```
CS5600, Handout week 5.b

1. Simple deadlock example

  T1:
    acquire(mutexA);
    acquire(mutexB);

    // do some stuff

    release(mutexB);
    release(mutexA);

  T2:
    acquire(mutexB);
    acquire(mutexA);

    // do some stuff

    release(mutexA);
    release(mutexB);



2. More subtle deadlock example

    Let M be a monitor (shared object with methods protected by mutex)
    Let N be another monitor

    class M {
      private:
        Mutex mutex_m;

        // instance of monitor N
        N another_monitor;

        // Assumption: no other objects in the system hold a pointer
        // to our "another_monitor"

      public:
        M();
        ~M();
        void methodA();
        void methodB();
    };
```



```
    class N {
      private:
        Mutex mutex_n;
        Cond cond_n;
        int navailable;

      public:
        N();
        ~N();
        void* alloc(int nwanted);
        void free(void*);
    }

    int
    N::alloc(int nwanted) {
      acquire(&mutex_n);
      while (navailable < nwanted) {
        wait(&cond_n, &mutex_n);
      }

      // peel off the memory

      navailable -= nwanted;
      release(&mutex_n);
    }

    void
    N::free(void* returning_mem) {

      acquire(&mutex_n);

      // put the memory back

      navailable += returning_mem;

      broadcast(&cond_n, &mutex_n);

      release(&mutex_n);
    }

    void
    M::methodA() {

      acquire(&mutex_m);

      void* new_mem = another_monitor.alloc(int nbytes);

      // do a bunch of stuff using this nice
      // chunk of memory n allocated for us

      release(&mutex_m);
    }

    void
     M::methodB() {

       acquire(&mutex_m);

       // do a bunch of stuff

       another_monitor.free(some_pointer);

       release(&mutex_m);
     }

QUESTION: What's the problem?
```

```
112
113   3. Locking brings a performance vs. complexity trade-off
114
115   /*
116   * linux/mm/filemap.c
117   *
118   * Copyright (C) 1994-1999 Linus Torvalds
119   */
120
121   /*
122   * This file handles the generic file mmap semantics used by
123   * most "normal" filesystems (but you don't /have/ to use this:
124   * the NFS filesystem used to do this differently, for example)
125   */
126   #include <linux/export.h>
127   #include <linux/compiler.h>
128   #include <linux/dax.h>
129   #include <linux/fs.h>
130   #include <linux/sched/signal.h>
131   #include <linux/uaccess.h>
132   #include <linux/capability.h>
133   #include <linux/kernel_stat.h>
134   #include <linux/gfp.h>
135   #include <linux/mm.h>
136   #include <linux/swap.h>
137   #include <linux/mman.h>
138   #include <linux/pagemap.h>
139   #include <linux/file.h>
140   #include <linux/uio.h>
141   #include <linux/hash.h>
142   #include <linux/writeback.h>
143   #include <linux/backing-dev.h>
144   #include <linux/pagevec.h>
145   #include <linux/blkdev.h>
146   #include <linux/security.h>
147   #include <linux/cpuset.h>
148   #include <linux/hugetlb.h>
149   #include <linux/memcontrol.h>
150   #include <linux/cleancache.h>
151   #include <linux/shmem_fs.h>
152   #include <linux/rmap.h>
153   #include "internal.h"
154
155   #define CREATE_TRACE_POINTS
156   #include <trace/events/filemap.h>
157
158   /*
159   * FIXME: remove all knowledge of the buffer layer from the core VM
160   */
161   #include <linux/buffer_head.h> /* for try_to_free_buffers */
162
163   #include <asm/mman.h>
164
165   /*
166   * Shared mappings implemented 30.11.1994. It's not fully working yet,
167   * though.
168   *
169   * Shared mappings now work. 15.8.1995 Bruno.
170   *
171   * finished 'unifying' the page and buffer cache and SMP-threaded the
172   * page-cache, 21.05.1999, Ingo Molnar <mingo@redhat.com>
173   *
174   * SMP-threaded pagemap-LRU 1999, Andrea Arcangeli <andrea@suse.de>
175   */
176
177   /*
178   * Lock ordering:
179   *
180   * ->i_mmap_rwsem (truncate_pagecache)
181   *   ->private_lock (__free_pte->__set_page_dirty_buffers)
182   *     ->swap_lock (exclusive_swap_page, others)
183   *         ->i_pages lock
184   *
185   * ->i_mutex
```



```
186   *   ->i_mmap_rwsem (truncate->unmap_mapping_range)
187   *
188   * ->mmap_sem
189   *   ->i_mmap_rwsem
190   *     ->page_table_lock or pte_lock (various, mainly in memory.c)
191   *       ->i_pages lock (arch-dependent flush_dcache_mmap_lock)
192   *
193   * ->mmap_sem
194   *   ->lock_page (access_process_vm)
195   *
196   * ->i_mutex (generic_perform_write)
197   *   ->mmap_sem (fault_in_pages_readable->do_page_fault)
198   *
199   * bdi->wb.list_lock
200   *   sb_lock (fs/fs-writeback.c)
201   *   ->i_pages lock (__sync_single_inode)
202   *
203   * ->i_mmap_rwsem
204   *   ->anon_vma.lock (vma_adjust)
205   *
206   * ->anon_vma.lock
207   *   ->page_table_lock or pte_lock (anon_vma_prepare and various)
208   *
209   * ->page_table_lock or pte_lock
210   *   ->swap_lock (try_to_unmap_one)
211   *   ->private_lock (try_to_unmap_one)
212   *   ->i_pages lock (try_to_unmap_one)
213   *   ->zone_lru_lock(zone) (follow_page->mark_page_accessed)
214   *   ->zone_lru_lock(zone) (check_pte_range->isolate_lru_page)
215   *   ->private_lock (page_remove_rmap->set_page_dirty)
216   *   ->i_pages lock (page_remove_rmap->set_page_dirty)
217   *   bdi.wb->list_lock (page_remove_rmap->set_page_dirty)
218   *   ->inode->i_lock (page_remove_rmap->set_page_dirty)
219   *   ->memcg->move_lock (page_remove_rmap->lock_page_memcg)
220   *   bdi.wb->list_lock (zap_pte_range->set_page_dirty)
221   *   ->inode->i_lock (zap_pte_range->set_page_dirty)
222   *   ->private_lock (zap_pte_range->__set_page_dirty_buffers)
223   *
224   * ->i_mmap_rwsem
225   *   ->tasklist_lock (memory_failure, collect_procs_ao)
226   */
227
228   static int page_cache_tree_insert(struct address_space *mapping,
229   struct page *page, void **shadowp)
230   {
231       struct radix_tree_node *node;
232       .....
233
234   [the point is: fine-grained locking leads to complexity.]
235
```