```
**********************************************************
  187466 Sun Nov  9 09:32:15 2014
new/usr/src/uts/common/vm/vm_page.c
5302 vm: remove 'nopageage' static global
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 1986, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*      Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989  AT&T    */
/*        All Rights Reserved   */

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

/*
 * VM - physical page management.
 */

#include <sys/types.h>
#include <sys/t_lock.h>
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/time.h>
#include <sys/vnode.h>
#include <sys/vm.h>
#include <sys/vtrace.h>
#include <sys/swap.h>
#include <sys/cmn_err.h>
#include <sys/tuneable.h>
#include <sys/sysmacros.h>
#include <sys/cpuvar.h>
#include <sys/callb.h>
#include <sys/debug.h>
#include <sys/tnf_probe.h>
#include <sys/condvar_impl.h>
#include <sys/mem_config.h>
#include <sys/mem_cage.h>
```



```
#include <sys/kmem.h>
#include <sys/atomic.h>
#include <sys/strlog.h>
#include <sys/mman.h>
#include <sys/ontrap.h>
#include <sys/lgrp.h>
#include <sys/vfs.h>

#include <vm/hat.h>
#include <vm/anon.h>
#include <vm/page.h>
#include <vm/seg.h>
#include <vm/pvn.h>
#include <vm/seg_kmem.h>
#include <vm/vm_dep.h>
#include <sys/vm_usage.h>
#include <fs/fs_subr.h>
#include <sys/ddi.h>
#include <sys/modctl.h>

static int nopageage = 0;

static pgcnt_t max_page_get;    /* max page_get request size in pages */
pgcnt_t total_pages = 0;        /* total number of pages (used by /proc) */

/*
 * freemem_lock protects all freemem variables:
 * availrmem. Also this lock protects the globals which track the
 * availrmem changes for accurate kernel footprint calculation.
 * See below for an explanation of these
 * globals.
 */
kmutex_t freemem_lock;
pgcnt_t availrmem;
pgcnt_t availrmem_initial;

/*
 * These globals track availrmem changes to get a more accurate
 * estimate of tke kernel size. Historically pp_kernel is used for
 * kernel size and is based on availrmem. But availrmem is adjusted for
 * locked pages in the system not just for kernel locked pages.
 * These new counters will track the pages locked through segvn and
 * by explicit user locking.
 *
 * pages_locked : How many pages are locked because of user specified
 * locking through mlock or plock.
 *
 * pages_useclaim,pages_claimed : These two variables track the
 * claim adjustments because of the protection changes on a segvn segment.
 *
 * All these globals are protected by the same lock which protects availrmem.
 */
pgcnt_t pages_locked = 0;
pgcnt_t pages_useclaim = 0;
pgcnt_t pages_claimed = 0;


/*
 * new_freemem_lock protects freemem, freemem_wait & freemem_cv.
 */
static kmutex_t new_freemem_lock;
static uint_t   freemem_wait;   /* someone waiting for freemem */
static kcondvar_t freemem_cv;

/*
 * The logical page free list is maintained as two lists, the 'free'
```

```
 126  * and the 'cache' lists.
 127  * The free list contains those pages that should be reused first.
 128  *
 129  * The implementation of the lists is machine dependent.
 130  * page_get_freelist(), page_get_cachelist(),
 131  * page_list_sub(), and page_list_add()
 132  * form the interface to the machine dependent implementation.
 133  *
 134  * Pages with p_free set are on the cache list.
 135  * Pages with p_free and p_age set are on the free list,
 136  *
 137  * A page may be locked while on either list.
 138  */

 140 /*
 141  * free list accounting stuff.
 142  *
 143  *
 144  * Spread out the value for the number of pages on the
 145  * page free and page cache lists.  If there is just one
 146  * value, then it must be under just one lock.
 147  * The lock contention and cache traffic are a real bother.
 148  *
 149  * When we acquire and then drop a single pcf lock
 150  * we can start in the middle of the array of pcf structures.
 151  * If we acquire more than one pcf lock at a time, we need to
 152  * start at the front to avoid deadlocking.
 153  *
 154  * pcf_count holds the number of pages in each pool.
 155  *
 156  * pcf_block is set when page_create_get_something() has asked the
 157  * PSM page freelist and page cachelist routines without specifying
 158  * a color and nothing came back.  This is used to block anything
 159  * else from moving pages from one list to the other while the
 160  * lists are searched again.  If a page is freeed while pcf_block is
 161  * set, then pcf_reserve is incremented.  pcgs_unblock() takes care
 162  * of clearning pcf_block, doing the wakeups, etc.
 163  */

 165 #define MAX_PCF_FANOUT NCPU
 166 static uint_t pcf_fanout = 1; /* Will get changed at boot time */
 167 static uint_t pcf_fanout_mask = 0;

 169 struct pcf {
 170         kmutex_t        pcf_lock;       /* protects the structure */
 171         uint_t          pcf_count;      /* page count */
 172         uint_t          pcf_wait;       /* number of waiters */
 173         uint_t          pcf_block;      /* pcgs flag to page_free() */
 174         uint_t          pcf_reserve;    /* pages freed after pcf_block set */
 175         uint_t          pcf_fill[10];   /* to line up on the caches */
 176 };
_____unchanged_portion_omitted_

2625 /*
2626  * Put page on the "free" list.
2627  * The free list is really two lists maintained by
2628  * the PSM of whatever machine we happen to be on.
2629  */
2630 void
2631 page_free(page_t *pp, int dontneed)
2632 {
2633         struct pcf      *p;
2634         uint_t          pcf_index;

2636         ASSERT((PAGE_EXCL(pp) &&
2637             !page_iolock_assert(pp)) || panicstr);
```



```
2639         if (PP_ISFREE(pp)) {
2640                 panic("page_free: page %p is free", (void *)pp);
2641         }

2643         if (pp->p_szc != 0) {
2644                 if (pp->p_vnode == NULL || IS_SWAPFSVP(pp->p_vnode) ||
2645                     PP_ISKAS(pp)) {
2646                         panic("page_free: anon or kernel "
2647                             "or no vnode large page %p", (void *)pp);
2648                 }
2649                 page_demote_vp_pages(pp);
2650                 ASSERT(pp->p_szc == 0);
2651         }

2653         /*
2654          * The page_struct_lock need not be acquired to examine these
2655          * fields since the page has an "exclusive" lock.
2656          */
2657         if (hat_page_is_mapped(pp) || pp->p_lckcnt != 0 || pp->p_cowcnt != 0 ||
2658             pp->p_slckcnt != 0) {
2659                 panic("page_free pp=%p, pfn=%lx, lckcnt=%d, cowcnt=%d "
2660                     "slckcnt = %d", (void *)pp, page_pptonum(pp), pp->p_lckcnt,
2661                     pp->p_cowcnt, pp->p_slckcnt);
2662                 /*NOTREACHED*/
2663         }

2665         ASSERT(!hat_page_getshare(pp));

2667         PP_SETFREE(pp);
2668         ASSERT(pp->p_vnode == NULL || !IS_VMODSORT(pp->p_vnode) ||
2669             !hat_ismod(pp));
2670         page_clr_all_props(pp);
2671         ASSERT(!hat_page_getshare(pp));

2673         /*
2674          * Now we add the page to the head of the free list.
2675          * But if this page is associated with a paged vnode
2676          * then we adjust the head forward so that the page is
2677          * effectively at the end of the list.
2678          */
2679         if (pp->p_vnode == NULL) {
2680                 /*
2681                  * Page has no identity, put it on the free list.
2682                  */
2683                 PP_SETAGED(pp);
2684                 pp->p_offset = (u_offset_t)-1;
2685                 page_list_add(pp, PG_FREE_LIST | PG_LIST_TAIL);
2686                 VM_STAT_ADD(pagecnt.pc_free_free);
2687                 TRACE_1(TR_FAC_VM, TR_PAGE_FREE_FREE,
2688                     "page_free_free:pp %p", pp);
2689         } else {
2690                 PP_CLRAGED(pp);

2692                 if (!dontneed) {
2694                 if (!dontneed || nopageage) {
2693                         /* move it to the tail of the list */
2694                         page_list_add(pp, PG_CACHE_LIST | PG_LIST_TAIL);

2696                         VM_STAT_ADD(pagecnt.pc_free_cache);
2697                         TRACE_1(TR_FAC_VM, TR_PAGE_FREE_CACHE_TAIL,
2698                             "page_free_cache_tail:pp %p", pp);
2699                 } else {
2700                         page_list_add(pp, PG_CACHE_LIST | PG_LIST_HEAD);

2702                         VM_STAT_ADD(pagecnt.pc_free_dontneed);
```

```
2703                        TRACE_1(TR_FAC_VM, TR_PAGE_FREE_CACHE_HEAD,
2704                            "page_free_cache_head:pp %p", pp);
2705                }
2706        }
2707        page_unlock(pp);

2709        /*
2710         * Now do the ‘freemem’ accounting.
2711         */
2712        pcf_index = PCF_INDEX();
2713        p = &pcf[pcf_index];

2715        mutex_enter(&p->pcf_lock);
2716        if (p->pcf_block) {
2717                p->pcf_reserve += 1;
2718        } else {
2719                p->pcf_count += 1;
2720                if (p->pcf_wait) {
2721                        mutex_enter(&new_freemem_lock);
2722                        /*
2723                         * Check to see if some other thread
2724                         * is actually waiting.  Another bucket
2725                         * may have woken it up by now.  If there
2726                         * are no waiters, then set our pcf_wait
2727                         * count to zero to avoid coming in here
2728                         * next time.  Also, since only one page
2729                         * was put on the free list, just wake
2730                         * up one waiter.
2731                         */
2732                        if (freemem_wait) {
2733                                cv_signal(&freemem_cv);
2734                                p->pcf_wait--;
2735                        } else {
2736                                p->pcf_wait = 0;
2737                        }
2738                        mutex_exit(&new_freemem_lock);
2739                }
2740        }
2741        mutex_exit(&p->pcf_lock);

2743        /* freemem is approximate, so this test OK */
2744        if (!p->pcf_block)
2745                freemem += 1;
2746 }
```

_____***unchanged_portion_omitted_***