

# [PATCH 6/9] clockpro-clockpro.patch

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-12/msg08480.html>

- *From:* Peter Zijlstra <a.p.zijlstra@xxxxxxxx>
- *Date:* Fri, 30 Dec 2005 23:43:34 +0100

From: Peter Zijlstra <a.p.zijlstra@xxxxxxxx>

The flesh of the clockpro implementation.

The paper: <http://www.cs.wm.edu/hpcs/WWW/HTML/publications/abs05-3.html> describes the algorithm approximated. It described a clock which has three hands, hand cold, hot and test. The table below describes the actions each hand performs.

res	hot	tst	ref	Hcold	Hhot	Htst	Flt
1	1	0	1	1101	1100	1101	
1	1	0	0	1100	1000	1100	
1	0	1	1	1100	1001	1001	
1	0	1	0	X0010	1000	1000	
1	0	0	1	1010	1001	1001	
1	0	0	0	X0000	1000	1000	
0	0	1	1		1100		
0	0	1	0	0010	X0000	X0000	
0	0	0	1		1010		

The approximation made is the removal of the nonresident pages from the one clock. The conceptual model is two clocks superimposed, one containing the resident and one containing the nonresident pages.

Implementation wise I use something based on Rik van Riel's nonresident code which actually approximates a clock with reduced order.

The resident clock with two hands is implemented using two lists which are to be seen as laid head to tail to form the clock. When one hand laps the other the lists are swapped.

Each page has 3 state bits:

hot -> PageHot()

[PATCH 6/9] clockpro-clockpro.patch

test -> PageTest()  
ref -> page\_referenced()

(PG\_active will be renamed to PG\_hot in a following patch, since the semantics changed also change the name in order to avoid confusion))

The HandCold rotation is driven by page reclaim needs. HandCold in turn drives HandHot, for every page HandCold promotes to hot HandHot needs to degrade one hot page to cold.

Changing the cold page target number also has influence on the HandHot rotation speed, when incremented the actual number of cold pages will be less than the desired number and hence we need to degrade some extra hot pages. When decreased, the actual number of cold pages is too large, so we would need to inhibit the degradation of hot pages.

The cold page target count is maintained in zone->nr\_cold\_target; it is incremented when a page is referenced in its test period and decremented when a page's test period expires.

The nonresident CLOCK is coupled to HandHot and is rotated so that when all resident zone CLOCKS have made one revolution, it too has made one whole revolution.

Signed-off-by: Peter Zijlstra <a.p.zijlstra@xxxxxxxxxx>

include/linux/mm\_page\_replace.h | 40 +-  
include/linux/mmzone.h | 13  
mm/Makefile | 2  
mm/clockpro.c | 557 +++  
mm/page\_alloc.c | 20 -  
mm/vmscan.c | 15 -  
6 files changed, 603 insertions(+), 44 deletions(-)

Index: linux-2.6-git/include/linux/mmzone.h

-----  
--- linux-2.6-git.orig/include/linux/mmzone.h  
+++ linux-2.6-git/include/linux/mmzone.h  
@@ -140,12 +140,13 @@ struct zone {  
ZONE\_PADDING(\_pad1\_)

/\* Fields commonly accessed by the page reclaim scanner \*/  
- spinlock\_t lru\_lock;  
- struct list\_head active\_list;  
- struct list\_head inactive\_list;  
- unsigned long nr\_scan\_active;  
- unsigned long nr\_active;  
- unsigned long nr\_inactive;  
+ spinlock\_t lru\_lock;  
+ struct list\_head list\_hand[2];  
+ unsigned long nr\_resident;

```

+ unsigned long nr_cold;
+ unsigned long nr_cold_target;
+ unsigned long nr_nonresident_scale;
+
unsigned long pages_scanned; /* since last reclaim */
int all_unreclaimable; /* All pages pinned */

```

Index: linux-2.6-git/mm/clockpro.c

=====

```

--- /dev/null
+++ linux-2.6-git/mm/clockpro.c
@@ -0,0 +1,557 @@
+/*
+ * mm/clockpro.c
+ *
+ * Written by Peter Zijlstra <a.p.zijlstra@xxxxxxxxxx>
+ * Released under the GPLv2, see the file COPYING for details.
+ *
+ * res | h/c | tst | ref || Hcold | Hhot | Htst || Flt
+ * ----+-----+-----+-----++-----+-----+-----++-----
+ * 1 | 1 | 0 | 1 ||=1101 | 1100 |=1101 ||
+ * 1 | 1 | 0 | 0 ||=1100 | 1000 |=1100 ||
+ * ----+-----+-----+-----++-----+-----+-----++-----
+ * 1 | 0 | 1 | 1 || 1100 | 1001 | 1001 ||
+ * 1 | 0 | 1 | 0 ||X0010 | 1000 | 1000 ||
+ * 1 | 0 | 0 | 1 || 1010 |=1001 |=1001 ||
+ * 1 | 0 | 0 | 0 ||X0000 |=1000 |=1000 ||
+ * ----+-----+-----+-----++-----+-----+-----++-----
+ * ----+-----+-----+-----++-----+-----+-----++-----
+ * 0 | 0 | 1 | 1 || | | || 1100
+ * 0 | 0 | 1 | 0 ||=0010 |X0000 |X0000 ||
+ * 0 | 0 | 0 | 1 || | | || 1010
+ *
+ * h/c -> PageHot()
+ * tst -> PageTest()
+ * ref -> page_referenced()
+ *
+ * The HandCold rotation is driven by page reclaim needs. HandCold in turn
+ * drives HandHot, for every page HandCold promotes to hot HandHot needs to
+ * degrade one hot page to cold.
+ *
+ * Changing the cold page target number also has influence on the HandHot
+ * rotation speed, when incremented the actual number of cold pages will be
+ * less than the desired number and hence we need to degrade some extra hot
+ * pages. When decreased, the actual number of cold pages is too large, so
+ * we would need to inhibit the degradation of hot pages.
+ *
+ * The cold page target count is maintained in zone->nr_cold_target; it is
+ * incremented when a page is referenced in its test period and decremented
+ * when a page's test period expires.
+ *

```

[PATCH 6/9] clockpro-clockpro.patch

```
+ * The nonresident CLOCK is coupled to HandHot and is rotated so that when
+ * all resident zone CLOCKS have made one revolution, it too has made one
+ * whole revolution (see __nonres_term()).
+ *
+ * All functions that are prefixed with '__' assume that zone->lru_lock is taken.
+ */
+
+#include <linux/mm_page_replace.h>
+#include <linux/rmap.h>
+#include <linux/buffer_head.h>
+#include <linux/pagevec.h>
+#include <linux/bootmem.h>
+#include <linux/init.h>
+
+#include <asm/div64.h>
+
+/*
+ * From 0 .. 100. Higher means more swappy.
+ */
+int vm_swappiness = 100;
+static long total_memory;
+
+static int __init page_replace_init(void)
+{
+ total_memory = nr_free_pagecache_pages();
+ return 0;
+}
+
+module_init(page_replace_init)
+
+/* Called to initialize the clockpro parameters */
+void __init page_replace_init_zone(struct zone *zone)
+{
+ INIT_LIST_HEAD(&zone->list_hand[0]);
+ INIT_LIST_HEAD(&zone->list_hand[1]);
+ zone->nr_resident = 0;
+ zone->nr_cold = 0;
+ zone->nr_cold_target = zone->pages_high;
+ zone->nr_nonresident_scale = 0;
+}
+
+/*
+ * Increase the cold pages target; limit it to the total number of resident
+ * pages present in the current zone.
+ *
+ * @zone: current zone
+ * @dct: intended increase
+ */
+static void __cold_target_inc(struct zone *zone, unsigned long dct)
+{
+ if (zone->nr_cold_target < zone->nr_resident - dct)
```

```

+ zone->nr_cold_target += dct;
+ else
+ zone->nr_cold_target = zone->nr_resident;
+ }
+
+ /*
+ * Decrease the cold pages target; limit it to the high watermark in order
+ * to always have some pages available for quick reclaim.
+ *
+ * @zone: current zone
+ * @dct: intended decrease
+ */
+static void __cold_target_dec(struct zone *zone, unsigned long dct)
+{
+ if (zone->nr_cold_target > zone->pages_high + dct)
+ zone->nr_cold_target -= dct;
+ else
+ zone->nr_cold_target = zone->pages_high;
+ }
+
+static void swap_lists(struct zone *zone)
+{
+ LIST_HEAD(tmp);
+ list_splice_init(&zone->list_hand[0], &tmp);
+ list_splice_init(&zone->list_hand[1], &zone->list_hand[0]);
+ list_splice(&tmp, &zone->list_hand[1]);
+ }
+
+static inline
+void __select_list_hand(struct zone *zone, struct list_head *list)
+{
+ if (list_empty(list))
+ swap_lists(zone);
+ }
+
+ /*
+ * Insert page into @zones clock and update adaptive parameters.
+ *
+ * Several page flags are used for insertion hints:
+ * PG_active – insert as an active page
+ * PG_test – use the use-once logic
+ *
+ * For now we will ignore the active hint; the use once logic is
+ * explained below.
+ *
+ * @zone: target zone.
+ * @page: new page.
+ */
+void __page_replace_insert(struct zone *zone, struct page *page)
+{
+ unsigned int rflags;

```

```

+
+ rflags = nonresident_get(page_mapping(page), page_index(page));
+
+ /* ignore the PG_active hint */
+ ClearPageActive(page);
+
+ /* abuse the PG_test flag for pagecache use-once */
+ if (!TestClearPageTest(page)) {
+ /*
+ * Insert (hot) when found in the nonresident list, otherwise
+ * insert as (cold,test). Insert at the head of the Hhot list,
+ * ie. right behind Hcold.
+ */
+ if (rflags & NR_found) {
+ SetPageActive(page);
+ __cold_target_inc(zone, 1);
+ } else {
+ SetPageTest(page);
+ ++zone->nr_cold;
+ }
+ ++zone->nr_resident;
+ __select_list_hand(zone, &zone->list_hand[hand_hot]);
+ list_add(&page->lru, &zone->list_hand[hand_hot]);
+ } else {
+ /*
+ * Pagecache insert; we want to avoid activation on the first
+ * reference (which we know will come); use-once logic.
+ *
+ * This is accomplished by inserting the page one state lower
+ * than usual so the activation that does come ups it to the
+ * normal insert state. Also we insert right behind Hhot so
+ * 1) Hhot cannot interfere; and 2) we lose the first reference
+ * quicker.
+ *
+ * Insert (cold,test)/(cold) so the following activation will
+ * elevate the state to (hot)/(cold,test). (NOTE: the activation
+ * will take care of the cold target increment).
+ */
+ BUG_ON(PageTest(page));
+
+ if (rflags & NR_found) {
+ SetPageTest(page);
+ }
+ ++zone->nr_cold;
+ ++zone->nr_resident;
+ __select_list_hand(zone, &zone->list_hand[hand_cold]);
+ list_add(&page->lru, &zone->list_hand[hand_cold]);
+ }
+
+ BUG_ON(!PageLRU(page));
+ }

```

```

+
+/*
+ * zone->lru_lock is heavily contended. Some of the functions that
+ * shrink the lists perform better by taking out a batch of pages
+ * and working on them outside the LRU lock.
+ *
+ * For pagecache intensive workloads, this function is the hottest
+ * spot in the kernel (apart from copy_*_user functions).
+ *
+ * @nr_to_scan: The number of pages to look through on the list.
+ * @src: The LRU list to pull pages off.
+ * @dst: The temp list to put pages on to.
+ * @scanned: The number of pages that were scanned.
+ *
+ * returns how many pages were moved onto *@dst.
+ */
+static int isolate_lru_pages(struct zone * zone, int nr_to_scan,
+ struct list_head *src, struct list_head *dst, int *scanned)
+{
+ int nr_taken = 0;
+ struct page *page;
+ int scan = 0;
+
+ spin_lock_irq(&zone->lru_lock);
+ __select_list_head(zone, src);
+ while (scan++ < nr_to_scan && !list_empty(src)) {
+ page = lru_to_page(src);
+ prefetchw_prev_lru_page(page, src, flags);
+
+ if (!TestClearPageLRU(page))
+ BUG();
+ list_del(&page->lru);
+ if (get_page_testone(page)) {
+ /*
+ * It is being freed elsewhere
+ */
+ __put_page(page);
+ SetPageLRU(page);
+ list_add(&page->lru, src);
+ continue;
+ } else {
+ list_add(&page->lru, dst);
+ nr_taken++;
+ if (!PageActive(page))
+ --zone->nr_cold;
+ }
+ }
+ zone->nr_resident -= nr_taken;
+ zone->pages_scanned += scan;
+ spin_unlock_irq(&zone->lru_lock);
+

```

[PATCH 6/9] clockpro-clockpro.patch

```
+ *scanned = scan;
+ return nr_taken;
+ }
+
+ /*
+ * Add page to a release pagevec, temp. drop zone lock to release pagevec if full.
+ * Set PG_lru, update zone->nr_cold and zone->nr_resident.
+ *
+ * @zone: @pages zone.
+ * @page: page to be released.
+ * @pvec: pagevec to collect pages in.
+ */
+static void __page_release(struct zone *zone, struct page *page,
+ struct pagevec *pvec)
+{
+ if (TestSetPageLRU(page))
+ BUG();
+ if (!PageActive(page))
+ ++zone->nr_cold;
+ ++zone->nr_resident;
+
+ if (!pagevec_add(pvec, page)) {
+ spin_unlock_irq(&zone->lru_lock);
+ if (buffer_heads_over_limit)
+ pagevec_strip(pvec);
+ __pagevec_release(pvec);
+ spin_lock_irq(&zone->lru_lock);
+ }
+ }
+
+ /*
+ * Try to reclaim a specified number of pages.
+ *
+ * Reclaim candidates have:
+ * - PG_lru cleared
+ * - 1 extra ref
+ *
+ * NOTE: hot pages are also returned but will be spit back by try_pageout()
+ * this to preserve CLOCK order.
+ *
+ * @zone: target zone to reclaim pages from.
+ * @nr_to_scan: nr of pages to try for reclaim.
+ *
+ * returns candidate list.
+ */
+void page_replace_candidates(struct zone *zone, int nr_to_scan, struct list_head *page_list)
+{
+ int nr_scan;
+
+ isolate_lru_pages(zone, nr_to_scan,
+ &zone->list_hand[hand_cold],
```

```

+ page_list, &nr_scan);
+
+ if (current_is_kswapd())
+ mod_page_state_zone(zone, pgscan_kswapd, nr_scan);
+ else
+ mod_page_state_zone(zone, pgscan_direct, nr_scan);
+ }
+
+ /*
+ * Activate a cold page:
+ * cold, !test -> cold, test
+ * cold, test -> hot
+ *
+ * @page: page to activate
+ */
+void page_replace_activate(struct page *page)
+{
+ int hot, test;
+
+ hot = PageActive(page);
+ test = PageTest(page);
+
+ if (hot) {
+ BUG_ON(test);
+ } else {
+ if (test) {
+ SetPageActive(page);
+ /*
+ * Leave PG_test set for new hot pages in order to
+ * recognise then in reinsert() and do accounting.
+ */
+ } else {
+ SetPageTest(page);
+ }
+ }
+ }
+
+static int reclaim_mapped(struct zone *);
+static void rotate_hot(struct zone *, int, int, struct pagevec *);
+
+ /*
+ * Reinsert those candidate pages that were not freed by try_pageout().
+ * Account pages that were promoted to hot by page_replace_activate().
+ * Rotate hand hot to balance the new hot and lost cold pages vs.
+ * the cold pages target.
+ *
+ * Candidate pages have:
+ * - PG_lru cleared
+ * - 1 extra ref
+ * undo that.
+ */

```

[PATCH 6/9] clockpro-clockpro.patch

```
+ * @zone: zone we're working on.
+ * @page_list: the left over pages.
+ */
+void page_replace_reinsert(struct zone *zone, struct list_head *page_list)
+{
+ struct pagevec pvec;
+ unsigned long dct = 0;
+
+ pagevec_init(&pvec, 1);
+ spin_lock_irq(&zone->lru_lock);
+ __select_list_hand(zone, &zone->list_hand[hand_hot]);
+ while (!list_empty(page_list)) {
+ struct page *page = lru_to_page(page_list);
+ prefetchw_prev_lru_page(page, page_list, flags);
+
+ if (PageActive(page) && PageTest(page)) {
+ ClearPageTest(page);
+ ++dct;
+ }
+
+ list_move(&page->lru, &zone->list_hand[hand_hot]);
+ __page_release(zone, page, &pvec);
+ }
+ __cold_target_inc(zone, dct);
+ spin_unlock_irq(&zone->lru_lock);
+
+ /*
+ * Limit the hot hand to a full revolution.
+ */
+ if (zone->nr_cold < zone->nr_cold_target) {
+ int i, nr = zone->nr_resident / SWAP_CLUSTER_MAX;
+ int rm = reclaim_mapped(zone);
+ for (i = 0; zone->nr_cold < zone->nr_cold_target && i < nr; ++i)
+ rotate_hot(zone, SWAP_CLUSTER_MAX, rm, &pvec);
+ }
+
+ pagevec_release(&pvec);
+ }
+
+ /*
+ * Puts cold pages that have their test bit set on the non-resident lists.
+ *
+ * @zone: dead pages zone.
+ * @page: dead page.
+ */
+void page_replace_remember(struct zone *zone, struct page *page)
+{
+ if (TestClearPageTest(page)) {
+ int list = nonresident_put(page_mapping(page),
+ page_index(page), NR_b1, NR_b1);
+ if (list != NR_free)
```

[PATCH 6/9] clockpro-clockpro.patch

```
+ __cold_target_dec(zone, 1);
+ }
+}
+
+static unsigned long estimate_pageable_memory(void)
+{
+#if 0
+ static unsigned long next_check;
+ static unsigned long total = 0;
+
+ if (!total || time_after(jiffies, next_check)) {
+ struct zone *z;
+ total = 0;
+ for_each_zone(z)
+ total += z->nr_resident;
+ next_check = jiffies + HZ/10;
+ }
+
+ // gave 0 first time, SIGFPE in kernel sucks
+ // hence the !total
+#else
+ unsigned long total = 0;
+ struct zone *z;
+ for_each_zone(z)
+ total += z->nr_resident;
+#endif
+ return total;
+}
+
+static int reclaim_mapped(struct zone *zone)
+{
+ long mapped_ratio;
+ long distress;
+ long swap_tendency;
+
+ /*
+ * `distress' is a measure of how much trouble we're having reclaiming
+ * pages. 0 -> no problems. 100 -> great trouble.
+ */
+ distress = 100 >> zone->prev_priority;
+
+ /*
+ * The point of this algorithm is to decide when to start reclaiming
+ * mapped memory instead of just pagecache. Work out how much memory
+ * is mapped.
+ */
+ mapped_ratio = (read_page_state(nr_mapped) * 100) / total_memory;
+
+ /*
+ * Now decide how much we really want to unmap some pages. The mapped
+ * ratio is downgraded - just because there's a lot of mapped memory
```

[PATCH 6/9] clockpro-clockpro.patch

```
+ * doesn't necessarily mean that page reclaim isn't succeeding.
+ *
+ * The distress ratio is important – we don't want to start going oom.
+ *
+ * A 100% value of vm_swappiness overrides this algorithm altogether.
+ */
+ swap_tendency = mapped_ratio / 2 + distress + vm_swappiness;
+
+ /*
+ * Now use this metric to decide whether to start moving mapped memory
+ * onto the inactive list.
+ */
+ if (swap_tendency >= 100)
+ return 1;
+
+ return 0;
+ }
+
+ /*
+ * Rotate the non-resident hand; scale the rotation speed so that when all
+ * hot hands have made one full revolution the non-resident hand will have
+ * too.
+ *
+ * @zone: current zone
+ * @dh: number of pages the hot hand has moved
+ */
+static void __nonres_term(struct zone *zone, unsigned long dh)
+{
+ unsigned long long cycles;
+ /*
+ * |b1| Rhot |B| Rhot
+ * Rtest = ----- ~ -----
+ * |r1| |R|
+ *
+ * NOTE depends on |B|, hence include the nonresident_del patch
+ */
+ cycles = zone->nr_nonresident_scale + (unsigned long long)dh * nonresident_estimate();
+ zone->nr_nonresident_scale = do_div(cycles, estimate_pageable_memory() + 1UL);
+ __get_cpu_var(nonres_cycle) += (u32)cycles;
+ __cold_target_dec(zone, cycles);
+ }
+
+ /*
+ * Rotate hand hot;
+ *
+ * @zone: current zone
+ * @nr_to_scan: batch quanta
+ * @reclaim_mapped: whether to demote mapped pages too
+ * @pvec: release pagevec
+ */
+static void rotate_hot(struct zone *zone, int nr_to_scan, int reclaim_mapped,
```

```

+ struct pagevec *pvec)
+{
+ LIST_HEAD(l_hold);
+ LIST_HEAD(l_tmp);
+ unsigned long dh = 0, dct = 0;
+ int pgscanned;
+ int pgdeactivate = 0;
+ int nr_taken;
+
+ nr_taken = isolate_lru_pages(zone, nr_to_scan,
+ &zone->list_hand[hand_hot],
+ &l_hold, &pgscanned);
+
+ mod_page_state_zone(zone, pgrefill, pgscanned);
+
+ while (!list_empty(&l_hold)) {
+ struct page *page = lru_to_page(&l_hold);
+ prefetchw_prev_lru_page(page, &l_hold, flags);
+
+ if (PageActive(page)) {
+ BUG_ON(PageTest(page));
+
+ /*
+ * Ignore the swap token; this is not actual reclaim
+ * and it will give a better reflection of the actual
+ * hotness of pages.
+ *
+ * XXX do something with this reclaim_mapped stuff.
+ */
+ if (!(reclaim_mapped && mapped) || !mapped) ||
+ (total_swap_pages == 0 && PageAnon(page))) && */
+ !page_referenced(page, 0, 1)) {
+ ClearPageActive(page);
+ ++pgdeactivate;
+ }
+
+ ++dh;
+ } else {
+ if (TestClearPageTest(page))
+ ++dct;
+ }
+ list_move(&page->lru, &l_tmp);
+
+ cond_resched();
+
+ spin_lock_irq(&zone->lru_lock);
+ while (!list_empty(&l_tmp)) {
+ struct page *page = lru_to_page(&l_tmp);
+ prefetchw_prev_lru_page(page, &l_tmp, flags);
+ list_move(&page->lru, &zone->list_hand[hand_cold]);

```

[PATCH 6/9] clockpro-clockpro.patch

```
+ __page_release(zone, page, pvec);
+ }
+ __nonres_term(zone, nr_taken);
+ __cold_target_dec(zone, dct);
+ spin_unlock_irq(&zone->lru_lock);
+
+ mod_page_state(pgdeactivate, pgdeactivate);
+}
Index: linux-2.6-git/include/linux/mm_page_replace.h
=====
--- linux-2.6-git.orig/include/linux/mm_page_replace.h
+++ linux-2.6-git/include/linux/mm_page_replace.h
@@ -7,6 +7,7 @@
#include <linux/mm.h>
#include <linux/list.h>
#include <linux/page-flags.h>
+#include <linux/swap.h>

#define lru_to_page(_head) (list_entry((_head)->prev, struct page, lru))

@@ -38,44 +39,41 @@
#define prefetchw_prev_lru_page(_page, _base, _field) do { } while (0)
#endif

+enum {
+ hand_hot = 0,
+ hand_cold = 1
+};
+
+void __init page_replace_init_zone(struct zone *);
+void __page_replace_insert(struct zone *, struct page *);
+void page_replace_candidates(struct zone *, int, struct list_head *);
-
-static inline
-void page_replace_activate(struct page *page)
-{
- SetPageActive(page);
-}
-
+void page_replace_activate(struct page *);
+void page_replace_reinsert(struct zone *, struct list_head *);
+void page_replace_remember(struct zone *, struct page *);

+
+/*
+ * Make page available for direct reclaim.
+ *
+ * @zone: page's zone.
+ * @page: page.
+ */
static inline
```

[PATCH 6/9] clockpro-clockpro.patch

```
void __page_replace_rotate_reclaimable(struct zone *zone, struct page *page)
{
if (PageLRU(page) && !PageActive(page)) {
- list_move_tail(&page->lru, &zone->inactive_list);
+ list_move_tail(&page->lru, &zone->list_hand[hand_cold]);
inc_page_state(pgrotated);
}
}
```

```
static inline void
-add_page_to_active_list(struct zone *zone, struct page *page)
-{
- list_add(&page->lru, &zone->active_list);
- zone->nr_active++;
-}
-
```

```
-static inline void
del_page_from_lru(struct zone *zone, struct page *page)
{
list_del(&page->lru);
- if (PageActive(page)) {
- ClearPageActive(page);
- zone->nr_active--;
- } else {
- zone->nr_inactive--;
- }
+ --zone->nr_resident;
+ if (!TestClearPageActive(page))
+ --zone->nr_cold;
}
```

```
#endif /* __KERNEL__ */
```

```
Index: linux-2.6-git/mm/vmscan.c
```

```
=====
```

```
--- linux-2.6-git.orig/mm/vmscan.c
+++ linux-2.6-git/mm/vmscan.c
@@ -339,10 +339,11 @@ static try_pageout_t try_pageout(struct
int may_enter_fs;
int referenced;

- if (TestSetPageLocked(page))
+ if (PageActive(page))
goto keep;

- BUG_ON(PageActive(page));
+ if (TestSetPageLocked(page))
+ goto keep;

sc->nr_scanned++;
/* Double the slab pressure for mapped and swapcache pages */
@@ -467,6 +468,7 @@ static try_pageout_t try_pageout(struct
```

```

#ifdef CONFIG_SWAP
if (PageSwapCache(page)) {
swp_entry_t swap = { .val = page_private(page) };
+ page_replace_remember(page_zone(page), page);
__delete_from_swap_cache(page);
write_unlock_irq(&mapping->tree_lock);
swap_free(swap);
@@ -475,6 +477,7 @@ static try_pageout_t try_pageout(struct
}
#endif /* CONFIG_SWAP */

+ page_replace_remember(page_zone(page), page);
__remove_from_page_cache(page);
write_unlock_irq(&mapping->tree_lock);
__put_page(page);
@@ -572,7 +575,7 @@ shrink_zone(struct zone *zone, struct sc

atomic_inc(&zone->reclaim_in_progress);

- nr_inactive = (zone->nr_inactive >> sc->priority) + SWAP_CLUSTER_MAX;
+ nr_inactive = (zone->nr_resident >> sc->priority) + SWAP_CLUSTER_MAX;
nr_inactive &= ~(SWAP_CLUSTER_MAX - 1);

sc->nr_to_scan = SWAP_CLUSTER_MAX;
@@ -667,7 +670,7 @@ int try_to_free_pages(struct zone **zone
continue;

zone->temp_priority = DEF_PRIORITY;
- lru_pages += zone->nr_active + zone->nr_inactive;
+ lru_pages += zone->nr_resident;
}

for (priority = DEF_PRIORITY; priority >= 0; priority--) {
@@ -811,14 +814,14 @@ loop_again:
zone->temp_priority = priority;
if (zone->prev_priority > priority)
zone->prev_priority = priority;
- lru_pages += zone->nr_active + zone->nr_inactive;
+ lru_pages += zone->nr_resident;

atomic_inc(&zone->reclaim_in_progress);
shrink_zone(zone, &sc);
atomic_dec(&zone->reclaim_in_progress);

if (zone->pages_scanned >=
- (zone->nr_active + zone->nr_inactive) * 4)
+ (zone->nr_resident) * 4)
zone->all_unreclaimable = 1;
}
reclaim_state->reclaimed_slab = 0;
Index: linux-2.6-git/mm/Makefile

```

```

-----
--- linux-2.6-git.orig/mm/Makefile
+++ linux-2.6-git/mm/Makefile
@@ -10,7 +10,7 @@ mmu-$(CONFIG_MMU) := fremap.o highmem.o
obj-y := bootmem.o filemap.o mempool.o oom_kill.o fadvise.o \
page_alloc.o page-writeback.o pdflush.o \
readahead.o slab.o swap.o truncate.o vmscan.o \
- prio_tree.o page_replace.o $(mmu-y)
+ prio_tree.o clockpro.o $(mmu-y)

```

```

obj-$(CONFIG_SWAP) += page_io.o swap_state.o swapfile.o thrash.o \
nonresident.o
Index: linux-2.6-git/mm/page_alloc.c

```

```

=====
--- linux-2.6-git.orig/mm/page_alloc.c
+++ linux-2.6-git/mm/page_alloc.c
@@ -1263,8 +1263,8 @@ void __get_zone_counts(unsigned long *ac
*inactive = 0;
*free = 0;
for (i = 0; i < MAX_NR_ZONES; i++) {
- *active += zones[i].nr_active;
- *inactive += zones[i].nr_inactive;
+ *active += zones[i].nr_resident - zones[i].nr_cold;
+ *inactive += zones[i].nr_cold;
*free += zones[i].free_pages;
}
}
@@ -1387,8 +1387,8 @@ void show_free_areas(void)
" min:%lukB"
" low:%lukB"
" high:%lukB"
- " active:%lukB"
- " inactive:%lukB"
+ " resident:%lukB"
+ " cold:%lukB"
" present:%lukB"
" pages_scanned:%lu"
" all_unreclaimable? %s"
@@ -1398,8 +1398,8 @@ void show_free_areas(void)
K(zone->pages_min),
K(zone->pages_low),
K(zone->pages_high),
- K(zone->nr_active),
- K(zone->nr_inactive),
+ K(zone->nr_resident),
+ K(zone->nr_cold),
K(zone->present_pages),
zone->pages_scanned,
(zone->all_unreclaimable ? "yes" : "no")
@@ -2156,8 +2156,8 @@ static int zoneinfo_show(struct seq_file
"\n min %lu"

```

```
"\n low %lu"
"\n high %lu"
- "\n active %lu"
- "\n inactive %lu"
+ "\n resident %lu"
+ "\n cold %lu"
"\n scanned %lu"
"\n spanned %lu"
"\n present %lu",
@@ -2165,8 +2165,8 @@ static int zoneinfo_show(struct seq_file
zone->pages_min,
zone->pages_low,
zone->pages_high,
- zone->nr_active,
- zone->nr_inactive,
+ zone->nr_resident,
+ zone->nr_cold,
zone->pages_scanned,
zone->spanned_pages,
zone->present_pages);
-
```

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>

---

- **Follow-Ups:**

- ◆ **Re: [PATCH 6/9] clockpro-clockpro.patch**

- ◆ *From:* Marcelo Tosatti

- **References:**

- ◆ **[PATCH] vm: page-replace and clockpro**

- ◆ *From:* Peter Zijlstra

- Prev by Date: **[PATCH 05/14] page-replace-remove-loop.patch**
- Next by Date: **[PATCH 11/14] page-replace-move-refill.patch**
- Previous by thread: **[PATCH 05/14] page-replace-remove-loop.patch**
- Next by thread: **Re: [PATCH 6/9] clockpro-clockpro.patch**
- Index(es):

- ◆ **Date**

- ◆ **Thread**