Split page table lock

Originally, mm->page_table_lock spinlock protected all page tables of the mm_struct. But this approach leads to poor page fault scalability of multi-threaded applications due high contention on the lock. To improve scalability, split page table lock was introduced.

With split page table lock we have separate per-table lock to serialize access to the table. At the moment we use split lock for PTE and PMD tables. Access to higher level tables protected by mm->page_table_lock.

There are helpers to lock/unlock a table and other accessor functions:

- pte_offset_map_lock()
  maps pte and takes PTE table lock, returns pointer to the taken lock;
- pte_unmap_unlock()
  unlocks and unmaps PTE table;
- pte_alloc_map_lock()
  allocates PTE table if needed and take the lock, returns pointer to taken lock or NULL if allocation failed;
- pte_lockptr()
  returns pointer to PTE table lock;
- pmd_lock()
  takes PMD table lock, returns pointer to taken lock;
- pmd_lockptr()
  returns pointer to PMD table lock;

Split page table lock for PTE tables is enabled compile-time if CONFIG_SPLIT_PTLOCK_CPUS (usually 4) is less or equal to NR_CPUS. If split lock is disabled, all tables are guarded by mm->page_table_lock.

Split page table lock for PMD tables is enabled, if it's enabled for PTE tables and the architecture supports it (see below).

Hugetlb and split page table lock

Hugetlb can support several page sizes. We use split lock only for PMD level, but not for PUD.

Hugetlb-specific helpers:

- huge_pte_lock()
  takes pmd split lock for PMD_SIZE page, mm->page_table_lock otherwise;
- huge_pte_lockptr()
  returns pointer to table lock;

Support of split page table lock by an architecture

There's no need in special enabling of PTE split page table lock: everything required is done by pgtable_pte_page_ctor() and pgtable_pte_page_dtor(), which must be called on PTE table allocation / freeing.

Make sure the architecture doesn't use slab allocator for page table allocation: slab uses page->slab_cache for its pages. This field shares storage with page->ptl.

PMD split lock only makes sense if you have more than two page table levels.

PMD split lock enabling requires pgtable_pmd_page_ctor() call on PMD table allocation and pgtable_pmd_page_dtor() on freeing.

Allocation usually happens in pmd_alloc_one(), freeing in pmd_free() and pmd_free_tlb(), but make sure you cover all PMD table allocation / freeing paths: i.e X86_PAE preallocate few PMDs on pgd_alloc().

With everything in place you can set CONFIG_ARCH_ENABLE_SPLIT_PMD_PTLOCK.

NOTE: pgtable_pte_page_ctor() and pgtable_pmd_page_ctor() can fail -- it must be handled properly.

page->ptl

page->ptl is used to access split page table lock, where 'page' is struct page of page containing the table. It shares storage with page->private (and few other fields in union).

To avoid increasing size of struct page and have best performance, we use a trick:

- if spinlock_t fits into long, we use page->ptr as spinlock, so we can avoid indirect access and save a cache line.
- if size of spinlock_t is bigger then size of long, we use page->ptl as pointer to spinlock_t and allocate it dynamically. This allows to use split lock with enabled DEBUG_SPINLOCK or DEBUG_LOCK_ALLOC, but costs one more

cache line for indirect access;

The spinlock_t allocated in pgtable_pte_page_ctor() for PTE table and in pgtable_pmd_page_ctor() for PMD table.

Please, never access page->ptl directly -- use appropriate helper.