PAGE REPLACEMENT ALGORITHM

Operating System

Linux Kernel 2.4.32

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

An optimal page replacement strategy exists.[Modern Op Sys] If there are n pages in the and each of them will be referenced after i instructions, then the page, that should be moved to swap area, is the one where the number of instructions i is the highest. This implies that at every point, during the execution of the program, the page that should be swapped out might be a different page. I.e. the decision is dependent on the current state the system is in. Imagine a program that just referenced a page P1. At this point a page needs to be swapped out. The page P1 won't be referenced for the next I instructions, before it is needed again. After those i instruction have been executed, the page P1 is needed, and so it is brought back from the swap area into the physical memory. Say now the system needs to swap out a page again. The page that won't be needed for the longest time is page P2 9 now. As long as P1 is not the page that will be unused for the longest time it will be kept in the main memory. This example attempts to illustrate why this algorithm is optimal. First-in First-out The first in first out page replacement algorithm keeps a list of all pages. When a page is loaded to the memory it is added to the head of the list. The pages to be swapped out are the ones at the tail of the list. This way the pages, that have been in the memory for the longest time are removed. The reasoning behind this algorithm is that the pages that have been in the physical memory for the longest time are the ones that are least likely to be used ever again, because the process that was using them has hopefully already finished. This idea might have worked well on systems where only ever one program is running, but with more programs running this scheme is not very efficient.

Least recently used

The Least Recently Used (LRU) algorithm is based on generally noted memory usage patterns of many programs. A page that is just used will probably be used again very soon, and a page that has not been used for a long time, will probably remain unused. LRU can be implemented by keeping a sorted list of all pages in memory. The list is sorted by time when the page was last used. This list is also called LRU stack. In practice this means that on every clock tick the position of the pages, used during that tick, must be updated.

LRU is relatively simple to implement and it has constant space and time overhead for given memory. LRU uses only page access recency as the base of the decision.
As discussed earlier, many common memory access patterns follow the principle of locality and LRU does work well with these. Typically, workloads with strong locality have the most page references with reuse distance small enough to allow these pages to fit in the main memory.

Not frequently used

Not Frequently Used (NFU) [16] is another approximation of LRU. In NFU, every page has an associated usage counter which is incremented on every clock tick the page is used. When a page needs to be evicted, the page with the lowest counter value is selected. The downside of this approach is that once some process uses some pages heavily, they tend to stay there for a while, even if they are not actively used anymore. This program model of doing computation in distinct phases is very common. Also programs, that have just been started, do not get much space in the main memory as the counters start from zero.

Page Replacement Algorithm Performance

A good page replacement algorithm has to have the following two properties:
It must swap out pages in a way to minimise the number of page faults and writes to the swap area
it must spend as little time as possible examining the pages in the system, before making a decision
So one page replacement algorithm is considered to be performing better than other page replacement algorithm if A process, that has been executed, on a system with better page replacement algorithm would spent less processor time in the kernel mode, than the same process executed on a system using not so good page replacement algorithm. Processor time spent in kernel mode is used to handle page faults and to choose which pages to swap out. A process running on a system with a better page replacement algorithm would produce less page faults, than the same process running on a system with a not so good page replacement algorithm. These are the criteria that will be used to evaluate the new page replacement algorithm that will be implemented in the Linux kernel.

Motivation for Guaranteeing a Process a Fixed Amount of Physical Memory

If process has all pages belonging to its working set in the physical memory, then it runs faster, because it does not have to wait for the operating system, to handle the page faults, it would otherwise produce. It will be possible to modify any of the page replacement algorithms, described in this chapter, not to remove pages, belonging to a certain process, once removing a page would cause the number of pages, and this process has in the physical memory, to drop below a certain minimum.

There are two problems with taking this approach:

1. By keeping pages of one process in the memory it is possible that for another process, with n pages in its working set, there will be less than n pages of physical memory available. The result of this will be that every time this process will get scheduled to run it will produce a page fault that the operating system will have to handle. This is going to slow down all the processes running on the system.

2. If the page replacement algorithm would just keep certain number of pages belonging to a process in the physical memory, then it is possible that those pages are no longer in the working set of that process. Why would this happen? Once the number of pages, this process has in the physical memory drops, to the guaranteed minimum, then swap out algorithm won't swap out any more pages belonging to the process, no matter whether they are likely to be used again or not. The working set of a process is likely to change, as the process allocates new memory, or uses addresses in different set of pages. Once this happens it is possible that the newly allocated pages will be the ones that will get swapped out. The solution to the first problem described in the paragraph above is simple: adjust the scheduling priority of a process that has the most memory guaranteed to be the highest. That way the processes that have little physical memory available are less likely to get scheduled and thus be executed and thus produce a page fault.
The solution to the second problem described above is not as straightforward. It is clear that making sure that the swap out algorithm scans all the pages belonging to a process before it scans the same page again. Therefore, once the process has more pages in physical memory than it has guaranteed (because it allocated more memory for itself or because they were brought back from swap area), a page that is least likely to be used again in the near future, can be swapped out (to approximate this one of the previously described algorithms are used).

Motivation for Limiting the Amount of Memory a Process May Use

It may not be easy to see at first why we should want to do anything like limiting the amount of memory a process may use. If we limit the amount of memory a process may use we can be sure that it won't use up memory that is needed for other, more important processes. Even if the system does not run out of virtual memory, if one process would use too much memory there may be not enough physical memory for other processes to run efficiently, without trashing.

IMPLEMENTATION:

Our aim was to work with the latest stable version of linux kernel: 3.19.2. However, due to a high change in replacement policies and approach, and also due to a lack of enough material, we had to shift to a simpler and earlier version of linux kernel: 2.4.23.

We understood the codes for page replacement in this kernel. The important files that needed to be understood are the following:

- Vmscan.c (it includes kswapd and balance_pgdat funcations)
- Swap.c(it includes function for adding and deleting from lru list and other important functions for page frame reclaimation)
- Filemap.c (it includes other support functions)

The understanding of the code (user manual to handle the code) is given below.

However, the kernel 2.4.23 could not be compiled on our systems due to:

- A different and obsolete method of compilation
- It had dependencies which are now obsolete. For example, the gedit used in linux version 2.6.xx onwards is different.

So we have implemented our printk analysis in kernel 3.19.2 using the basic understanding of the code of 2.4.23.

From Page Replacement to Page Frame reclamation

The page replacement policy in linux is now over ridden by a policy called page frame reclamation. Page frame reclamation suggests that page frames need to be reclaimed not at the point when the RAM is full. But instead of replacing pages, it is better to reclaim pages earlier to avert the “danger” of Page replacement.

In page replacement, when the RAM is full, the pages are swapped, i.e. the pages are copied from disk to memory and from memory to disk. This requires performing the heavy duty I/O operations. The I/O operations make the system very slow and most of the times the processor would be busy in performing I/O. Thus there arouse a need of removing pages when there is a signal from the RAM that the number of free pages are low.

In linux, it is carried out by 3 main unsigned long variables: nr_pages, nr_low, nr_high.

nr_low: it is a variable with a constant value which defines a low threshold of pages.

Nr_high: it is a variable with a constant value which defines the high threshold. It suggests that the no. of free pages in the RAM are enough and we do not need to reclaim pages.

Nr_pages: it is the variable that stores the number of free pages.

Thus, whenever nr_pages < nr_high, the page frame reclamation algorithm is called and the pages are reclaimed according to the need. A priority variable decides how urgent it is, to reclaim pages. It is dependent on how close nr_pages is to nr_low.

The full page reclamation algorithm starts with the kswapd() function which is a swap daemon, rather a thread which is asleep whenever there are enough number of pages. As soon as nr_pages < nr_high, kswapd is woken up and pages are reclaimed.

The full functioning is explained hereby.

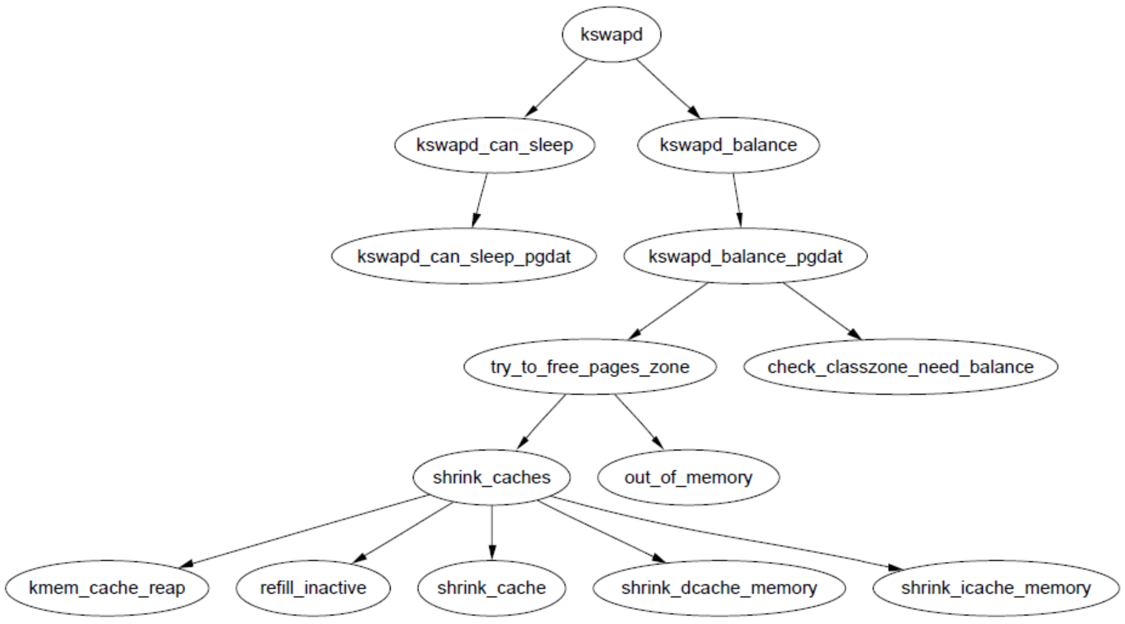
Linux Page Frame Reclamation Flow (version 2.4.23) – User Manual

Pageout Daemon- Kswapd

At system start, a kernel thread called kswapd is started from kswapd_init() which continuously executes the function kswapd() in mm/vmscan.c which usually sleeps. This daemon is responsible for reclaiming pages when memory is running low. Historically, kswapd used to wake up every 10 seconds but now it is only woken by the physical page allocator when the pages_low number of free pages in a zone is reached.

It is this daemon that performs most of the tasks needed to maintain the page cache correctly, shrink slab caches and swap out processes if necessary. Un-like swapout daemons such as Solaris [MM01] which is woken up with increas-ing frequency as there is memory pressure, kswapd keeps freeing pages until the pages_high watermark is reached. Under extreme memory pressure, pro-cesses will do the work of kswapd synchronously by calling balance_classzone() which calls try_to_free_pages_zone(). The physical page allocator will also call try_to_free_pages_zone() when the zone it is allocating from is under heavy pressure.
When kswapd is woken up, it performs the following:

- Calls kswapd_can_sleep() which cycles through all zones checking the need_balance field in the struct zone_t. If any of them are set, it cannot sleep;
- If it cannot sleep, it is removed from the kswapd_wait wait queue;
- kswapd_balance() is called which cycles through all zones. It will free pages in a zone with try_to_free_pages_zone() if need_balance is set and will keep freeing until the pages_high watermark is reached;
- The task queue for tq_disk is run so that pages queued will be written out;
- Add kswapd back to the kswapd_wait queue and go back to the first step.



Call Graph: `kswapd()`

Page cache

The page cache is a list of pages that are backed by regular files, block devices or swap. There are basically four types of pages that exist in the cache:

- Pages that were faulted in as a result of reading a memory mapped file;
- Blocks read from a block device or file system are packed into special pages called buffer pages. The number of blocks that may fit depends on the size of the block and the page size of the architecture;
- Anonymous pages first enter the page cache with no backing storage but are allocated slots in the backing storage when the kernel needs to swap them out;
- Pages belonging to shared memory regions which are treated in a similar fash-ion to anonymous pages. The only difference is that shared pages are added to the swap cache and space reserved in backing storage immediately after the first write to the page.

Pages exist in this cache for two reasons. The first is to eliminate unnecessary disk reads. Pages read from disk are stored in a page hash table hashed on the struct address_space and the offset. This table is always searched before the disk is accessed. The second reason is that the page cache forms the queue as a basis for the page replacement algorithm to select which page to discard or pageout.

The cache collectively consists of two lists defined in mm/page_alloc.c called active_list and inactive_list which broadly speaking store the “hot” and “cold” pages respectively. The lists are protected by the pagemap_lru_lock. An API is provided that is responsible for manipulating the page cache.

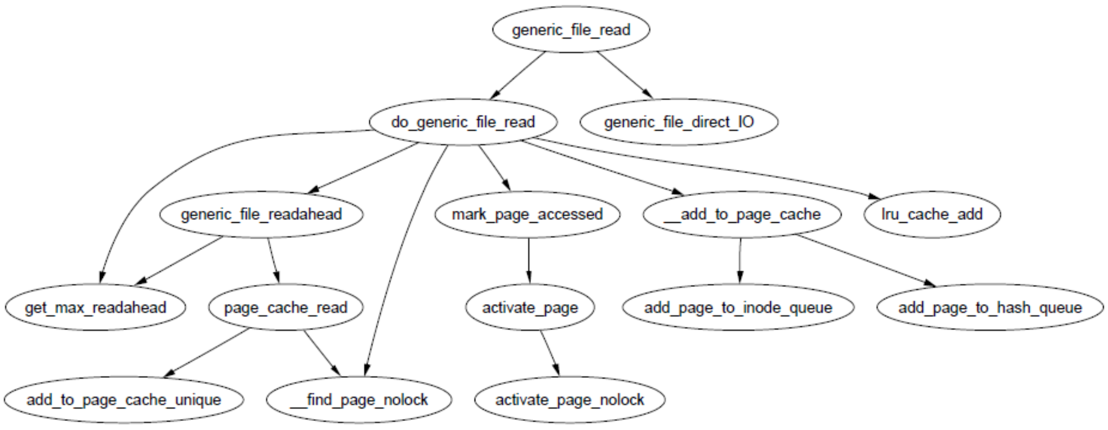
Manipulating the page cache

This section begins with how pages are added to the page cache. It will then cover how pages are moved from the active_list to the inactive_list. Lastly we will cover how pages are reclaimed from the page cache.

Adding page to page cache

Pages which are read from a file or block device are added to the page cache by calling __add_to_page_cache() during generic_file_read().

All filesystems use the high level function generic_file_read() so that oper-ations will take place through the page cache. It calls do_generic_file_read() which first checks if the page exists in the page cache. If it does not, the information is read from disk and added to the cache with __add_to_page_cache().



Call Graph: generic_file_read()

Anonymous pages are added to the page cache the first time they are about to be swapped out. The only real difference between anonymous pages and file backed pages as far as the page cache is concerned is that anonymous pages will use swapper_space as the struct address_space.

Shared memory pages are added during one of two cases. The first is during shmem_getpage_locked() which is called when a page has to be either fetched from swap or allocated as it is the first reference. The second is when the swapout code calls shmem_unuse(). This occurs when a swap area is being deactivated and a page, backed by swap space, is found that does not appear to belong to any process. The inodes related to shared memory are exhaustively searched until the correct page is found. In both cases, the page is added with add_to_page_cache().

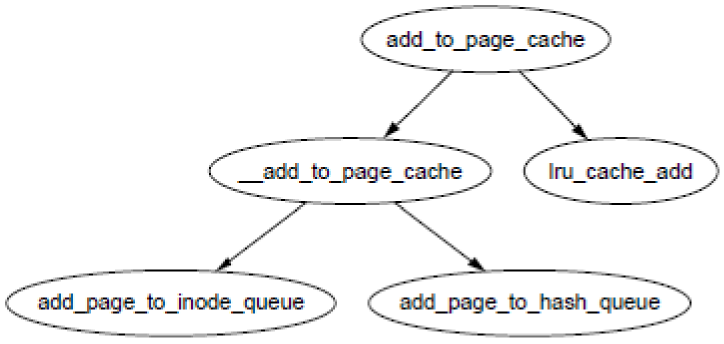
Refilling inactive_list

When caches are being shrunk, pages are moved from the active_list to the inactive_list by the function refill_inactive(). It takes as a parameter the number of pages to move, which is calculated in shrink_caches() as a ratio de-pending on nr_pages, the number of pages in active_list and the number of pages in inactive_list. The number of pages to move is calculated as

$$\text{pages} = \text{nr\_pages} * \frac{\text{nr\_active\_pages}}{2 * (\text{nr\_inactive\_pages} + 1)}$$

This keeps the active_list about two thirds the size of the inactive_list and the number of pages to move is determined as a ratio based on how many pages we desire to swap out (nr_pages).

Pages are taken from the end of the active_list. If the PG_referenced flag is set, it is cleared and the page is put back at top of the active_list as it has been recently used and is still “hot”. If the flag is cleared, it is moved to the inactive_list and the PG_referenced flag set so that it will be quickly promoted to the active_list if necessary.



Call Graph: `add_to_page_cache()`

Reclaiming pages from the page cache

The function shrink_cache() is the part of the replacement algorithm which takes pages from the inactive_list and decides how they should be swapped out. The two starting parameters which determine how much work will be performed are nr_pages and priority. nr_pages starts out as SWAP_CLUSTER_MAX and priority starts as DEF_PRIORITY.

Two parameters, max_scan and max_mapped determine how much work the function will do and are affected by the priority. Each time the function shrink_caches() is called without enough pages being freed, the priority will be decreased until the highest priority 1 is reached.

max_scan is the maximum number of pages will be scanned by this function and is simply calculated as

$$\text{max\_scan} = \frac{\text{nr\_inactive\_pages}}{\text{priority}}$$

where nr_inactive_pages is the number of pages in the inactive_list. This means that at lowest priority 6, at most one sixth of the pages in the inactive_list will be scanned and at highest priority, all of them will be.

The second parameter is max_mapped which determines how many process pages are allowed to exist in the page cache before whole processes will be swapped out. This is calculated as the minimum of either one tenth of max_scan or

$$\text{max\_mapped} = \text{nr\_pages} * 2^{(10-\text{priority})}$$

In other words, at lowest priority, the maximum number of mapped pages allowed is either one tenth of max_scan or 16 times the number of pages to swap out (nr_pages) whichever is the lower number. At high priority, it is either one tenth of max_scan or 512 times the number of pages to swap out.

From there, the function is basically a very large for-loop which scans at most max_scan pages to free up nr_pages pages from the end of the inactive_list or until the inactive_list is empty. After each page, it checks to see whether it should reschedule itself so that the swapper does not monopolise the CPU.

For each type of page found on the list, it makes a different decision on what to do. The page types and actions are as follows:

Page is mapped by a process. The max_mapped count is decremented. If it reaches 0, the page tables of processes will be linearly searched and swapped out by the function swap_out()

Page is locked and the PG_launder bit is set. A reference to the page is taken with page_cache_get() so that the page will not disappear and wait_on_page() is called which sleeps until the IO is complete. Once it is completed, the reference count is decremented with page_cache_release(). When the count reaches zero, it is freed.

Page is dirty, is unmapped by all processes, has no buffers and belongs to a device or file mapping. The PG_dirty bit is cleared and the PG_launder bit is set. A reference to the page is taken with page_cache_get() so the page will not disappear prematurely and then the writepage() function provided by the mapping is called to clean the page. The last case will pick up this page during the next pass and wait for the IO to complete if necessary.

Page has buffers associated with data on disk. A reference is taken to the page and an attempt is made to free the pages with try_to_release_page(). If it succeeds and is an anonymous page, the page can be freed. If it is backed by a file or device, the reference is simply dropped and the page will be freed later. However it is unclear how a page could have both associated buffers and a file mapping.

Page is anonymous belonging to a process and has no associated buffers. The LRU is unlocked and the page is unlocked. The max_mapped count is decremented. If it reaches zero, then swap_out() is called to start swapping out entire processes as there are too many process mapped pages in the page cache. An anonymous page may have associated buffers if it is backed by a swap file. In this case, the page is treated as a buffer page and normal block IO syncs the page with the backing storage.

Page has no references to it. If the page is in the swap cache, it is deleted from it as it is now stored in the swap area. If it is part of a file, it is removed from the inode queue. The page is then deleted from the page cache and freed.

Shrinking all caches

The function responsible for shrinking the various caches is shrink_caches() which takes a few simple steps to free up some memory. The maximum number of pages that will be written to disk in any given pass is nr_pages which is initialised by try_to_free_pages_zone() to be SWAP_CLUSTER_MAX[1]. The limitation is there so that if kswapd schedules a large number of pages to be swapped to disk, it will sleep occasionally to allow the IO to take place. As pages are freed, nr_pages is decremented to keep count.

The amount of work that will be performed also depends on the priority ini-tialised by try_to_free_pages_zone() to be DEF_PRIORITY[2]. For each pass that does not free up enough pages, the priority is decremented for the highest priority been 1.

The function first calls kmem_cache_reap() which selects a slab cache to shrink. If nr_pages number of pages are freed, the work is complete and the function returns otherwise it will try to free nr_pages from other caches.

If other caches are to be affected, refill_inactive() will move pages from the active_list to the inactive_list before shrinking the page cache by reclaiming pages at the end of the inactive_list with shrink_cache().

Finally, it shrinks three special caches, the dcache (shrink_dcache_memory()), the icache (shrink_icache_memory()) and the dqcache (shrink_dqcache_memory()). These objects are quite small in themselves but a cascading effect allows a lot more pages to be freed in the form of buffer and disk caches.

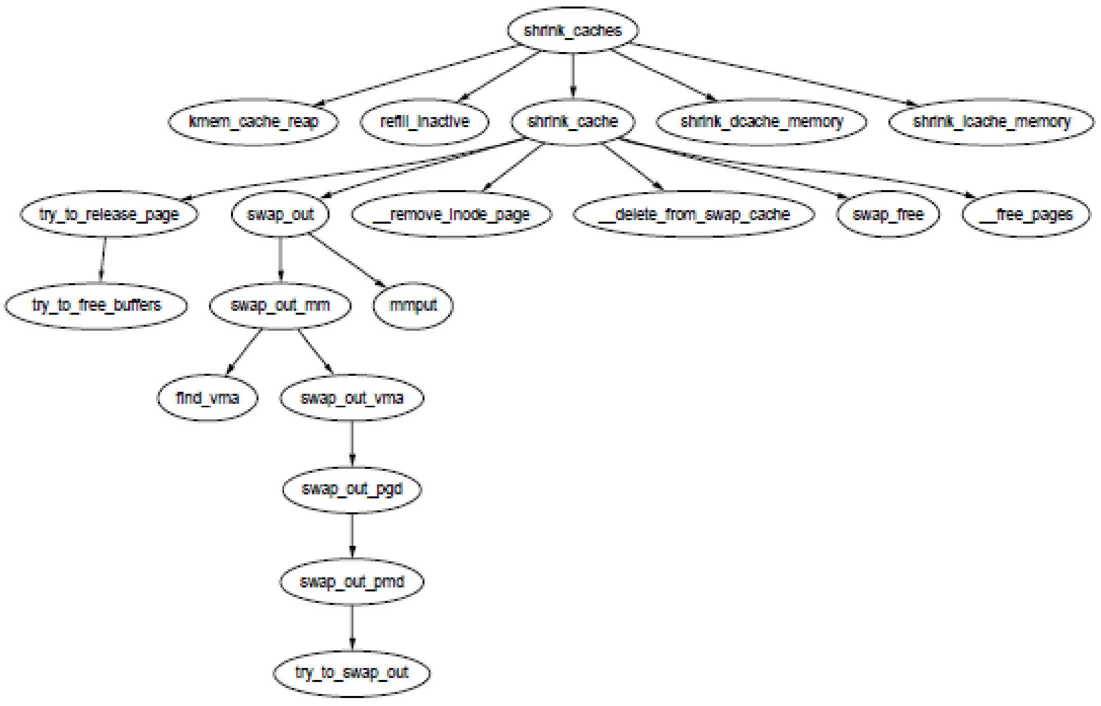
Swapping out process pages

When max_mapped pages have been found in the page cache, swap_out() is called to start swapping out process pages. Starting from the mm_struct pointed to by swap_mm and the address mm→swap_address, the page tables are searched forward until nr_pages have been freed.

All process mapped pages are examined regardless of where they are in the lists or when they were last referenced but pages which are part of the active_list or have been recently referenced will be skipped over. The examination of hot pages is a bit costly but insignificant in comparison to linearly searching all processes for the PTEs that reference a particular struct page.

Once it has been decided to swap out pages from a process, an attempt will be made to swap out at least SWAP_CLUSTER number of pages and the full list of mm_structs will only be examined once to avoid constant looping when no pages are available. Writing out the pages in bulk increases the chance that pages close together in the process address space will be written out to adjacent slots on disk.

swap_mm is initialised to point to init_mm and the swap_address is initialised to 0 the first time it is used. A task has been fully searched when the swap_address is equal to TASK_SIZE. Once a task has been selected to swap pages from, the ref-erence count to the mm_struct is incremented so that it will not be freed early and swap_out_mm() is called with the selected mm_struct as a parameter. This func-tion walks each VMA the process holds and calls swap_out_vma() for it. This is to avoid having to walk the entire page table which will be largely sparse. swap_out_pgd() and swap_out_pmd() walk the page tables for given VMA until finally try_to_swap_out() is called on the actual page and PTE.



Call Graph: shrink_caches()

try_to_swap_out() first checks to make sure the page is not part of the active_list, been recently referenced or part of a zone that we are not interested in. Once it has been established this is a page to be swapped out, it is removed from the page tables of the process and further work is performed. It is at this point the PTE is checked to see if it is dirty. If it is, the struct page flags will be updated to reflect that so that it will get laundered. Pages with buffers are not handled further as they cannot be swapped out to backing storage so the PTE for the process is simply established again and the page will be flushed later.

Page Replacement policy

During discussions the page replacement policy is frequently said to be a Least Recently Used (LRU)-based algorithm but this is not strictly speaking true as the lists are not strictly maintained in LRU order. The objective is for the active_list to contain the working set of all processes and the inactive_list. As all reclaimable pages are contained in just two lists and all pages may be selected to reclaim rather than just the faulting process, the replacement policy is a global one.

The lists resemble a simplified LRU 2Q where two lists called Am and A1 are maintained. With LRU 2Q, pages when first allocated are placed on a FIFO queue called A1. If they are referenced while on that queue, they are placed in a normal LRU managed list called Am. This is roughly analogous to using lru_cache_add() to place pages on a queue called inactive_list (A1) and using mark_page_accessed() to get moved to the active_list (Am). The algorithm describes how the size of the two lists have to be tuned but Linux takes a sim-pler approach by using refill_inactive() to move pages from the bottom of active_list to inactive_list to keep active_list about two thirds the size of the total page cache.

Functions for page replacement:

add_to_page_cache(struct page * page, struct address_space * mapping, unsigned long offset)
Adds a page to the page cache with lru_cache_add() in addition to adding it to the inode queue and page hash tables. Important for pages backed by files on disk.

remove_inode_page(struct page *page)
This function removes a page from both the inode queue with remove_page_from_inode_queue() and from the hash queues with remove_page_from_hash_queue(). This effectively removes the page from the page cache.

lru_cache_add(struct page * page)
Add a cold page to the inactive_list. Will be followed by mark_page_accessed() if known to be a hot page, such as when a page is faulted in.

lru_cache_del(struct page *page)
Removes a page from the page cache by calling either del_page_from_active_list() or del_page_from_inactive_list(), whichever is appropriate.

mark_page_accessed(struct page *page)
Mark that the page has been accessed. If it was not recently referenced (in the inactive_list and PG_referenced flag not set), the referenced flag is set. If it is referenced a second time, activate_page() is called, which marks the page hot, and the referenced flag is cleared

page_cache_get(struct page *page)
Increases the reference count to a page already in the page cache

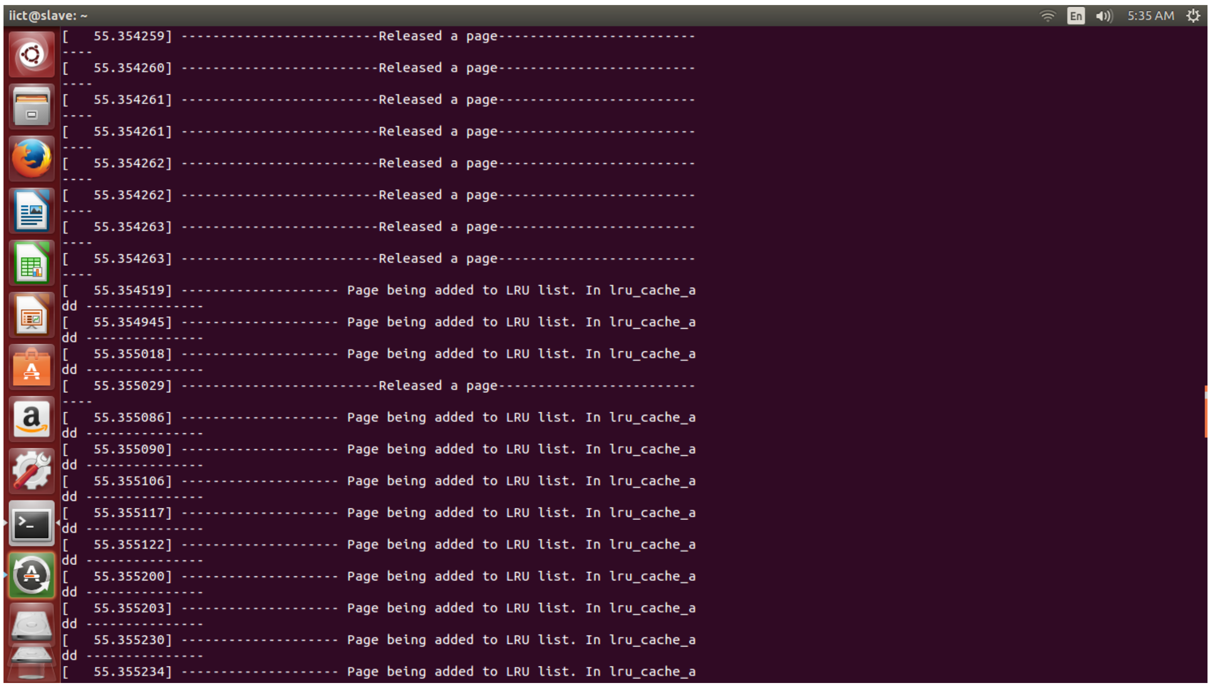
page_cache_release(struct page *page)
An alias for __free_page(). The reference count is decremented and if it drops to 0, the page will be freed

activate_page(struct page * page)
Removes a page from the inactive_list and places it on active_list. It is very rarely called directly as the caller has to know the page is on the inactive list. mark_page_accessed() should be used instead

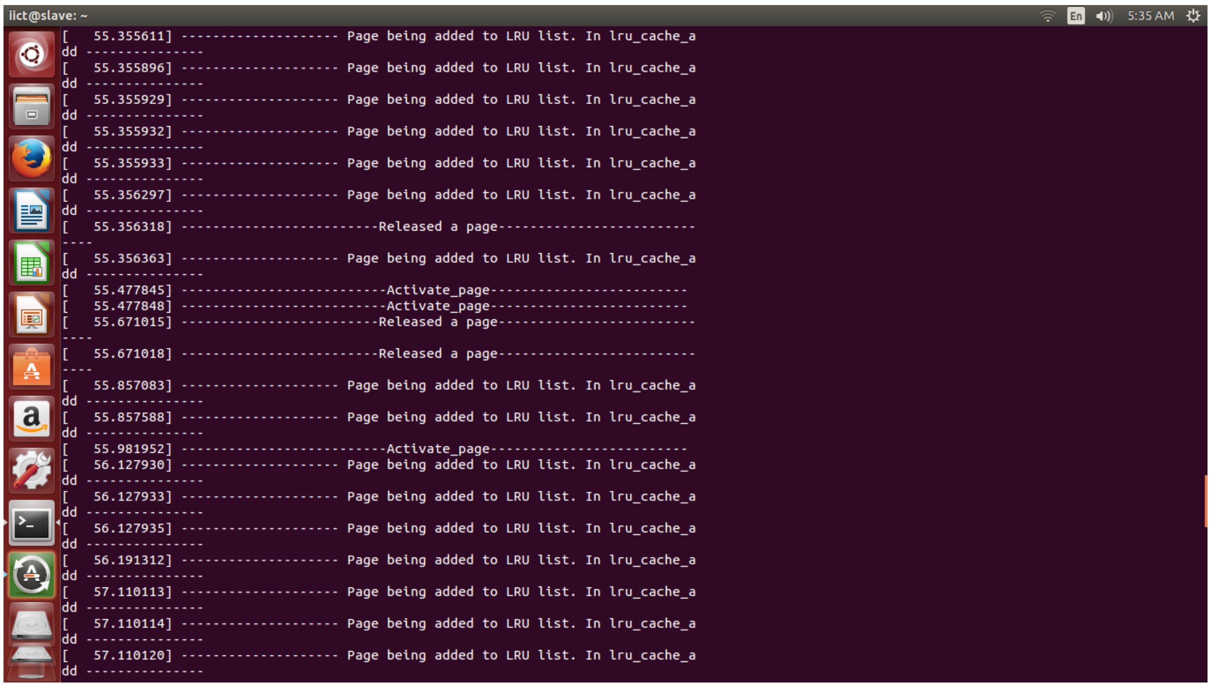
Printk analysis of code

After understanding the code, we tried and understand how exactly the code replaces pages and how often it makes a page active/inactive. Thus we applied printk statements in various functions of the code, compiled the codes and saw the output on the terminal via the command: dmesg.

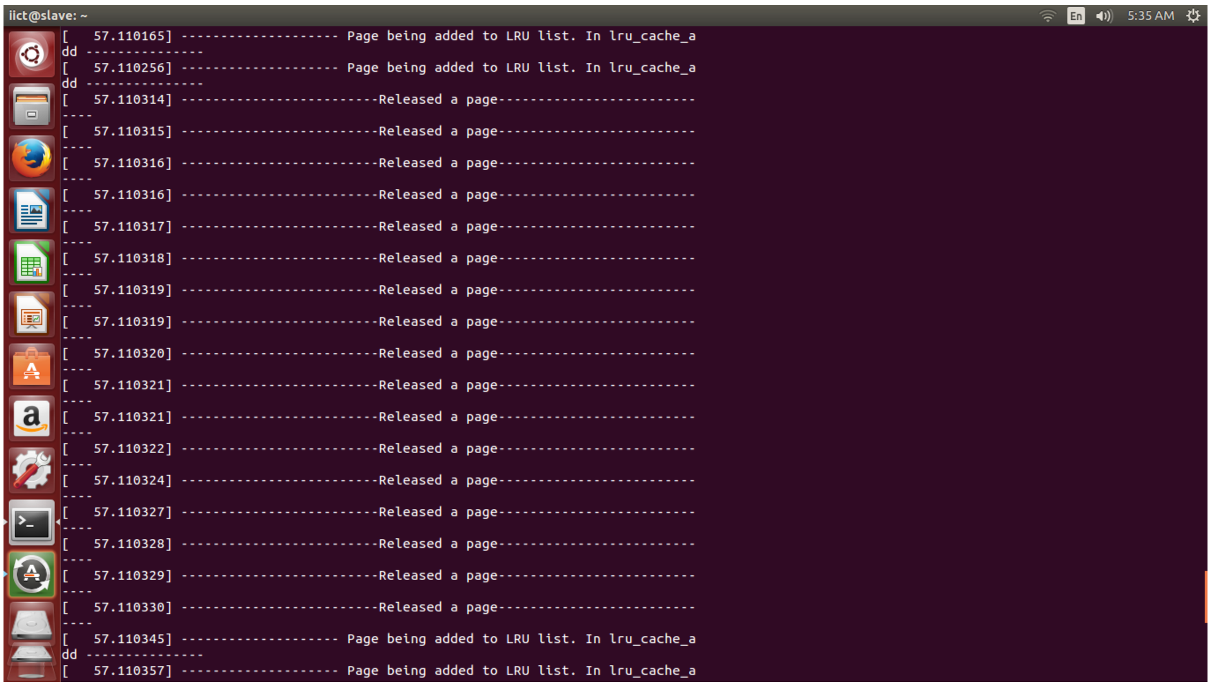
During normal functioning, i.e. when we did not explicitly open any cumbersome software, there was a normal freeing and adding of pages according to the usage of OS and other background processes which can be seen in the following images:



```
[   55.354259] -----------------------Released a page-----------------------
----
[   55.354260] -----------------------Released a page-----------------------
----
[   55.354261] -----------------------Released a page-----------------------
----
[   55.354261] -----------------------Released a page-----------------------
----
[   55.354262] -----------------------Released a page-----------------------
----
[   55.354262] -----------------------Released a page-----------------------
----
[   55.354263] -----------------------Released a page-----------------------
----
[   55.354263] -----------------------Released a page-----------------------
----
[   55.354519] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.354945] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355018] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355029] -----------------------Released a page-----------------------
----
[   55.355086] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355090] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355106] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355117] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355122] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355200] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355203] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355230] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355234] ------------------- Page being added to LRU list. In lru_cache_a
```



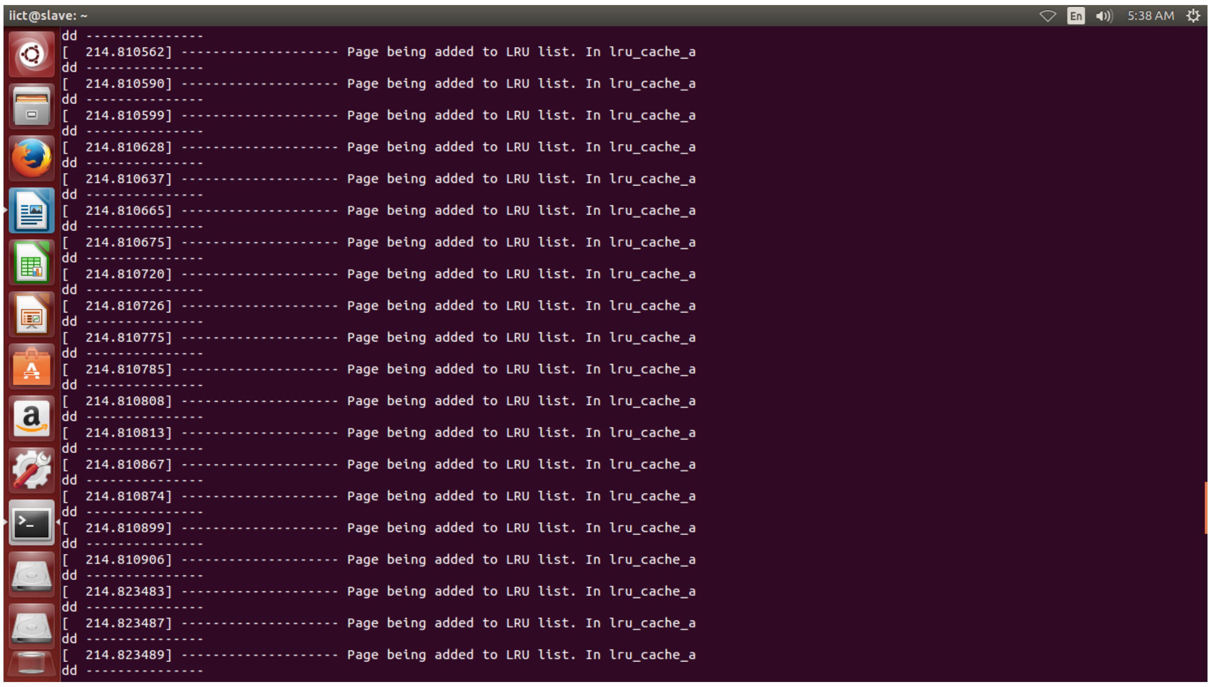
```
[   55.355611] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355896] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355929] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355932] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.355933] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.356297] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.356318] -----------------------Released a page-----------------------
----
[   55.356363] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.477845] ------------------------Activate_page-----------------------
[   55.477848] ------------------------Activate_page-----------------------
[   55.671015] -----------------------Released a page-----------------------
----
[   55.671018] -----------------------Released a page-----------------------
----
[   55.857083] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.857588] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   55.981952] ------------------------Activate_page-----------------------
[   56.127930] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   56.127933] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   56.127935] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   56.191312] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   57.110113] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   57.110114] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   57.110120] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
```



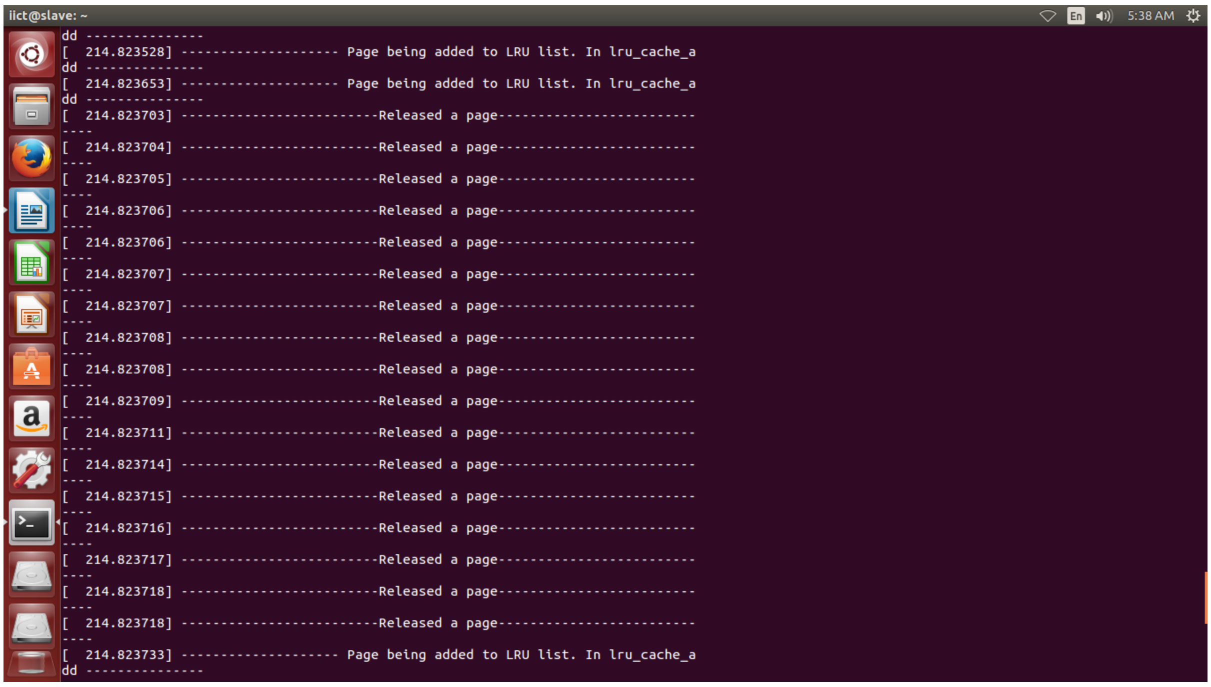
```
[   57.110165] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   57.110256] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   57.110314] ------------------------Released a page------------------------
----
[   57.110315] ------------------------Released a page------------------------
----
[   57.110316] ------------------------Released a page------------------------
----
[   57.110316] ------------------------Released a page------------------------
----
[   57.110317] ------------------------Released a page------------------------
----
[   57.110318] ------------------------Released a page------------------------
----
[   57.110319] ------------------------Released a page------------------------
----
[   57.110319] ------------------------Released a page------------------------
----
[   57.110320] ------------------------Released a page------------------------
----
[   57.110321] ------------------------Released a page------------------------
----
[   57.110321] ------------------------Released a page------------------------
----
[   57.110322] ------------------------Released a page------------------------
----
[   57.110324] ------------------------Released a page------------------------
----
[   57.110327] ------------------------Released a page------------------------
----
[   57.110328] ------------------------Released a page------------------------
----
[   57.110329] ------------------------Released a page------------------------
----
[   57.110330] ------------------------Released a page------------------------
----
[   57.110345] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[   57.110357] ------------------- Page being added to LRU list. In lru_cache_a
```

Now as we opened up certain software and utilities there was a remarkable change in the pattern of releasing of page and activation of page.

Libre Office Writer



```
dd --------------
[  214.810562] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810590] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810599] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810628] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810637] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810665] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810675] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810720] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810726] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810775] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810785] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810808] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810813] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810867] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810874] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810899] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.810906] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.823483] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.823487] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
[  214.823489] ------------------- Page being added to LRU list. In lru_cache_a
dd --------------
```

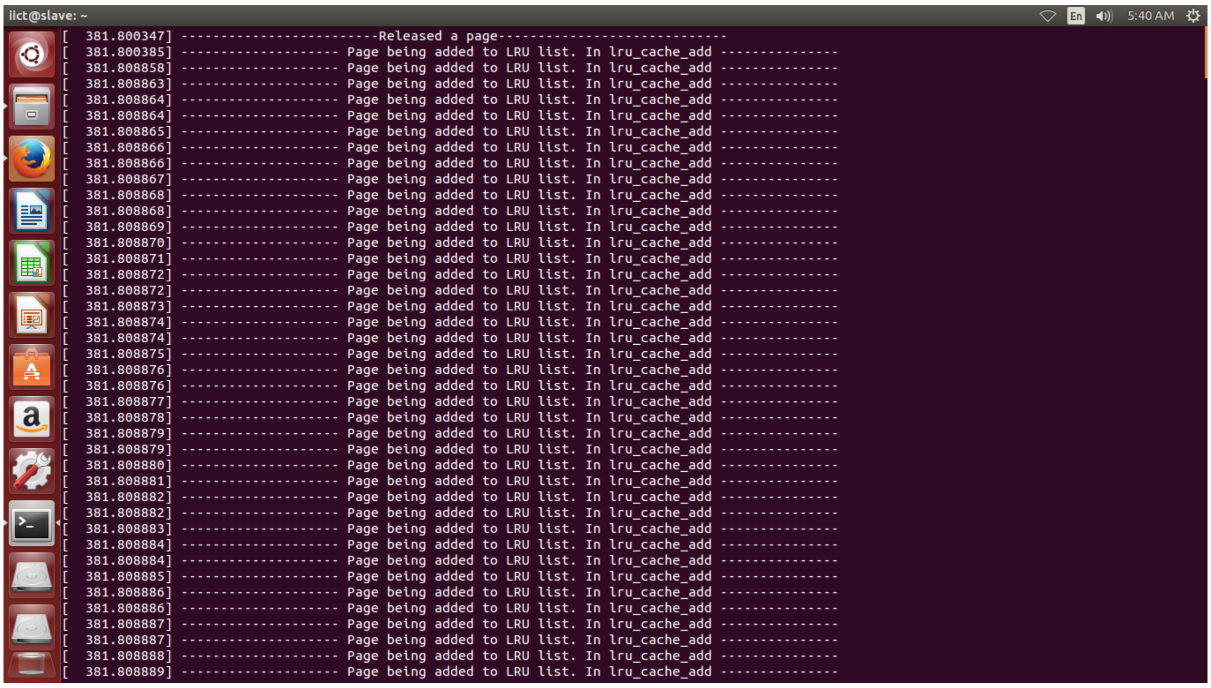


```
iict@slave: ~
dd ---------------
[  214.823528] ------------------- Page being added to LRU list. In lru_cache_a
dd ---------------
[  214.823653] ------------------- Page being added to LRU list. In lru_cache_a
dd ---------------
[  214.823703] ------------------------Released a page------------------------
----
[  214.823704] ------------------------Released a page------------------------
----
[  214.823705] ------------------------Released a page------------------------
----
[  214.823706] ------------------------Released a page------------------------
----
[  214.823706] ------------------------Released a page------------------------
----
[  214.823707] ------------------------Released a page------------------------
----
[  214.823707] ------------------------Released a page------------------------
----
[  214.823708] ------------------------Released a page------------------------
----
[  214.823708] ------------------------Released a page------------------------
----
[  214.823709] ------------------------Released a page------------------------
----
[  214.823711] ------------------------Released a page------------------------
----
[  214.823714] ------------------------Released a page------------------------
----
[  214.823715] ------------------------Released a page------------------------
----
[  214.823716] ------------------------Released a page------------------------
----
[  214.823717] ------------------------Released a page------------------------
----
[  214.823718] ------------------------Released a page------------------------
----
[  214.823718] ------------------------Released a page------------------------
----
[  214.823733] ------------------- Page being added to LRU list. In lru_cache_a
dd ---------------
```

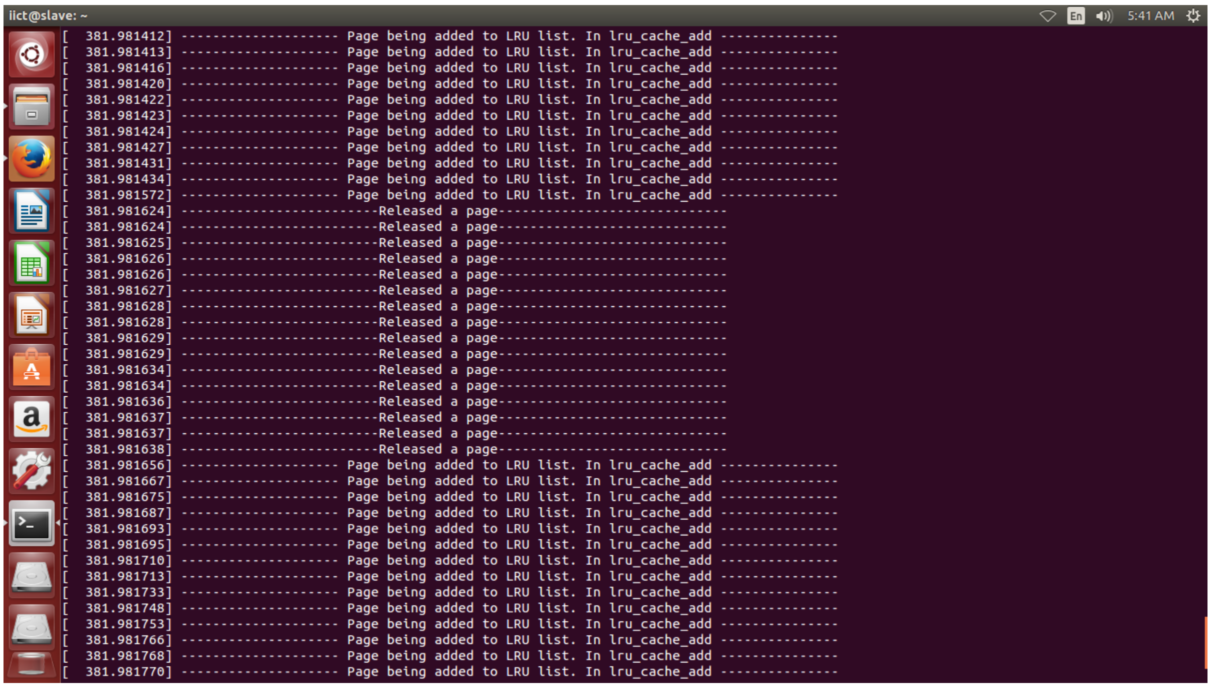
As soon as the libre office application was opened, there was a long array of the message: Page being added to LRU list. And then a long array of: released a page. This shows that libre office writer takes up a considerable amount of RAM frames.Thus relatively large number of pages become inactive when libre office is executed. And hence many pages are shifted to lru. Then as pages reach below nr_pages watermark, pages are started to be released till nr_high is achieved.

Mozilla Firefox

We see a similar pattern in Mozilla as in Libre Office here:



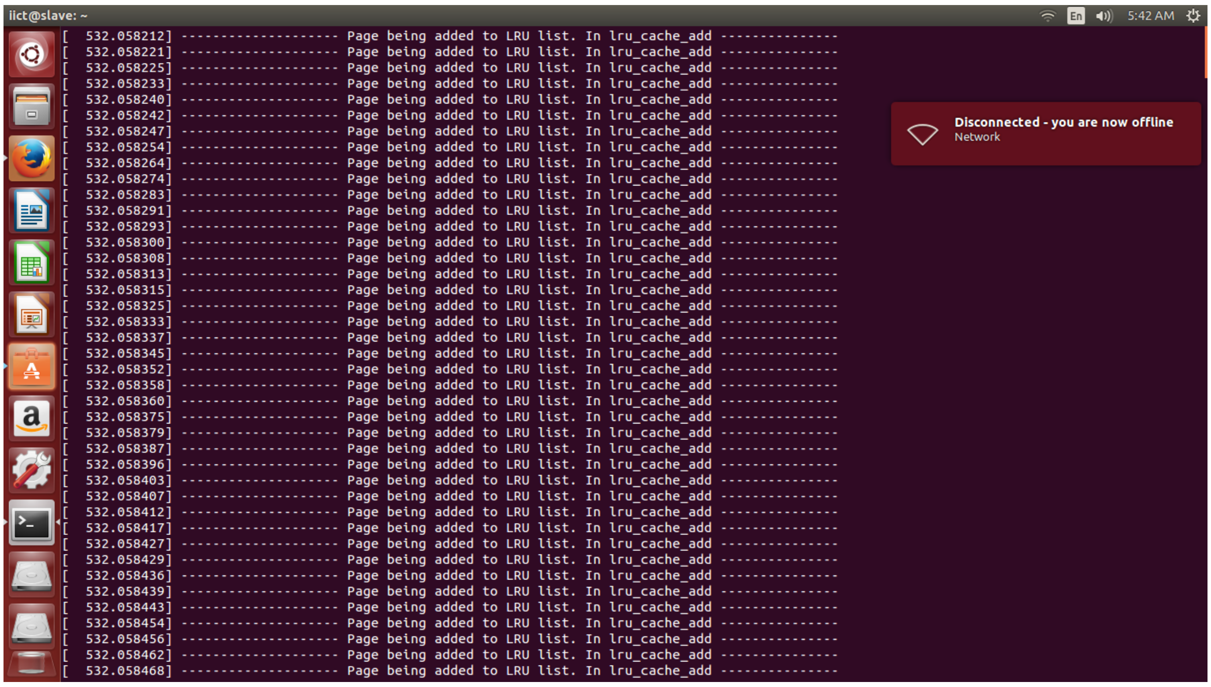
```
iict@slave: ~
[  381.800347] ------------------------Released a page--------------------------------
[  381.800385] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808858] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808863] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808864] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808864] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808865] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808866] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808866] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808867] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808868] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808868] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808869] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808870] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808871] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808872] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808872] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808873] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808874] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808874] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808875] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808876] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808876] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808877] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808878] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808879] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808879] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808880] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808881] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808882] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808882] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808883] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808884] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808884] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808885] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808886] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808886] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808887] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808887] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808888] ------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.808889] ------------------- Page being added to LRU list. In lru_cache_add ---------------
```



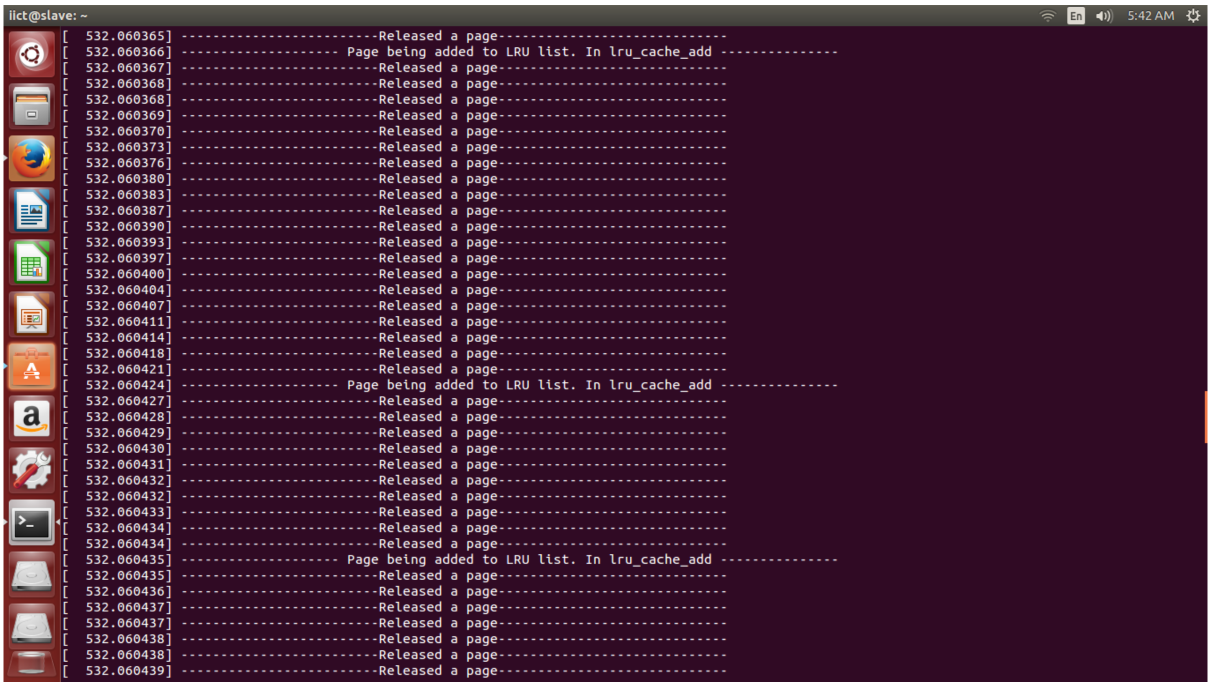
```
[  381.981412] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981413] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981416] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981420] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981422] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981423] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981424] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981427] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981431] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981434] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981572] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981624] -------------------------Released a page----------------------------
[  381.981624] -------------------------Released a page----------------------------
[  381.981625] -------------------------Released a page----------------------------
[  381.981626] -------------------------Released a page----------------------------
[  381.981626] -------------------------Released a page----------------------------
[  381.981627] -------------------------Released a page----------------------------
[  381.981628] -------------------------Released a page----------------------------
[  381.981628] -------------------------Released a page----------------------------
[  381.981629] -------------------------Released a page----------------------------
[  381.981629] -------------------------Released a page----------------------------
[  381.981634] -------------------------Released a page----------------------------
[  381.981634] -------------------------Released a page----------------------------
[  381.981636] -------------------------Released a page----------------------------
[  381.981637] -------------------------Released a page----------------------------
[  381.981637] -------------------------Released a page----------------------------
[  381.981638] -------------------------Released a page----------------------------
[  381.981656] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981667] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981675] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981687] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981693] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981695] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981710] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981713] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981733] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981748] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981753] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981766] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981768] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  381.981770] -------------------- Page being added to LRU list. In lru_cache_add ---------------
```

Ubuntu Software centre

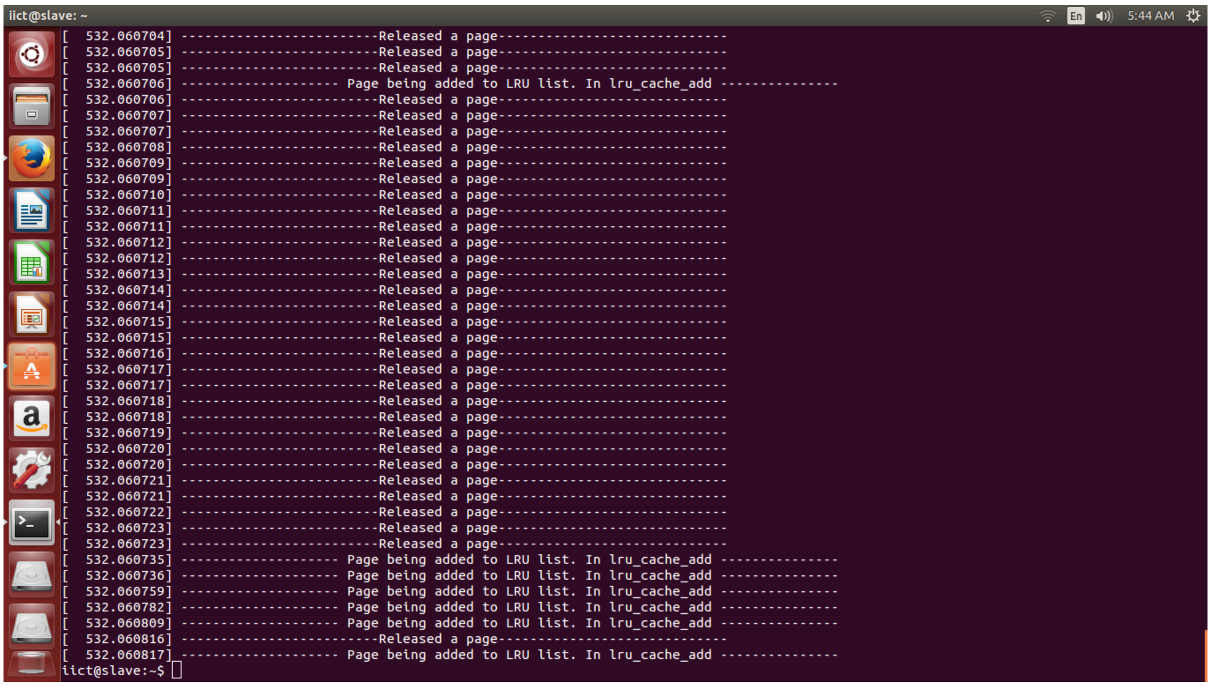
Here we again see a similar pattern but with increased intensity of releasing pages. It might be the case that the software centre requires a larger number of pages than Mozilla and libreoffice. Thus we observe a large number of lru transitions and releasing of pages.



```
[  532.058212] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058221] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058225] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058233] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058240] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058242] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058247] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058254] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058264] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058274] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058283] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058291] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058293] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058300] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058308] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058313] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058315] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058325] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058333] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058337] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058345] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058352] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058358] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058360] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058375] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058379] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058387] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058396] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058403] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058407] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058412] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058417] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058427] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058429] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058436] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058439] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058443] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058454] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058456] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058462] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.058468] -------------------- Page being added to LRU list. In lru_cache_add ---------------
```



```
[  532.060365] -------------------------Released a page-----------------------------
[  532.060366] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.060367] -------------------------Released a page-----------------------------
[  532.060368] -------------------------Released a page-----------------------------
[  532.060368] -------------------------Released a page-----------------------------
[  532.060369] -------------------------Released a page-----------------------------
[  532.060370] -------------------------Released a page-----------------------------
[  532.060373] -------------------------Released a page-----------------------------
[  532.060376] -------------------------Released a page-----------------------------
[  532.060380] -------------------------Released a page-----------------------------
[  532.060383] -------------------------Released a page-----------------------------
[  532.060387] -------------------------Released a page-----------------------------
[  532.060390] -------------------------Released a page-----------------------------
[  532.060393] -------------------------Released a page-----------------------------
[  532.060397] -------------------------Released a page-----------------------------
[  532.060400] -------------------------Released a page-----------------------------
[  532.060404] -------------------------Released a page-----------------------------
[  532.060407] -------------------------Released a page-----------------------------
[  532.060411] -------------------------Released a page-----------------------------
[  532.060414] -------------------------Released a page-----------------------------
[  532.060418] -------------------------Released a page-----------------------------
[  532.060421] -------------------------Released a page-----------------------------
[  532.060424] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.060427] -------------------------Released a page-----------------------------
[  532.060428] -------------------------Released a page-----------------------------
[  532.060429] -------------------------Released a page-----------------------------
[  532.060430] -------------------------Released a page-----------------------------
[  532.060431] -------------------------Released a page-----------------------------
[  532.060432] -------------------------Released a page-----------------------------
[  532.060432] -------------------------Released a page-----------------------------
[  532.060433] -------------------------Released a page-----------------------------
[  532.060434] -------------------------Released a page-----------------------------
[  532.060434] -------------------------Released a page-----------------------------
[  532.060435] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.060435] -------------------------Released a page-----------------------------
[  532.060436] -------------------------Released a page-----------------------------
[  532.060437] -------------------------Released a page-----------------------------
[  532.060437] -------------------------Released a page-----------------------------
[  532.060438] -------------------------Released a page-----------------------------
[  532.060438] -------------------------Released a page-----------------------------
[  532.060439] -------------------------Released a page-----------------------------
```



```
[  532.060704] -------------------------Released a page-----------------------------
[  532.060705] -------------------------Released a page-----------------------------
[  532.060705] -------------------------Released a page-----------------------------
[  532.060706] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.060706] -------------------------Released a page-----------------------------
[  532.060707] -------------------------Released a page-----------------------------
[  532.060707] -------------------------Released a page-----------------------------
[  532.060708] -------------------------Released a page-----------------------------
[  532.060709] -------------------------Released a page-----------------------------
[  532.060709] -------------------------Released a page-----------------------------
[  532.060710] -------------------------Released a page-----------------------------
[  532.060711] -------------------------Released a page-----------------------------
[  532.060711] -------------------------Released a page-----------------------------
[  532.060712] -------------------------Released a page-----------------------------
[  532.060712] -------------------------Released a page-----------------------------
[  532.060713] -------------------------Released a page-----------------------------
[  532.060714] -------------------------Released a page-----------------------------
[  532.060714] -------------------------Released a page-----------------------------
[  532.060715] -------------------------Released a page-----------------------------
[  532.060715] -------------------------Released a page-----------------------------
[  532.060716] -------------------------Released a page-----------------------------
[  532.060717] -------------------------Released a page-----------------------------
[  532.060717] -------------------------Released a page-----------------------------
[  532.060718] -------------------------Released a page-----------------------------
[  532.060718] -------------------------Released a page-----------------------------
[  532.060719] -------------------------Released a page-----------------------------
[  532.060720] -------------------------Released a page-----------------------------
[  532.060720] -------------------------Released a page-----------------------------
[  532.060721] -------------------------Released a page-----------------------------
[  532.060721] -------------------------Released a page-----------------------------
[  532.060722] -------------------------Released a page-----------------------------
[  532.060723] -------------------------Released a page-----------------------------
[  532.060723] -------------------------Released a page-----------------------------
[  532.060735] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.060736] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.060759] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.060782] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.060809] -------------------- Page being added to LRU list. In lru_cache_add ---------------
[  532.060816] -------------------------Released a page-----------------------------
[  532.060817] -------------------- Page being added to LRU list. In lru_cache_add ---------------
iict@slave:~$ 
```

Thus it can be seen that when heavy applications are opened there is a large activity of page swapping and releasing in RAM. It can be assumed that the largest amount of swapping could occur while processing or viewing a large video file, which occupies a considerable amount of RAM and utilizes large processor resources.

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

http://web.cs.wpi.edu/~jb/CS502/Project/proj2.html

http://www.comp.nus.edu.sg/~lubomir/PROOFS/proj4.pdf

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

http://www.csn.ul.ie/~mel/projects/

http://www.csn.ul.ie/~mel/projects/vm/guide/pdf/code.pdf

http://www.csn.ul.ie/~mel/projects/vm/guide/pdf/understand.pdf