

# 伙伴系统的功能实现



—:

- 代表 **空闲内存块**，即当前未被分配的页块。
- 在 Buddy 系统中，这种符号通常用于表示可以分配或未占用的内存。

!:

- 可能表示 **已分配的内存块**。
- 这些内存块已经被某个进程或程序使用，暂时不可被分配给其他任务。

?:

- 用于标记 **不确定或错误状态** 的页块。
- 例如，某些页块可能被分配了但未正确初始化，或由于程序错误进入了异常状态。

符号按照一定的行和列分布，对应不同大小的页块：

- 一行表示一个内存区段。
- 每个符号代表一个具体的页块（如4KB或8KB大小）。

通过符号的分布，可以快速观察哪个区域是空闲的、哪个区域被占用了。

## Buddy 系统的状态

总页数:

- 图片中标注了

```
buddy_total_pages: ea
```

和

```
buddy_total_pages: dc00
```

:

- `ea` (十六进制 234) 页，表示第一个状态下系统有 234 页。
- `dc00` (十六进制 56320) 页，表示内存规模更大。

Buddy 级别:

- `buddy_queues (4K-4M): 2 2 1 2 1 2 0 0 0`

:

- 这一队列信息表示在不同大小的 Buddy 分块中（从4KB到4MB），每种块的空闲数目。
- 例如：
  - 第一位 `2` 表示 4KB 块有 2 个空闲。
  - 第二位 `2` 表示 8KB 块有 2 个空闲。
  - 以此类推，最后几位 `0` 表示更大的块（如2MB、4MB）没有空闲。

## 相关代码的改动

### 1.新增list.h文件

这个文件定义了 `list_head_t` 结构体，表示链表的节点，使用 `prev` 和 `next` 指针连接形成双向链表。特别是它使用了循环链表（当链表为空时，头节点的 `next` 和 `prev` 指向自己）。这种结构对于高效的内存分配、回收和合并操作非常重要，尤其在伙伴系统中，内存块需要频繁地进行合并与拆分。

### 在伙伴系统中的作用

在伙伴系统的内存管理中，通常需要处理内存块的分配和回收，以及块的合并和拆分。链表的结构为这些操作提供了便捷的支持。以下是几个具体的应用场景：

**内存块的管理：**可以使用链表来管理不同大小的空闲内存块。每个空闲块的大小可以作为链表节点的一部分，而这些块按照大小或者其他标准组织成链表。每当有一个内存请求时，你可以快速地从链表中查找合适大小的空闲块。

**内存块的合并与拆分：**当释放一个内存块时，如果相邻的块也为空闲块，可以将它们合并为一个更大的块。通过链表操作，可以非常方便地进行这种合并。例如，通过 `list_del` 可以删除一个节点，`list_add` 可以将合并后的块重新加入链表。

**内存分配与回收的高效性：**使用循环链表来组织内存块，避免了对链表头尾的频繁操作，确保了内存分配和回收的效率。在伙伴系统中，链表为内存块的管理提供了灵活性和扩展性。

```
#ifndef QQ3_LIST_H
#define QQ3_LIST_H
/*author qq3小组
*/

/*the host structures will be organised into a circle link-list */

typedef struct list_head{
    struct list_head *prev;
    struct list_head *next;
}list_head_t;

#define INIT_LIST_HEAD(l)\
do{\
    (l)->prev = (l)->next = l;\
} while(0)

static inline void __list_add(list_head_t *new, list_head_t *prev,
                              list_head_t *next){
    new->next = next;
    next->prev = new;
    new->prev = prev;
    prev->next = new;
}
```

## • 代码主要内容简介

### 1. 数据结构

#### list\_head\_t

- **描述：**双向循环链表的节点结构。每个节点有两个指针，`prev` 和 `next`，分别指向前一个和后一个节点，链表是循环的，即头节点的 `next` 指向头节点本身，尾节点的 `prev` 指向尾节点本身。
- **作用：**用于管理内存块或其他结构的链表。

### 2. 宏定义

#### INIT\_LIST\_HEAD(l)

- **描述：**初始化链表头节点。
- **功能：**将链表头的 `prev` 和 `next` 指向自己，表示链表为空。

### 3. 链表操作函数

#### `__list_add(list_head_t *new, list_head_t *prev, list_head_t *next)`

- **描述：**在链表中插入一个新的节点。
- **功能：**将 `new` 节点插入到 `prev` 和 `next` 之间。

#### `list_add(list_head_t *new, list_head_t *head)`

- **描述：**将一个新的节点插入到链表的头部。
- **功能：**调用 `__list_add`，将节点插入到链表头部，`head` 成为新节点的前驱。

#### `list_add_tail(list_head_t *new, list_head_t *head)`

- **描述：**将一个新的节点插入到链表的尾部。
- **功能：**调用 `__list_add`，将节点插入到链表尾部，`head` 的 `prev` 成为新节点的后继。

#### `__list_del(list_head_t *prev, list_head_t *next)`

- **描述：**从链表中删除一个节点。
- **功能：**通过调整 `prev` 和 `next` 指针，删除指定的节点。

#### `list_del(list_head_t *entry)`

- **描述：**删除链表中的指定节点。
- **功能：**调用 `__list_del`，通过修改前后节点的指针将 `entry` 节点移出链表。

#### `list_del_init(list_head_t *entry)`

- **描述：**删除链表中的指定节点并重新初始化它。
- **功能：**删除节点后，重新初始化该节点的 `prev` 和 `next` 指针，使其变为一个空链表节点。

#### `list_empty(list_head_t *entry)`

- **描述：**检查链表是否为空。
- **功能：**判断链表头的 `next` 是否指向头节点自己，若是，则链表为空。

#### `list_meet_tail(list_head_t *first, list_head_t *entry)`

- **描述：**检查指定节点是否是链表尾节点。
- **功能：**判断节点的 `next` 是否指向链表头，若是，说明该节点是链表的尾节点。

### 4. 宏定义与辅助函数

#### `LIST_FIND2(stru_t, mb_t, root, key, value, result)`

- **描述：**在链表中查找具有特定键值的节点。
- **功能：**遍历链表，查找与指定键值 `value` 匹配的节点，将匹配节点指针存储在 `result` 中。

## hashtable\_add(list\_head\_t \*hashtable, int hash, list\_head\_t \*new)

- **描述**: 将一个新节点插入到哈希表中。
- **功能**: 根据哈希值将节点插入到对应的哈希表位置。

## MB2STRU(stru\_type, mb\_addr, mb\_name)

- **描述**: 通过链表节点指针反向查找并获取包含该节点的结构体指针。
- **功能**: 通过给定的成员名 `mb_name` 计算出结构体的基地址, 进而返回包含该节点的结构体指针。

## container\_of(head, stru, member)

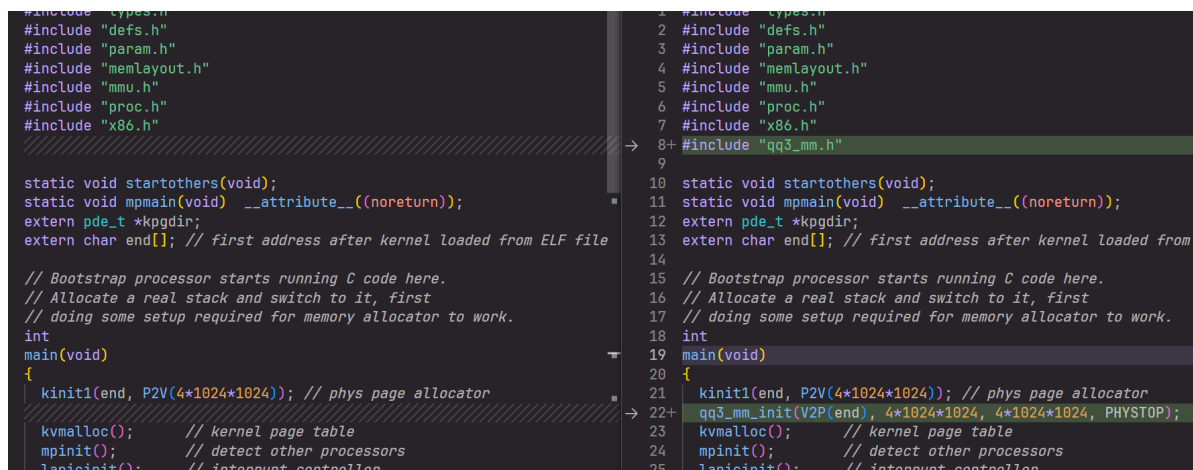
- **描述**: 获取链表节点所对应的结构体指针。
- **功能**: 通过链表头节点 `head` 和成员名 `member` 计算出该节点对应的结构体实例。

## list\_for\_each\_safe(root, container, mbname)

- **描述**: 安全地遍历链表。
- **功能**: 遍历链表并对每个节点执行操作, 同时在遍历过程中安全地删除节点。

## 2.修改main.c

新增了对 `qq3_mm_init` 函数的调用, 它负责初始化伙伴系统内存管理, 替代了原本 `kinit2` 的单一物理内存分配功能。伙伴系统管理内存的分配和回收, 提高了内存分配的效率和灵活性。  
具体的`qq3_mm_init`函数实现后续会介绍。



```
#include <types.h>
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"

static void startothers(void);
static void mpmain(void) __attribute__((noreturn));
extern pde_t *kpgdir;
extern char end[]; // first address after kernel loaded from ELF file

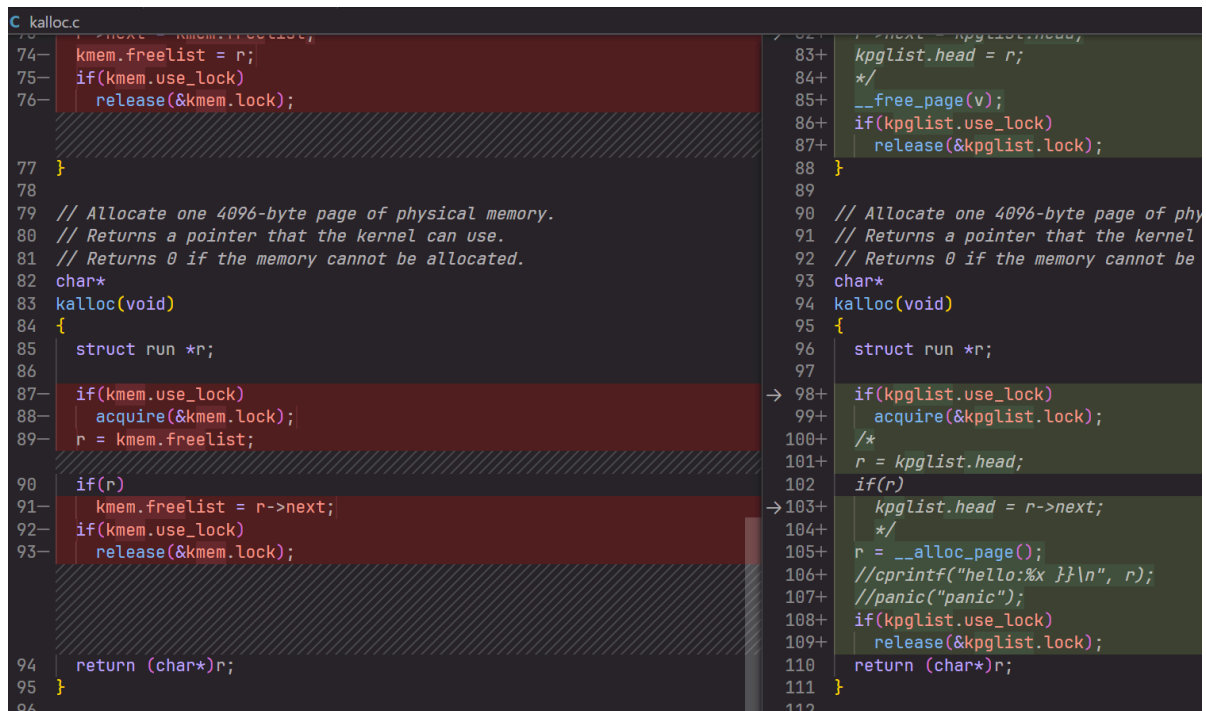
// Bootstrap processor starts running C code here.
// Allocate a real stack and switch to it, first
// doing some setup required for memory allocator to work.
int
main(void)
{
    kinit1(end, P2V(4*1024*1024)); // phys page allocator
    kvmalloc(); // kernel page table
    mpinit(); // detect other processors
    lapicinit(); // interrupt controller

    // ... (other code) ...

    qq3_mm_init(V2P(end), 4*1024*1024, 4*1024*1024, PHYSTOP);
    kvmalloc(); // kernel page table
    mpinit(); // detect other processors
    lapicinit(); // interrupt controller
}
```

## 3.修改kalloc.c

修改后的 `kalloc.c` 文件内容通过集成伙伴系统的功能, 改动了内存分配和回收机制, 使其能够更高效地分配和管理物理内存块。



## 主要改动点与作用

### 1. 使用伙伴系统替代原有的链表管理机制

**原代码：** 使用一个简单的链表 `freelist` 来管理空闲页块。空闲页块以链表节点的形式连接，分配时从头部取一个节点，回收时将页块重新插入到链表头部。

```
struct {
    struct spinlock lock;
    int use_lock;
    struct run *freelist;
} kmem;
```

**改动后：** 重命名为 `kpglist`，但更重要的是，用伙伴系统的分配和回收方法取代了原本的链表操作。伙伴系统的分配和回收通过新增的 `__alloc_page` 和 `__free_page` 函数实现，而非直接操作链表。

```
struct {
    struct spinlock lock;
    int use_lock;
    struct run *head;
} kpglist;
```

**作用：** 伙伴系统的内存管理具有更高效的内存块合并和拆分能力，可以减少内存碎片化问题，相比链表管理具有更高的性能和灵活性。

### 2. 修改 kinit1和kinit2 的初始化逻辑

**原代码：** 在 `kinit1` 和 `kinit2` 中，通过 `freerange` 初始化物理内存块，并将其添加到空闲链表中。

```
void kinit1(void *vstart, void *vend) {
    initlock(&kmem.lock, "kmem");
    kmem.use_lock = 0;
    freerange(vstart, vend);
}

void kinit2(void *vstart, void *vend) {
    freerange(vstart, vend);
    kmem.use_lock = 1;
}
```

**改动后：**注释掉了 `freerange` 调用，改为通过伙伴系统管理内存块，同时增加了一个全局变量 `global_use_clock` 来控制伙伴系统的初始化阶段。

```
void kinit1(void *vstart, void *vend) {
    initlock(&kpglist.lock, "kpglist");
    kpglist.use_lock = 0;
    global_use_clock = 0;
}

void kinit2(void *vstart, void *vend) {
    kpglist.use_lock = 1;
    global_use_clock = 1;
}
```

**作用：**通过伙伴系统的初始化逻辑，替代了原有的 `freerange` 方法，优化了内存初始化过程。新增的全局变量 `global_use_clock` 可能用于全局同步伙伴系统的状态。

### 3. 内存回收 (\*\*kfree\*\*) 的改动

**原代码：**使用链表管理空闲内存页，通过将释放的页块插入到链表头部来实现回收。

```
r = (struct run*)v;
r->next = kmem.freelist;
kmem.freelist = r;
```

**改动后：**使用 `__free_page(v)` 函数处理内存回收，而不再直接操作链表。

```
__free_page(v);
```

**作用：**将内存回收的逻辑交给伙伴系统管理，这样可以自动处理空闲块的合并操作，提高内存利用效率，并减少碎片化。

### 4. 内存分配 (\*\*kalloc\*\*) 的改动

**原代码：**从链表头部获取一个空闲页块。

```
r = kmem.freelist;
if(r)
    kmem.freelist = r->next;
```

**改动后：**改为使用 `__alloc_page` 函数分配页块，并注释掉了原链表操作。

```
r = __alloc_page();
```

**作用：**伙伴系统通过 `__alloc_page` 提供了更智能的分配机制，允许分配不同大小的内存块，同时保留空闲块的合并和拆分能力。

## 5. `freerange` 逻辑的保留

虽然 `freerange` 函数在 `kinit1` 和 `kinit2` 中被注释掉了，但其定义仍然存在，是为了兼容或者作为备用逻辑。

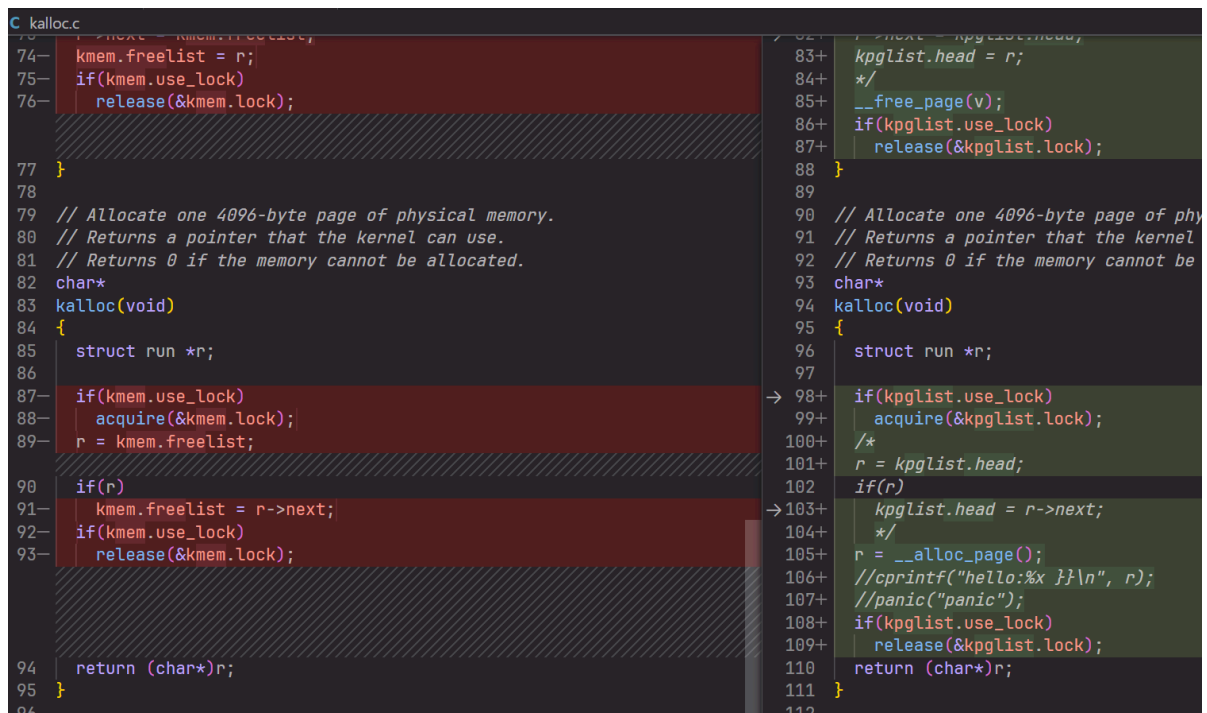
## 新增函数的作用

### `__alloc_page`

- **描述：**伙伴系统分配页块的核心函数。
- **作用：**根据伙伴系统的规则，从内存池中分配一个页块。

### `__free_page`

- **描述：**伙伴系统释放页块的核心函数。
- **作用：**将回收的页块重新加入伙伴系统的内存池中，并根据需要合并相邻空闲块。



```
C kalloc.c
74- kmem.freelist = r;
75- if(kmem.use_lock)
76-     release(&kmem.lock);
77- }
78-
79- // Allocate one 4096-byte page of physical memory.
80- // Returns a pointer that the kernel can use.
81- // Returns 0 if the memory cannot be allocated.
82- char*
83- kalloc(void)
84- {
85-     struct run *r;
86-
87-     if(kmem.use_lock)
88-         acquire(&kmem.lock);
89-     r = kmem.freelist;
90-
91-     if(r)
92-         kmem.freelist = r->next;
93-     if(kmem.use_lock)
94-         release(&kmem.lock);
95-
96-     return (char*)r;
97- }
98-
99- // next = kpglist.head;
100-
101- kpglist.head = r;
102- /*
103-  * __free_page(v);
104-  * if(kpglist.use_lock)
105-  *     release(&kpglist.lock);
106-  */
107- }
108-
109- // Allocate one 4096-byte page of phy
110- // Returns a pointer that the kernel
111- // Returns 0 if the memory cannot be
112- char*
113- kalloc(void)
114- {
115-     struct run *r;
116-
117-     if(kpglist.use_lock)
118-         acquire(&kpglist.lock);
119-     /*
120-      * r = kpglist.head;
121-      * if(r)
122-      *     kpglist.head = r->next;
123-      */
124-     r = __alloc_page();
125-     //cprintf("hello:%x }\n", r);
126-     //panic("panic");
127-     if(kpglist.use_lock)
128-         release(&kpglist.lock);
129-     return (char*)r;
130- }
```

# 4.新建.Bochsrc和bochsout并修改dot-bochsrc文件

## 1. `.bochsrc` 配置文件

`.bochsrc` 是 Bochs 模拟器的核心配置文件，它控制了整个虚拟机的行为和硬件模拟参数。这对我们演示伙伴系统的机制是非常必要的。



## 作用：

- 虚拟环境的基础配置：
  - 设置 CPU 参数（核心数量、IPS）。
  - 设置内存大小（如 `megs: 32`）。
  - 设置硬盘和启动顺序（如 `boot: disk`）。
- 伙伴算法运行的环境保障：
  - 确保模拟器能正确加载和运行操作系统（如加载 `xv6.img` 和 `fs.img`）。
- 定义外围设备（如 `floppya` 和 `ata0-master`）以确保文件系统和内核能够运行伙伴算法。

## 为什么需要它？

- `.bochsrc` 是运行伙伴算法所需虚拟环境的配置入口。如果没有正确配置，模拟器将无法正常运行伙伴算法所依赖的操作系统和相关程序。

```
# nas a selection of about 10 different display library implementations to
# different platforms. If you run configure with multiple --with-* option
# the display_library command lets you choose which one you want to run with
# If you do not write a display_library line, Bochs will choose a default for
# you.
#
# The choices are:
#   x                use X windows interface, cross platform
#   win32            use native win32 libraries
#   carbon           use Carbon library (for MacOS X)
#   beos             use native BeOS libraries
#   macintosh        use MacOS pre-10
#   amigaos          use native AmigaOS libraries
#   sdl              use SDL library, cross platform
#   svga             use SVGALIB library for Linux, allows graphics without
#   term             text only, uses curses/ncurses library, cross platform
#   rfb              provides an interface to AT&T's VNC viewer, cross platform
#   wx               use wxWidgets library, cross platform
#   nogui            no display at all
#
# NOTE: if you use the "wx" configuration interface, you must also use
# the "wx" display library.
#
# Specific options:
# Some display libraries now support specific option to control their
# behaviour. See the examples below for currently supported options
```

## 2. bochsout 日志文件

`bochsout` 是 Bochs 模拟器的输出日志文件，它记录了模拟器运行期间的所有信息，包括错误、警告和调试信息。

## 作用：

- 运行过程的追踪和诊断：
  - 如果伙伴算法运行失败，`bochsout` 中的日志可以帮助你定位错误是发生在操作系统加载阶段、文件系统访问阶段，还是伙伴算法本身。
- 性能监控和优化：

- 日志中可能包含 CPU 指令执行情况和内存使用情况，这对优化伙伴算法的运行效率很有帮助。

## 为什么需要它？

- `bochsout` 是调试伙伴算法的重要工具。如果没有日志，你将无法了解虚拟机内部的运行情况，尤其是当算法因虚拟硬件配置问题或软件错误崩溃时，日志是唯一的线索。

```
00000000000i[ ] Bochs x86 Emulator 2.6.8
00000000000i[ ] Built from SVN snapshot on May 3, 2015
00000000000i[ ] Compiled on Dec 20 2024 at 06:03:53
00000000000i[ ] System configuration
00000000000i[ ] processors: 2 (cores=1, HT threads=1)
00000000000i[ ] A20 line support: yes
00000000000i[ ] IPS is set to 10000000
00000000000i[ ] CPU configuration
00000000000i[ ] SMP support: yes, quantum=16
00000000000i[ ] level: 6
00000000000i[ ] APIC support: xapic
00000000000i[ ] FPU support: yes
00000000000i[ ] MMX support: yes
00000000000i[ ] 3dnow! support: no
00000000000i[ ] SEP support: yes
00000000000i[ ] SIMD support: sse2
00000000000i[ ] XSAVE support: no
00000000000i[ ] AES support: no
00000000000i[ ] SHA support: no
00000000000i[ ] MOVBE support: no
00000000000i[ ] ADX support: no
00000000000i[ ] x86-64 support: yes
00000000000i[ ] 1G paging support: no
00000000000i[ ] MWAIT support: yes
00000000000i[ ] VMX support: 1
00000000000i[ ] Optimization configuration
00000000000i[ ] RepeatSpeedups support: no
00000000000i[ ] Fast function calls: no
00000000000i[ ] Handlers Chaining speedups: no
00000000000i[ ] Devices configuration
00000000000i[ ] NE2000 support: no
00000000000i[ ] PCI support: yes, enabled=yes
00000000000i[ ] SB16 support: no
00000000000i[ ] USB support: no
00000000000i[ ] VGA extension support: vbe
00000000000i[MEM0 ] allocated memory at 0x7fea607f5010. after alignment, vector=0x7fea607f5010
00000000000i[MEM0 ] 32.00MB
00000000000i[MEM0 ] mem block size = 0x00100000, blocks=32
00000000000i[MEM0 ] rom at 0xffff0000/131072 ('/usr/local/share/bochs/BIOS-bochs-latest')
00000000000i[PLUGIN] init dev of 'pci' plugin device by virtual method
00000000000i[DEV ] i440FX PMC present at device 0, function 0
00000000000i[PLUGIN] init dev of 'pci2isa' plugin device by virtual method
00000000000i[DEV ] PIIX3 PCI-to-ISA bridge present at device 1, function 0
```

## 3. `dot-bochsrc` 配置文件

`dot-bochsrc` 是另一个模拟器配置文件，它的作用类似于 `.bochsrc`，但针对的是一些补充性或特定场景的配置。

### 作用：

- 自定义补充配置：
  - `dot-bochsrc` 中可能存在一些未在 `.bochsrc` 中指定的额外选项（如显示模式、鼠标、键盘类型等），这些选项可以为伙伴系统提供更加灵活的支持。
- 快速切换配置：

- 如果需要对伙伴系统进行不同硬件环境的测试，`dot-bochsrc` 提供了一种方便的方式来覆盖或补充默认配置。

## 为什么需要它？

- 在复杂的模拟环境中，可能需要为不同的测试场景提供不同的配置文件（如针对伙伴算法的显示优化、输入设备配置等），`dot-bochsrc` 就充当了一个额外的调试和测试工具。

```
# You may now use double quotes around pathnames, in case
# your pathname includes spaces.

=====
# CONFIG INTERFACE
#
# The configuration interface is a series of menus or dialog boxes that
# allows you to change all the settings that control Bochs's behavior.
# There are two choices of configuration interface: a text mode version
# called "textconfig" and a graphical version called "wx". The text
# mode version uses stdin/stdout and is always compiled in. The graphical
# version is only available when you use "--with-wx" on the configure
# command. If you do not write a config interface line, Bochs will
# choose a default for you.
#
# NOTE: if you use the "wx" configuration interface, you must also use
# the "wx" display library.
=====
#config interface: textconfig
#config interface: wx

=====
# DISPLAY LIBRARY
#
# The display library is the code that displays the Bochs VGA screen. Bochs
# has a selection of about 10 different display library implementations for
# different platforms. If you run configure with multiple --with-* options,
# the display library command lets you choose which one you want to run with.
# If you do not write a display library line, Bochs will choose a default for
# you.
#
# The choices are:
# x          use X windows interface, cross platform
# win32      use native win32 libraries
# carbon     use Carbon library (for MacOS X)
# beos       use native BeOS libraries
# macintosh  use MacOS pre-10
# amigaos    use native AmigaOS libraries
# sdl        use SDL library, cross platform
# svga       use SVGALIB library for Linux, allows graphics without X11
# term       text only, uses curses/ncurses library, cross platform
# rfb        provides an interface to AT&T's VNC viewer, cross platform
```

# 5.qq3\_page.h- 页面描述和虚拟/物理地址映射

```
#ifndef QQ3_PAGE_H
#define QQ3_PAGE_H
/*author qq3小组
*/

#include"memlayout.h"

#define PAGE_SHIFT 12
#define PAGE_SIZE 0x1000
#define PAGE_MASK (~0xfff)
#define pa_idx(paddr) (((unsigned)paddr)>>PAGE_SHIFT)
#define pa_pg pa_idx

/*TODO cancell the following two macros, and 'vpg', 'ppg' is a good transfe
#define PG_H10(pg_id) (pg_id>>10)
#define PG_L10(pg_id) (pg_id&(0x400-1))

/* page table/directory entry ==> linear address of target page */

// >一个普通的整形变成了union，其实是反而隐藏了类型信息。但不妨一试。
// 毕竟pte是特殊的，常常知道pte是个unsigned。
#pragma pack(push)
#pragma pack(1)
union pte{
    int value;
    struct {
        unsigned present: 1;
        unsigned writable: 1;
        unsigned user: 1;
        unsigned PWT: 1;
        unsigned PCD: 1;
        unsigned accessed: 1;
        unsigned dirty: 1;
        unsigned : 2;
        unsigned avl: 3;
    }
};
```

## 主要结构和功能：

- **pte (Page Table Entry) :**
  - 这是一个联合体，包含了页面表项的标志位和物理页面的地址。
  - 页面表项用于描述页表中的每个页面，包含如是否存在、是否可写、是否是用户空间可访问、页面是否已经被访问等标志。具体来说，结构体中的每一位表示一个标志位。
  - **physical** 字段表示该页面所在的物理地址，配合其他标志一起使用。
- **linear\_addr:**
  - 这是一个联合体，用于描述虚拟地址。通过将虚拟地址分成 **offset**、**tbl\_idx**（表项索引）和 **dir\_idx**（目录项索引）三个部分，帮助实现虚拟地址到物理地址的转换。

- `cr3` :
  - `cr3` 是 x86 架构下用于存储当前页目录基址的寄存器。它包含了一个物理地址，用于指向页目录。在这里，它用于表示页表的物理地址。
- `__pa` 和 `__va` 宏：
  - `__pa(vaddr)`：将虚拟地址 `vaddr` 转换为物理地址。
  - `__va(paddr)`：将物理地址 `paddr` 转换为虚拟地址。
  - 这些宏是操作系统中的关键部分，它们为伙伴算法的内存管理提供了物理地址和虚拟地址之间的转换接口。
- `pte2page()` 和 `__va2pte()` 函数：
  - `pte2page()` 用来将一个页表项转换为指向物理页面的指针。
  - `__va2pte()` 函数根据虚拟地址和页目录查找对应的页表项，用于访问内存。

这些定义和宏确保了操作系统能够通过页表操作虚拟内存和物理内存之间的转换，从而支持页面的分配与回收，这对于伙伴算法的实现至关重要。

## 6. `qq3_mm.h` - 内存管理的核心结构和函数声明

---

`qq3_mm.h` 文件定义了与内存分配和回收相关的核心数据结构，并声明了与伙伴算法实现相关的内存管理函数。

```

qq3_mm.h
1  #ifndef QQ3_MM_H
2  #define QQ3_MM_H
3  /* author qq3小组
4  **/
5
6  #include "types.h"
7  #include "defs.h"
8  #include "qq3_list.h"
9  #include "qq3_page.h"
0  #include "spinlock.h"
1
2
3  /*physical page descriptor
4  * 页描述暂不设锁，由上层路径加锁
5  */
6  typedef struct page{
7      struct list_head lru;    //用于buddy系统的链表头
8      int _count;
9      //代表该page作为buddy头的order,最大10, 约定32为特殊值表示已被分配
0      //约定负数表示作为buddy followee
1      char private;
2      char zone_id;
3  } page_t;
4
5
6  #define page_idx(page_t) ((unsigned)((page_t) - mem_map))
7  // #define pte_pfn(pte) ((pte)>>PAGE_SHIFT)
8  // #define pfn_page(pfn) (mem_map + (pfn))
9  // #define pte_page(pte) ( pfn_page( pte_pfn(pte) ) )
0  // #define page_va(page) __va( (page - mem_map) << PAGE_SHIFT)
1  // #define virt_to_page(vaddr) pfn_page( __pa(vaddr) >> PAGE_S
2
3  #define MAX_ORDER 10

```

## 主要结构和功能:

- `struct page`:
  - 该结构体代表每个物理页面。在伙伴算法中，每个 `struct page` 都对应系统中的一个物理页面。
  - 它可能包含一些元数据，如与其他页面的链接、空闲标志、页的大小等。在伙伴算法中，每个页面需要被标记和管理，以便进行有效的分配和释放。
- `struct free_area`:
  - `free_area` 用于管理不同大小的空闲块。在伙伴算法中，内存会被划分成若干大小的块，每个大小对应一个链表。`free_area` 包含一个链表，链表中的每个元素都是一个空闲内存块。每个内存块的大小是 2 的幂。
  - 在分配内存时，系统会在 `free_area` 中查找足够大的空闲块。如果没有找到合适的块，系统会递归地将更大的块拆分成小块。
- `alloc_pages()` 和 `free_pages()`:

- `alloc_pages()` 用于从 `free_area` 中分配一个或多个页面。当系统收到一个内存分配请求时，它会检查 `free_area` 中是否有足够大的空闲块。如果没有，系统会继续拆分更大的块，直到满足请求的大小。
- `free_pages()` 用于释放页面并将其归还给 `free_area`。在释放页面时，系统会检查相邻的块是否可以合并。如果可以，系统会将相邻的块合并成更大的块，这样有助于减少内存碎片。
- **zone 和 pageblock:**
  - `zone` 是系统中一个内存区域的定义，通常由多个 `free_area` 构成，代表某个物理区域的内存。
  - `pageblock` 是一组连续页面的集合，通常用于批量分配和回收。`zone` 内的页面被划分为若干 `pageblock`，并按大小维护空闲列表。

## 7. qq3\_mm.c - 伙伴算法的实现

---

`qq3_mm.c` 文件实现了伙伴算法的核心逻辑，包括内存的分配、释放、拆分和合并等操作。



```

C qq3_mm.c
1  /* author qq3小组
2  */
3
4  #include "qq3_mm.h"
5  #include "qq3_page.h"
6  #include "qq3_global.h"
7
8
9  #define MEMBER_OFFSET(stru_type, member_name) \
10     ( (unsigned)&((stru_type *)0)->member_name) )
11
12 //演示free时的合并行为 屏幕打印 -表示发生一个合并 !表示不能继续合并, 入链
13 int debug_demo_free = 1;
14 //演示alloc时的拆分行 屏幕打印 %表示发生一个拆分 !表示order匹配终止拆
15 int debug_demo_alloc = 1;
16
17 int page_is_buddy(struct page *page, int order);
18 void init_free_area(int zone_id, int start_idx);
19 void __free_pages_bulk(struct page *page, zone_t *zone, int or
20 void cleave(free_area_t *free_area, int order);
21
22
23 struct spinlock page_big_lock;
24 #define PGNUM_MAX 1024*128
25 struct page mem_map[PGNUM_MAX]; //TODO 大数组, 过大会越过xv6的4M初
26 zone_t memzones[2];
27 void info_zone(zone_t *zone){
28     cprintf("\nbuddy total pages: %x \nbuddy quenes (4k-4M):",
29     free_area_t *area = zone->free_area;
30     for(int i = 0; i <= MAX_ORDER; i++){
31         cprintf("%x ", area[i].nr_free);
32     }
33     cprintf("\n");
34 }
35
36 void qq3_mm_init(unsigned va, unsigned end_va, unsigned va2, u
37     memset(mem_map, 0, PGNUM_MAX*sizeof(page_t));
38     initlock(&page_big_lock, "page_big_lock");

```

## 主要功能:

- `alloc_pages()`:
  - 该函数实现了内存的分配。它会遍历 `free_area` 中的空闲链表, 查找符合要求的空闲块。如果找到较大的块, 系统会继续将其拆分成更小的块, 直到找到满足请求的块。
  - 如果无法找到合适的块, 系统会继续向上查找更大的空闲块, 直到找到足够大的块或无法找到。
- `free_pages()`:
  - 该函数实现了内存的释放。在释放页面时, 系统会尝试将相邻的空闲块合并成一个更大的块, 以减少内存碎片。
  - 如果释放的块大小是 2 的幂次方, 它会检查相邻的块是否可以合并。如果可以, 系统会将其合并为更大的块, 并将合并后的块归还给 `free_area`。



- `cleave()` 和 `__free_pages_bulk()` :
  - `cleave()` 函数用于拆分较大的块，将其分为两个较小的块。它是实现伙伴算法的关键步骤之一。
  - `__free_pages_bulk()` 函数在释放多个页面时，会在释放时处理块的合并和拆分操作。
- 伙伴合并：
  - 在伙伴算法中，当释放的页面无法与邻近的页面合并时，它们会被加入到空闲列表中，并等待分配。通过合并相邻的块，可以有效减少内存碎片并提高内存利用率。

## 8.qq3\_global.h - 全局变量和宏定义

`qq3_global.h` 主要定义了一些全局变量、常量和宏，用于内存管理和伙伴算法的配置。

```
C qq3_global.h
1  #ifndef QQ3_GLOBAL_H
2  #define QQ3_GLOBAL_H
3  //author qq3小组
4
5  #include "types.h"
6  #include "defs.h"
7
8  int global_use_clock;
9  void assert_func(char*exp,char*file,char*base_file,int i)
10
11  #define cassert(exp)
12      do{
13          if(!(exp)) assert_func(#exp,__FILE__,__BASE_
14      } while(0)
15
16  #endif
17
```

### 主要功能：

- `V2P` 和 `P2V` 宏：
  - 这两个宏在系统中起到了非常重要的作用，它们分别用于虚拟地址与物理地址之间的转换。伙伴算法依赖于物理页面的分配和释放，而这些宏提供了从虚拟地址到物理地址的转换接口。
- `ZONE_SIZE` 和 `MAX_ORDER` 宏：
  - `ZONE_SIZE` 定义了每个内存区域的大小，而 `MAX_ORDER` 则定义了支持的最大内存块大小（即 `order`），这些参数控制了伙伴算法中内存块的划分和管理范围。
- `init_zone()` 和 `init_free_area()` :
  - 这两个函数用于初始化内存区域和空闲块链表，为伙伴算法的运行做准备。

## 9.qq3\_global.c - 全局内存管理初始化和信息输出

qq3\_global.c 文件负责内存管理的初始化，确保在系统启动时正确配置内存区域、空闲链表等。

```
C qq3_global.c
1  #include "qq3_global.h"
2
3  void assert_func(char*exp, char*file, char*base_fi
4      |    cprintf("assert failure>>>exp:%s,file:%s,bas
5      |    panic("\nASSERT SPIN !!!");
6  }
7
```

### 主要功能：

- `init_zone()` :
  - 初始化内存区域，为后续的内存管理工作做准备。通过这个函数，系统会为每个内存区域设置初始状态，并为每个区域的 `free_area` 链表分配内存。
- `info_zone()` :
  - 打印系统当前内存区域的信息，便于调试和验证内存管理是否正常工作。通过输出内存区域的状态，可以帮助开发者检查系统在初始化过程中是否存在问题。

## 10.qq3\_list.h - 链表操作

- `qq3_list.h` 提供了链表操作的工具函数，用于管理空闲页面链表。

```

C qq3_list.h
1  #ifndef QQ3_LIST_H
2  #define QQ3_LIST_H
3
4  /*the host structures will be organised into a circle link-list */
5
6  typedef struct list_head{
7      struct list_head *prev;
8      struct list_head *next;
9  }list_head_t;
10
11 #define INIT_LIST_HEAD(l)\
12     do{\
13         (l)->prev = (l)->next = l;\
14     } while(0)
15
16 static inline void __list_add(list_head_t *new, list_head_t *prev,
17                               list_head_t *next){
18     new->next = next;
19     next->prev = new;
20     new->prev = prev;
21     prev->next = new;
22 }
23
24 /**
25  * 1,assume only one entry in list, (new)'s 'next' pointer will point to
26  * 'head', and (head)'s prev will point to 'new'. So, the list is circle
27  */
28
29 static inline void list_add(list_head_t *new, list_head_t *head){
30     __list_add(new, head, head->next);
31 }
32
33 static inline void list_add_tail(list_head_t *new, list_head_t *head){

```

## 主要功能：

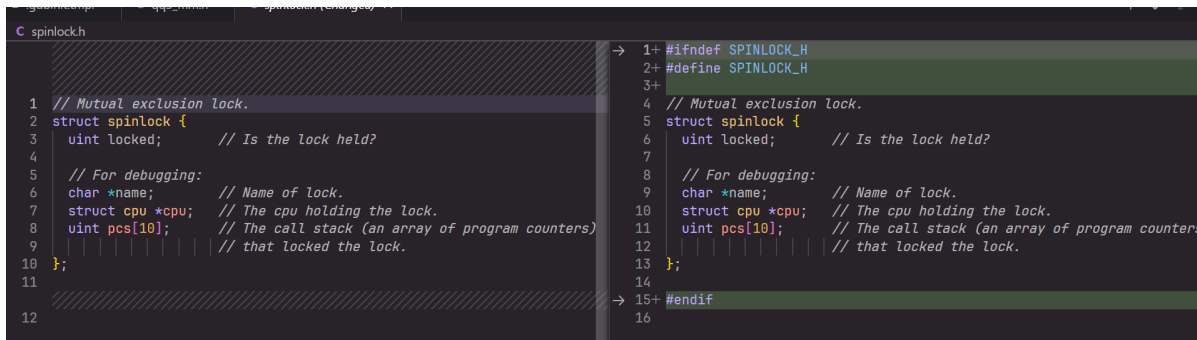
- `list_add()` 和 `list_del()` :
  - 这两个函数用于在链表中添加和删除元素。在伙伴算法中，当页面被分配或释放时，它们会被从 `free_area` 的链表中删除或添加。
- `list_for_each_safe()` :
  - 安全地遍历链表，在遍历过程中可以删除当前节点，而不会破坏链表的结构。对于内存的分配和释放，链表操作非常频繁，因此这些操作的安全性至关重要。

# 11.spinlock.h - 自旋锁的修改

自旋锁的修改主要为并发环境下的内存管理提供了保证。在多核系统中，多个处理器可能会同时访问和修改 `free_area` 链表，导致数据竞争问题。通过自旋锁保护对空闲链表的访问，可以确保每次只有一个 CPU 可以对内存块进行操作，避免了并发访问引发的错误。

**\*\*struct spinlock\*\***：定义了自旋锁的结构，确保操作系统在进行内存分配和回收时能够同步地访问共享资源。

**锁的保护**：在 `alloc_pages()` 和 `free_pages()` 等内存管理函数中，通过加锁和解锁的方式确保内存块的分配、释放以及合并操作是线程安全的。



## 12. tags - 方便对项目代码进行管理

包含了代码标记 (tags) 文件的内容, 用于项目中的代码导航和搜索工具 (如ctags) 来帮助开发人员在大型代码库中快速找到相关函数、结构体或变量的定义和引用。

```
> Users > 32951 > Desktop > xv6 > xv6-public-master > tags
1  |!_TAG_FILE_FORMAT  2  /extended format; --format=1 will not append ;" to lin
2  !_TAG_FILE_SORTED  1  /0=unsorted, 1=sorted, 2=foldcase/
3  !_TAG_PROGRAM_AUTHOR  Darren Hiebert  /dhiebert@users.sourceforge.net/
4  !_TAG_PROGRAM_NAME  Exuberant Ctags //
5  !_TAG_PROGRAM_URL  http://ctags.sourceforge.net    /official site/
6  !_TAG_PROGRAM_VERSION  5.9~svn20110310 //
7  ALT kbd.h  11;"    d
8  AS  Makefile  /^AS = $(TOOLPREFIX)gas$/" m
9  ASFLAGS Makefile  /^ASFLAGS = -m32 -gdwarf-2 -Wa,-divide$/" m
10 ASSERT  lapic.c 25;"    d  file:
11 Align  umalloc.c /^typedef long Align;$/" t  file:
12 BACK  sh.c  12;"    d  file:
13 BACKSPACE  console.c 127;" d  file:
14 BBLOCK  fs.h  48;"    d
15 BCAST  lapic.c 28;"    d  file:
16 BIG  ustests.c 1452;" d  file:
17 BPB  fs.h  45;"    d
18 BSIZE  fs.h  6;" d
19 BUSY  lapic.c 29;"    d  file:
20 B_DIRTY buf.h  13;"    d
21 B_VALID buf.h  12;"    d
22 C  console.c 189;" d  file:
23 C  kbd.h  32;"    d
24 CAPSLOCK  kbd.h  13;"    d
25 CC  Makefile  /^CC = $(TOOLPREFIX)gcc$/" m
26 CFLAGS Makefile  /^CFLAGS = -fno-pic -static -fno-builtin -fno-strict-alias
27 CMOS_PORT  lapic.c 123;" d  file:
28 CMOS_RETURN lapic.c 124;" d  file:
29 CMOS_STATA  lapic.c 163;" d  file:
30 CMOS_STATA  lapic.c 164;" d  file:
```