NAME

rman, **rman_activate_resource**, **rman_adjust_resource**, **rman_deactivate_resource**, **rman_fini**, **rman_init**, **rman_init_from_resource**, **rman_is_region_manager**, **rman_manage_region**, **rman_first_free_region**, **rman_last_free_region**, **rman_release_resource**, **rman_reserve_resource**, **rman_reserve_resource_bound**, **rman_make_alignment_flags**, **rman_get_start**, **rman_get_end**, **rman_get_device**, **rman_get_size**, **rman_get_flags**, **rman_set_mapping**, **rman_get_mapping**, **rman_set_virtual**, **rman_get_virtual**, **rman_set_bustag**, **rman_get_bustag**, **rman_set_bushandle**, **rman_get_bushandle**, **rman_set_rid**, **rman_get_rid** **-** resource management functions

SYNOPSIS

#include <sys/types.h>
#include <sys/rman.h>

int
rman_activate_resource(*struct resource *r*);

int
rman_adjust_resource(*struct resource *r*, *rman_res_t start*, *rman_res_t end*);

int
rman_deactivate_resource(*struct resource *r*);

int
rman_fini(*struct rman *rm*);

int
rman_init(*struct rman *rm*);

int
rman_init_from_resource(*struct rman *rm*, *struct resource *r*);

int
rman_is_region_manager(*struct resource *r*, *struct rman *rm*);

int
rman_manage_region(*struct rman *rm*, *rman_res_t start*, *rman_res_t end*);

int
rman_first_free_region(*struct rman *rm*, *rman_res_t *start*, *rman_res_t *end*);

int
rman_last_free_region(*struct rman *rm*, *rman_res_t *start*, *rman_res_t *end*);

int
rman_release_resource(*struct resource *r*);

*struct resource **
rman_reserve_resource(*struct rman *rm*, *rman_res_t start*, *rman_res_t end*, *rman_res_t count*,
u_int flags, *device_t dev*);

*struct resource **
rman_reserve_resource_bound(*struct rman *rm*, *rman_res_t start*, *rman_res_t end*, *rman_res_t count*,
rman_res_t bound, *u_int flags*, *device_t dev*);

uint32_t
rman_make_alignment_flags(*uint32_t size*);

rman_res_t
rman_get_start(*struct resource *r*);

rman_res_t
rman_get_end(*struct resource *r*);

device_t
rman_get_device(*struct resource *r*);

rman_res_t
rman_get_size(*struct resource *r*);

u_int
rman_get_flags(*struct resource *r*);

void
rman_set_mapping(*struct resource *r*, *struct resource_map *map*);

void
rman_get_mapping(*struct resource *r*, *struct resource_map *map*);

void
rman_set_virtual(*struct resource *r*, *void *v*);

*void **
rman_get_virtual(*struct resource *r*);

void
rman_set_bustag(*struct resource *r*, *bus_space_tag_t t*);

bus_space_tag_t
rman_get_bustag(*struct resource *r*);

void
rman_set_bushandle(*struct resource *r*, *bus_space_handle_t h*);

bus_space_handle_t
rman_get_bushandle(*struct resource *r*);

void
rman_set_rid(*struct resource *r*, *int rid*);

int
rman_get_rid(*struct resource *r*);

DESCRIPTION

The **rman** set of functions provides a flexible resource management abstraction. It is used extensively by the bus management code. It implements the abstractions of region and resource. A region descriptor is used to manage a region; this could be memory or some other form of bus space.

Each region has a set of bounds. Within these bounds, allocated segments may reside. Each segment, termed a resource, has several properties which are represented by a 16-bit flag register, as follows.

```
#define RF_ALLOCATED    0x0001 /* resource has been reserved */
#define RF_ACTIVE       0x0002 /* resource allocation has been activated */
#define RF_SHAREABLE    0x0004 /* resource permits contemporaneous sharing */
#define RF_FIRSTSHARE   0x0020 /* first in sharing list */
#define RF_PREFETCHABLE 0x0040 /* resource is prefetchable */
#define RF_UNMAPPED     0x0100 /* don’t map resource when activating */
```

Bits 15:10 of the flag register are used to represent the desired alignment of the resource within the region.

The **rman_init**() function initializes the region descriptor, pointed to by the *rm* argument, for use with the resource management functions. It is required that the fields *rm_type* and *rm_descr* of *struct rman* be set before calling **rman_init**(). The field *rm_type* shall be set to RMAN_ARRAY. The field *rm_descr* shall be set to a string that describes the resource to be managed. The *rm_start* and *rm_end* fields may be set to limit the range of acceptable resource addresses. If these fields are not set, **rman_init**() will initialize them to allow the entire range of resource addresses. It also initializes any mutexes associated with the structure. If **rman_init**() fails to initialize the mutex, it will return ENOMEM; otherwise it will return 0 and *rm* will be initialized.

The **rman_fini**() function frees any structures associated with the structure pointed to by the *rm* argument. If any of the resources within the managed region have the RF_ALLOCATED flag set, it will return EBUSY; otherwise, any mutexes associated with the structure will be released and destroyed, and the function will return 0.

The **rman_manage_region**() function establishes the concept of a region which is under **rman** control. The *rman* argument points to the region descriptor. The *start* and *end* arguments specify the bounds of the region. If successful, **rman_manage_region**() will return 0. If the region overlaps with an existing region, it will return EBUSY. If any part of the region falls outside of the valid address range for *rm*, it will return EINVAL. ENOMEM will be returned when **rman_manage_region**() failed to allocate memory for the region.

The **rman_init_from_resource**() function is a wrapper routine to create a resource manager backed by an existing resource. It initializes *rm* using **rman_init**() and then adds a region to *rm* corresponding to the address range allocated to *r* via **rman_manage_region**().

The **rman_first_free_region**() and **rman_last_free_region**() functions can be used to query a resource manager for its first (or last) unallocated region. If *rm* contains no free region, these functions will return ENOENT. Otherwise, **start* and **end* are set to the bounds of the free region and zero is returned.

The **rman_reserve_resource_bound**() function is where the bulk of the **rman** logic is located. It attempts to reserve a contiguous range in the specified region *rm* for the use of the device *dev*. The caller can specify the *start* and *end* of an acceptable range, as well as a boundary restriction and required alignment, and the code will attempt to find a free segment which fits. The *start* argument is the lowest acceptable starting value of the resource. The *end* argument is the highest acceptable ending value of the resource. Therefore, *start* + *count* - 1 must be <= *end* for any allocation to happen. The alignment requirement (if any) is specified in *flags*. The *bound* argument may be set to specify a boundary restriction such that an allocated region may cross an address that is a multiple of the boundary. The *bound* argument must be a power of two. It may be set to zero to specify no boundary restriction. A shared segment will be allocated if the RF_SHAREABLE flag is set, otherwise an exclusive segment will be allocated. If this shared segment already exists, the caller has its device added to the list of consumers.

The **rman_reserve_resource**() function is used to reserve resources within a previously established region. It is a simplified interface to **rman_reserve_resource_bound**() which passes 0 for the *bound* argument.

The **rman_make_alignment_flags**() function returns the flag mask corresponding to the desired alignment *size*. This should be used when calling **rman_reserve_resource_bound**().

The **rman_is_region_manager**() function returns true if the allocated resource *r* was allocated from *rm*. Otherwise, it returns false.

The **rman_adjust_resource**() function is used to adjust the reserved address range of an allocated resource to reserve *start* through *end*. It can be used to grow or shrink one or both ends of the resource range. The current implementation does not support entirely relocating the resource and will fail with EINVAL if the new resource range does not overlap the old resource range. If either end of the resource range grows and the new resource range would conflict with another allocated resource, the function will fail with EBUSY. The **rman_adjust_resource**() function does not support adjusting the resource range for shared resources and will fail such attempts with EINVAL. Upon success, the resource *r* will have a start address of *start* and an end address of *end* and the function will return zero. Note that none of the constraints of the original allocation request such as alignment or boundary restrictions are checked by **rman_adjust_resource**(). It is the caller’s responsibility to enforce any such requirements.

The **rman_release_resource**() function releases the reserved resource *r*. It may attempt to merge adjacent free resources.

The **rman_activate_resource**() function marks a resource as active, by setting the RF_ACTIVE flag. If this is a time shared resource, and the caller has not yet acquired the resource, the function returns EBUSY.

The **rman_deactivate_resource**() function marks a resource *r* as inactive, by clearing the RF_ACTIVE flag. If other consumers are waiting for this range, it will wakeup their threads.

The **rman_get_start**(), **rman_get_end**(), **rman_get_size**(), and **rman_get_flags**() functions return the bounds, size and flags of the previously reserved resource *r*.

The **rman_set_bustag**() function associates a *bus_space_tag_t t* with the resource *r*. The **rman_get_bustag**() function is used to retrieve this tag once set.

The **rman_set_bushandle**() function associates a *bus_space_handle_t h* with the resource *r*. The

rman_get_bushandle() function is used to retrieve this handle once set.

The **rman_set_virtual**() function is used to associate a kernel virtual address with a resource *r*. The **rman_get_virtual**() function can be used to retrieve the KVA once set.

The **rman_set_mapping**() function is used to associate a resource mapping with a resource *r*. The mapping must cover the entire resource. Setting a mapping sets the associated bus_space(9) handle and tag for *r* as well as the kernel virtual address if the mapping contains one. These individual values can be retrieved via **rman_get_bushandle**(), **rman_get_bustag**(), and **rman_get_virtual**().

The **rman_get_mapping**() function can be used to retrieve the associated resource mapping once set.

The **rman_set_rid**() function associates a resource identifier with a resource *r*. The **rman_get_rid**() function retrieves this RID.

The **rman_get_device**() function returns a pointer to the device which reserved the resource *r*.

SEE ALSO

bus_activate_resource(9), bus_adjust_resource(9), bus_alloc_resource(9), bus_map_resource(9), bus_release_resource(9), bus_set_resource(9), bus_space(9), mutex(9)

AUTHORS

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