Additional information

TABLE OF CONTENTS

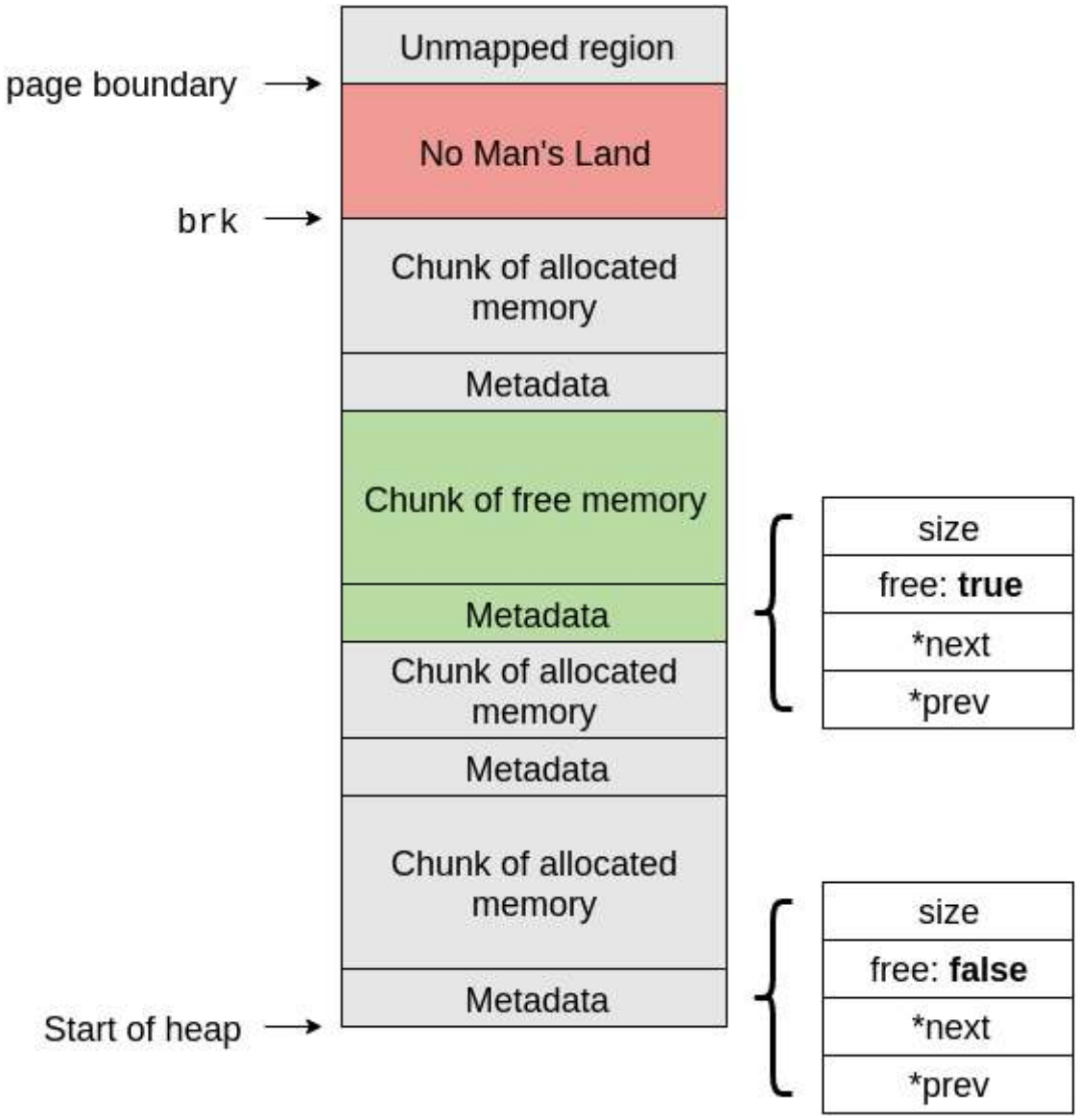
1 Unmapped region and no man's land

2 Resource limits

Unmapped region and no man's land

We saw earlier that the break marks the end of the mapped virtual address space. By this assumption, accessing addresses above the break should trigger an error ("bus error" or "segmentation fault").

The virtual address space is mapped in quanta of pages (usually some multiple of 4096 bytes). When `sbrk` is called, the operating system will have to map more memory to the heap. To do that, it maps an entire page from physical memory to the mapped region of the heap. Now, it is possible that the break doesn't end up exactly on a page boundary. In this situation, what is the status of the memory between the break and the page boundary? It turns out that this memory is accessible, even though it is above the break and thus should be unmapped in theory. Bugs related to this issue are particularly insidious, because no error will occur if you read from or write to this "no man's land."



Resource limits

In Homework 0, you briefly explored the `getrlimit` syscall. In modern operating systems like Linux, processes have limits on their resource usage. For example, the maximum size of the stack and heap are governed by limits on the resources `RLIMIT_STACK` and `RLIMIT_DATA` (see `man 2 getrlimit`). Each of these resources has a *hard limit* and a *soft limit*. A process can raise its own soft limits; the soft limit exists to catch bugs (e.g. resource leaks) early by causing an error if a process uses more resources than expected. The hard limit can only be raised by the superuser (`root`), and exists to prevent resource abuse. For this assignment, do not place an upper bound on the stack size or heap size. You do not need to implement resource limits.
