```
**********************************************************
  100184 Thu Apr 25 02:44:40 2019
new/usr/src/uts/i86pc/vm/vm_machdep.c
10806 mnode_range_setup() makes assumptions about mnodes
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Toomas Soome <tsoome@me.com>
**********************************************************
```

```
/*
 * 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) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 * Copyright 2019, Joyent, Inc.
 * Copyright 2018 Joyent, Inc.
 */

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

/*
 * Portions of this source code were derived from Berkeley 4.3 BSD
 * under license from the Regents of the University of California.
 */

/*
 * UNIX machine dependent virtual memory support.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/user.h>
#include <sys/proc.h>
#include <sys/kmem.h>
#include <sys/vmem.h>
#include <sys/buf.h>
#include <sys/cpuvar.h>
#include <sys/lgrp.h>
#include <sys/disp.h>
#include <sys/vm.h>
#include <sys/mman.h>
#include <sys/vnode.h>
#include <sys/cred.h>
#include <sys/exec.h>
```



```
#include <sys/exechdr.h>
#include <sys/debug.h>
#include <sys/vmsystm.h>
#include <sys/swap.h>
#include <sys/dumphdr.h>
#include <sys/random.h>

#include <vm/hat.h>
#include <vm/as.h>
#include <vm/seg.h>
#include <vm/seg_kp.h>
#include <vm/seg_vn.h>
#include <vm/page.h>
#include <vm/seg_kmem.h>
#include <vm/seg_kpm.h>
#include <vm/vm_dep.h>

#include <sys/cpu.h>
#include <sys/vm_machparam.h>
#include <sys/memlist.h>
#include <sys/bootconf.h> /* XXX the memlist stuff belongs in memlist_plat.h */
#include <vm/hat_i86.h>
#include <sys/x86_archext.h>
#include <sys/elf_386.h>
#include <sys/cmn_err.h>
#include <sys/archsystm.h>
#include <sys/machsystm.h>
#include <sys/secflags.h>

#include <sys/vtrace.h>
#include <sys/ddidmareq.h>
#include <sys/promif.h>
#include <sys/memnode.h>
#include <sys/stack.h>
#include <util/qsort.h>
#include <sys/taskq.h>

#ifdef __xpv

#include <sys/hypervisor.h>
#include <sys/xen_mmu.h>
#include <sys/balloon_impl.h>

/*
 * domain 0 pages usable for DMA are kept pre-allocated and kept in
 * distinct lists, ordered by increasing mfn.
 */
static kmutex_t io_pool_lock;
static kmutex_t contig_list_lock;
static page_t *io_pool_4g;      /* pool for 32 bit dma limited devices */
static page_t *io_pool_16m;     /* pool for 24 bit dma limited legacy devices */
static long io_pool_cnt;
static long io_pool_cnt_max = 0;
#define DEFAULT_IO_POOL_MIN     128
static long io_pool_cnt_min = DEFAULT_IO_POOL_MIN;
static long io_pool_cnt_lowater = 0;
static long io_pool_shrink_attempts; /* how many times did we try to shrink */
static long io_pool_shrinks;    /* how many times did we really shrink */
static long io_pool_grows;      /* how many times did we grow */
static mfn_t start_mfn = 1;
static caddr_t io_pool_kva;     /* use to alloc pages when needed */

static int create_contig_pfnlist(uint_t);

/*
 * percentage of phys mem to hold in the i/o pool
```

```
 */
#define DEFAULT_IO_POOL_PCT     2
static long io_pool_physmem_pct = DEFAULT_IO_POOL_PCT;
static void page_io_pool_sub(page_t **, page_t *, page_t *);
int ioalloc_dbg = 0;

#endif /* __xpv */

uint_t vac_colors = 1;

int largepagesupport = 0;
extern uint_t page_create_new;
extern uint_t page_create_exists;
extern uint_t page_create_putbacks;
/*
 * Allow users to disable the kernel's use of SSE.
 */
extern int use_sse_pagecopy, use_sse_pagezero;

/*
 * combined memory ranges from mnode and memranges[] to manage single
 * mnode/mtype dimension in the page lists.
 */
typedef struct {
        pfn_t   mnr_pfnlo;
        pfn_t   mnr_pfnhi;
        int     mnr_mnode;
        int     mnr_memrange;           /* index into memranges[] */
        int     mnr_next;               /* next lower PA mnoderange */
        int     mnr_exists;
        /* maintain page list stats */
        pgcnt_t mnr_mt_clpgcnt;         /* cache list cnt */
        pgcnt_t mnr_mt_flpgcnt[MMU_PAGE_SIZES]; /* free list cnt per szc */
        pgcnt_t mnr_mt_totcnt;          /* sum of cache and free lists */
#ifdef DEBUG
        struct mnr_mts {                /* mnode/mtype szc stats */
                pgcnt_t mnr_mts_pgcnt;
                int     mnr_mts_colors;
                pgcnt_t *mnr_mtsc_pgcnt;
        }       *mnr_mts;
#endif
} mnoderange_t;
```

*unchanged_portion_omitted_*

```
pfn_t *memranges = &arch_memranges[0];
int nranges = NUM_MEM_RANGES;

/*
 * This combines mem_node_config and memranges into one data
 * structure to be used for page list management.
 */
static mnoderange_t *mnoderanges;
static int mnoderangecnt;
static int mtype4g;
static int mtype16m;
static int mtypetop;
mnoderange_t    *mnoderanges;
int             mnoderangecnt;
int             mtype4g;
int             mtype16m;
int             mtypetop;       /* index of highest pfn'ed mnoderange */

/*
 * 4g memory management variables for systems with more than 4g of memory:
 *
 * physical memory below 4g is required for 32bit dma devices and, currently,
 * for kmem memory. On systems with more than 4g of memory, the pool of memory
```



```
 * below 4g can be depleted without any paging activity given that there is
 * likely to be sufficient memory above 4g.
 *
 * physmax4g is set true if the largest pfn is over 4g. The rest of the
 * 4g memory management code is enabled only when physmax4g is true.
 *
 * maxmem4g is the count of the maximum number of pages on the page lists
 * with physical addresses below 4g. It can be a lot less then 4g given that
 * BIOS may reserve large chunks of space below 4g for hot plug pci devices,
 * agp aperture etc.
 *
 * freemem4g maintains the count of the number of available pages on the
 * page lists with physical addresses below 4g.
 *
 * DESFREE4G specifies the desired amount of below 4g memory. It defaults to
 * 6% (desfree4gshift = 4) of maxmem4g.
 *
 * RESTRICT4G_ALLOC returns true if freemem4g falls below DESFREE4G
 * and the amount of physical memory above 4g is greater than freemem4g.
 * In this case, page_get_* routines will restrict below 4g allocations
 * for requests that don't specifically require it.
 */

#define DESFREE4G       (maxmem4g >> desfree4gshift)

#define RESTRICT4G_ALLOC                                                \
        (physmax4g && (freemem4g < DESFREE4G) && ((freemem4g << 1) < freemem))

static pgcnt_t  maxmem4g;
static pgcnt_t  freemem4g;
static int      physmax4g;
static int      desfree4gshift = 4;     /* maxmem4g shift to derive DESFREE4G */

/*
 * 16m memory management:
 *
 * reserve some amount of physical memory below 16m for legacy devices.
 *
 * RESTRICT16M_ALLOC returns true if an there are sufficient free pages above
 * 16m or if the 16m pool drops below DESFREE16M.
 *
 * In this case, general page allocations via page_get_{free,cache}list
 * routines will be restricted from allocating from the 16m pool. Allocations
 * that require specific pfn ranges (page_get_anylist) and PG_PANIC allocations
 * are not restricted.
 */

#define FREEMEM16M      MTYPE_FREEMEM(mtype16m)
#define DESFREE16M      desfree16m
#define RESTRICT16M_ALLOC(freemem, pgcnt, flags) \
        (mtype16m != -1 && (freemem != 0) && ((flags & PG_PANIC) == 0) && \
        ((freemem != 0) && ((flags & PG_PANIC) == 0) &&          \
            ((freemem >= (FREEMEM16M)) || \
            (FREEMEM16M  < (DESFREE16M + pgcnt))))

static pgcnt_t  desfree16m = 0x380;

/*
 * This can be patched via /etc/system to allow old non-PAE aware device
 * drivers to use kmem_alloc'd memory on 32 bit systems with > 4Gig RAM.
 */
int restricted_kmemalloc = 0;

#ifdef VM_STATS
struct {
        ulong_t pga_alloc;
```

```
 281         ulong_t pga_notfullrange;
 282         ulong_t pga_nulldmaattr;
 283         ulong_t pga_allocok;
 284         ulong_t pga_allocfailed;
 285         ulong_t pgma_alloc;
 286         ulong_t pgma_allocok;
 287         ulong_t pgma_allocfailed;
 288         ulong_t pgma_allocempty;
 289 } pga_vmstats;
_____unchanged_portion_omitted_

1392 static int
1393 mnoderange_cmp(const void *v1, const void *v2)
1394 {
1395         const mnoderange_t *m1 = v1;
1396         const mnoderange_t *m2 = v2;

1398         if (m1->mnr_pfnlo < m2->mnr_pfnlo)
1399                 return (-1);
1400         return (m1->mnr_pfnlo > m2->mnr_pfnlo);
1401 }

1392 /*
1393  * mnode_range_setup() initializes mnoderanges.
1394  */
1403 void
1404 mnode_range_setup(mnoderange_t *mnoderanges)
1405 {
1406         mnoderange_t *mp;
1407         size_t nr_ranges;
1408         size_t mnode;
1398         mnoderange_t *mp = mnoderanges;
1399         int     mnode, mri;
1400         int     mindex = 0;     /* current index into mnoderanges array */
1401         int     i, j;
1402         pfn_t   hipfn;
1403         int     last, hi;

1410         for (mnode = 0, nr_ranges = 0, mp = mnoderanges;
1411             mnode < max_mem_nodes; mnode++) {
1412                 size_t mri = nranges - 1;

1405         for (mnode = 0; mnode < max_mem_nodes; mnode++) {
1414                 if (mem_node_config[mnode].exists == 0)
1415                         continue;

1409                 mri = nranges - 1;

1417                 while (MEMRANGEHI(mri) < mem_node_config[mnode].physbase)
1418                         mri--;

1420                 while (mri >= 0 && mem_node_config[mnode].physmax >=
1421                     MEMRANGELO(mri)) {
1422                         mp->mnr_pfnlo = MAX(MEMRANGELO(mri),
1416                         mnoderanges->mnr_pfnlo = MAX(MEMRANGELO(mri),
1423                             mem_node_config[mnode].physbase);
1424                         mp->mnr_pfnhi = MIN(MEMRANGEHI(mri),
1418                         mnoderanges->mnr_pfnhi = MIN(MEMRANGEHI(mri),
1425                             mem_node_config[mnode].physmax);
1426                         mp->mnr_mnode = mnode;
1427                         mp->mnr_memrange = mri;
1428                         mp->mnr_next = -1;
1429                         mp->mnr_exists = 1;
1430                         mp++;
1431                         nr_ranges++;
1420                         mnoderanges->mnr_mnode = mnode;
```



```
1421                         mnoderanges->mnr_memrange = mri;
1422                         mnoderanges->mnr_exists = 1;
1423                         mnoderanges++;
1424                         mindex++;
1432                         if (mem_node_config[mnode].physmax > MEMRANGEHI(mri))
1433                                 mri--;
1434                         else
1435                                 break;
1436                 }
1437         }

1439         /*
1440          * mnoderangecnt can be larger than nr_ranges when memory DR is
1441          * supposedly supported.
1433          * For now do a simple sort of the mnoderanges array to fill in
1434          * the mnr_next fields.  Since mindex is expected to be relatively
1435          * small, using a simple O(N^2) algorithm.
1442          */
1443         VERIFY3U(nr_ranges, <=, mnoderangecnt);

1445         qsort(mnoderanges, nr_ranges, sizeof (mnoderange_t), mnoderange_cmp);

1447         /*
1448          * If some intrepid soul takes the axe to the memory DR code, we can
1449          * remove ->mnr_next altogether, as we just sorted by ->mnr_pfnlo order.
1450          *
1451          * The VERIFY3U() above can be "==" then too.
1452          */
1453         for (size_t i = 1; i < nr_ranges; i++)
1454                 mnoderanges[i].mnr_next = i - 1;

1456         mtypetop = nr_ranges - 1;
1457         mtype16m = pfn_2_mtype(PFN_16MEG - 1); /* Can be -1 ... */
1458         if (physmax4g)
1459                 mtype4g = pfn_2_mtype(0xfffff);
1437         for (i = 0; i < mindex; i++) {
1438                 if (mp[i].mnr_pfnlo == 0)       /* find lowest */
1439                         break;
1440         }
1441         ASSERT(i < mindex);
1442         last = i;
1443         mtype16m = last;
1444         mp[last].mnr_next = -1;
1445         for (i = 0; i < mindex - 1; i++) {
1446                 hipfn = (pfn_t)(-1);
1447                 hi = -1;
1448                 /* find next highest mnode range */
1449                 for (j = 0; j < mindex; j++) {
1450                         if (mp[j].mnr_pfnlo > mp[last].mnr_pfnlo &&
1451                             mp[j].mnr_pfnlo < hipfn) {
1452                                 hipfn = mp[j].mnr_pfnlo;
1453                                 hi = j;
1454                         }
1455                 }
1456                 mp[hi].mnr_next = last;
1457                 last = hi;
1458         }
1459         mtypetop = last;
1460 }
_____unchanged_portion_omitted_

1958 /*
1959  * Called once at startup to configure page_coloring data structures and
1960  * does the 1st page_free()/page_freelist_add().
1961  */
1962 void
```

```
1963 page_coloring_setup(caddr_t pcmemaddr)
1964 {
1965         int     i;
1966         int     j;
1967         int     k;
1968         caddr_t addr;
1969         int     colors;

1971         /*
1972          * do page coloring setup
1973          */
1974         addr = pcmemaddr;

1976         mnoderanges = (mnoderange_t *)addr;
1977         addr += (mnoderangecnt * sizeof (mnoderange_t));

1979         mnode_range_setup(mnoderanges);

1981         if (physmax4g)
1982                 mtype4g = pfn_2_mtype(0xfffff);

1981         for (k = 0; k < NPC_MUTEX; k++) {
1982                 fpc_mutex[k] = (kmutex_t *)addr;
1983                 addr += (max_mem_nodes * sizeof (kmutex_t));
1984         }
1985         for (k = 0; k < NPC_MUTEX; k++) {
1986                 cpc_mutex[k] = (kmutex_t *)addr;
1987                 addr += (max_mem_nodes * sizeof (kmutex_t));
1988         }
1989         page_freelists = (page_t ****)addr;
1990         addr += (mnoderangecnt * sizeof (page_t ***));

1992         page_cachelists = (page_t ***)addr;
1993         addr += (mnoderangecnt * sizeof (page_t **));

1995         for (i = 0; i < mnoderangecnt; i++) {
1996                 page_freelists[i] = (page_t ***)addr;
1997                 addr += (mmu_page_sizes * sizeof (page_t **));

1999                 for (j = 0; j < mmu_page_sizes; j++) {
2000                         colors = page_get_pagecolors(j);
2001                         page_freelists[i][j] = (page_t **)addr;
2002                         addr += (colors * sizeof (page_t *));
2003                 }
2004                 page_cachelists[i] = (page_t **)addr;
2005                 addr += (page_colors * sizeof (page_t *));
2006         }
2007 }
```

_____***unchanged_portion_omitted_***