L1：多处理器内核上的物理内存管理

2020.4.10
181240035
刘春旭

实验进度：

- 完成了`kalloc`和`malloc`，支持最多8个线程4KB以下的快速并行内存分配；
- **个人觉得本次实验有一些值得写的东西，故实验报告多于2页纸，请见谅。**

> 拖延真的不是好习惯。 ——鲁迅（没说过这话）

1. 代码思路：

本实验的基本思路就是处理两种类型的内存分配，slow path和fast path。每个线程都保留自己的链表头，这个页面头连接着不同大小的slab页面：

```
typedef struct __head{
  page_t *_32;
  ...
  page_t *_4096;
}list_head;
```

其中，寻找对应slab大小的函数为位运算（round up to the nearest power of 2）：

```
size_t ALIGN(size_t v){
  v--;
  v |= v >> 1;
  v |= v >> 2;
  v |= v >> 4;
  v |= v >> 8;
  v |= v >> 16;
  v++;
  return v;
}
```

然后根据当前线程的序号、大小的类型就可以从链表的头部找到对应的slab页面的头部，然后进行内存分配。思路基本同课上讲解。

2. BUG之路

> 什么吗，这个实验也没有多难嘛...（希望之花.mp3）

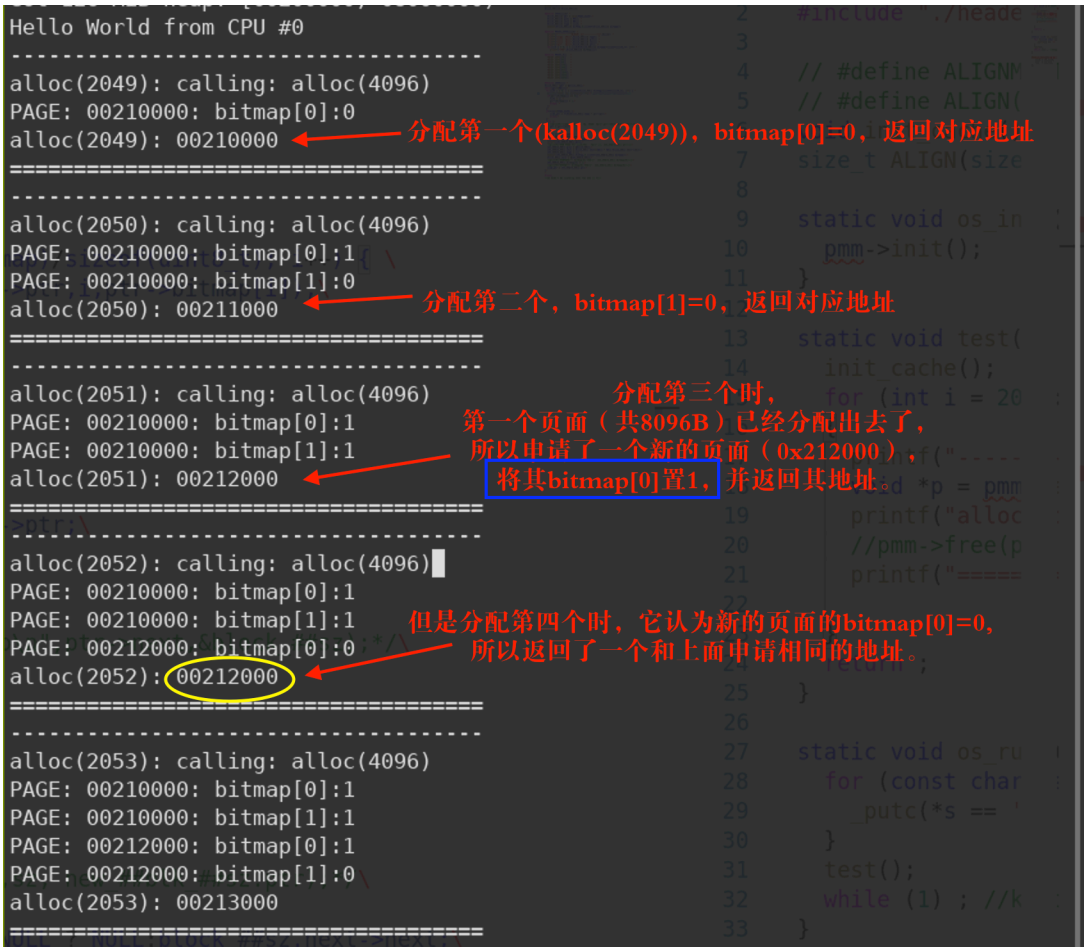
回过头来看确实这个实验的每一步，蒋老师都有过贴心的提示。所以其实这个实验最值得提的就是debug之路了。

- *刚开始的时候没有意识到可以把指针强制类型转换成结构体指针*

我真傻，真的。刚开始做的时候觉得，我该怎么把页面写到堆区呢？我当时真的觉得只有malloc才能帮我实现把数据分配到堆区，但是现在要自己实现malloc，我人就傻了。直到ddl前三天的一个午后，我睡醒惊觉，把`void *`转化成`page_t *`之后不就可以随意读写结构体对应位置的值了吗。（午睡有益身心健康）

- *栈溢出*

  这个真的是一个大问题，由于上面第一点提到的问题，我觉得只能把头部和数据区分开，这样不也可以正常返回堆区的地址吗，所以我就把头部声明在**栈区、静态数据区**了（当时我没意识到这个想法的危害）。于是随着页面越来越多...某个位置的值就一直写不上了。



当时我：

> （1）怀疑可能编译器优化掉了这条给bitmap赋值的指令，故我换成了汇编代码：
>
> ```
> __asm__ __volatile__("mov $1, %0" : :"r"(new_##blk_##sz.bitmap[0]) :
> );
> ```
>
> 注释掉printf后依然有问题。
>
> （2）怀疑是编译器乱序执行指令的锅：
>
> 但是我尝试在这条赋值的后面加入了`barrier()`之后
>
> ```
> #define barrier() __asm__ __volatile__("": : :"memory")
> ```
>
> 所以我觉得可能是在这条指令后面调用了函数的话，这条指令的执行顺序就在某种程度上被保证了吧。

但是这其实是栈溢出的原因。首先，我原先在函数内部分配的变量是分配到栈上的，函数结束之后是会被销毁的；其次实验讲义中要求过不要把过多的内容分配到静态数据区。

解决方案：意识到第一点后重写。

- *写了很多宏*

  写了大概100多行的宏...很难受，这也是我下决心重写代码的原因之一。刚开始写宏的时候很开心，觉得很方便，但后来就越写越多...

  解决方案：可以先预留出足够的空间，足够容纳下无论哪种大小的slab的bitmap，然后就可以统一定义了。另外，bitmap也不够bit，我定义成了bytemap。反正要求对齐的话，是肯定要浪费一部分内存的。

- *fast path的数据竞争?*

  在实验的后期，hard test后几个点轮流出问题。接近崩溃的时候，和同学讨论发现，有很多同学都在bitmap上加了锁。确实，实验的讲义上也给每个页面都加了锁。但是我认为fast path的内存都已经是线程独享的了，无论在哪个线程free，我再分配的时候也不会出现数据竞争的问题，因此不用给bitmap加锁。可能出现数据竞争的位置只有可能是sbrk的时候，因为毕竟只有堆区才是数据共享区。所以后来我发现，是自己的unlock没有用原子指令.....

  解决方案：unlock改用原子指令，且只在sbrk处加一把大锁，未在bitmap上加锁。