Chapter 2.18
Swap Token:
Rethink the Application of the LRU Principle on Paging to Remove System Thrashing

Song Jiang
Wayne State University, USA

ABSTRACT

Most computer systems use the global page replacement policy based on the LRU principle to reduce page faults. The LRU principle for the global page replacement dictates that a Least Recently Used (LRU) page, or the least active page in a general sense, should be selected for replacement in the entire user memory space. However, in a multiprogramming environment under high memory load, an indiscriminate use of the principle can lead to system thrashing, in which all processes spend most of their time waiting for the disk service instead of making progress. In this chapter, we will rethink the application of the LRU principle on global paging to identify one of root causes for thrashing, and describe a mechanism, named as swap token, to solve the issue. The mechnaim is simple in its design and implementation but highly effective in alleviating or removing thrashing. A key feature of the swap token mechanism is that it can distinguish the conditions for an LRU page, or a page that has not been used for relatively long period of time, to be generated and accordingly categorize LRU pages into two types: true and false LRU pages. The mechanism identifies false LRU pages to avoid use of the LRU principle on these pages, in order to remove thrashing.

A prototype implementation of the swap token mechanism in the Linux kernel as well as some experiment measurements are presented. The experiment results show that the mechanism can consistently reduce the program execution slowdown in a multiprogramming environment including SPEC2000 programs and other memory-intensive applications by up to 67%. The slowdown reductions mainly come from reductions of up to 95% of total page faults during program interactions. This chapter also shows that the mechanism introduces little overhead to program executions, and its implementations on Linux (and Unix) systems are straightforward.

DOI: 10.4018/978-1-61350-456-7.ch2.18

INTRODUCTION

The virtual memory system allocates physical memory to multiple concurrently running programs in a computer system through a global page replacement algorithm, especially when the aggregate memory demand is larger than the available physical memory space. A commonly used replacement algorithm in a virtual memory management is the global Least Recent Used (LRU) replacement, which selects an LRU memory page, or the least actively used page, for replacement throughout the entire user memory space of the system. According to the observed common memory reference behavior, the LRU replacement policy takes the assumption that a page will not be used again in the near future if it has not been accessed for a certain period of time. In a single programming environment where only one process is running at a time, this assumption as well as the corresponding LRU principle, which always selects LRU pages for replacement -- hold well for many application programs, leading to an efficient memory use for their execution. However, as the assumption and the principle are directly adopted in memory management designs and implementations for multiprogramming systems, many of computing practitioners can experience following difficulty in their program executions. When the aggregate memory demand of multiple concurrently running programs exceeds the available user memory space to a certain degree, the system starts thrashing --- none of the processes are able to establish their working sets, causing a large number of page faults in the system, low CPU utilization, and a long delay for each process. Although a large amount of CPU cycles are wasted due to the excessive page faults in the shared use of the memory, people seem to have accepted this reality, and to believe that these additional cycles are unavoidable due to the memory shortage and due to the fairness requirement for the concurrently running programs.

As the LRU principle is based on access patterns exhibited in one program's execution, a direct application of the principle on the concurrently running programs is problematic and may cause system thrashing. Let us take a close look into the way an LRU replacement policy is implemented in a multiprogramming system. An allocated memory page of a process will become a replacement candidate according to the LRU principle if the page has not been accessed for a certain period of time under two conditions: (1) the process does not need to access the page; and (2) the process is conducting page faults (a sleeping process) so that it is *not able* to access the page although it might have done so without the page faults. We call the LRU pages generated on the first condition *true LRU pages*, and those on the second condition *false LRU pages*. These false LRU pages are produced by the time delay of page faults, not by the access delay of the process. Therefore, this delay does not necessarily hint that the page is not going to be accessed again by the process soon, or the LRU assumption is not applicable for the false LRU pages. However, LRU page replacement implementations do not distinguish these two types of LRU pages, and treats them equally by attempting to replace any LRU pages!

Whenever page faults occur due to memory shortage in a multiprogramming environment, false LRU pages of a process can be generated, which will weaken the ability of the process to achieve its working set. For example, if a process does not access its already obtained memory pages on the false LRU condition, these pages may become replacement candidates (the LRU pages) when the memory space is being demanded by other processes. When the process is ready to use these pages in its execution turn, these LRU pages may have been replaced to satisfy memory demands from other processes. The process then has to ask the virtual memory system to retrieve these pages back probably by generating and replacing false LRU pages from other processes.