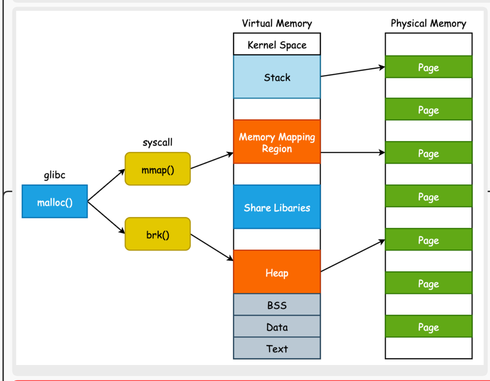
Linux 内存杂记(01)

关于 malloc()

首先需要明确的是，malloc() 是 glibc 一个库函数，并不是系统调用。malloc() 本质上其实就是封装了 brk() 以及 mmap() 等系统调用，同时在内部进行了一些内存管理



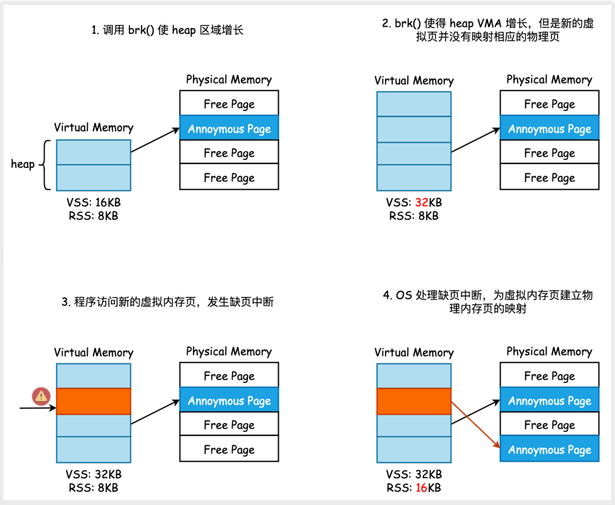
在进程的虚拟内存中，诸如 Stack、Heap 以及 BSS 等区域，我们会称为其为 VMA，即 Virtual Memory Area

VMA 其实就是一段连续的、功能相同的一段虚拟内存

如上图所示，malloc() 的本质就是在其实现内部通过调用 brk() 或者是 mmap() 来改变进程的虚拟内存空间，从而给进程分配虚拟内存的。而虚拟内存到物理内存之间的映射，由操作系统管理，用户进程无法干预此过程

VSS vs RSS

VSS，即 Virtual Set Size，表示进程所使用的虚拟内存大小；RSS，即 Resident Set Size，即进程所使用的物理内存大小。进程的 VSS 和 RSS 可以通过 top/htop 命令进行查看，对应结果中的 VIRT 以及 RES



因此，我们一定要明确，使用 malloc() 申请虚拟内存，并不代表着物理内存的分配，只有应用程序访问了申请的虚拟内存区域时，操作系统才会为进程分配物理内存页

Experiment

我们可以通过将 overcommit_memory 设置为 1 来允许进程申请不超过虚拟内存大小的内存，从而观察内存申请与内存分配之间的区别

```
sudo swapoff -a
sudo sh -c 'echo 1 >/proc/sys/vm/overcommit_memory'
```

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    for (int i = 0; i < 200000; i++) {
        char *p = malloc(1024 * 1024 * 1024);
        if (p == NULL) {
            printf("malloc failed! \n");
            exit(1);
        }
        else {
            printf("malloc success: %d \n", i);
        }
    }
}
```

运行上面的简易程序就会发现，我们使用 malloc() 能够成功申请的内存数量要远超于物理内存的大小

虚拟内存和物理内存之间的映射，本质上就是一种 Lazy Load，或者说懒加载，对提高系统性能和负载有着至关重要的作用

关于 overcommit_memory

Memory Overcommit 表示操作系统承诺给进程的内存大小超过了实际可用的物理内存，注意是物理内存。而 overcommit_memory 这一内核参数决定了操作系统如何对待 Memory Overcommit

Value	含义
0	内核允许合理的 overcommit，但禁止明目张胆的 overcommit。单次申请的内存总和不允许超过 [free memory + free swap + pagecache size + SLAB 可回收部分]
1	允许随意的 overcommit
2	禁止 overcommit，即进程不允许申请超过物理内存总量的虚拟内存。部署数据库应用的机器通常采用该值，以避免 OOM 导致 DB 进程被内核 KILL

当 overcommit_memory = 2 时，可通过设置 overcommit_ratio 来设置可申请内存的阈值

MemoryLimit = (Physical RAM * vm.overcommit_ratio / 100) + Swap

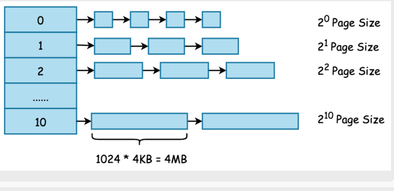
同时，在 glibc 中，malloc/free 也是使用内存池实现的，也就是说，每次 malloc/free 并不会对应着一次系统调用，glibc 很可能攒够了一波内存之后才将其归还给内核

页框管理

Buddy

Linux 内核采用 Buddy，即伙伴算法来管理页框，以满足快速内存申请的需求

Buddy 算法本身并不复杂，内存区中的页框被组合成一些内存块，每个内存块包含 2^k 个连续的页框，这些不同大小的内存块使用一个数组保存，并且相同大小的内存块之间采用链表进行连接



如上图所示，对于数组中的第 0 个元素来说，保存了由 2^0 = 1 个连续页框组成的不同内存块。而对于第 2 个元素来说，链表中的内存块大小为 2^2 = 4 个 Page，一个 Page 的大小为 4096KB

因此，当数组的长度为 10 时，最大的连续内存块即为 1024 个连续页框，组成大小为 4MB 的连续内存

可通过 /proc/buddyinfo 来查看当前系统的 buddy 数组情况

```
smart@stable:~/workspace$ cat /proc/buddyinfo
Node 0, zone      DMA      1      0      0      0      2      1      1      0      1      1      3
Node 0, zone    DMA32      2      2      2      1      1      3      2      2      2      3    745
Node 0, zone   Normal    502    363    229     69    147     73     27     15      5      6   1856
```

通常我们只需要关注 Normal 所在的那一行，以 1856 为例，这表示系统中存在 1856 个由 1024 个页框所组成的连续内存块，它们总计占用了约 7G 的内存空间

Slab

slab 中文可以翻译为厚块、厚板，但是相对于 Buddy 数组中的内存块而言，slab 应该被认为是"薄板"

Buddy 内存分配的粒度还是比较大的，最小的连续内存为一个页框，即 4KB。但是我们有时需要分配更小的内存，拿着一页内存来用就太浪费了，因此就有了 slab，把 Buddy 中的一块内存继续切割成多个小对象，供内核进行使用