

**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:

- |                   |   |
|-------------------|---|
| <i>header</i>     | Head node of a circular, doubly linked list of <i>struct vm_map_entry</i> objects. Each object defines a particular region within this map's address space. |
| <i>lock</i>       | Used to serialize access to the structure.  |
| <i>system_mtx</i> | A mutex which is used if the map is a system map.   |

<i>nentries</i>	A count of the members in use within the circular map entry list.
<i>size</i>	Specifies the size of the virtual address space.
<i>timestamp</i>	Used to determine if the map has changed since its last access.
<i>needs_wakeup</i>	Indicates if a thread is waiting for an allocation within the map. Used only by system maps.
<i>system_map</i>	Set to TRUE to indicate that map is a system map; otherwise, it belongs to a user process.
<i>flags</i>	Map flags, described below.
<i>root</i>	Root node of a binary search tree used for fast lookup of map entries.
<i>pmap</i>	Pointer to the underlying physical map with which this virtual map is associated.
<i>busy</i>	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\_PREFER\_DEFAULT\_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\_size* 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 <i>struct vm_object</i> 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>.