lsOSLab 实验报告

171240501 匡舒磊

L1 内核内存分配

代码架构：使用了两个结构体`freelist` 与 `runlist`，结构体内部包括了指向两个链表开始的head (head不存储任何的内存信息)和这个链表的size，(为方便描述，下用`freelist`与`runlist`代指两个链表)然后把整块可用的内存都加到freelist中，链表中的每个block存储了这个内存块的size，begin_addr,end_addr,和指向上一个与下一个block的指针，还包含了一个state(0表示这个block未被使用，1表示在freelist中，2表示在runlist中)，每次分配时从freelist的开始找size满足的块，如满足则分配给他，如果这个块大于需要的块的大小的话，则把剩余的再加入到freelist中，free时则在runlist中遍历寻找，再加入到freelist中，如果有可以合并的块的话则合并，在freelist中块按照地址排序，runlist则无排序

说明： 由于是用数组存放的block，故而runlist和freelist中一共只能最多包含4096个块，如果需要更多的话，需要在修改一下数组大小

bug ： 我讨厌链表...一不小心可能处理是就有东西未指向什么...

L2 内核多线程

说明

在没开kvm的make run 运行下，tty要等很久才能使用（我自己测大概要等一分钟），但是make run2和make run4大概十几秒就可以了

代码架构

用c_task指向所有的task，runtask数组指向每个cpu正在运行的cpu，信号量对应的taskid也用数组存储，每一次切换进程或者信号量的wait和唤醒都直接数组遍历寻找空位或者找到满足条件的位置。值得说明的是，对于每一个task在create的时候，都直接将他绑定在了一个cpu上。

印象深刻的**bug**

之前的实现没有考虑跨核调度，然后每次运行产生的bug都不同（会在不同的位置挂掉，甚至加一些无关的printf也会不同），一开始怀疑自己alloc出错导致了内存被污染了，检查了一遍感觉没啥问题，后来发现可能是跨核调度的问题，即可能产生一个进程中断后在另一个核上被调用，然后调用时中断返回，导致这个进程同时在两个cpu上运行，导致其stack smash。

感想

有基础理论知识和一知半解做lab完全是两种体验，一开始做的时候是有点懵逼的做的，然后期末考试复习完再看自己的代码的时候，debug的时候就比较容易看出自己的一些错误想法。

L3 文件系统

类似讲义的方式实现了一个不太完整的文件系统，采用链表存储inode，并将大多数文件信息存储其中，对应文件只包含一个content，inode中有指向内容的指针

除了初始的外，在/ 目录中有一个`hello.c`文件，可以用于测试cat

实现了指令：

> cat + 路径（可cat文件与proc中的文件） ls + 路径 mkdir + 路劲 rmdir：只删除文件夹 rm：只删除文件 cd +路径

以上均支持相对路劲与绝对路劲