操作系统课程实验
Lab0：实验准备

陈　渝

清华大学计算机系

大纲

- 安装实验环境
  - 在虚拟机上使用安装好的ubuntu实验环境

大纲

- 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：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

大纲

- 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：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

大纲

- 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：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

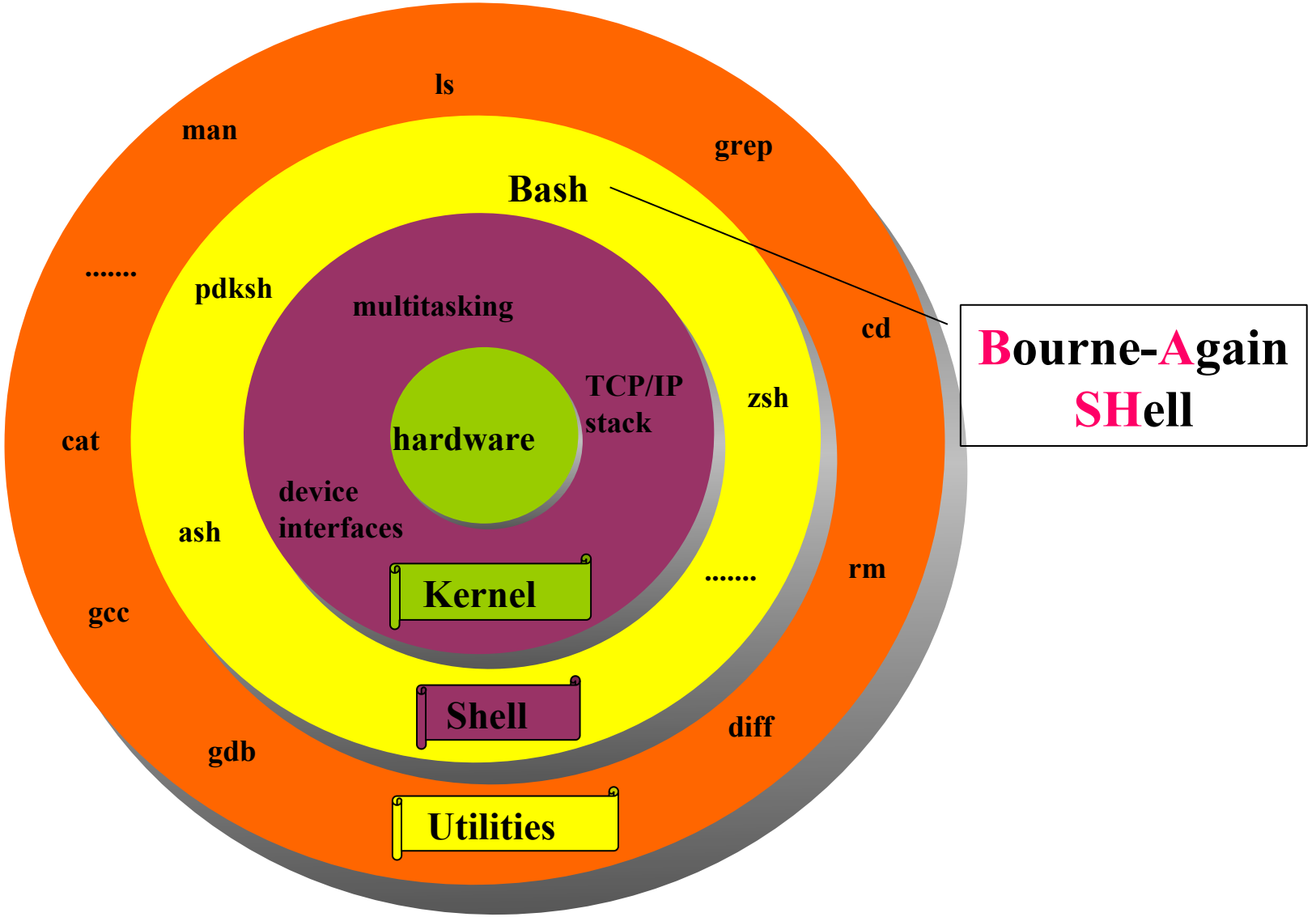
安装实验环境

- 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位于
    http://pan.baidu.com/s/1gdePM6J
  - 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>

大纲

- 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：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

使用命令行工具和实验工具



使用命令行工具和实验工具

- 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

使用命令行工具和实验工具

- 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

使用命令行工具和实验工具

- 源码阅读与编辑工具：understand、gedit、vim
  - 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

使用命令行工具和实验工具

- 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

使用命令行工具和实验工具

- 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。

使用命令行工具和实验工具

- 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

大纲

- 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：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。

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。

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-内存架构

- 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

了解x86-32硬件-内存架构

- 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。

了解x86-32硬件-内存架构

- 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看到的地址空间。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。

了解x86-32硬件-内存架构

- 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-内存架构

- 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, ...

大纲

- 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：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。

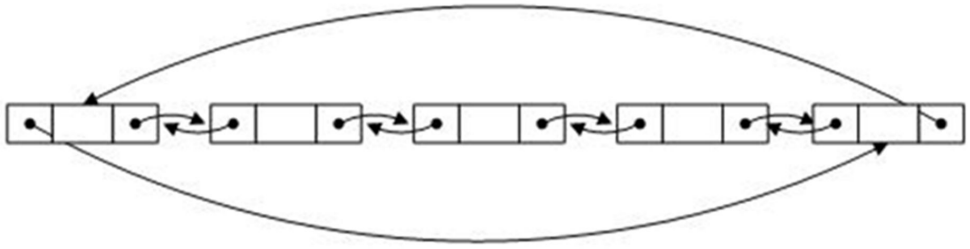
/lab2/kern/mm/pmm.h

```
------------------------------
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;
```



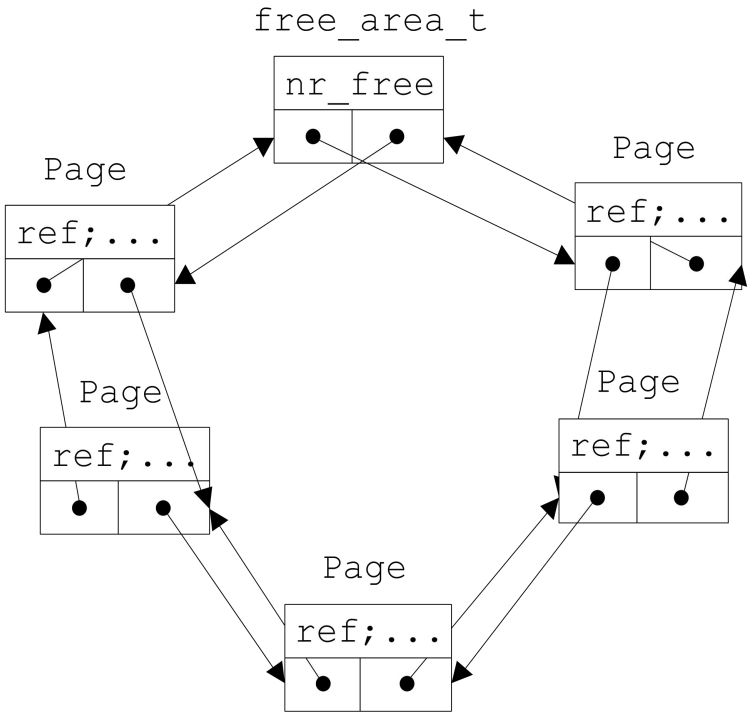
双向循环链表

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;
};
```

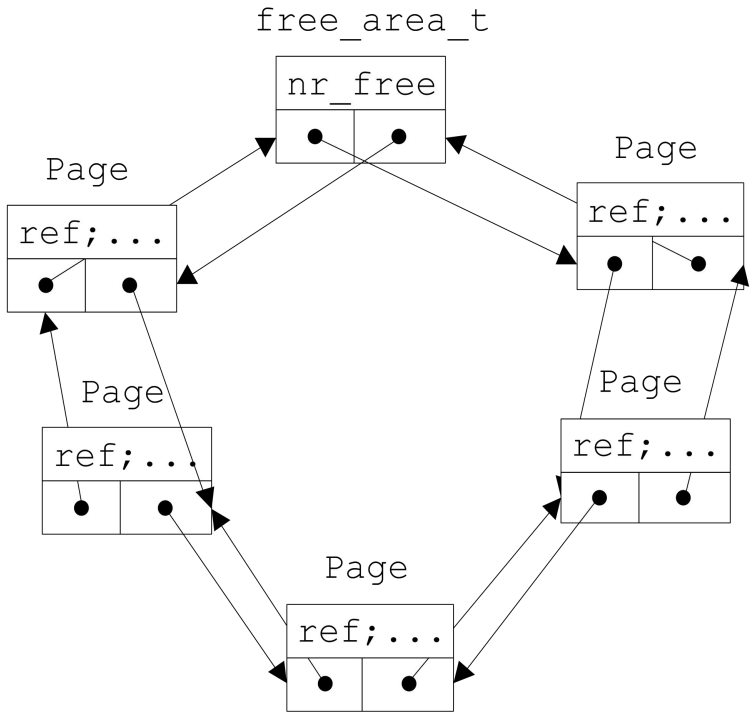


了解ucore编程方法和通用数据结构

- 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;
```



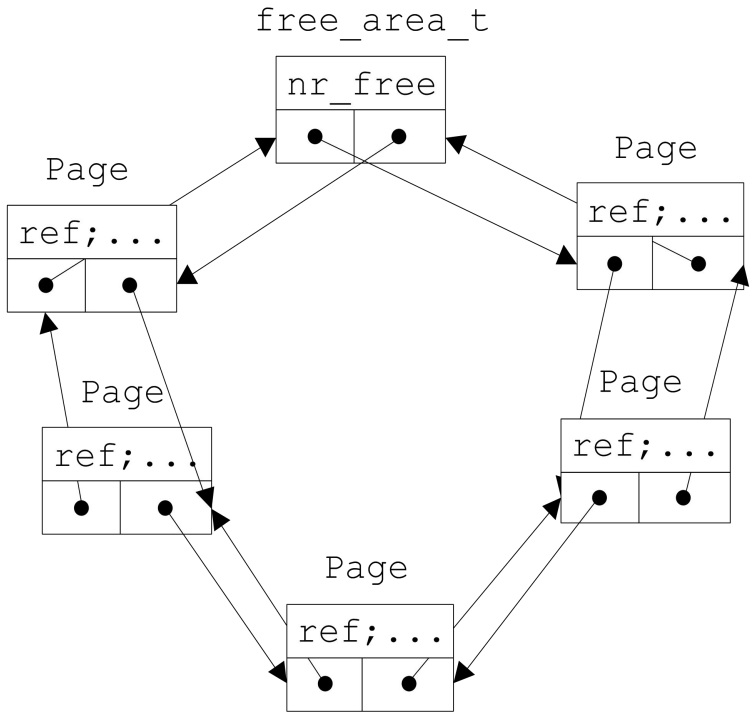
了解ucore编程方法和通用数据结构

- 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;
};
```

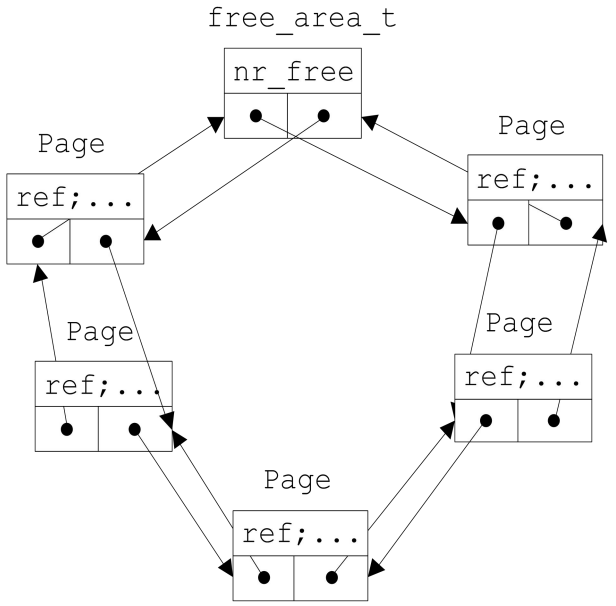


了解ucore编程方法和通用数据结构

- 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)

了解ucore编程方法和通用数据结构

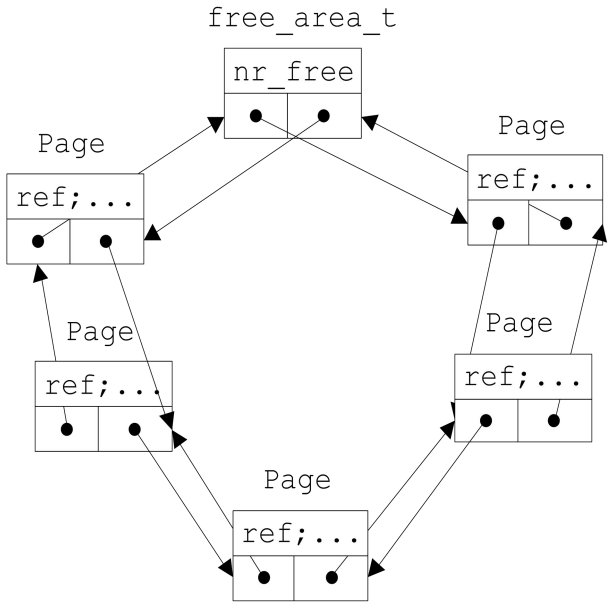
- 链表操作函数
  - list_init(list_entry_t *elm)
  - list_add_after和list_add_before
  - list_del(list_entry_t *listelm)
- 访问链表节点所在的宿主数据结构




```
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

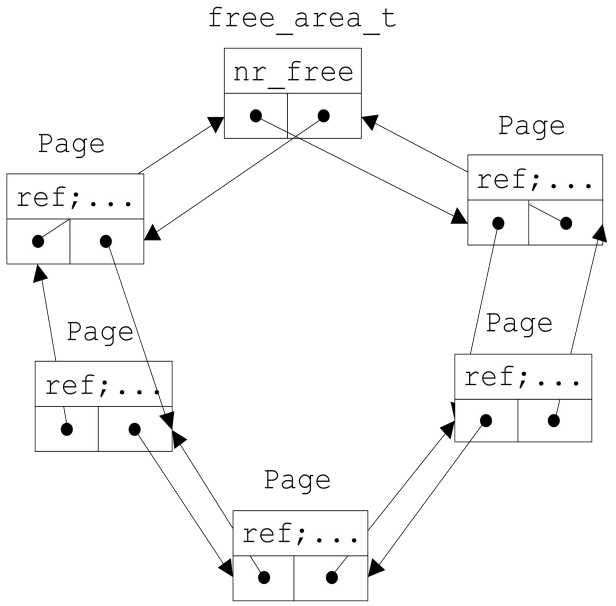
- 链表操作函数
  - list_init(list_entry_t *elm)
  - list_add_after和list_add_before
  - list_del(list_entry_t *listelm)
- 访问链表节点所在的宿主数据结构



#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

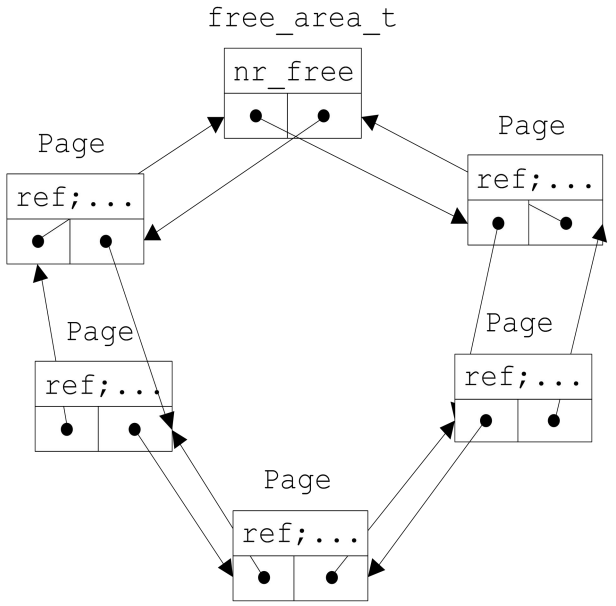



```
#define le2page(le, member)      to_struct((le), struct Page, member)

#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))
```