NAME

vm_map_entry_resize_free - vm map free space algorithm

SYNOPSIS

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

void
vm_map_entry_resize_free(*vm_map_t map*, *vm_map_entry_t entry*);

DESCRIPTION

This manual page describes the *vm_map_entry* fields used in the VM map free space algorithm, how to maintain consistency of these variables, and the **vm_map_entry_resize_free**() function.

VM map entries are organized as both a doubly-linked list (*prev* and *next* pointers) and as a binary search tree (*left* and *right* pointers). The search tree is organized as a Sleator and Tarjan splay tree, also known as a "self-adjusting tree".

```
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;
        ...
};
```

The free space algorithm adds two fields to *struct vm_map_entry*: *adj_free* and *max_free*. The *adj_free* field is the amount of free address space adjacent to and immediately following (higher address) the map entry. This field is unused in the map header. Note that *adj_free* depends on the linked list, not the splay tree and that *adj_free* can be computed as:

```
entry->adj_free = (entry->next == &map->header ?
    map->max_offset : entry->next->start) - entry->end;
```

The *max_free* field is the maximum amount of contiguous free space in the entry's subtree. Note that *max_free* depends on the splay tree, not the linked list and that *max_free* is computed by taking the maximum of its own *adj_free* and the *max_free* of its left and right subtrees. Again, *max_free* is unused in the map header.

These fields allow for an **O**(*log n*) implementation of **vm_map_findspace**(). Using *max_free*, we can immediately test for a sufficiently large free region in an entire subtree. This makes it possible to find a first-fit free region of a given size in one pass down the tree, so **O**(*log n*) amortized using splay trees.

When a free region changes size, we must update *adj_free* and *max_free* in the preceding map entry and propagate *max_free* up the tree. This is handled in **vm_map_entry_link**() and **vm_map_entry_unlink**() for the cases of inserting and deleting an entry. Note that **vm_map_entry_link**() updates both the new entry and the previous entry, and that **vm_map_entry_unlink**() updates the previous entry. Also note that *max_free* is not actually propagated up the tree. Instead, that entry is first splayed to the root and then the change is made there. This is a common technique in splay trees and is also how map entries are linked and unlinked into the tree.

The **vm_map_entry_resize_free**() function updates the free space variables in the given *entry* and propagates those values up the tree. This function should be called whenever a map entry is resized in-place, that is, by modifying its *start* or *end* values. Note that if you change *end*, then you should resize that entry, but if you change *start*, then you should resize the previous entry. The map must be locked before calling this function, and again, propagating *max_free* is performed by splaying that entry to the root.

EXAMPLES

Consider adding a map entry with **vm_map_insert**().

```
ret = vm_map_insert(map, object, offset, start, end, prot,
    max_prot, cow);
```

In this case, no further action is required to maintain consistency of the free space variables. The **vm_map_insert**() function calls **vm_map_entry_link**() which updates both the new entry and the previous entry. The same would be true for **vm_map_delete**() and for calling **vm_map_entry_link**() or **vm_map_entry_unlink**() directly.

Now consider resizing an entry in-place without a call to **vm_map_entry_link**() or **vm_map_entry_unlink**().

```
entry->start = new_start;
if (entry->prev != &map->header)
```

```
vm_map_entry_resize_free(map, entry->prev);
```

In this case, resetting *start* changes the amount of free space following the previous entry, so we use **vm_map_entry_resize_free**() to update the previous entry.

Finally, suppose we change an entry's *end* address.

```
entry->end = new_end;
vm_map_entry_resize_free(map, entry);
```

Here, we call **vm_map_entry_resize_free**() on the entry itself.

SEE ALSO

vm_map(9), vm_map_findspace(9)

Daniel D. Sleator and Robert E. Tarjan, "Self-Adjusting Binary Search Trees", *JACM*, vol. 32(3), pp. 652-686, July 1985.

HISTORY

Splay trees were added to the VM map in FreeBSD 5.0, and the **O**(*log n*) tree-based free space algorithm was added in FreeBSD 5.3.

AUTHORS

The tree-based free space algorithm and this manual page were written by Mark W. Krentel <*krentel@dreamscape.com*>.