NAME

vm_map - virtual address space portion of virtual memory subsystem

SYNOPSIS

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

DESCRIPTION

The **vm_map** subsystem is used to manage virtual address spaces. This section describes the main data structures used within the code.

The *struct vm_map* is a generic representation of an address space. This address space may belong to a user process or the kernel. The kernel actually uses several maps, which are maintained as subordinate maps, created using the vm_map_submap(9) function.

```
struct vm_map {
        struct vm_map_entry header;
        struct sx lock;
        struct mtx system_mtx;
        int nentries;
        vm_size_t size;
        u_int timestamp;
        u_char needs_wakeup;
        u_char system_map;
        vm_flags_t flags;
        vm_map_entry_t root;
        pmap_t pmap;
        int busy;
};
```

The fields of *struct vm_map* are as follows:

header Head node of a circular, doubly linked list of *struct vm_map_entry* objects. Each object defines a particular region within this map's address space.

lock Used to serialize access to the structure.

system_mtx A mutex which is used if the map is a system map.

nentries A count of the members in use within the circular map entry list.

size Specifies the size of the virtual address space.

timestamp Used to determine if the map has changed since its last access.

needs_wakeup Indicates if a thread is waiting for an allocation within the map. Used only by system maps.

system_map Set to TRUE to indicate that map is a system map; otherwise, it belongs to a user process.

flags Map flags, described below.

root Root node of a binary search tree used for fast lookup of map entries.

pmap Pointer to the underlying physical map with which this virtual map is associated.

busy Map busy counter, prevents forks.

Possible map flags:

MAP_WIREFUTURE Wire all future pages in this map.

MAP_BUSY_WAKEUP There are waiters for the map busy status.

The following flags can be passed to vm_map_find(9) and vm_map_insert(9) to specify the copy-on-write properties of regions within the map:

MAP_COPY_ON_WRITE The mapping is copy-on-write.

MAP_NOFAULT The mapping should not generate page faults.

MAP_PREFAULT The mapping should be prefaulted into physical memory.

MAP_PREFAULT_PARTIAL The mapping should be partially prefaulted into physical memory.

MAP_DISABLE_SYNCER Do not periodically flush dirty pages; only flush them when absolutely necessary.

MAP_DISABLE_COREDUMP
: Do not include the mapping in a core dump.

MAP_PREFAULT_MADVISE
: Specify that the request is from a user process calling madvise(2).

MAP_ACC_CHARGED
: Region is already charged to the requestor by some means.

MAP_ACC_NO_CHARGE
: Do not charge for allocated region.

The *struct vm_map_entry* is a generic representation of a region. The region managed by each entry is associated with a *union vm_map_object*, described below.

```
struct vm_map_entry {
        struct vm_map_entry *prev;
        struct vm_map_entry *next;
        struct vm_map_entry *left;
        struct vm_map_entry *right;
        vm_offset_t start;
        vm_offset_t end;
        vm_offset_t avail_ssize;
        vm_size_t adj_free;
        vm_size_t max_free;
        union vm_map_object object;
        vm_ooffset_t offset;
        vm_eflags_t eflags;
        /* Only in task maps: */
        vm_prot_t protection;
        vm_prot_t max_protection;
        vm_inherit_t inheritance;
        int wired_count;
        vm_pindex_t lastr;
};
```

The fields of *struct vm_map_entry* are as follows:

prev
: Pointer to the previous node in a doubly-linked, circular list.

next
: Pointer to the next node in a doubly-linked, circular list.

left
: Pointer to the left node in a binary search tree.

right Pointer to the right node in a binary search tree.

start Lower address bound of this entry's region.

end Upper address bound of this entry's region.

avail_ssize If the entry is for a process stack, specifies how much the entry can grow.

adj_free The amount of free, unmapped address space adjacent to and immediately following this map entry.

max_free The maximum amount of contiguous free space in this map entry's subtree.

object Pointer to the *struct vm_map_object* with which this entry is associated.

offset Offset within the *object* which is mapped from *start* onwards.

eflags Flags applied to this entry, described below.

The following five members are only valid for entries forming part of a user process's address space:

protection Memory protection bits applied to this region.

max_protection Mask for the memory protection bits which may be actually be applied to this region.

inheritance Contains flags which specify how this entry should be treated during fork processing.

wired_count Count of how many times this entry has been wired into physical memory.

lastr Contains the address of the last read which caused a page fault.

The following flags may be applied to each entry, by specifying them as a mask within the *eflags* member:

MAP_ENTRY_NOSYNC The system should not flush the data associated with this map periodically, but only when it needs to.

MAP_ENTRY_IS_SUB_MAP If set, then the *object* member specifies a subordinate map.

MAP_ENTRY_COW Indicate that this is a copy-on-write region.

MAP_ENTRY_NEEDS_COPY	Indicate that a copy-on-write region needs to be copied.
MAP_ENTRY_NOFAULT	Specifies that accesses within this region should never cause a page fault. If a page fault occurs within this region, the system will panic.
MAP_ENTRY_USER_WIRED	Indicate that this region was wired on behalf of a user process.
MAP_ENTRY_BEHAV_NORMAL	The system should use the default paging behaviour for this region.
MAP_ENTRY_BEHAV_SEQUENTIAL	The system should depress the priority of pages immediately preceding each page within this region when faulted in.
MAP_ENTRY_BEHAV_RANDOM	Is a hint that pages within this region will be accessed randomly, and that prefetching is likely not advantageous.
MAP_ENTRY_IN_TRANSITION	Indicate that wiring or unwiring of an entry is in progress, and that other kernel threads should not attempt to modify fields in the structure.
MAP_ENTRY_NEEDS_WAKEUP	Indicate that there are kernel threads waiting for this region to become available.
MAP_ENTRY_NOCOREDUMP	The region should not be included in a core dump.

The *inheritance* member has type *vm_inherit_t*. This governs the inheritance behaviour for a map entry during fork processing. The following values are defined for *vm_inherit_t*:

VM_INHERIT_SHARE	The object associated with the entry should be cloned and shared with the new map. A new *struct vm_object* will be created if necessary.
VM_INHERIT_COPY	The object associated with the entry should be copied to the new map.
VM_INHERIT_NONE	The entry should not be copied to the new map.
VM_INHERIT_DEFAULT	Specifies the default behaviour, VM_INHERIT_COPY.

The *union vm_map_object* is used to specify the structure which a *struct vm_map_entry* is associated

with.

The fields of *union vm_map_object* are as follows:

```
union vm_map_object {
        struct vm_object *vm_object;
        struct vm_map *sub_map;
};
```

Normally, the *sub_map* member is only used by system maps to indicate that a memory range is managed by a subordinate system map. Within a user process map, each *struct vm_map_entry* is backed by a *struct vm_object*.

SEE ALSO

pmap(9), vm_map_check_protection(9), vm_map_delete(9), vm_map_entry_resize_free(9), vm_map_find(9), vm_map_findspace(9), vm_map_inherit(9), vm_map_init(9), vm_map_insert(9), vm_map_lock(9), vm_map_lookup(9), vm_map_madvise(9), vm_map_max(9), vm_map_min(9), vm_map_pmap(9), vm_map_protect(9), vm_map_remove(9), vm_map_simplify_entry(9), vm_map_stack(9), vm_map_submap(9), vm_map_sync(9), vm_map_wire(9)

AUTHORS

This manual page was written by Bruce M Simpson <*bms@spc.org*>.