The rest of the paper is structured as follows. Section 2 describes related works on the starvation problem in the Linux kernel. The different experiments to detect the starvation problem in CPU-bound processes when there are network I/O-bound processes in the system are described in Section 3. Section 4 describes an analysis of different Linux schedulers identifying the root causes of the starvation problem. Section 5 proposes some ideas in the design of the Linux schedulers in order to minimize the starvation problem. Finally, Section 6 makes some concluding remarks and presents future works.

2. Related work

Several research works have focused on the study of the Linux kernels and the starvation problem. An in-depth documentation about how *Linux 2.6 CFS* and *2.6 O(1)* schedulers work is given in [2,3]. The author points out the importance of treating tasks with a certain degree of fairness and to ensure that threads never starve in the *scheduler*.

One of the major changes in the *Linux kernel 2.6* was an entirely new scheduling algorithm. This scheduling algorithm was advertised for its *O(1)* time complexity, and thus it was widely called *Linux 2.6 O(1) scheduler*. This scheduler gained wide acceptance among the Linux community over the years. This scheduler was designed to achieve the following new features: (i) *O(1)* complexity on the time needed to choose the next process to run. (ii) Quick response to interactive processes under high system load. (iii) An acceptable level of prevention of both starving and hogging. (iv) Scalability and task affinity under symmetric multiprocessing environment. (v) Improved performance when having a small number of processes [4,5].

Linux 2.6 O(1) scheduler has recently been replaced in the *Linux kernel 2.6.23* with a new scheduling algorithm, namely, the *Completely Fair Scheduler* (*CFS*). The *CFS* was designed to maximize CPU utilization and the system interactive performance by fixing the deficiencies found in the previous *O(1)* scheduler, especially its complex interactivity heuristics. The basic idea behind the Linux CFS is to emulate an ideal and precise multitasking uniprocessor CPU, which is able to run all the processes in the system in parallel and with an equal speed of *1/n*, where n is the number of runnable tasks in the system, by making an equal and fair division of the CPU bandwidth among all tasks in the system. One of the main features of CFS, besides its "fair scheduling" algorithm, is its modular design, where processes are scheduled according to their scheduling classes. Another important difference is that this scheduler has no notion of a specific pre-set timeslice per process. Rather, the timeslice is totally dynamic and its calculation is implicitly accomplished by the CFS algorithm [6].

Kang et al. [7] describe why the *Linux 2.6 O(1) scheduler* is not suitable to support real-time tasks. The authors experimentally prove unexpected execution latency of real-time tasks which in many cases are due to starvation problems. They also provide a new Linux scheduling algorithm based on weighted average priority inheritance protocol (WAPIP), a variation of the priority inheritance protocol (PIP) [8] which assigns priorities to kernel-level processes at runtime by monitoring the activities of user-level real-time tasks. The new algorithm improves significantly the latency of the real-time tasks. Li et al. [9] also provide a non-preemptive algorithm suitable for soft real-time systems, now based on the usage of dynamic grouping of tasks with deadlines that are very close to each other and schedules tasks within the group.

Torrey et al. [4] propose a new Linux scheduler and they compare the new proposal with the current *Linux 2.6 O(1) scheduler* providing comparable results in all response time tests and showing some inadvertent improvements in turnaround time. An advantage of this new scheduler is that there is no method for differentiating interactive tasks, which minimizes the overheads associated with the scheduler. Bozyigit et al. [10] provided a modified version of the Linux kernel (including the scheduler) which enabling an integrated task migration for clusters of workstations.

Wong et al. [11] provide a comparison of 2.6 *O(1)* and *CFS* Linux schedulers noting that *CFS* is fairer in CPU time distribution without compromising the performance for interactive processes significantly higher than *O(1)*. Moreover, authors prove that CFS is more efficient than *O(1)* mainly due to the complex algorithms used to identify interactive tasks for the *O(1) scheduler*. Wang et al. [12] also provides a comparison of the Linux kernels: 2.6 O(1), CFS and Rotating Staircase Deadline Scheduler (RSDL) analyzing and measuring fairness, interactivity and multi-processors performance in micro, real and synthesis applications. Authors prove that there are notable differences in fairness and interactivity under micro benchmarks, while minor differences in synthesis and real applications.

Salah et al. [13,14] provide an interruption handling scheme for the Linux kernel which minimizes the overheads caused by heavy incoming network traffic. Their past research efforts focused on avoiding the starvation of user processes as a result of the time spent to handle incoming interrupt requests. Moreover, they prove experimentally that their proposal improves significantly the results for general-purpose network desktops or servers running network I/O bound applications, when subjecting such network hosts to both light and heavy traffic loads [15].