English Title

Name:Name StdID:171860658 Date:May，2019

L1:kalloc

1 设计架构

在实验前看了诸如 Buddy System, Slab System 等实现方法，但最终还是选择了简单的以链表为结构的方法。

```
struct node {
  struct node *next, *pre;
  uintptr_t start, end;
};
```

每一个 node 指示了一个分配出去的空间，即每次 alloc 会额外占用 sizeof(node) 的空间用于储存链表信息。head, tail 是两个分别位于可用空间头和尾的空分配（即只占用 sizeof(node) 空间）。每次分配时，从头遍历，判断 p->next->begin-p->end 是否大于 size ，若大于则插入节点，分配空间。

该实现较为简单，利用自旋锁锁住插入节点和删除节点部分即可。

为了方便后面的实验，添加了两个宏。DEBUG 用于输出一些调试信息，CORRECTNESS_FIRST 使用只分配不释放的方法实现。

2 印象深刻的 Bug

(a) 考虑分配空间时为链表节点分配的空间：由于链表节点与分配的空间一起分配空间，在一开始的实现中，p->end 没有加上该部分的偏移，导致 assert(p->next->pre==p) 不满足。

(b) 把函数定义写到头文件里了：头文件应当写的是函数的声明。以及需要使用宏使头文件只被引用一次。

L2:kthreads

3 设计架构

我的代码中对于 irq 和 tasks 使用数组实现，有数量上限，最多支持 1024 个 tasks 的注册。os 和 kmt 模块实现起来话的时间不多，但花了大量时间 debug，而且查的欲生欲死。

最后还是成功把终端跑起来了，令人开心。

4 印象深刻的 bug

Bug 太多了，我挑选一些记得的和调了很久的记录。

(a) dead lock：编程中死锁出现的次数很多，但是触发死锁的原因很多，查起来就很难。主要有两个原因，一个是没有正确的关中断，另一个是 ABBA 的出现。解决这个 bug 的一个好的调试方法是先使用一个 cpu 跑着调试。

(b) 锁未初始化就使用：出问题后打印锁的名称，发现是乱码。

(c) ud2：这是一个较为精彩（坎坷）的 bug。首先是出现了 relock，根据调试信息发现是收到了错误中断导致 relock，加上判断后发现是 Invalid opcode，根据汇编发现是指令 ud2，查阅资料发现是程序中有 undefined behaviour，确认到程序中，发现是 assert 中的条件写错了。

(d) 中断中中断：不确定关中断后 os_trap 是否会来中断，为此实验了多次，并且其它 bug 的存在影响了对这个的判断，导致浪费了很多时间。

(e) spinlock 错误：在检测

(f) cpu 空转：当可运行的 threads 数量小于 cpu 总数时，存在 cpu 无法获得进程。为此特别设置了“空”进程，当没有可运行的进场时运行。

(g) debug 错误：这类错误即程序本身没有错误，但是加进去的 debug 信息反而有问题导致 bug。这分三类，第一种是 assert 中的条件本身就不应该成立（或没有考虑到所有情况），第二种是调试信息输出有误，比如给 Log 加锁后再让其输出是否关中断。第三类是改变了程序中的值。这类 bug 在所有 bug 中占比不小。