



本科生实验报告

实验课程: 操作系统实验

实验名称: 第六章内存管理

专业名称: 信息与计算科学

学生姓名: 张文沁

学生学号: 20337268

实验地点:

实验成绩:

报告时间: 2022.5.30

第六章内存管理

仰之弥高，钻之弥坚。瞻之在前，忽焉在后。

1. 实验概述

在本次实验中，我们首先学习如何使用位图和地址池来管理资源。然后，我们将实现在物理地址空间下的内存管理。接着，我们将会学习并开启二级分页机制。在开启分页机制后，我们将实现在虚拟地址空间下的内存管理。

本次实验最精彩的地方在于分页机制。基于分页机制，我们可以将连续的虚拟地址空间映射到不连续的物理地址空间。同时，对于同一个虚拟地址，在不同的页目录表和页表下，我们会得到不同的物理地址。这为实现虚拟地址空间的隔离奠定了基础。但是，本实验最令人困惑的地方也在于分页机制。开启了分页机制后，程序中使用的地址是虚拟地址。我们需要结合页目录表和页表才能确定虚拟地址对应的物理地址。而我们常常会忘记这一点，导致了我们的不知道某些虚拟地址表示的具体含义。

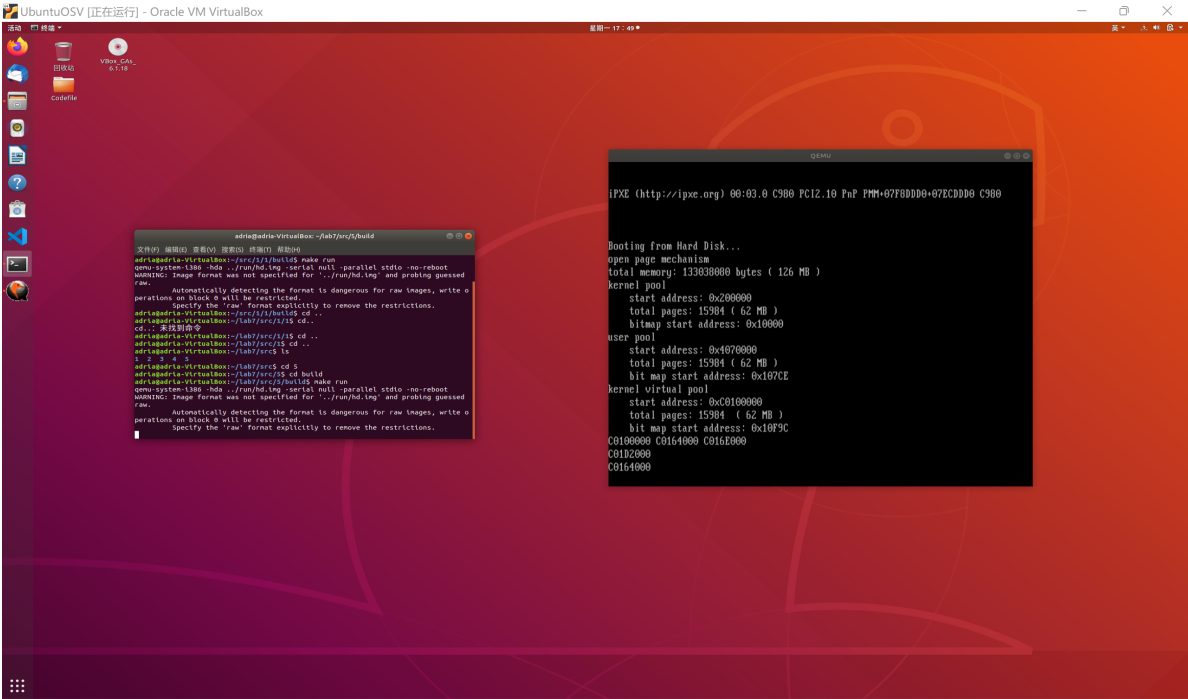
实验要求

1. 实验不限语言，C/C++/Rust都可以。
2. 实验不限平台，Windows、Linux和MacOS等都可以。
3. 实验不限CPU，ARM/Intel/Risc-V都可以

Assignment 1

复现参考代码，实现二级分页机制，并能够在虚拟机地址空间中进行内存管理，包括内存的申请和释放等，截图并给出过程解释。

代码复现结果如下：



```
QEMU

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDD0+07ECDDD0 C980

Booting from Hard Disk...
open page mechanism
total memory: 133038080 bytes ( 126 MB )
kernel pool
  start address: 0x200000
  total pages: 15984 ( 62 MB )
  bitmap start address: 0x10000
user pool
  start address: 0x4070000
  total pages: 15984 ( 62 MB )
  bit map start address: 0x107CE
kernel virtual pool
  start address: 0xC0100000
  total pages: 15984 ( 62 MB )
  bit map start address: 0x10F9C
C0100000 C0164000 C016E000
C01D2000
C0164000
```

解释：

分析：在二级分页机制中，内存的申请步骤分一下三步进行：

- 从虚拟地址池中分配对应的连续的虚拟页。
- 尝试从物理地址池中分配对应多个页，
不同于虚拟页的分配，物理页是一个一个分配并且不需要连续。如果物理页的数量不够分配时，则需要返回-1并且将已经分配好的物理页和对应的虚拟页内存全部释放，以避免内存泄漏
- 建立虚拟页和物理页之间的映射关系。
- 根据上图的结果，输出的地址全是虚拟地址。可以发现，内核的虚拟地址池的起始地址是0xC0100000。因为初始并无分配内存，所以第一个分配内存页的p1的虚拟地址的起始地址就是虚拟地址池的起始地址。而由于p1申请了100页，所以p2对应的起始地址就是在P1的基础上加上100个页表项的大小，如图所示，P3同上。如此，内存的申请变完成了。
而内存的释放同申请差不多：先释放物理页再释放对应的虚拟页。

Assignment 2

参照理论课上的学习的物理内存分配算法如first-fit, best-fit等实现动态分区算法等，或者自行提出自己的算法。

解释：

Assignment2要实现的是动态的内存分配，其原理属于连续的内存分配，具体实现算法有：

- 首次适应（又分从头检索的和从上次位置检索）
- 最佳适应
- 最差适应
- 三种算法的差别在于：首次适应是按照查找到的第一个足够大的空间分配的；而最佳适应是查找到的能满足要求的最小空间；而最差适应则是跟最佳适应相反，从大空间优先分配。
- 在本次Assignment2中，实现了最佳适应
- 但是src/3中已经实现了页这一结构，所以在不想破坏底层结构的情况下只能利用页结构实现动态内存分配。这种方式跟原本的动态内存分配相比在结构上多了内部碎片（动态内存分配原本是连续的，所以没内部碎片，但有外部碎片），但是跟分页比又多了连续性，在分配内存的时候的页是连续分配的。

代码修改如下：

原代码：从头检查直到发现手段满足要求的空间便分配，首次适应算法

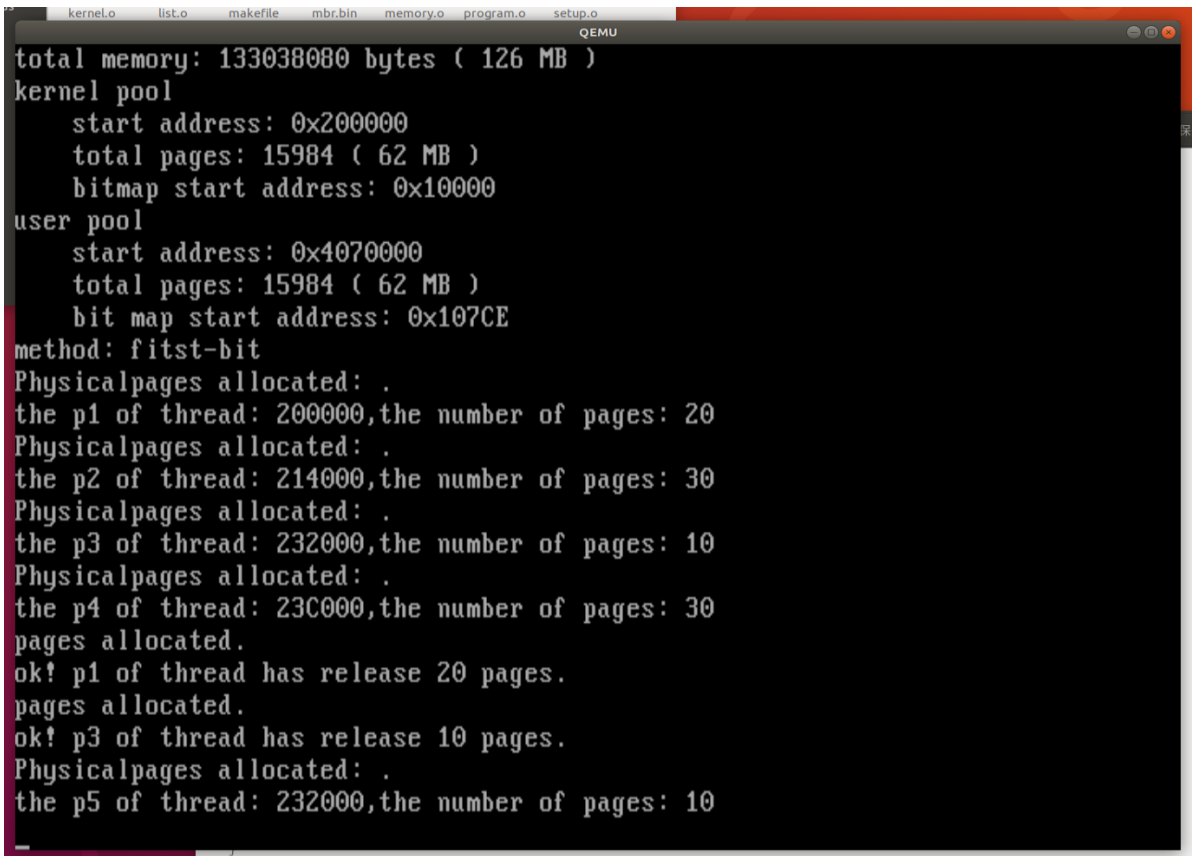
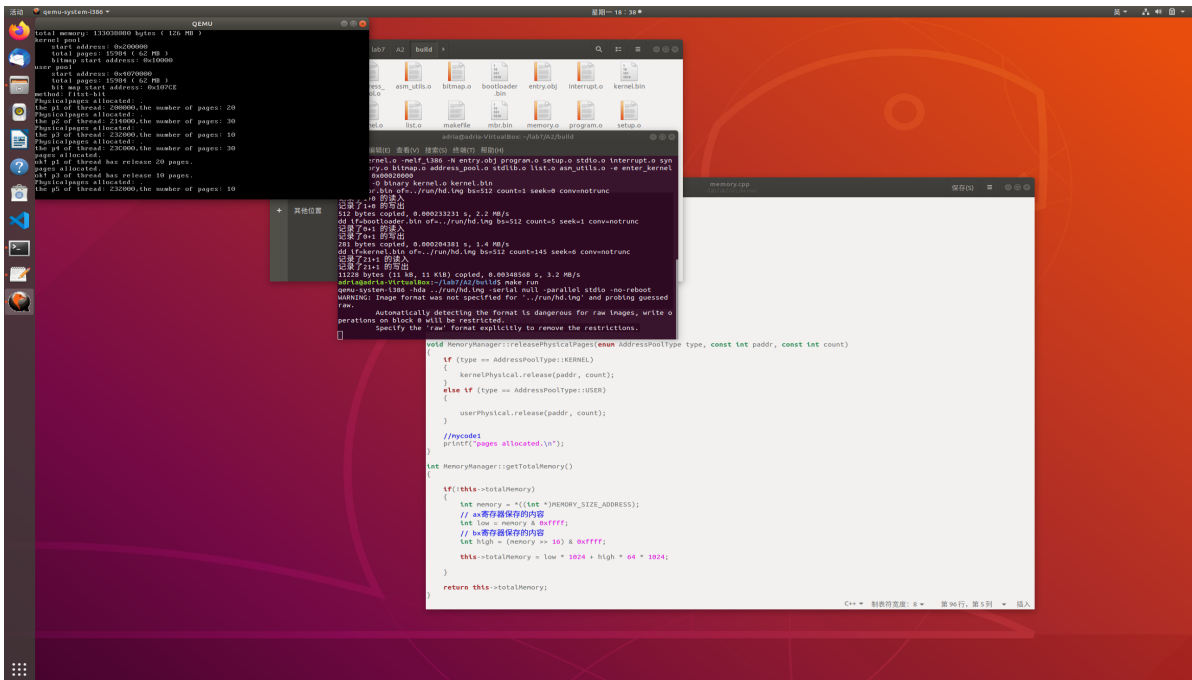
修改代码：设置两个变量记录最小的满足要求的空间的起点和大小，即检测整个区间大小，在遍历完之后才会进行分配。

```
// 检查是否存在从index开始的连续count个资源
empty = 0;
start = index;
while ((index < length) && (!get(index)))//&& (empty < count)
{
    ++empty;
    ++index;
}
if(empty>=count && min > empty)
{
    min=empty;
    min_s=start;
}
if(empty>=count && (Is_change==0))
{
    Is_change=1;
    min_s=start;
}
/*
while ((index < length) && (!get(index)) && (empty < count))
{
    ++empty;
    ++index;
}

// 存在连续的count个资源
if (empty == count)
{
    for (int i = 0; i < count; ++i)
    {
        set(start + i, true);
    }

    return start;
}
*/
```

运行结果如下：



结果解释：

一共进行了四次申请，P1 20页,P2 30页,p3 10页,P4 30页，释放P1，P3，再申请10页。如果是首次适应算法，P1对应的20页内存空间大于10，则第五次申请的起始位置应该是P1的起始地址，如果是最佳适应算法则因为P3是10页，所以第五次的起始位置会是P3的起始位置。如图所示，符合。

Assignment 3

参照理论课上虚拟内存管理的页面置换算法如FIFO、LRU等，实现页面置换，也可以提出自己的算法。

实现解释：

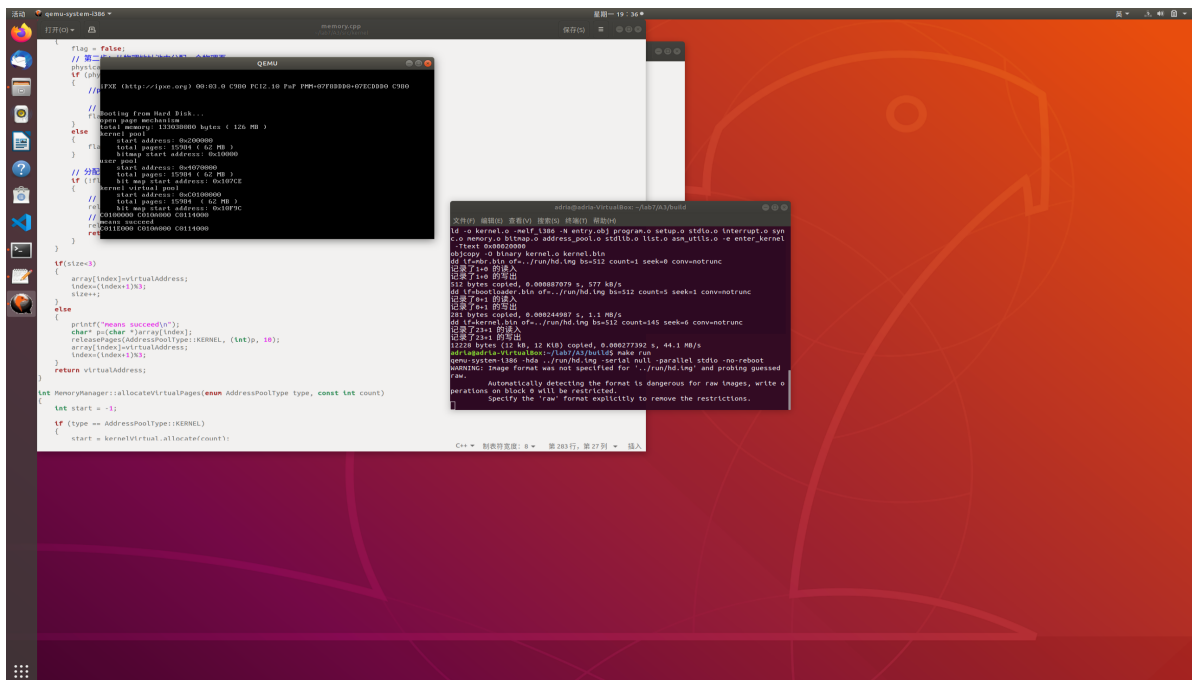
任务3要求实现页面替换，课本中大致有三种方案，FIFO,LRU,最优算法。此处实现FIFO算法，但是因为没有和外存相关的函数所以页面替换不完整，只实现了分配内存。

实现原理：用数组和下标表示空间和被分配内存的下标如果数组满则替换下标对应的虚拟地址，不满则直接分配好插入数组。

修改代码：

```
if(size<3)
{
    array[index]=virtualAddress;
    index=(index+1)%3;
    size++;
}
else
{
    printf("means succeed\n");
    char* p=(char *)array[index];
    releasePages(AddressPoolType::KERNEL, (int)p, 10);
    array[index]=virtualAddress;
    index=(index+1)%3;
}
```

展示：



```
QEMU

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD+07ECDDDD C980

Booting from Hard Disk...
open page mechanism
total memory: 133038080 bytes ( 126 MB )
kernel pool
  start address: 0x200000
  total pages: 15984 ( 62 MB )
  bitmap start address: 0x10000
user pool
  start address: 0x4070000
  total pages: 15984 ( 62 MB )
  bit map start address: 0x107CE
kernel virtual pool
  start address: 0xC0100000
  total pages: 15984 ( 62 MB )
  bit map start address: 0x10F9C
C0100000 C010A000 C0114000
means succeed
C011E000 C010A000 C0114000
```

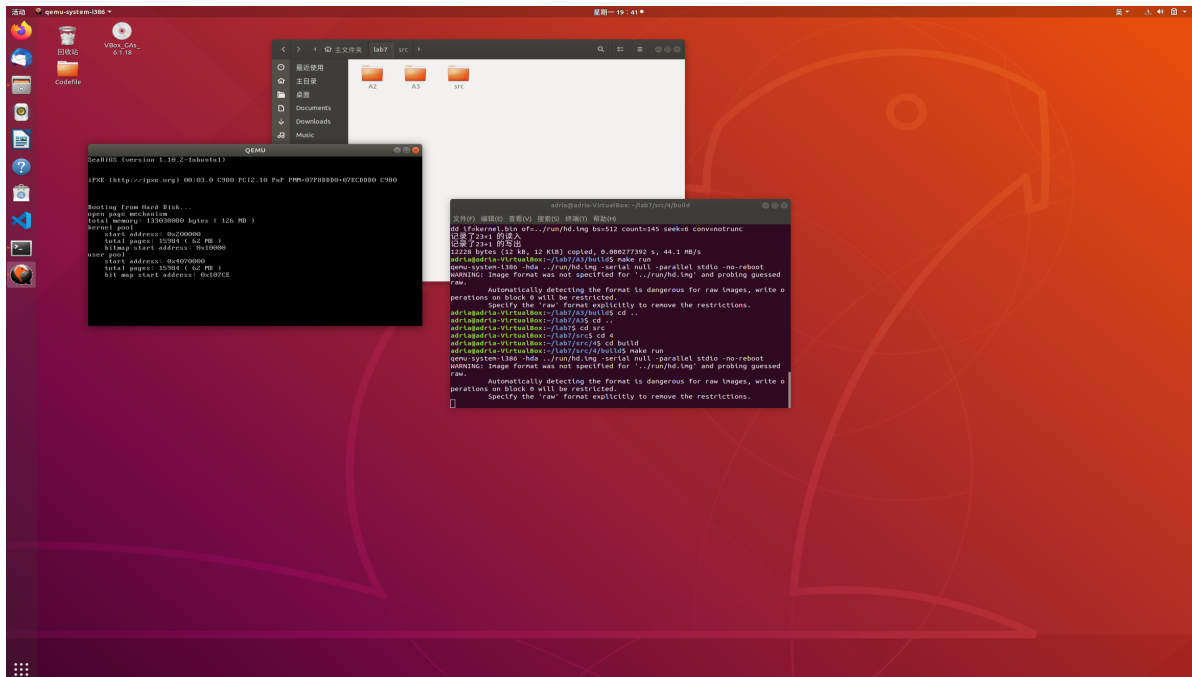
解释：

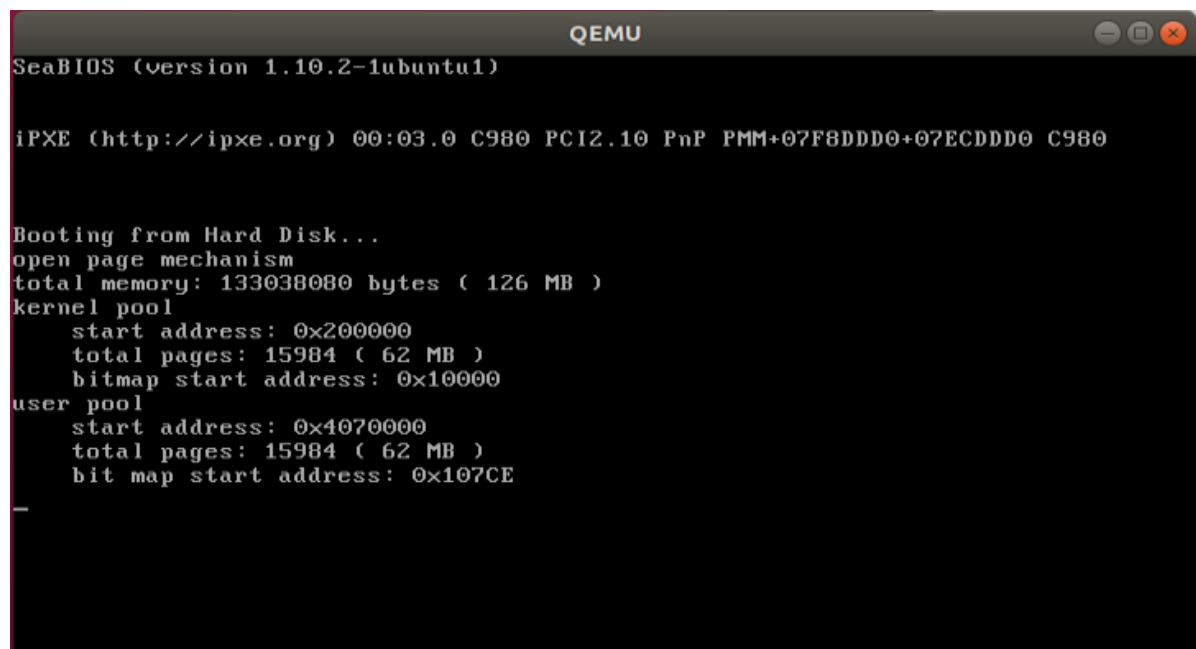
设置了一共大小为3的数组，先输出三个数的地址，然后申请新的内存，用FIFO算法进行替换得到新的地址。

Assignment 4

复现“虚拟页内存管理”一节的代码，完成如下要求。

- 结合代码分析虚拟页内存分配的三步过程和虚拟页内存释放。
- 构造测试例子来分析虚拟页内存管理的实现是否存在bug。如果存在，则尝试修复并再次测试。否则，结合测例简要分析虚拟页内存管理的实现的正确性。



A screenshot of a QEMU virtual machine window. The title bar says 'QEMU'. The terminal output shows the SeaBIOS boot process. It starts with 'SeaBIOS (version 1.10.2-1ubuntu1)', followed by 'iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD+07ECDDDD C980'. Then it says 'Booting from Hard Disk...', 'open page mechanism', 'total memory: 133038080 bytes (126 MB)', 'kernel pool', 'start address: 0x200000', 'total pages: 15984 (62 MB)', 'bitmap start address: 0x10000', 'user pool', 'start address: 0x4070000', 'total pages: 15984 (62 MB)', and 'bit map start address: 0x107CE'. The output ends with a single hyphen character.

```
SeaBIOS (version 1.10.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD+07ECDDDD C980

Booting from Hard Disk...
open page mechanism
total memory: 133038080 bytes ( 126 MB )
kernel pool
  start address: 0x200000
  total pages: 15984 ( 62 MB )
  bitmap start address: 0x10000
user pool
  start address: 0x4070000
  total pages: 15984 ( 62 MB )
  bit map start address: 0x107CE
-
```

实现了二级分页的机制，解决了内存分配不连续的问题，原理和任务1差不多。未发现bug