#include <assert.h>

#include <string.h>

#include <sys/types.h>

#include <unistd.h>

void \*malloc(size\_t size) {

void \*p = sbrk(0);

void \*request = sbrk(size);

if (request == (void\*) -1) {

return NULL; // sbrk failed.

} else {

assert(p == request); // Not thread safe.

return p;

}

}

void \*malloc(size\_t size) {

struct block\_meta \*block;

// TODO: align size?

if (size <= 0) {

return NULL;

}

if (!global\_base) { // First call.

block = request\_space(NULL, size);

if (!block) {

return NULL;

}

global\_base = block;

} else {

struct block\_meta \*last = global\_base;

block = find\_free\_block(&last, size);

if (!block) { // Failed to find free block.

block = request\_space(last, size);

if (!block) {

return NULL;

}

} else { // Found free block

// TODO: consider splitting block here.

block->free = 0;

block->magic = 0x77777777;

}

}

return(block+1);

}

void free(void \*ptr) {

if (!ptr) {

return;

}

// TODO: consider merging blocks once splitting blocks is implemented.

struct block\_meta\* block\_ptr = get\_block\_ptr(ptr);

assert(block\_ptr->free == 0);

assert(block\_ptr->magic == 0x77777777 || block\_ptr->magic == 0x12345678);

block\_ptr->free = 1;

block\_ptr->magic = 0x55555555;

}

void \*realloc(void \*ptr, size\_t size) {

if (!ptr) {

// NULL ptr. realloc should act like malloc.

return malloc(size);

}

struct block\_meta\* block\_ptr = get\_block\_ptr(ptr);

if (block\_ptr->size >= size) {

// We have enough space. Could free some once we implement split.

return ptr;

}

// Need to really realloc. Malloc new space and free old space.

// Then copy old data to new space.

void \*new\_ptr;

new\_ptr = malloc(size);

if (!new\_ptr) {

return NULL; // TODO: set errno on failure.

}

memcpy(new\_ptr, ptr, block\_ptr->size);

free(ptr);

return new\_ptr;

}

void \*calloc(size\_t nelem, size\_t elsize) {

size\_t size = nelem \* elsize; // TODO: check for overflow.

void \*ptr = malloc(size);

memset(ptr, 0, size);

return ptr;

}