Range allocator

Synopsis

```
typedef void *ralloc_t;

typedef enum
{
     ALLOCATE_ANY,
     ALLOCATE_EXACT,
     ALLOCATE_ABOVE,
     ALLOCATE_BELOW,
} allocation_flags;

typedef uintptr_t vaddr_t;

ralloc_t create_range_allocator(
          vaddr_t base, size_t length, size_t granularity);

void destroy_range_allocator(ralloc_t ralloc);

vaddr_t allocate_range(
          ralloc_t ralloc, size_t length,
          allocation_flags flags, vaddr_t optional_hint);

void free_range(
          ralloc_t ralloc, vaddr_t base, size_t length);
```

Description

For certain subsystems rather than allocating physical memory, we are more interested in effectively managing address spaces. Write a small, simple range allocator conforming to the API above.

The function ***`create_range_allocator()`*** creates, and returns an opaque handle, to a range allocator representing the range ***`[base, base+length)`***. The parameter ***`granularity`*** specifies the required granularity for the allocations: all allocations shall be rounded to a size multiple of the granularity.

The function ***`destroy_range_allocator()`*** frees all control structures associated with the specified range allocator.

The function ***allocate_range()*** must allocate a range of the specified length and return the base address. The allocation flags parameter must be interpreted as follows:

- ***ALLOCATE_ANY*** : allocate in any available address big enough to contain the requested ***length***. The parameter ***optional_hint*** must be ignored.
- ***ALLOCATE_EXACT*** : allocate (if possible) the requested ***length*** exactly at the address specified by ***optional_hint***.
- ***ALLOCATE_ABOVE*** : allocate the requested ***length*** above the address specified by ***optional_hint***.
- ***ALLOCATE_BELOW*** : allocate the requested ***length*** below the address specified by ***optional_hint***. The complete allocated range must reside below the hint, not just the starting address.

If the allocation cannot be satisfied, ***allocate_range()*** shall return ***(vaddr_t)-1***.

Finally, the function ***free_range()*** must release a range (or part of a range) previously allocated.

Submission Format

Answers to this exercise must be submitted as a ready to compile C or C++ file named either ***rangeAllocator.c*** or ***rangeAllocator.cpp***. This file must not contain a ***main()*** function.

Any submission must conform to the following requirements:

- Must be correct in all behaviors.
- Must conform to the API described in the document.
- Ready to compile C (or C++) source code in a file named either ***rangeAllocator.c*** or ***rangeAllocator.cpp***. This file must not contain a ***main()*** function.
- Code must be strictly conforming to either C89, C++98 or C++03 standards.

Optionally (not required but can improve the quality of submission) you can also include, in a subdirectory, the following:

- The test battery they used to verify their own code.

- A ***README*** file stating design, bugs, limitations and/or idiosyncrasies of your implementation.

Your code will be evaluated as follows:

- It will be run against a test battery of our own to have a quick assessment of correctness.
- The same test battery will also perform a quick assessment of the effectiveness of the address space management.
- The code and optional documentation will be reviewed by one of our engineers.