Operating systems

Page Replacement Algorithms

Abstract

Rujuta Shah (121044)
Sohum Shah (121056)

Paging

Memory management in an operating system, deals with allocating memory to programs when they need it and freeing the memory when it is no longer needed. This requires the frequent I/O actions of copying memory from secondary storage to primary storage and vice versa. One such scheme that deals with this is **paging**.

Paging manages the storage and retrieval of data from the secondary storage, for usage in the main memory. The OS retrieves data from the secondary storage in blocks, known as pages. The main advantage of paging over other memory management schemes is that it allows the physical address space of a process to ne non-contiguous. Paging is especially important when a process generates data larger than the RAM, or it tries to access a relatively large memory. In this case, all processes are allowed to access only some of the pages in the physical memory, and rest in a **virtual memory.**

Why Page replacement

Page Fault

Page fault is the scenario when a running program tries to access a memory page, which even though is mapped into virtual address space, but not loaded (or mapped) to the physical memory. Processes are allowed to run with only some of the pages in their address spaces actually resident in the physical computer memory. As long as they only reference the code and data that is resident, there is no problem. As soon as they reference a "virtual memory" location that is not resident in physical memory (page fault), the operating system must bring in the page that was referenced, replacing some other page if necessary.

Need of a page replacement algorithm

If a page fault occurs, then there arises a need of loading a page from the virtual memory into the physical RAM. If there is not enough available RAM, then an existing page needs to be evicted from the RAM and should be replaced by the page in need, from the virtual memory. This is known as **page replacement.**

Page replacement algorithm

Page replacement algorithms decide which memory pages to page out (swap out, write to disk) when a page of memory needs to be allocated. If the page that is to be evicted has been dynamically allocated by a program, or if a program has modified the contents in it (i.e. the page has become dirty), the page needs to be rewritten into a secondary storage before eviction. If at a later stage, a process makes reference to that memory page in the virtual memory, another page-fault occurs, and another empty page needs to be fetched from the RAM or a page must be evicted and replaced.

This method involves constant I/O actions, which are very slow. This determines the **quality** of the algorithm. The less time waiting for page-ins (page replacements) better is the algorithm. A page replacement algorithm looks at the limited information about accesses to the pages provided by hardware, and tries to guess which pages should be replaced to minimize the total number of page

misses, while balancing this with the costs (primary storage and processor time) of the algorithm itself.

Efficient paging systems must determine the page frame to empty by choosing one that is least likely to be needed within a short time. There are various page replacement algorithms that try to do this. Most operating systems use some approximation of the least recently used (LRU) page replacement algorithm.

Existing Algorithms

Theoretically optimal page replacement algorithm

It works as follows: when a page needs to be loaded in the RAM, the operating system replaces out the page whose next use, according to the algorithm, will occur farthest in the future. This algorithm cannot be implemented efficiently because it is impossible to compute reliably how long it will be before a page is going to be used.

First-in First-out

The operating system keeps track of all the pages in memory in a queue, with the most recent arrival at the back, and the oldest arrival in front. When a page needs to be replaced, the page at the front of the queue is selected. While FIFO is cheap and intuitive, it performs poorly in practical application.

Least recently used

LRU works on the idea that pages that have been most heavily used in the past few instructions are most likely to be used heavily in the next few instructions too. While LRU can provide near-optimal performance in theory, it is rather expensive to implement in practice.

Not frequently used

In this algorithm, counters are managed that keep track of how frequently a page has been used. Thus, the page with the lowest counter can be swapped out when necessary.

Implementation

Our goal would be to understand various page replacement algorithms and to identify the one which is feasible and efficient to be implemented. The goal would be to change the policies of pages in Linux.

In Linux, the routine that looks after the efficiency of the memory management unit is the Kernel Swap Daemon (kswapd()). A code: vmscan.c looks after the page replacement in Linux, which essentially uses kswapd routine to decide which pages must be replaced.

Kswapd()

The basic task of kswapd is to swap modified pages into the swap file. Linus OS maintains a kernel swap timer. The daemon is initialized when init process is called during the startup time. It then waits till the kernel swap timer periodically times out. Every time the timer expires, the swap daemon looks to see if the number of free pages in the system is getting too low. If there are enough

free pages, the swap daemon sleeps until its timer expires again, otherwise the swap daemon tries three ways to reduce the number of physical pages being used by the system:

- Reducing the size of the buffer and page caches,
- Swapping out shared pages,
- Swapping out or discarding pages.

By default, the swap daemon tries to free up 4 pages each time it runs. The above methods are each tried in turn until enough pages have been freed. The swap daemon then sleeps again until its timer expires.

This task is accomplished in two stages:

First, it looks around for pages that can be released without having to write data to the disk. If it can recover enough pages this way, it goes back to sleep. If not, it enters swap stage (page replacement).

The swap stage scans through its page tables looking for victims. Every page is a potential victim. The reason why a page won't be evicted if that page is in some weird state (e.g., some other thread has the page locked at the moment). Mainly, though, those pages are going to be evicted.

We would try to understand this routine, and the code that uses it: vmscan.c. We would also try and modify the code to see if it can be made more efficient or feasible.

References

http://www.science.unitn.it/~fiorella/guidelinux/tlk/node39.html

http://en.wikipedia.org/wiki/Paging

http://en.wikipedia.org/wiki/Page_replacement_algorithm