**Report for Lab07**

**Stu：金泽文**

**No.PB15111604**

**实验目的：**

在实验6的基础上，通过实现动态内存管理，固定内存管理，以及内存有效性检查，学习并理解操作系统内存管理的原理与细节。

**实验内容：**

1. 检测物理内存：实现内存有效性检查方案以检查你所管理的动态内存。
   * 为简单起见，允许直接从1MB开始。
   * 接口：pMemStart（？=1MB）和pMemSize。
   * 操作系统初始化过程中进行检查，并汇报pMemStart和pMemSize的值（要求16进制表示）
   * 附加（非强制要求）：从0地址开始。
2. 实现动态分区算法：dPartition
   * 给定任意大小的内存区域，按照动态分区算法对其进行初始化。接口：dPartitionInit()。（要求：若内存区域太小，小于最小大小，进行失败提示）
   * 从动态分区中分配一个指定大小的分区（允许进行4字节、8字节或你自定义的某个大小对齐）。接口：dPartitionAlloc()
   * 释放一个分区。接口：dPartitionFree()
   * 将检测到的动态物理内存，按照动态分区的方式进行管理
   * 重新封装动态物理内存的分配和回收接口，提供malloc和free接口给userApp。
3. 实现等大小固定分区算法：eFPartition
   * 给定分区大小和分区个数，结合你的内部管理数据结构开销，计算出总大小。接口：eFPartitionTotalSize()
   * 将给定内存区域初始化为若干个等大小固定分区。初始化接口：eFPartitionInit()
   * 分配一个固定大小分区。接口：eFPartitionAlloc()
   * 释放一个固定大小分区。接口：eFPartitionFree()
   * 修改你的任务池分配算法：从动态物理内存中，分配一个动态分区，该动态分区能容纳指定个数的任务和内部数据管理开销，使用等大小固定分区算法管理这个动态分区。
4. 修改osStart原语，以增加内存管理功能

 实验报告内容

**实验分析与实验过程：**

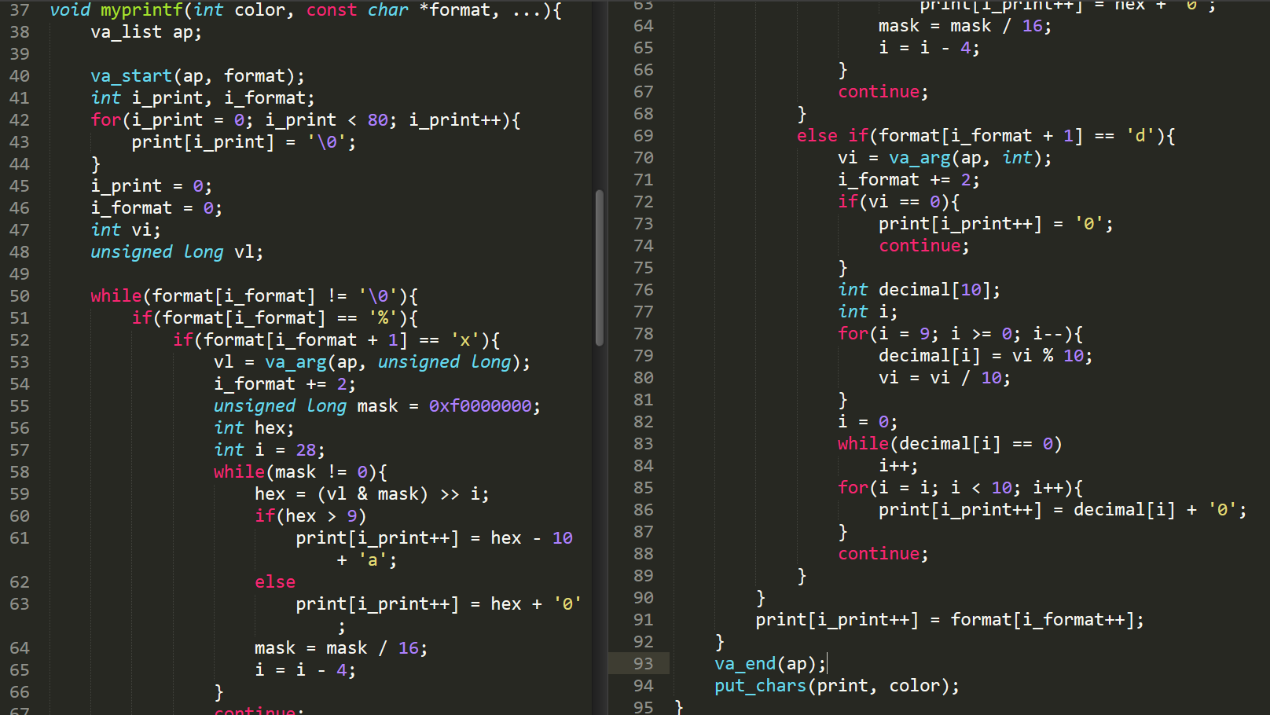
本次实验重点在于实现。

**内存有效性检查**：

首先是内存有效性检查。实现方式很简答， 因为老师上课说过实现方法：依次写入并读取内存，如果发现写入值与读取值不同，则认为内存无效。另外值得一提的是，我额外多写入，读取了另一个值，以防之前的读写一致是偶然所致。

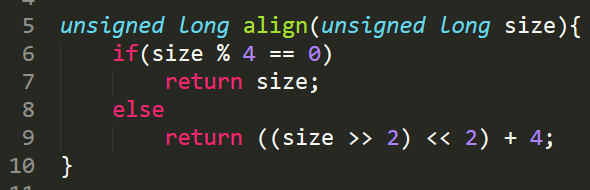
在os开头调用，并且输出对应结果。

**函数myprintf**：

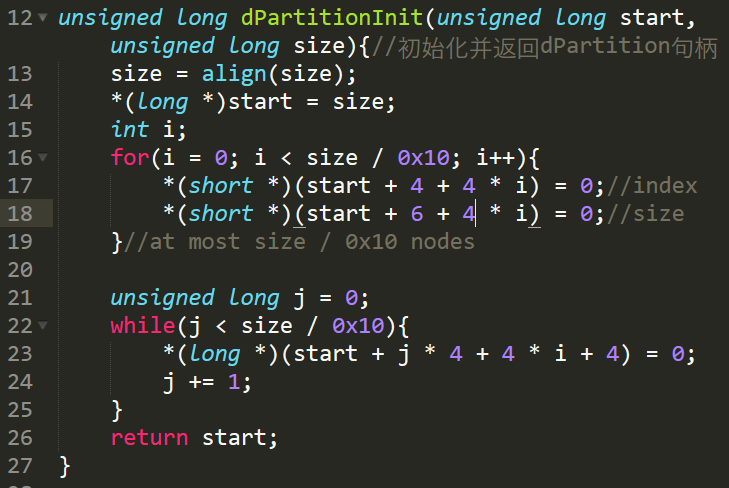
为了实现可变长参数的函数，使用stdarg头文件，通过[维基百科-stdarg.h](https://zh.wikipedia.org/wiki/Stdarg.h)中的阐述，学会实现方式。值得一提的是，实现myprintf过程中，我是用的put\_char和put\_chars函数移植于陈老师之前代码的思路，并且有较大改动。另外，为了将更多的精力放在实验上，我实现的myprintf只支持\n \r %d %x的特殊字符。

**align**：

为了实现对齐而设计的函数。



**dPartitionInit**：

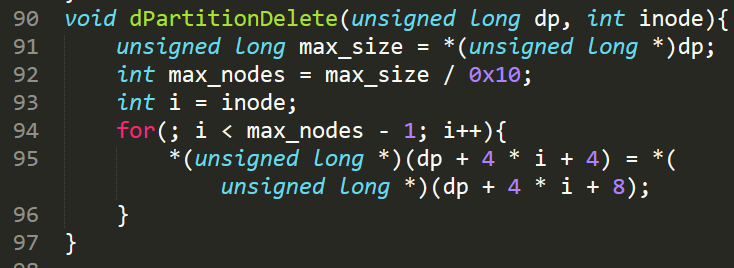
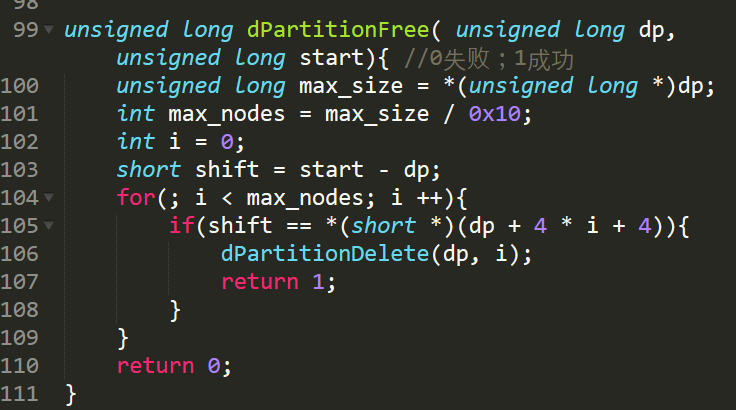
 我采用的不是链表或者数组的实现方式，而是针对每一个partition，在前面分出一head。第一个32位存放这个partition的大小。之后根据这个大小确定最多的节点个数，位size/0x10，这样做能保证节点个数动态变化。之后的size/0x10个32位内存每个存放一个页内偏移index。

**dPartitionAlloc ：**

 为了实现这个函数，额外写了dPartitionInsert 函数，用来动态插入的。

主要是根据**dPartitionInit** 中所设计的表头，依次查看，按照first fit算法分配内存。需要注意块头前面那些index信息要实时更新，保证中间没有空index。

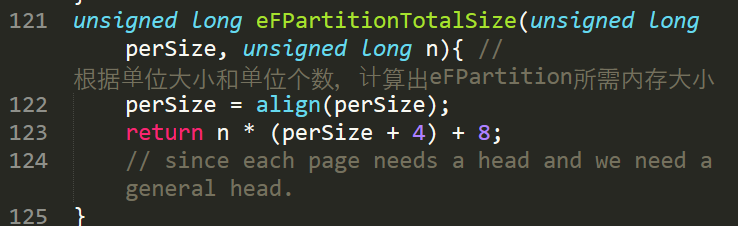
**dPartitionFree** ：

为了实现这个函数，额外写了dPartitionDelete函数，用来动态删除。

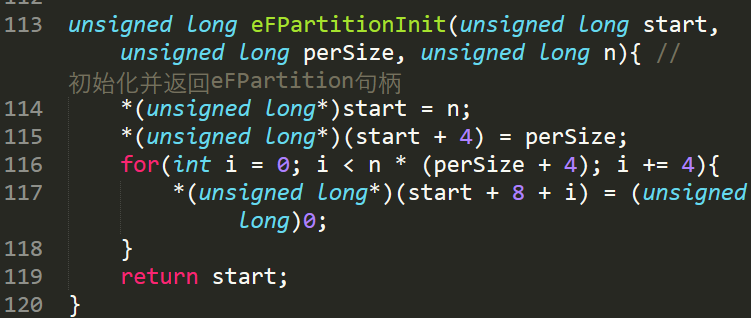
**eFPartitionTotalSize**：

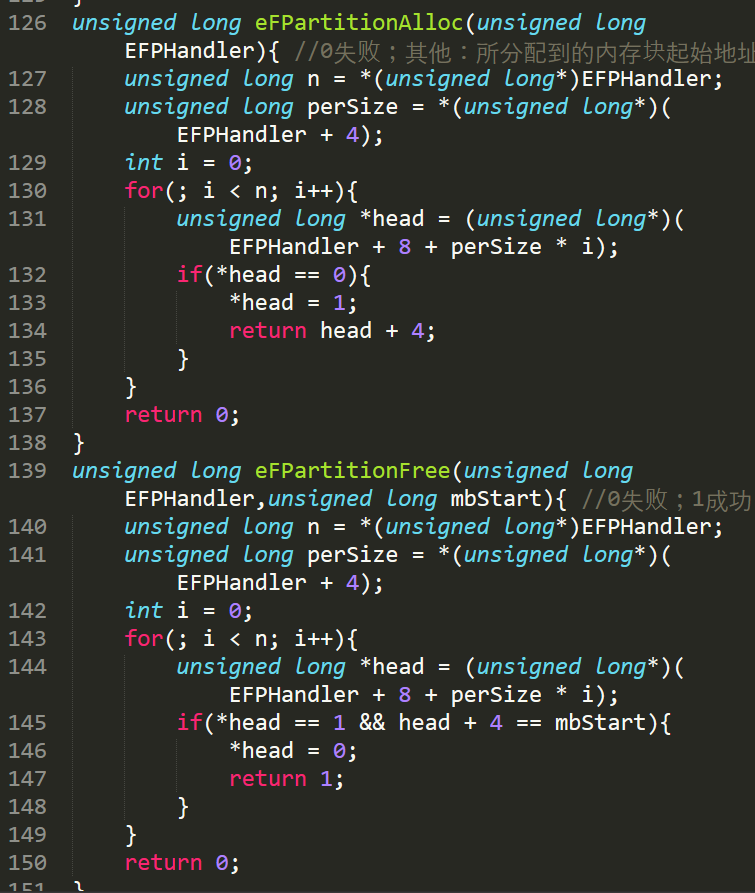
为了保存节点头信息，需要为partition设置表头，保存n和perSize。

另外为了节点标记，所以每个块需要弄一个标志位。所以，最后的totalsize就是

n \* (perSize + 4) + 8。另外要注意对齐。

**eFPartitionInit**：

 按照**eFPartitionTotalSize**中所说的实现。

**eFPartitionAlloc**和**eFPartitionFree**：  
由于分配时只需要一个参数，所以第一个空闲的就是要分配的，无须比较size。Free时依次比较地址是否相等。

**数据结构的接口，设计说明**：

由于内存的管理都是在每个partition头部设置对应的参数，动态管理对应size和索引表，固定管理对应perSize和n。详细地说一下。

**动态**管理数据结构：（以0x1000的dp，0x40的size为例）

0x1000（保存size）-- 0x40

0x1004（开始保存索引表）-- index-0 （保存对应偏移量。）

0x1006（这一部分short保存size）--index-0-size

0x1008— index-1 （保存对应偏移量。）

0x100a（这一部分short保存size）--index-1-size

0x100c— index-2

0x100e（这一部分short保存size）--index-2-size

0x1010— index-3

0x1012（这一部分short保存size）--index-3-size

（索引表到这结束，因为最多0x40/0x10 = 4个）

0x1014-- index-0 （偏移0x14）

**固定**管理数据结构：

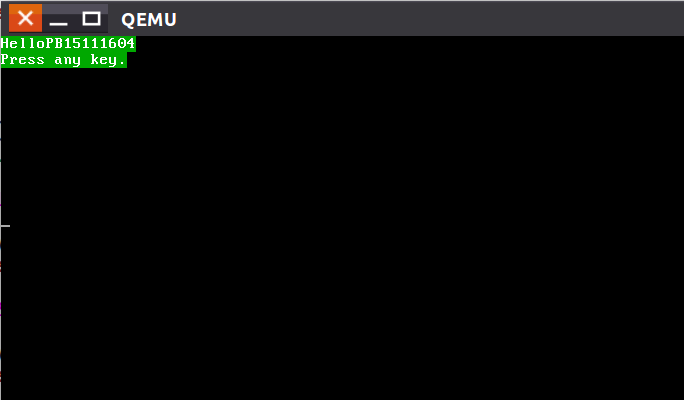
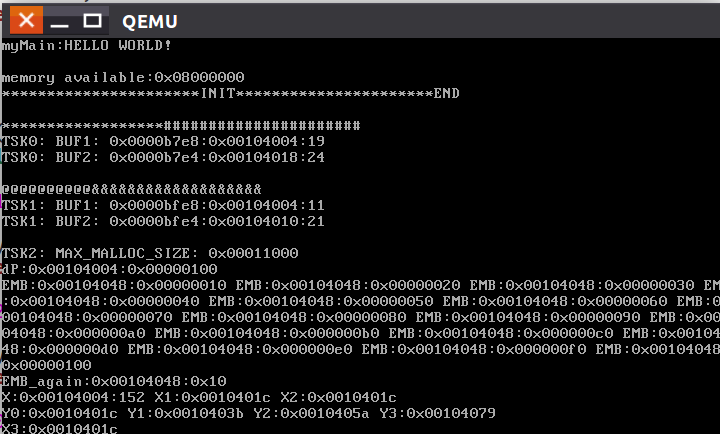
这一部分较为简单，大致说一下。

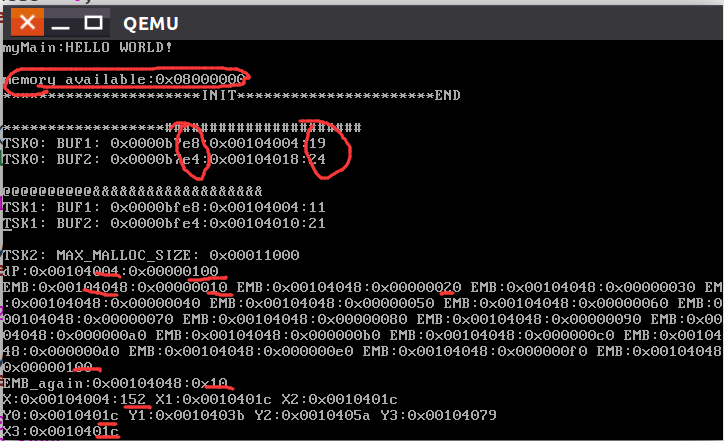
开头第一个32位保存块数n，第二个32位保存每个块对应的大小（对齐之后）。之后每一个块第一个32位保存标志，0表示这个块是available的，1表示allocated。

**原语的接口、设计说明**：

这一部分上面针对每个函数都有详尽的说明。

**测试与运行：**



 可以看到，有内存有效性检查输出，以及各个tsk对应的正确输出。