操作系统

Operating System

内容提要

- 实验内容简介
- 实验环境介绍

实验内容简介

实验内容简介

- 前言
- 国内外现状
- 实验课程设计
- 效果和存在的问题
- 小结

前言

- 对操作系统课程教学的理解（Why）
  - 计算机科学与计算机工程相结合
  - 原理和实验教学内容并行进行
    - -->...实验-->原理-->实验-->原理-->...
  - 强调动手编程实践
  - 实验需求
    - 理解系统　　循序渐进　　阅读代码
    - 把握全局　　功能完善　　改进创新

国内外现状——国外

- MIT：xv6 和JOS
  - 7千行以下，C语言，支持X86 SMP架构
- Harvard：OS161-1.4.1
  - 1万1千行代码，C语言，支持MIPS架构
- Columbia：Linux
  - 部分Linux核心代码，C语言
- Berkeley: Nachos
  - 1万行左右，C++ / java语言，模拟MIPS架构
- Stanford：PintOS
  - 1万1千行代码，C语言，
- Univ. of Maryland: geek OS
  - <10000行代码，C语言，x86

国内外现状——国内

- ucore
  - 清华　基于jos/xv6/OS161/linux，200~10000行 C语言，以x86-32为主，且支持X86-64/ARM/MIPS…
- xv6和JOS
  - 北大
- Linux
  - 国防科大、浙大、西邮
- MINIX
  - 上海交大，南开
- Nachos
  - 南开，山大
- Solaris, Windows WRK, Wince, RTEMS, uCos-II, eCos ..

实验课程设计

- 目标：在OS原理和实现中建立一个桥梁
  - 对原理知识的补充和完善
    - 讲课内容和实验内容同步
  - 让学生对操作系统设计有一个全局的理解
    - 操作系统要小巧且覆盖面全
  - 适合不同层次学生的需求
    - 存在高中低三类学生

实验课程设计

- 设计思路
  - 差异化教学
    - 高水平学生：鼓励创新
    - 中等学生：完成实验
    - 较弱学生：理解实验内容

实验课程设计

- 设计思路
  - 方便且利用理解细节
    - 大量采用开源软件
    - 实验环境：Windows/Linux
    - 源码阅读工具：understand
    - 源码文档自动生成工具：Doxygen
    - 编译环境：gcc，make，Binutils
    - 真实/虚拟运行环境：X86机器或QEMU
    - 调试工具：改进的QEMU + (GDB O.R. IDE)
    - IDE工具：Eclipse-CDT

实验课程设计

- 设计思路
  - 采用小巧全面的操作系统ucore并进行改进，需要覆盖操作系统的关键点，为此增加：
    - 外设：I/O管理/中断管理
    - 内存：虚存管理/页表/缺页处理/页替换算法
    - CPU：进程管理/调度器算法
    - 并发：信号量实现和同步互斥应用
    - 存储：基于链表/FAT的文件系统
  - 完整代码量控制在10000行以内
  - 提供实验讲义和源码分析文档

实验课程设计

- 实验内容
  - 1 OS启动、中断与设备管理： 0200~1800行
  - 2 物理内存管理： 1800~2500行
  - 3 虚拟内存管理： 2500~3200行
  - 4 内核线程管理： 3200~3600行
  - 5 用户进程管理： 3600~4300行
  - 6 处理器调度： 4300~5100行
  - 7 同步互斥： 5100~6400行
  - 8 文件系统： 6400~9999行

实验课程设计


段式内存管理
init子系统
库函数子系统
cprintf/str*...
Monitor监控子系统
远程debug调试子系统
中断管理子系统
软件层
设备驱动层
网卡
串口
并口
CGA
硬盘
键盘
时钟
中断控制器
bootloader
CPU
I/O地址空间
MEM地址空间
硬件层
外设控制器/中断控制器
网卡
串口
并口
CGA
硬盘
键盘
时钟

实验课程设计


各种用户态应用和测试用例
用户态函数库
用户态
系统调用接口
内核态
进程管理子系统
进程间共享库支持
进程调度算法
进程调度框架
进程生命周期管理
文件管理子系统
FAT文件系统
UNIX文件系统
Buffer Cache
网络
TCP/IP协议栈
进程间通信
消息队列
PIPE
内存管理子系统
不连续地址空间分配算法
连续地址空间分配算法
虚拟内存分配管理
物理内存分配管理
页式内存管理
写时复制
按需分页
页故障管理
页替换算法
swap管理
同步互斥/死锁
解决死锁问题的实例
同步互斥应用实例
semaphore实现
Lock实现

实验课程设计

- **Lab1: Bootloader/Interrupt/Device Driver**
  - 启动操作系统的bootloader，了解操作系统启动前的状态和要做的准备工作，了解运行操作系统的硬件支持，操作系统如何加载到内存中，理解两类中断--“外设中断”，“陷阱中断”等；
    - 理管储存的制机段分于基
    - 念概本基的理管备设
    - PC动启bootloader程过的
    - bootloader成组件文的
    - 行运译编bootloader程过的
    - 试调bootloader法方的
    - 程过理处和构结的栈解了级编汇在
    - 制机理处断中
    - 口串过通/口并/CGA法方的符字出输

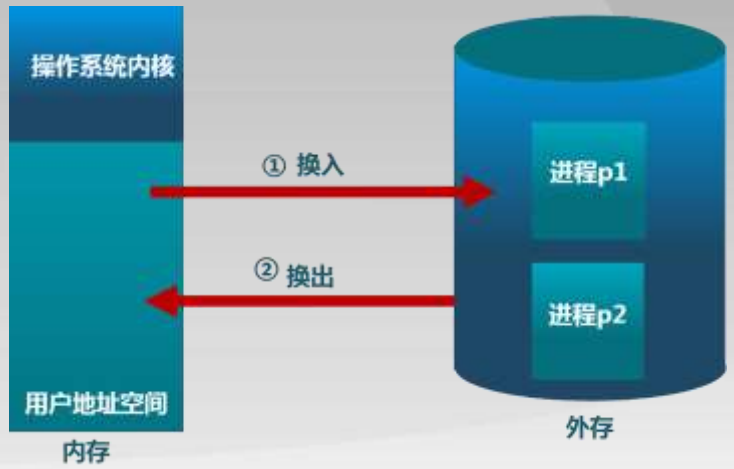
```
proj1 /
|-- boot
|   |-- asm.h
|   |-- bootasm.S
|   `-- bootmain.c
|-- libs
|   |-- types.h
|   `-- x86.h
|-- Makefile
`-- tools
    |-- function.mk
    `-- sign.c

3 directories, 8 files
```

实验课程设计

- Lab2：物理内存管理
  - 理解x86分段/分页模式，了解操作系统如何管理连续空间的物理内存
    - 理解内存地址的转换和保护
    - 实现页表的建立和使用方法
    - 实现物理内存的管理方法
    - 了解常用的减少碎片的方法

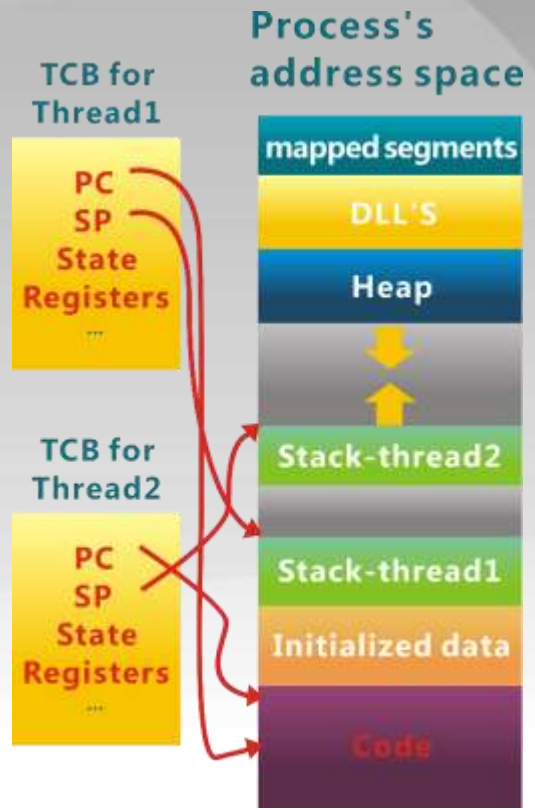
实验课程设计

- Lab3：虚拟内存管理
  - 了解页表机制和换出（swap）机制，以及中断-“故障中断”、缺页故障处理等，基于页的内存替换算法；
    - 理解换页的软硬件协同机制
    - 实现虚拟内存的Page Fault异常处理
    - 实现页替换算法



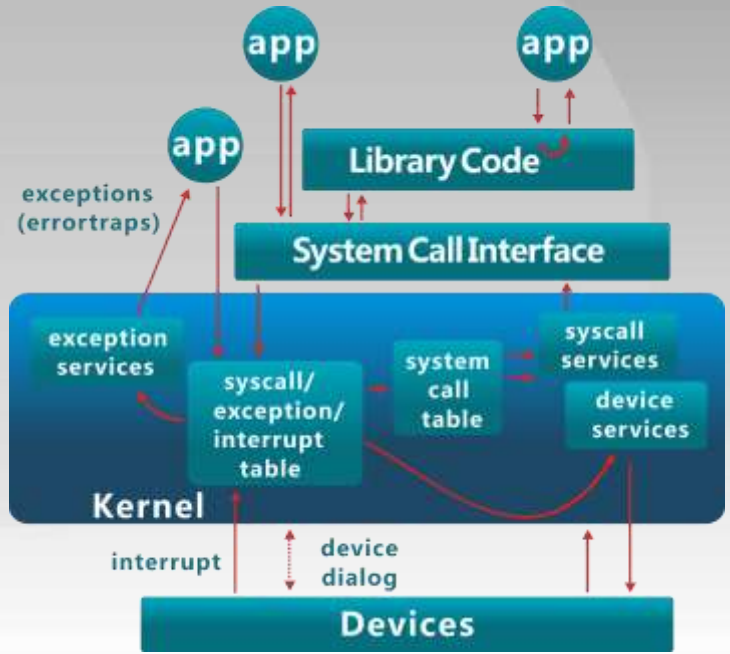
实验课程设计

- Lab4: 内核线程管理
  - 了解如果利用CPU来高效地完成各种工作的设计与实现基础，如何创建相对与用户进程更加简单的内核态线程，如果对内核线程进行动态管理等；
    - 建立内核线程的关键息信
    - 实现内核线程的管理方法



实验课程设计

- Lab5: 用户进程管理
  - 了解用户态进程创建、执行、切换和结束的动态管理过程，了解在用户态通过系统调用得到内核态的内核服务的过程
    - 建立用户进程的关键信息
    - 实现用户进程管理
    - 分析进程和内存管理的关系
    - 实现系统调用的处理过程

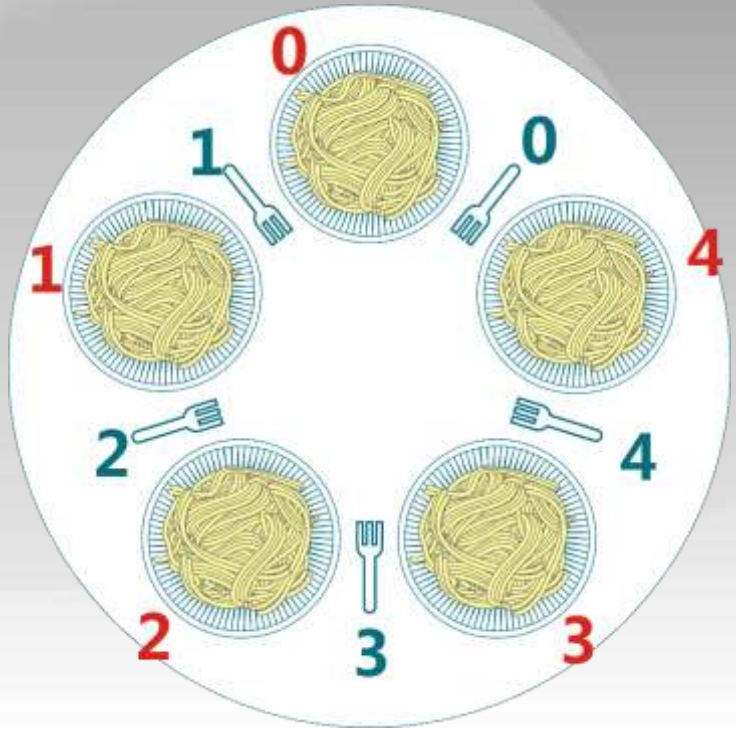


实验课程设计

- Lab6：进程调度
  - 用于理解操作系统的调度过程和调度算法
    - 熟悉 ucore 的系统调度器框架，以及内置的 Round-Robin 调度算法。
    - 基于调度器框架实现一个调度器算法

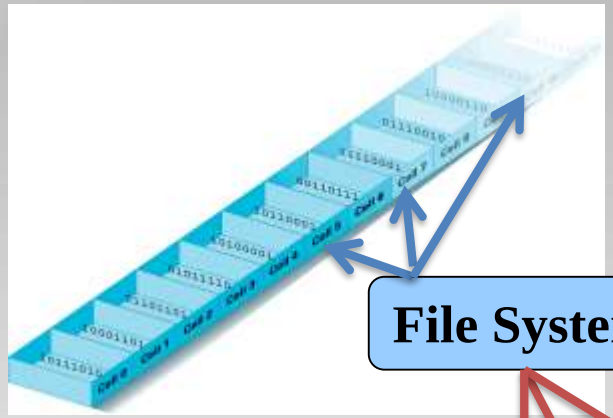
实验课程设计

- Lab7：同步互斥
  - 了解进程间如何进行信息交换和共享，并了解同步互斥的具体实现以及对系统性能的影响，研究死锁产生的原因，以及如何避免死锁；
    - 熟悉 ucore 的同步互斥机制
    - 理解基本的spinlock、semphpore、condition variable的实现
    - 用各种同步机制解决同步问题



实验课程设计

- Lab8：文件系统
  - 了解文件系统的具体实现，与进程管理等的关系，了解缓存对操作系统IO访问的性能改进，了解虚拟文件系统（VFS）、buffer cache和disk driver之间的关系。



- 掌握基本的文件系统系统调用的实现方法；
- 了解一个基于索引节点组织方式的Simple FS文件系统的设计与实现；
- 了解文件系统抽象层-VFS的设计与实现；

实验课程设计

- **扩展实验**

U0: ucore porting on x86-64
Status: 100%, ucorer: wnz

U1: local page replacement framework with different algorithms of local page replacement
status: 100%, ucorer: yxh

U2: ucore支持ARM CPU(with mmu)
Status: 100 %, ucorer: wjf, ykl, xb
…
U9: ucore文件系统框架：支持在VFS下同时支持FAT32等文件系统，实现更加简化的VFS、FAT和SFS，并能够实现高性能的基于DMA方式的磁盘访问；
status: 100%, ucorer: qz,rsw
…
U12:ucore支持GO programming
Status: 100 %, ucorer: cr,fjy

效果

- 好的方面
  - 理论和实验能够较好地结合起来，不再感到OS课是一个只要死记硬背的课程了
  - 理解了一个OS的全局设计实现，而不是一个一个分离的知识点
  - 掌握了许多OS原理上没有涉及或涉及不够的东西，比如中断/系统调用的实现，X86的段页机制，进程上下文如何切换的，内核态和用户态的具体区别是什么
  - 这是大学期碰到的最复杂的软件，学习了分析和设计大型系统软件的方法

小结

对覆盖大部分OS关键知识点的微型OS开展实验对学好OS课程有极大的促进作用

你值得尝试！

实验环境介绍

内容提要

- 安装实验环境
  - 在虚拟机上使用安装好的ubuntu实验环境
- 使用实验工具
  - shell命令：ls、cd、rm、pwd...
  - 系统维护工具：apt、git
  - 源码阅读与编辑工具： eclipse-CDT 、 understand、gedit、vim
  - 源码比较工具：diff、meld
  - 开发编译调试工具：gcc 、gdb 、make
  - 硬件模拟器：qemu

内容提要

- 了解x86-32硬件
  - Intel 80386运行模式概述
  - Intel 80386内存架构概述
  - Intel 80386寄存器概述
- 了解ucore编程方法和通用数据结构
  - 面向对象编程方法
  - 通用数据结构

安装实验环境

- 在虚拟机上使用安装好的ubuntu实验环境
  - 下载安装VirtualBox虚拟机软件
    https://www.virtualbox.org/
  - VirtualBox软件和虚拟硬盘文件压缩包等信息可查询
    https://github.com/chyyuu/os_course_info
  - 解压压缩包后，可得到如下内容（大约4GB多）
    \mooc-os\mooc-os.vbox
    \mooc-os\mooc-os.vbox-prev
    \mooc-os\mooc-os.vdi
  - 解压压缩包后，可得到如下内容（大约4GB多）
  - 用户名是 moocos 口令是 <空格键>

内容提要

- 安装实验环境
  - 在虚拟机上使用安装好的ubuntu实验环境
- 使用实验工具
  - shell命令：ls、cd、rm、pwd...
  - 系统维护工具：apt、git
  - 源码阅读与编辑工具：eclipse-CDT、understand、gedit、vim
  - 源码比较工具：diff、meld
  - 开发编译调试工具：gcc 、gdb 、make
  - 硬件模拟器：qemu
- 了解x86-32硬件
  - Intel 80386运行模式概述
  - Intel 80386内存架构概述
  - Intel 80386寄存器概述
- 了解ucore编程方法和通用数据结构
  - 面向对象编程方法
  - 通用数据结构

使用实验工具

- shell命令：ls、cd、rm、pwd、mkdir、find
  - 基于bash（Bourne-Again SHell）
  - 完成对文件、目录的基本操作

使用实验工具

- 系统维护工具：apt、git
  - apt：安装管理各种软件
  - git：开发版本维护软件

使用实验工具

- 源码编辑工具：Eclipse-CDT、understand、gedit、vim
  - Eclipse-CDT：基于Eclipse的C/C++集成开发环境
    跨平台、丰富的分析理解代码的功能
    可与qemu结合，联机源码级Debug uCore OS。
  - Understand：跨平台、丰富的分析理解代码的功能
    Windows上有类似的sourceinsight软件
  - gedit：Linux中的常用文本编辑
    Windows上有类似的notepad
  - vim: Linux/unix中的传统编辑器
    类似有emacs等
    可通过exuberant-ctags、cscope等实现代码定位

使用实验工具

- 源码比较工具：diff、meld
  - 比较不同目录或不同文件的区别
  - diff是命令行工具，使用简单
  - meld是图形界面的工具，功能相对直观和方便

使用实验工具

- 开发编译调试工具：gcc 、gdb 、make
  - gcc：C语言编译器
  - gdb：执行程序调试器
  - make：软件工程管理工具

    make命令执行时，需要一个 makefile 文件，以告诉make命令如何去编译和链接程序。

使用实验工具

- 硬件模拟器：qemu
  - qemu可模拟多种CPU硬件环境，本实验中，用于模拟一台intel x86-32的计算机系统

参考资料

- apt-get
  - http://wiki.ubuntu.org.cn/Apt-get%E4%BD%BF%E7%94%A8%E6%8C%87%E5%8D%97
- gcc
  - http://wiki.ubuntu.org.cn/Gcchowto
  - http://wiki.ubuntu.org.cn/Compiling_Cpp
  - http://wiki.ubuntu.org.cn/C_Cpp_IDE
  - http://wiki.ubuntu.org.cn/C%E8%AF%AD%E8%A8%80%E7%AE%80%E8%A6%81%E8%AF%AD%E6%B3%95%E6%8C%87%E5%8D%97
- gdb
  - http://wiki.ubuntu.org.cn/%E7%94%A8GDB%E8%B0%83%E8%AF%95%E7%A8%8B%E5%BA%8F
- make & makefile
  - http://wiki.ubuntu.com.cn/index.php?title=%E8%B7%9F%E6%88%91%E4%B8%80%E8%B5%B7%E5%86%99Makefile&variant=zh-cn

参考资料

- shell
  - http://wiki.ubuntu.org.cn/Shell%E7%BC%96%E7%A8%8B%E5%9F%BA%E7%A1%80
  - http://wiki.ubuntu.org.cn/%E9%AB%98%E7%BA%A7Bash%E8%84%9A%E6%9C%AC%E7%BC%96%E7%A8%8B%E6%8C%87%E5%8D%97
- understand
  - http://blog.csdn.net/qwang24/article/details/4064975
- vim
  - http://www.httpy.com/html/wangluobiancheng/Perljiaocheng/2014/0613/93894.html
  - http://wenku.baidu.com/view/4b004dd5360cba1aa811da77.html
- meld
  - https://linuxtoy.org/archives/meld-2.html
  - 类似的工具还有 kdiff3、diffmerge、P4merge
- qemu
  - http://wenku.baidu.com/view/04c0116aa45177232f60a2eb.html
- Eclipse-CDT
  - http://blog.csdn.net/anzhu_111/article/details/5946634

内容提要

- 安装实验环境
  - 在虚拟机上使用安装好的ubuntu实验环境
- 使用实验工具
  - shell命令：ls、cd、rm、pwd...
  - 系统维护工具：apt、git
  - 源码阅读与编辑工具：eclipse-CDT 、understand、gedit、vim
  - 源码比较工具：diff、meld
  - 开发编译调试工具：gcc 、gdb 、make
  - 硬件模拟器：qemu
- 了解x86-32硬件
  - Intel 80386运行模式概述
  - Intel 80386内存架构概述
  - Intel 80386寄存器概述
- 了解ucore编程方法和通用数据结构
  - 面向对象编程方法
  - 通用数据结构

了解x86-32硬件-运行模式

- 80386有四种运行模式：

  实模式、保护模式、SMM模式和虚拟8086模式。

- 实模式：80386加电启动后处于实模式运行状态，在这种状态下软件可访问的物理内存空间不能超过1MB，且无法发挥Intel 80386以上级别的32位CPU的4GB内存管理能力。

- 保护模式：支持内存分页机制，提供了对虚拟内存的良好支持。保护模式下80386支持多任务，还支持优先级机制，不同的程序可以运行在不同的优先级上。优先级一共分0～3 4个级别，操作系统运行在最高的优先级0上，应用程序则运行在比较低的级别上；配合良好的检查机制后，既可以在任务间实现数据的安全共享也可以很好地隔离各个任务。

了解x86-32硬件-内存构架

- 地址是访问内存空间的索引。
- 80386是32位的处理器，即可以寻址的物理内存地址空间为2^32=4G字节
- 物理内存地址空间是处理器提交到总线上用于访问计算机系统中的内存和外设的最终地址。一个计算机系统中只有一个物理地址空间。
- 线性地址空间是在操作系统的虚存管理之下每个运行的应用程序能访问的地址空间。每个运行的应用程序都认为自己独享整个计算机系统的地址空间，这样可让多个运行的应用程序之间相互隔离。
- 逻辑地址空间是应用程序直接使用的地址空间。

段机制启动、页机制未启动：逻辑地址->段机制处理->线性地址=物理地址

段机制和页机制都启动：逻辑地址->段机制处理->线性地址->页机制处理->物理地址

了解x86-32硬件-寄存器

- 80386的寄存器可以分为8组：
  - 通用寄存器
  - 段寄存器
  - 指令指针寄存器
  - 标志寄存器
  - 控制寄存器
  - 系统地址寄存器，调试寄存器，测试寄存器

了解x86-32硬件-寄存器

- 通用寄存器
  - EAX：累加器
  - EBX：基址寄存器
  - ECX：计数器
  - EDX：数据寄存器
  - ESI：源地址指针寄存器
  - EDI：目的地址指针寄存器
  - EBP：基址指针寄存器
  - ESP：堆栈指针寄存器

了解x86-32硬件-寄存器

- 段寄存器
  - CS：代码段(Code Segment)
  - DS：数据段(Data Segment)
  - ES：附加数据段(Extra Segment)
  - SS：堆栈段(Stack Segment)
  - FS：附加段
  - GS：附加段

了解x86-32硬件-寄存器

- 指令寄存器和标志寄存器
  - EIP：指令寄存器

    EIP的低16位就是8086的IP，它存储的是下一条要执行指令的内存地址，在分段地址转换中，表示指令的段内偏移地址。
  - EFLAGS：标志寄存器

    IF(Interrupt Flag)：中断允许标志位,由CLI，STI两条指令来控制；设置 IF 使CPU可识别外部（可屏蔽）中断请求。复位 IF 则禁止中断。 IF 对不可屏蔽外部中断和故障中断的识别没有任何作用。

    CF,PF, ZF, ...

内容提要

- 安装实验环境
  - 在虚拟机上使用安装好的ubuntu实验环境
- 使用命令行工具和实验工具
  - shell命令：ls、cd、rm、pwd...
  - 系统维护工具：apt、git
  - 源码阅读与编辑工具： eclipse-CDT 、understand、gedit、vim
  - 源码比较工具：diff、meld
  - 开发编译调试工具：gcc 、gdb 、make
  - 硬件模拟器：qemu
- 了解x86-32硬件
  - Intel 80386运行模式概述
  - Intel 80386内存架构概述
  - Intel 80386寄存器概述
- 了解ucore编程方法和通用数据结构
  - 面向对象编程方法
  - 通用数据结构

了解ucore编程方法和通用数据结构

- ucore主要基于C语言设计，采用了一定的面向对象编程方法。

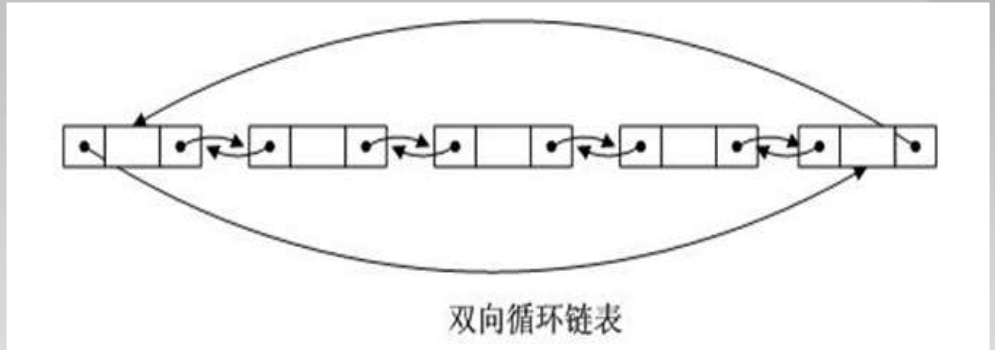
/lab2/kern/mm/pmm.h

```
------------------------------
struct pmm_manager {
    const char *name;
    void (*init)(void);
    void (*init_memmap)(struct Page *base,
size_t n);
    struct Page *(*alloc_pages)(size_t n);
    void (*free_pages)(struct Page *base, size_t
n);
    size_t (*nr_free_pages)(void);
    void (*check)(void);
};
```

了解ucore编程方法和通用数据结构

- **双向循环链表**

```
typedef struct foo {
    ElemType data;
    struct foo *prev;
    struct foo *next;
} foo_t;
```

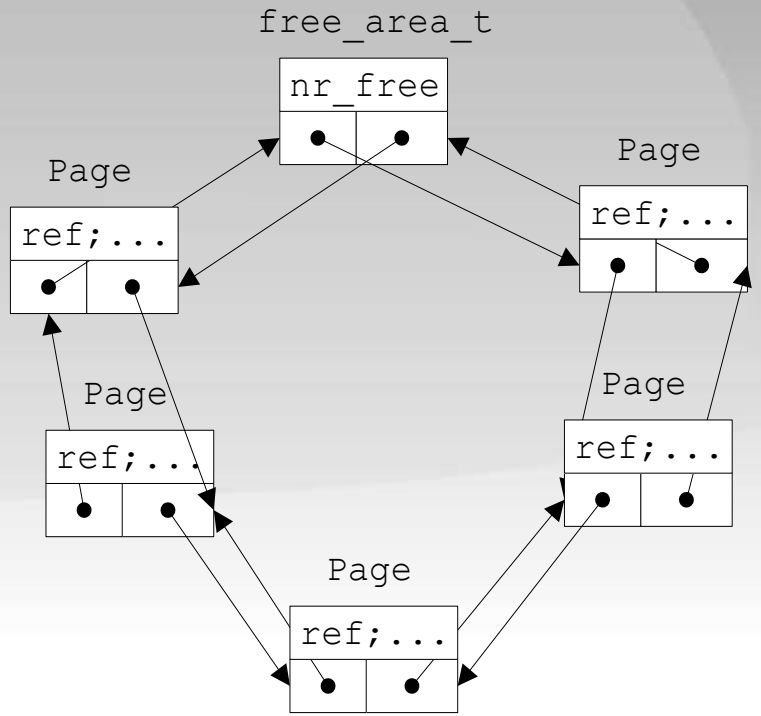


需要为每种特定数据结构类型定义针对这个数据结构的特定链表插入、删除等各种操作，会导致代码冗余。

了解ucore编程方法和通用数据结构

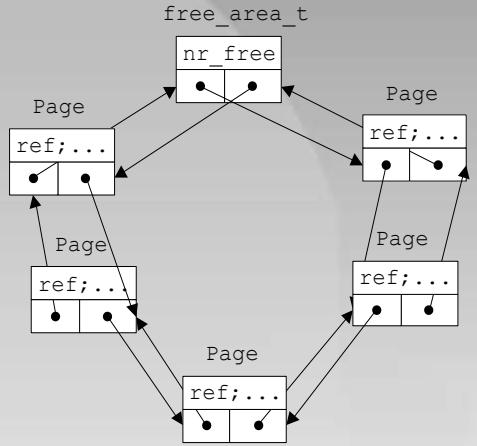
- uCore的双向链表结构定义

```
struct list_entry {
    struct list_entry *prev, *next;
};

typedef struct {
    list_entry_t free_list;
    unsigned int nr_free;
} free_area_t;

struct Page {
    atomic_t ref;
     ……
    list_entry_t page_link;
};
```



了解ucore编程方法和通用数据结构

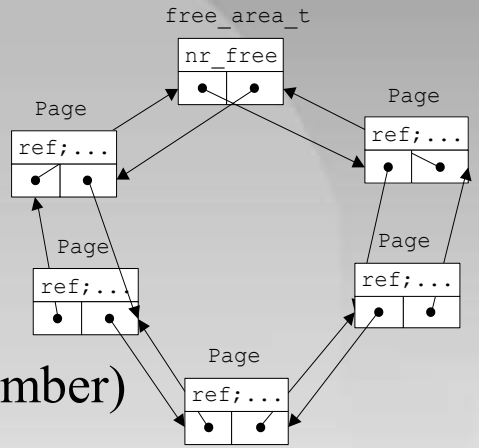
- **链表操作函数**
  - list_init(list_entry_t *elm)
  - list_add_after和list_add_before
  - list_del(list_entry_t *listelm)
- **访问链表节点所在的宿主数据结构**




```
free_area_t free_area;
list_entry_t * le = &free_area.free_list;
while((le=list_next(le)) != &free_area.free_list) {
      struct Page *p = le2page(le, page_link);        ……
}
```

了解ucore编程方法和通用数据结构

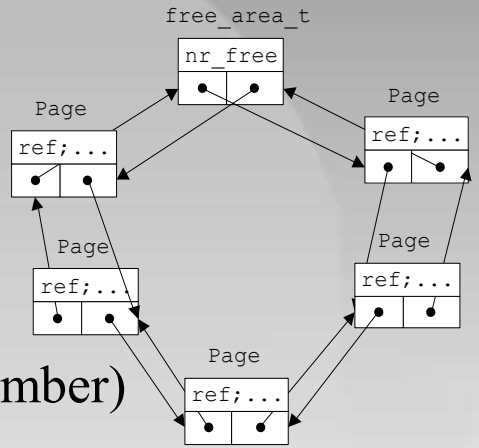
- **链表操作函数**
  - list_init(list_entry_t *elm)
  - list_add_after和list_add_before
  - list_del(list_entry_t *listelm)
- **访问链表节点所在的宿主数据结构**



#define le2page(le, member) to_struct((le), struct Page, member)

了解ucore编程方法和通用数据结构

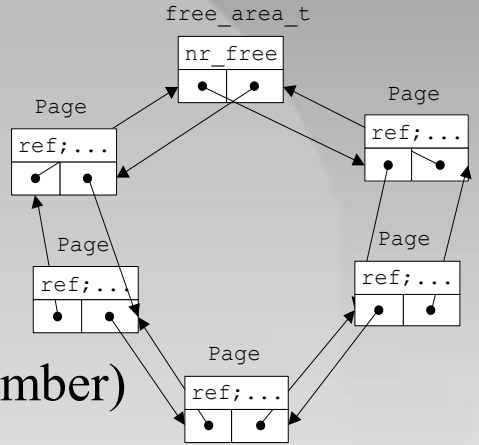
- **链表操作函数**
  - list_init(list_entry_t *elm)
  - list_add_after和list_add_before
  - list_del(list_entry_t *listelm)
- **访问链表节点所在的宿主数据结构**




```
#define le2page(le, member)      to_struct((le), struct Page, member)

#define to_struct(ptr, type, member)                                \
    ((type *)((char *)(ptr) - offsetof(type, member)))
```

了解ucore编程方法和通用数据结构

- **链表操作函数**
  - list_init(list_entry_t *elm)
  - list_add_after和list_add_before
  - list_del(list_entry_t *listelm)
- **访问链表节点所在的宿主数据结构**




```
#define le2page(le, member)      to_struct((le), struct Page, member)

#define to_struct(ptr, type, member)                        \
    ((type *)((char *)(ptr) - offsetof(type, member)))

#define offsetof(type, member)                              \
    ((size_t)(&((type *)0)->member))
```

小结

- 安装实验环境
  - 在虚拟机上使用安装好的Ubuntu Linux实验环境
- 使用实验工具
  - shell命令：ls、cd、rm、pwd...
  - 系统维护工具：apt、git
  - 源码阅读与编辑工具：eclipse-CDT、understand、gedit、vim
  - 源码比较工具：diff、meld
  - 开发编译调试工具：gcc 、gdb 、make
  - 硬件模拟器：qemu
- 了解x86-32硬件
  - Intel 80386运行模式概述
  - Intel 80386内存架构概述
  - Intel 80386寄存器概述
- 了解ucore编程方法和通用数据结构
  - 面向对象编程方法
  - 通用数据结构