NAME

vm_page_busied, **vm_page_busy_downgrade**, **vm_page_busy_sleep**, **vm_page_sbusied**, **vm_page_sunbusy**, **vm_page_trysbusy**, **vm_page_tryxbusy**, **vm_page_xbusied**, **vm_page_xunbusy**, **vm_page_assert_sbusied**, **vm_page_assert_unbusied**, **vm_page_assert_xbusied** - protect page identity changes and page content references

SYNOPSIS

#include <sys/param.h>
#include <vm/vm.h>
#include <vm/vm_page.h>

int
vm_page_busied(*vm_page_t m*);

void
vm_page_busy_downgrade(*vm_page_t m*);

bool
vm_page_busy_sleep(*vm_page_t m*, *const char *msg*, *int allocflags*);

int
vm_page_sbusied(*vm_page_t m*);

void
vm_page_sunbusy(*vm_page_t m*);

int
vm_page_trysbusy(*vm_page_t m*);

int
vm_page_tryxbusy(*vm_page_t m*);

int
vm_page_xbusied(*vm_page_t m*);

void
vm_page_xunbusy(*vm_page_t m*);

options INVARIANTS
options INVARIANT_SUPPORT

void
vm_page_assert_sbusied(*vm_page_t m*);

void
vm_page_assert_unbusied(*vm_page_t m*);

void
vm_page_assert_xbusied(*vm_page_t m*);

DESCRIPTION

Page identity is usually protected by higher level locks like vm_object locks and vm page locks. However, sometimes it is not possible to hold such locks for the time necessary to complete the identity change. In such case the page can be exclusively busied by a thread which needs to own the identity for a certain amount of time.

In other situations, threads do not need to change the identity of the page but they want to prevent other threads from changing the identity themselves. For example, when a thread wants to access or update page contents without a lock held the page is shared busied.

Before busying a page the vm_object lock must be held. The same rule applies when a page is unbusied. This makes the vm_object lock a real busy interlock.

The **vm_page_busied**() function returns non-zero if the current thread busied *m* in either exclusive or shared mode. Returns zero otherwise.

The **vm_page_busy_downgrade**() function must be used to downgrade *m* from an exclusive busy state to a shared busy state.

The **vm_page_busy_sleep**() checks the busy state of the page *m* and puts the invoking thread to sleep if the page is busy. The VM object for the page must be locked. The *allocflags* parameter must be either 0, in which case the function will sleep if the page is busied, or VM_ALLOC_IGN_SBUSY, in which case the function will sleep only if the page is exclusively busied. A return value of true indicates that the invoking thread was put to sleep and that the object was unlocked. A return value of false indicates that the invoking thread did not sleep and the object remains locked. The parameter *msg* is a string describing the sleep condition for userland tools.

The **vm_page_busied**() function returns non-zero if the current thread busied *m* in shared mode. Returns zero otherwise.

The **vm_page_sunbusy**() function shared unbusies *m*.

The **vm_page_trysbusy**() attempts to shared busy *m*. If the operation cannot immediately succeed **vm_page_trysbusy**() returns 0, otherwise a non-zero value is returned.

The **vm_page_tryxbusy**() attempts to exclusive busy *m*. If the operation cannot immediately succeed **vm_page_tryxbusy**() returns 0, otherwise a non-zero value is returned.

The **vm_page_xbusied**() function returns non-zero if the current thread busied *m* in exclusive mode. Returns zero otherwise.

The **vm_page_xunbusy**() function exclusive unbusies *m*. Assertions on the busy state allow kernels compiled with **options INVARIANTS** and **options INVARIANT_SUPPORT** to panic if they are not respected.

The **vm_page_assert_sbusied**() function panics if *m* is not shared busied.

The **vm_page_assert_unbusied**() function panics if *m* is not unbusied.

The **vm_page_assert_xbusied**() function panics if *m* is not exclusive busied.

SEE ALSO

vm_page_aflag(9), vm_page_alloc(9), vm_page_deactivate(9), vm_page_free(9), vm_page_grab(9), vm_page_insert(9), vm_page_lookup(9), vm_page_rename(9), VOP_GETPAGES(9)