实验 4《内存管理》讲义

1 实验 4 说明

实验名：内存管理。

本实验是系列实验中的第四个。

1.1 实验 4 基础

本实验在实验 3 的基础上进行。

在实验 3 提交的截止时间过后，同学们可以就实验 3 的内容互通有无。

实验 4 可以在其他学实验 3 的基础上进行：

- 无论你使用哪一个（包括自己的），请在实验报告中标注，实验 3 的基础来自哪个同学（可以是自己）；
- 给你使用的实验 3 打分。
- 也可以参考助教提供的实验 3（或实验 4）框架，这个框架已经帮同学们完成了较为关键的部分代码。若直接使用了助教提供的框架，请说明。

2 实验 4 目的

实现一个具有内存管理功能的 OS。

3 实验 4 要求

- 【必须】内存检测，确定动态内存的范围
- 【必须】提供动态分区管理机制 dPartition
- 【必须】提供等大小固定分区管理机制 ePartition
- 【必须】使用动态分区管理机制来管理所有动态内存
- 【必须】提供 kmalloc/kfree 和 malloc/free 两套接口，分别提供给内核和用户
- 【可选】kmalloc/kfree、malloc/free 相互隔离
- 【必须】自测
  - shell 改用 malloc/free 来动态注册新命令（将涉及部分字符串处理)
  - 自编测试用例，并添加到 shell 命令列表中，执行新增的 shell 命令

4 实验 4 准备（预备知识和准备工作）

4.1 关于内存布局

X86 的内存布局中，1MB 以内的略微复杂，除了常规内存之外，还包括 VGA 显存、BIOS 等，本实验跳过不计，仅使用 1M 以上的内存作为可用内存。若有同学希望将 1MB 以内的空间管理起来，也是可以的，这个作为可选项。

假设只有 1 块连续的内存空间（从 1MB 开始），我们将操作系统内核代码（包括数据等）放在从 1MB 开始的内存中（即低端，静态分配）。此外，还要考虑到我们在何处进行了其他静态分配。例如，最初的栈，我们放在了哪里？建议静态分配的内存，顺着内核代码、数据往后排，以留出一整块初始的动态内存。

此外，有的平台上，可能会有非连续的若干块内存空间。这种情况，本实验不做要求。若有同学希望能够管理非连续的多块内存空间，也是可以的，这个作为可选项。

提示：若因静态分配的缘故，或者考虑 1MB 以内的可用空间，或者考虑其他非连续的多块内存空间，等等，导致动态内存分散多处，可以考虑的解决方案有多种。其中一种思路，可以把不可用的内存，当作已经（永久）分配出去的内存。这样无需另外设计相关的数据结构，而只需要使用动态内存管理相关的数据结构即可。

4.2 关于静态分配和动态分配

静态分配与动态分配相对应。动态分配管理的内存空间，我们称其为动态内存。所谓动态分配，主要是指通过我们即将提供的 malloc 等接口对内存进行分配管理，我们还可以通过 free 等接口回收内存。这样，通过动态分配/回收内存，可以复用内存空间。（注意，栈上的分配也是一种动态分配，但这不是 OS 设计要考虑的，此处忽略。）

静态分配的内存，一般不纳入动态内存管理。比如内核的代码、数据等，它们所占用的存储空间，在编译链接的过程中确定，这部分被占用的空间，一般不回收。也是一般所谓的“操作系统代码常驻内存”的意思。操作系统的代码和数据所占用的内存空间，可以结合链接描述文件来得知。我们在链接描述文件中，定义了几个符号，通过这几个符号，可以得到这部分静态分配的内存的位置。

除此之外，在操作系统启动过程中，一开始，由于动态内存分配机制尚未完成初始化，或者其他的一些原因，操作系统设计者可能也会将某些内存用作某个固定用途，例如我们的第一个栈。

在为动态内存进行初始化的时候，设计者要注意避开静态分配的内存，以免发生访问冲

突。

4.3 简单的内存检测

我们通过简单的内存检测来试探可用内存。

检测的方法很简单，即核验写入的数据与读出的数据是否一致，即先写后读，如果一致就认为内存可用。

但要注意一点，如果我们检测的内存中，涉及到已用内存，那么直接先写后读，会导致写覆盖，因此要先缓存原有数据，然后检测，再检测后还需要写回事先缓存的数据。

另外，如果以字节为单位，检测每一个存储单元，可能过于浪费时间。可以考虑合适的步长，每步只抽检其中的一部分。例如，我们可以以 4KB 为单位，仅对 4KB 的头尾各两个字节进行检测，若两头都可以正确读写，即认为整个 4KB 均可读写。注意，这里选择的步长，建议最好是 2 的幂。

先读后写时，常见的验证内容为 0xAA55 和 0x55AA，这样可以覆盖每个 bit 的“0”“1”取值。

4.4 内存管理算法

4.4.1 等大小

等大小内存块管理，适用于需要集中管理某种对象/数据结构/缓冲区的场景。相当于 Linux 中的 slab 分配算法。我们可以实现一个简单的。

许多模块有等大小内存块的管理需求，例如某种频繁分配回收的对象/数据结构/缓冲区等等。有时候，我们在设计某个功能模块的时候，可能会静态分配一个数组，然后将这个数组组织为一个空闲链表，再为这个空闲链表提供分配/回收的接口，以便从其中动态分配和回收相关的数据结构，例如 TCB 的管理，由于 shell 命令的数据结构的管理。如果把这种管理加以抽象，就可以得到一种通用的内存管理算法，其实就是等大小内存块的管理。

假设我们要管理 TCB 这种数据结构。假设我们最多允许创建 100 个 TCB。那么我们可以首先申请一块较大的内存空间，这个内存空间能容纳 100 个 TCB（以及因管理这 100 个 TCB 而引入的一部分内存开销）。然后我们将等大小内存块的管理算法作用其上，当我们要分配一个 TCB 的时候，就调用相应的分配接口，而回收的时候，也调用相应的回收接口即可。

除了按数据结构，也可以直接按大小。例如操作系统中可能多处申请各种不同大小的缓冲区，或者为字符串申请，或者其他。在 Linux 中，就基于 slab 提供了专用 cache 和按大小的 cache。（注意，这个 cache 不是 CPU 中的 cache，而是 Linux 中对此功能的命名。）

4.4.2 不等大小（即动态分区管理）

总体上，操作系统各处有不同的内存块需求，它们的大小不等。类似于进程对动态内存的需求，在进程的地址空间中，堆和堆的管理算法，用于满足此类需求。我们可以实现一个简单一些的算法。算法的考虑，同动态分区管理算法类似。

建议整个系统的动态内存，分成两部分，一部分专门用于内核，剩下部分专门用于用户（相当于进程的堆），这两部分都可以使用动态分区管理算法进行管理。

内核用和用户用动态内存之间划分的关键，是如何确定内核用动态内存的大小。例如在 RTEMS 操作系统中，内核用的所有动态内存的大小都是可配置可计算的。我们的操作系统中，不做此要求。只需要大约估一个大小就可以，建议这个大小稍大一些，能满足内核的需要。万一空间不足，可以考虑报错，这样在调试的时候可以通过报错信息来调整内核用动态内存大小。

4.4.3 关于着色

在 Linux 的 slab 管理算法中有着色的概念。主要是为性能考虑。我们不做此要求。

4.4.4 关于隔离带

建议为内存块之间增加隔离带，以避免小范围的溢出带来的问题。调试过程中，若发现内存的内容被莫名奇妙修改了，可以考虑是否可能是内存溢出造成的。查小范围内存溢出的一种方法是，在隔离带内填写特殊的字符序列（例如全写 0x12345678，或者全 0/1，或者全 A 等等），并通过调试工具观察内存（甚至可以编写一段代码来检测隔离带中的内容），若隔离带中的内容被修改，即可判断内存溢出。

5 实验 4 内容

本次实验所涉及的大部分为课程所讲内容，重点是如何实现这些算法。 本实验所有需要实现的函数均使用 TODO 标出（使用 VSCode 中的 Todo Tree 插件可以清晰查看所有 TODO 任务）

完成实验的步骤推荐：

首先从流程上弄清楚从 OS 内核启动到进入 userAPP ，本次实验相较于上一次在流程上多做了什么？（如 osStart.c 中新增的 pMemInit(); 它背后又调用了哪些函数，实现了什么功能？）

其次从模块上弄清楚这次实验相较于上次新加了什么？新加的模块如何和原来的 OS 内核结合（即接口是什么，在哪里？）

在上面两个宏观层面搞明白后进入局部模块，即我们要完成的 kernel/mem ，首先还是从宏观层面理解每个函数的作用是什么？函数之间的依赖是什么？函数如何一步步封装，最后形成 malloc/free 接口的。

在这之后就可以开始着手实现每个函数了

一个易混淆的点：

可用内存检测是在检测可正常读写的内存大小，而不是空闲内存大小

几个难点：

对于空闲链表的理解以及它如何运作

释放内存后相邻空闲块的合并

一个提醒：

对于 KMEM 的实现，实际上就是用实现的内存管理算法去管理另一块内存空间。和 UMEM 实现大同小异

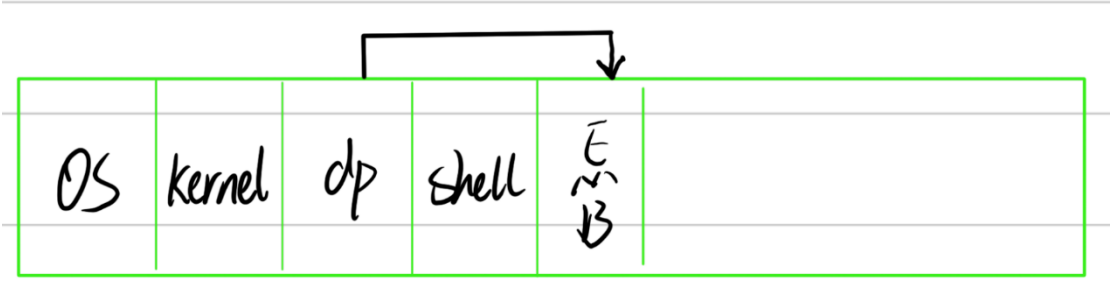
完成这些函数后，启动 shell 在其中运行 userApp/memTestCase.c 中 memTestCaseInit 函数新增的那些命令，进而实现测试。

6 实验 4 示例

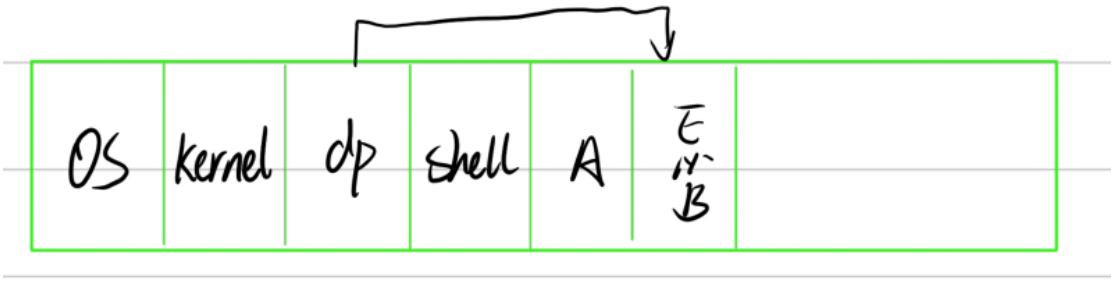
下面以 memTestCase.c 中 testdP2 为例写一下动态内存分配的流程

该过程依次分配 A，B，C，然后依次释放 A，B，C。

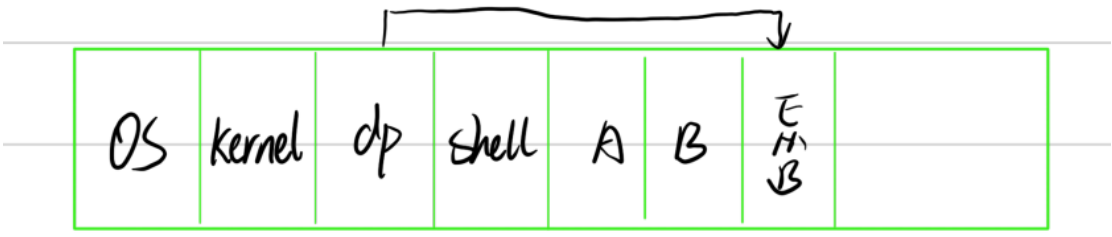
首先在实现 KMEM 情况下，在 OS 之后会分配 kernel 内存空间，然后对内存初始化，产生 dp 结构体和第 一个 EMB 结构体，调用 addNewCmd 函数之后会在 user 内核中占据一部分位置存储 shell 命令



申请 A 空间

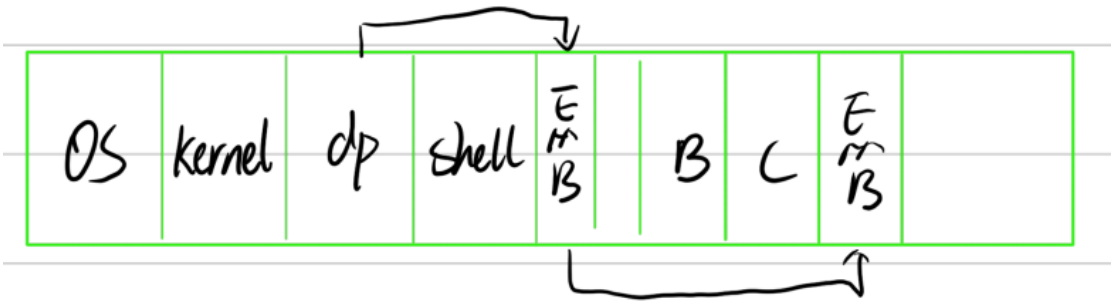


申请 B 空间

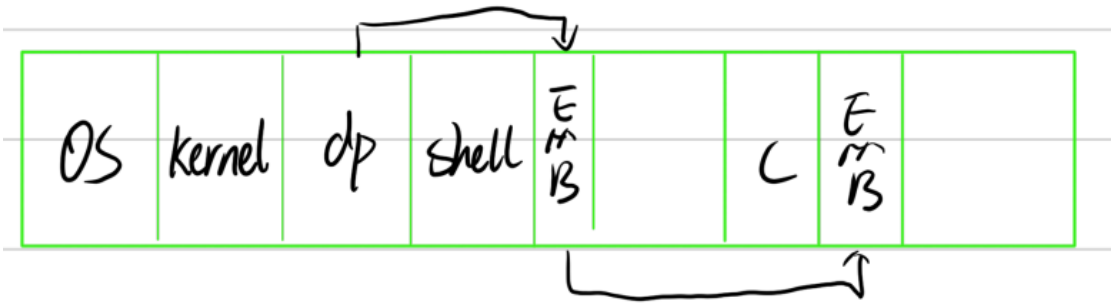


申请 C 空间

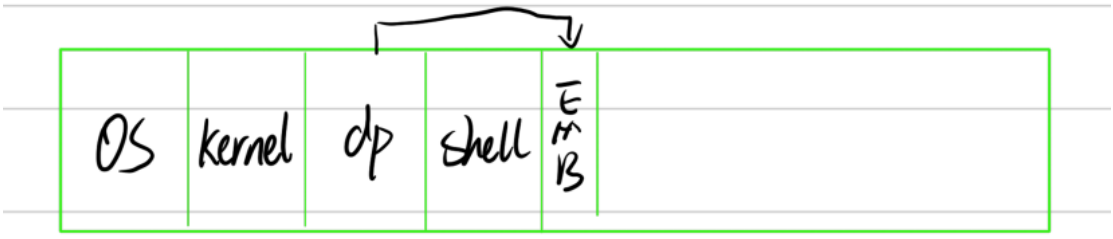
释放 A 空间（产生两个 EMB）



释放 B 空间（空闲块合并）



释放 C 空间



7 实验 4 提交要求

截止日期为 2024.6.21 晚 23.59 分

若要实现 KMEM 和 UMEM 分别管理，需要相应的新增 kmalloc.c 文件，在 memTestCase.h 中自己编写相应的测试。实现 KMEN 和 UMEM 会有额外十分的奖励

实验报告中需要说清楚运行 memTestCaseInit 那些新增的 shell 命令，会出现什么结果，即打印出什么信息（截图放到报告中）？是否符合你的预期，为什么会出现这样的结果。（详细地讲一两个运行结果，大同小异的 可以从简）

其他实验报告要求从简，提交格式要求同上次实验