Report for lab2, YiChen Mao

Part 1: Physical Page Management

Exercise 1.

Tasks: 完善以下函数

```
boot_alloc()
mem_init() (only up to the call to “check_page_free_list(1)”)
page_init()
page_alloc()
page_free()
```

```
函数 boot_alloc() 简单的物理内存分配器，只有在 JOS 建立虚拟内存的过程中使用，真正的分配器是
page_alloc()。
当 n>0 时分配连续的 n bytes 物理内存，不初始化内存，返回首地址。（注意要和PGSIZE对齐）
当 n=0 时返回首地址，不需要分配内存。
但超出分配内存超出空间，boot_alloc 应该调用 panic 终止。
static void *
boot_alloc(uint32_t n)
{
  static char *nextfree;
  char *result;
  if (!nextfree) {
    extern char end[];
    nextfree = ROUNDUP((char *) end, PGSIZE);
  }
  if (n>0){
    result = nextfree;
    nextfree = ROUNDUP(nextfree + n, PGSIZE);
    if (PGNUM(PADDR(nextfree))>=npages)
```

```
      panic("boot_alloc: out of memory");
  }
  else{
    result = nextfree;
  }
  return result;
}
```

由函数 i386_detect_memory() 知道总共要分配 npages 页物理页，因此先利用函数 boot_alloc 分配 sizeof(struct PageInfo) * npages 字节内存，这里 sizeof(struct PageInfo) 为 8字节（4字节为指针，4字节为引用次数）。并将分配来的空间先清空。

```
void
mem_init(void)
{
  uint32_t cr0;
  size_t n;
  i386_detect_memory();
  kern_pgdir = (pde_t *) boot_alloc(PGSIZE);
  memset(kern_pgdir, 0, PGSIZE);
  kern_pgdir[PDX(UVPT)] = PADDR(kern_pgdir) | PTE_U | PTE_P;

  pages = (struct PageInfo*)boot_alloc(sizeof(struct PageInfo) * npages);
  memset(pages,0,sizeof(struct PageInfo) * npages);

  page_init();

  check_page_free_list(1);
  ......
}
```

参数 IOPAGE，EXTPAGE，FREEPAGE 分别表示 IO hole，extended memory，extended memory 第一个空闲页面 的页号。

(1) page 0 标记为数用，因此将其引用数置为 1。

(2) 所有的 base memory 都是 free 的，因此将这一段引用数置为 0，同时用链表连接。

(3) 所有的 IO hole 不能被占用，因此将此段引用数置为 1。

(4) extended memory 前半段被占用，之后为free。

```
page_init(void)
{
  size_t i;
  pages[0].pp_ref = 1;
  pages[0].pp_link = NULL;

  size_t IOPAGE =  PGNUM(IOPHYSMEM);
  size_t EXTPAGE = PGNUM(EXTPHYSMEM);
  size_t FREEPAGE = PGNUM(PADDR(boot_alloc(0)));

  assert(page_free_list == NULL);
  assert(npages_basemem == IOPAGE);
```

```
    for (i = 1; i < IOPAGE; i++) {
        pages[i].pp_ref = 0;
        pages[i].pp_link = page_free_list;
        page_free_list = &pages[i];
    }
  for (i = IOPAGE; i < EXTPAGE; i++) {
    pages[i].pp_ref = 1;
    pages[i].pp_link = NULL;
  }
  for (i = EXTPAGE; i < FREEPAGE; i++) {
    pages[i].pp_ref = 1;
    pages[i].pp_link = NULL;
  }
    for (i = FREEPAGE; i < npages; i++) {
        pages[i].pp_ref = 0;
        pages[i].pp_link = page_free_list;
        page_free_list = &pages[i];
    }
  return;
}
```

page_free_list 指向当前空闲页面的链表的末尾。
如果 page_free_list 为 NULL 则返回 NULL。
将 page_free_list 作为本次分配的页面，将其从链表中删除，注意要将其pp_link 清空，否则 page_free 会 panic。
如果设置 (alloc_flags & ALLOC_ZERO) 那么需要将该物理页清为 '\0'。
还要注意到 reference count 会在调用函数中处理，因此在page_alloc 不需要处理。

```
struct PageInfo *
page_alloc(int alloc_flags)
{
  if (page_free_list == NULL)return NULL;
  struct PageInfo *alloc_page = page_free_list;
  page_free_list = alloc_page->pp_link;
  alloc_page->pp_link = NULL;

  if (alloc_flags & ALLOC_ZERO){
    memset(page2kva(alloc_page),'\0',PGSIZE);
  }
  return alloc_page;
}
```

```
如果 (pp->pp_ref != 0 || pp->pp_link !=NULL) 调用 panic
否则将当前页面放置在链表末尾。

void
page_free(struct PageInfo *pp)
{
  if (pp->pp_ref != 0 || pp->pp_link !=NULL)
    panic("Something went wrong at page_free");
  pp->pp_link = page_free_list;
  page_free_list = pp;
  return;
}
```

###

Part 2: Virtual Memory

Exercise 2.

Tasks: 阅读手册的第 5 章和第 6 章。

Exercise 3.

Tasks: 观察代码，判断 x 的数据类型。

Questions: x 的数据类型应当是什么?

Answer: x 应当是 uintptr_t，因为间接引用修改过 value 所指向地址的值，因此指针 value 中所存储的一定是虚拟内存。因此这里的 x 应当是 uintptr_t 数据类型。

Exercise 4.

Tasks: 完善下列函数

```
pgdir_walk()
boot_map_region()
page_lookup()
page_remove()
page_insert()
```

```
pgdir_walk() 给出 pgdir 和 虚拟地址 va，求解对应的 pte。
1. 首先由虚拟地址的高10位和 pgdir 得到pde。
```

2. 判断 pde 的 PTE_P 位是否为 1，为 0则表示该条目无效。
2.1 若该条目无效时，若 create 为假则返回空指针，否则创建新页。
3.用 page_alloc 申请新的一页，并要求清空。
4.增加页引用计数，修改 pde 条目权限。
5.再在虚拟地址空间中找到 Page Table对应页面，在根据中间10位求出pte。

```
pte_t *
pgdir_walk(pde_t *pgdir, const void *va, int create)
{
  if (!((create == 0) || (create == 1)))
    panic("pgdir_walk: create is wrong!!!");

  pde_t *pde = &pgdir[PDX(va)];
  if ((*pde & PTE_P) == 0){
    if (create == false){
      return NULL;
    }
    else{
      struct PageInfo *page = page_alloc(ALLOC_ZERO);
      if (page==NULL) return NULL;
      page->pp_ref++;
      *pde = page2pa(page) | PTE_SYSCALL;
    }
  }
  pte_t *pgtable = (pte_t *)KADDR(PTE_ADDR(*pde));
  return &pgtable[PTX(va)];
}
```

boot_map_region 将虚拟地址空间 [va,va+size) 映射到 物理地址空间 [pa,pa+size)
用函数 pgdir_walk 求解出 va+i 处虚拟页面的 pte，修改 pte。前 20 位 位物理空间地址，低12位为权限。

```
static void
boot_map_region(pde_t *pgdir, uintptr_t va, size_t size, physaddr_t pa, int perm)
{
  if (size % PGSIZE == 0){
    panic("boot_map_region: size % PGSIZE != 0");
  }
  if (PTE_ADDR(va) != va)
    panic("boot_map_region: va is not page_aligned");
  if (PTE_ADDR(pa) != pa)
    panic("boot_map_region: pa is not page_aligned");
  for (size_t i = 0;i < size; i +=PGSIZE){
    pte_t *pte = pgdir_walk(pgdir,(void *)va+i,1);
    *pte = PTE_ADDR(pa+i) | perm | PTE_P;
  }
}
```

page_insert() 将 物理页面 pp 映射到 虚拟地址 va 对应的页面中。
1.用pgdir_walk 求出 虚拟地址 va 对应的 pte。(若 pde 不存在的话在函数 pgdir_walk 中会创建并修改权限位)
2.若无法创建则返回 -E_NO_MEM。
3.先将页面引用数 +1。（否则在步骤 4 中删除时若原页面和pp相同时，会导致页面被回收）
4.再删除原 va 对应页面。这样即原页面与 pp 页面相同或不同都可以同样处理。

```
int
page_insert(pde_t *pgdir, struct PageInfo *pp, void *va, int perm)
{
  pte_t *pte = pgdir_walk(pgdir,va,1);
    if (pte == NULL) return -E_NO_MEM;
    pp->pp_ref++;
    if ((*pte & PTE_P) != 0) {
        page_remove(pgdir,va);
        tlb_invalidate(pgdir,va);
    }
    *pte = page2pa(pp) | perm | PTE_P;
  return 0;
}
```

page_lookup() 返回 虚拟地址 va 对应物理地址页面。
1.同样先求出pte，若pte为空则表示没有对应页面。
2.若 page_store 非空则存储 pte 地址。
3.返回物理地址。

```
struct PageInfo *
page_lookup(pde_t *pgdir, void *va, pte_t **pte_store)
{
  pte_t* pte = pgdir_walk(pgdir,va,0);
  if (pte == NULL) return NULL;
  if (pte_store != NULL)
    *pte_store = pte;
  return pa2page(PTE_ADDR(*pte));
}
```

page_remove() 清除 va 对应的物理页面之间的映射。
1.page_lookup 先求出 va 对应的物理页面地址，同时储存对应pte。
2.修改 pte 为 0，同时用 page_decref() 减少引用和释放页面。

```
void
page_remove(pde_t *pgdir, void *va)
{
  // Fill this function in
  pte_t* pte;
  struct PageInfo* page = page_lookup(pgdir, va, &pte);
  if(page != NULL){
    *pte = 0;
    page_decref(page);
```

```
    tlb_invalidate(pgdir, va);
  }
  return;
}
```

Part 3: Kernel Address Space

Exercise 5.

Task: 继续完善函数 mem_init()

```
  boot_map_region(kern_pgdir,UPAGES,PTSIZE,PADDR(pages),PTE_U);
```
将 [UPAGES,UPAGES+PTSIZE) 映射到物理空间 [pages,pages+PTSIZE)，权限是用户可读。

```
  boot_map_region(kern_pgdir,KSTACKTOP-KSTKSIZE,KSTKSIZE,PADDR(bootstack),PTE_W);
```
将 [KSTACKTOP-KSTKSIZE,KSTACKTOP) 映射到物理空间 [bootstack,bootstack+KSTACKTOP-KSTKSIZE)，权限是用户不可读写，超级用户可读写。

```
  assert(KERNBASE == 0xf0000000); // 0x100000000 - KERNBASE
  boot_map_region(kern_pgdir,KERNBASE,0x10000000,0x0,PTE_W);
```
将 [KERNBASE,0100000000) 映射到物理空间 [0x0,0x100000000 - KERNBASE)，权限是用户不可读写，超级用户可读写。

Questions

2.尽可能填写表格。(pd table)

Entry	Base Virtual Address	Points to (logically):
1023	0xffc00000	KERNBASE
……	……	……
960	0xf0000000	KERNBASE
959	0xefc00000	Stack
958	0xef800000	?
957	0xef400000	Page Director
956	0xef000000	PageInfo
……	……	……
1	0x00400000	?
0	0x00000000	Empty

3.为什么用户不能读写内核的内存，是什么机制保护了内核。

答：因为内核代码和数据中，PTE_U位置为0，因此用户不能访问内核内存。

4.这个操作系统最大可以支持多少物理内存，为什么。

答: boot_map_region(kern_pgdir,UPAGES,PTSIZE,PADDR(pages),PTE_U);将 pages 对应的物理空间映射到了UPAGES处。

因为 UPAGES 只有 4M，因此 pages 大小不能超过 4M，否则在映射过程中会覆盖 Page Director。每个 PageInfo 占 8字节，因此至多有

2^{19}个物理页面，每个物理页面有4K字节，故总共可支持2G的物理空间。

5.如果我们有最大的物理内存，那么需要多大的空间去管理，详细说明每部分空间占用。

答：1 个 Page Director 有 2^9 个条目，对齐占 1 页，共 4 K

2^9 个Page Table ，每个占 1 页，共 2 M

2^{19} 个PageInfo ，每个占 8字节，共 4M

故总共占 6148 K。

6.EIP是什么时候变得比KERNBASE 大，在这之前为什么能正常运行，以及为什么要转换。

答：执行完 0x10002d: jmp *%eax 语句后 EIP 跳转到比KERNBASE大的地方。

通过 entry_pgdir 建立虚拟地址 [0,4M) 到物理地址 [0,4M) 的映射时，转换之后使得可以在高地址执行代码。

Challenge： Extend the JOS kernel monitor

这里选择 第二个 challenge : Extend the JOS kernel monitor。

```
K> help
help - Display this list of commands
kerninfo - Display information about the kernel
backtrace - Display information about the stack backtrace
showmappings - Show physical pages mapped to specific virtual address area
setpermissions - Set permissions of specific virtual pages
clearpermissions - Clear permissions of specific virtual pages
showvirtualmemory - Show Virtual memory
va2pa - Convert virtual address to physical address
pa2va - Convert physical address to virtual address
```

这里完成了六个函数: showmappings，setpermissions，clearpermissions，showvirtualmemory，va2pa，pa2va五个函数，具体实现如下。

第一个函数功能是实现输出虚拟地址在 `[L,R]` 之间的虚拟页面权限以及对应物理页面地址。
先处理一些输入不合法情况。(其中包括参数个数，输入参数是否是合法数字以及参数是否对页面 `size` 对齐)。
枚举其中的所有虚拟页面，依次求出 `pde` 和 `pte` ，然后获得权限输出即可。

```
const char Bit2Sign[9][2] = {{'-','P'},{'-','W'},{'-','U'},{'-','T'},{'-','C'},{'-
','A'},{'-','D'},{'-','I'},{'-','G'}};
int
mon_showmappings(int argc,char **argv,struct Trapframe *tf){
  if(argc!=3){
    cprintf("mon_showmappings: The number of parameters is two.\n");
    return 0;
  }
  char *errChar;
  uintptr_t StartAddr = strtol(argv[1], &errChar, 0);
  if (*errChar){
    cprintf("mon_showmappings: The first argument is not a number.\n");
    return 0;
  }
  uintptr_t EndAddr = strtol(argv[2],&errChar,0);
  if (*errChar){
    cprintf("mon_showmappings: The second argument is not a number.\n");
    return 0;
  }
  if (StartAddr&0x3ff){
    cprintf("mon_showmappings: The first parameter is not aligned.\n");
    return 0;
  }
  if (EndAddr&0x3ff){
```

```
    cprintf("mon_showmappings: The second parameter is not aligned.\n");
    return 0;
  }
  if (StartAddr > EndAddr){
    cprintf("mon_shopmappings: The first parameter is larger than the second
parameter.\n");
    return 0;
  }

    cprintf(
        "G: Global             I: PT Attribute Index    D: Dirty\n"
        "A: Accessed           C: Cache Disable         T: Write-Through\n"
        "U: User/Supervisor    W: Writable              P: Present\n"
        "-------------------------------------------------------------------\n"
        "virtual_address        physica_address        GIDACTUWP\n");

  for (uintptr_t Address = StartAddr;Address < EndAddr; Address+=PGSIZE){
    pde_t *pde = &kern_pgdir[PDX(Address)];
    if (*pde & PTE_P){
      pte_t *pte = (pte_t *)KADDR(PTE_ADDR(*pde)) + PTX(Address);
      if (*pte & PTE_P){
        char permission[10];
        for (int i = 8 , perm = *pte - PTE_ADDR(*pte); i >= 0; i--,perm>>=1){
          permission[i] = Bit2Sign[8-i][(perm&1)];
        }
        permission[9]='\0';
        cprintf("0x%08x             0x%08x
%s\n",Address,PTE_ADDR(*pte),permission);
        continue;
      }
    }
    cprintf("0x%08x             unmapped               --------\n",Address);
  }
    return 0;
}
```

示例如下：

```
K> showmappings 0xf0000000 0xf0010000
G: Global              I: PT Attribute Index    D: Dirty
A: Accessed            C: Cache Disable         T: Write-Through
U: User/Supervisor     W: Writable              P: Present
-------------------------------------------------------------------------
virtual_address         physica_address        GIDACTUWP
0xf0000000              0x00000000             -------WP
0xf0001000              0x00001000             --DA---WP
0xf0002000              0x00002000             --DA---WP
0xf0003000              0x00003000             --DA---WP
0xf0004000              0x00004000             --DA---WP
0xf0005000              0x00005000             --DA---WP
0xf0006000              0x00006000             --DA---WP
0xf0007000              0x00007000             --DA---WP
0xf0008000              0x00008000             --DA---WP
0xf0009000              0x00009000             --DA---WP
0xf000a000              0x0000a000             --DA---WP
0xf000b000              0x0000b000             --DA---WP
0xf000c000              0x0000c000             --DA---WP
0xf000d000              0x0000d000             --DA---WP
0xf000e000              0x0000e000             --DA---WP
0xf000f000              0x0000f000             --DA---WP
```

第二个函数是设置虚拟地址位于 `[L,R]` 之间的页面权限。
类似前一个函数，先判断输入合法性，再将枚举页面，求出pte，修改pte权限即可。
完成后调用 `showmappings` 输出页面权限。

```
int Sign2Perm(char *s){
  int l = strlen(s);
  int Perm = 0;
  for (int i=0;i<l;i++){
    switch(s[i]){
      case 'P':Perm|=PTE_P;break;
      case 'W':Perm|=PTE_W;break;
      case 'U':Perm|=PTE_U;break;
      case 'T':Perm|=PTE_PWT;break;
      case 'C':Perm|=PTE_PCD;break;
      case 'A':Perm|=PTE_A;break;
      case 'D':Perm|=PTE_D;break;
      case 'I':Perm|=PTE_PS;break;
      case 'G':Perm|=PTE_G;break;
      default:return -1;
    }
  }
  return Perm;
}
int mon_setpermissions(int argc, char **argv, struct Trapframe *tf){
  if(argc!=4){
    cprintf("mon_setpermissions: The number of parameters is three.\n");
    return 0;
  }
  char *errChar;
  uintptr_t StartAddr = strtol(argv[1], &errChar, 0);
```

```
  if (*errChar){
    cprintf("mon_setpermissions: The first argument is not a number.\n");
    return 0;
  }
  uintptr_t EndAddr = strtol(argv[2],&errChar,0);
  if (*errChar){
    cprintf("mon_setpermissions: The second argument is not a number\n");
    return 0;
  }
  if (StartAddr&0x3ff){
    cprintf("mon_setpermissions: The first parameter is not aligned.\n");
    return 0;
  }
  if (EndAddr&0x3ff){
    cprintf("mon_setpermissions: The second parameter is not aligned.\n");
    return 0;
  }
  if (StartAddr > EndAddr){
    cprintf("mon_setpermissions: The first parameter is larger than the second
parameter.\n");
    return 0;
  }
  int Perm = Sign2Perm(argv[3]);
  if (Perm == -1){
    cprintf("mon_setpermissions: The permission bit is not set correctly.\n");
    return 0;
  }
  for (uintptr_t Address = StartAddr;Address < EndAddr; Address+=PGSIZE){
    pde_t *pde = &kern_pgdir[PDX(Address)];
    if (*pde & PTE_P){
      pte_t *pte = (pte_t *)KADDR(PTE_ADDR(*pde)) + PTX(Address);
      if (*pte & PTE_P){
        *pte = *pte | Perm;
        continue;
      }
    }
  }
    cprintf("Permission has been updated:\n");
    mon_showmappings(argc-1,argv,tf);
    return 0;
}
```

示例如下：

```
K> setpermissions 0xf0000000 0xf0010000 GT
Permission has been updated:
G: Global              I: PT Attribute Index    D: Dirty
A: Accessed            C: Cache Disable         T: Write-Through
U: User/Supervisor     W: Writable              P: Present
-------------------------------------------------------------------
virtual_address         physica_address         GIDACTUWP
0xf0000000              0x00000000              G----T-WP
0xf0001000              0x00001000              G-DA-T-WP
0xf0002000              0x00002000              G-DA-T-WP
0xf0003000              0x00003000              G-DA-T-WP
0xf0004000              0x00004000              G-DA-T-WP
0xf0005000              0x00005000              G-DA-T-WP
0xf0006000              0x00006000              G-DA-T-WP
0xf0007000              0x00007000              G-DA-T-WP
0xf0008000              0x00008000              G-DA-T-WP
0xf0009000              0x00009000              G-DA-T-WP
0xf000a000              0x0000a000              G-DA-T-WP
0xf000b000              0x0000b000              G-DA-T-WP
0xf000c000              0x0000c000              G-DA-T-WP
0xf000d000              0x0000d000              G-DA-T-WP
0xf000e000              0x0000e000              G-DA-T-WP
0xf000f000              0x0000f000              G-DA-T-WP
```

第三个函数是清空虚拟地址位于 `[L,R]` 之间的页面权限。

```
int mon_clearpermissions(int argc, char **argv, struct Trapframe *tf){
    if(argc!=4){
    cprintf("mon_clearpermissions: The number of parameters is three.\n");
    return 0;
  }
  char *errChar;
  uintptr_t StartAddr = strtol(argv[1], &errChar, 0);
  if (*errChar){
    cprintf("mon_clearpermissions: The first argument is not a number.\n");
    return 0;
  }
  uintptr_t EndAddr = strtol(argv[2],&errChar,0);
  if (*errChar){
    cprintf("mon_clearpermissions: The second argument is not a number.\n");
    return 0;
  }
  if (StartAddr&0x3ff){
    cprintf("mon_clearpermissions: The first parameter is not aligned.\n");
    return 0;
  }
  if (EndAddr&0x3ff){
    cprintf("mon_clearpermissions: The second parameter is not aligned.\n");
    return 0;
  }
  if (StartAddr > EndAddr){
```

```
    cprintf("mon_clearpermissions: The first parameter is larger than the second
parameter.\n");
    return 0;
  }
  int Perm = Sign2Perm(argv[3]);
  if (Perm == -1){
    cprintf("mon_clearpermissions: The permission bit is not set correctly.\n");
    return 0;
  }
  for (uintptr_t Address = StartAddr;Address < EndAddr; Address+=PGSIZE){
    pde_t *pde = &kern_pgdir[PDX(Address)];
    if (*pde & PTE_P){
      pte_t *pte = (pte_t *)KADDR(PTE_ADDR(*pde)) + PTX(Address);
      if (*pte & PTE_P){
        *pte = *pte & ~Perm;
        continue;
      }
    }
  }
    cprintf("Permission has been updated:\n");
    mon_showmappings(argc-1,argv,tf);

    return 0;
}
```

示例如下：

```
K> clearpermissions 0xf0000000 0xf0010000 AW
Permission has been updated:
G: Global             I: PT Attribute Index    D: Dirty
A: Accessed           C: Cache Disable         T: Write-Through
U: User/Supervisor    W: Writable              P: Present
-------------------------------------------------------------------------
virtual_address        physica_address        GIDACTUWP
0xf0000000             0x00000000             G----T--P
0xf0001000             0x00001000             G-D--T--P
0xf0002000             0x00002000             G-D--T--P
0xf0003000             0x00003000             G-D--T--P
0xf0004000             0x00004000             G-D--T--P
0xf0005000             0x00005000             G-D--T--P
0xf0006000             0x00006000             G-D--T--P
0xf0007000             0x00007000             G-D--T--P
0xf0008000             0x00008000             G-D--T--P
0xf0009000             0x00009000             G-D--T--P
0xf000a000             0x0000a000             G-D--T--P
0xf000b000             0x0000b000             G-D--T--P
0xf000c000             0x0000c000             G-D--T--P
0xf000d000             0x0000d000             G-D--T--P
0xf000e000             0x0000e000             G-D--T--P
0xf000f000             0x0000f000             G-D--T--P
```

第四个函数是输出 虚拟地址 [L,R] 之间的内容。
4字节对齐输出，因此要求L，R 4字节对齐。在枚举的过程中直接取地址即可，没四次输出换行。

```
int
mon_showvirtualmemory(int argc, char **argv, struct Trapframe *tf){
  if(argc!=3){
    cprintf("mon_showvvirtualmemory: The number of parameters is two.\n");
    return 0;
  }
  char *errChar;
  uintptr_t StartAddr = strtol(argv[1], &errChar, 0);
  if (*errChar){
    cprintf("mon_showvvirtualmemory: The first argument is not a number.\n");
    return 0;
  }
  uintptr_t EndAddr = strtol(argv[2],&errChar,0);
  if (*errChar){
    cprintf("mon_showvvirtualmemory: The second argument is not a number.\n");
    return 0;
  }
  if (StartAddr&0x3){
    cprintf("mon_clearpermissions: The first parameter is not aligned.\n");
    return 0;
  }
  if (EndAddr&0x3){
    cprintf("mon_clearpermissions: The second parameter is not aligned.\n");
    return 0;
  }
  if (StartAddr > EndAddr){
    cprintf("mon_showvvirtualmemory: The first parameter is larger than the second
parameter.\n");
    return 0;
  }
  int c = 0;
  for (uintptr_t Address = StartAddr;Address < EndAddr; Address+=4){
    switch (c){
      case 0:cprintf("0x%08x   :0x%08x    ",Address,*(int*)Address);break;
      case 1:cprintf("0x%08x    ",*(int*)Address);break;
      case 2:cprintf("0x%08x    ",*(int*)Address);break;
      case 3:cprintf("0x%08x\n",*(int*)Address);break;
    }
    c = (c+1)&3;
  }
  return 0;
}
```

示例如下：

```
K> showvirtualmemory 0xf0000000 0xf0000100
0xf0000000   :0xf000ff53    0xf000ff53    0xf000e2c3    0xf000ff53
0xf0000010   :0xf000ff53    0xf000ff53    0xf000ff53    0xf000ff53
0xf0000020   :0xf000fea5    0xf000e987    0xf000d62c    0xf000d62c
0xf0000030   :0xf000d62c    0xf000d62c    0xf000ef57    0xf000d62c
0xf0000040   :0xc0005479    0xf000f84d    0xf000f841    0xf000e3fe
0xf0000050   :0xf000e739    0xf000f859    0xf000e82e    0xf000efd2
0xf0000060   :0xf000d648    0xf000e6f2    0xf000fe6e    0xf000ff53
0xf0000070   :0xf000ff53    0xf000ff53    0xf0006924    0xc0008954
0xf0000080   :0xf000ff53    0xf000ff53    0xf000ff53    0xf000ff53
0xf0000090   :0xf000ff53    0xf000ff53    0xf000ff53    0xf000ff53
0xf00000a0   :0xf000ff53    0xf000ff53    0xf000ff53    0xf000ff53
0xf00000b0   :0xf000ff53    0xf000ff53    0xf000ff53    0xf000ff53
0xf00000c0   :0xf000ff53    0xf000ff53    0xf000ff53    0xf000ff53
0xf00000d0   :0xf000ff53    0xf000ff53    0xf000ff53    0xf000ff53
0xf00000e0   :0xf000ff53    0xf000ff53    0xf000ff53    0xf000ff53
0xf00000f0   :0xf000ff53    0xf000ff53    0xf000ff53    0xf000ff53
```

第五个函数是将虚拟地址转化为物理地址，同上求出pte，取前20和虚拟地址后12位即物理地址。

```
int
mon_va2pa(int argc, char **argv, struct Trapframe *tf){
  if(argc!=2){
    cprintf("mon_va2pa: The number of parameters is one.\n");
    return 0;
  }
  char *errChar;
  uintptr_t Address = strtol(argv[1], &errChar, 0);
  if (*errChar){
    cprintf("mon_va2pa: The argument is not a number.\n");
    return 0;
  }
  pde_t *pde = &kern_pgdir[PDX(Address)];
  if (*pde & PTE_P){
    pte_t *pte = (pte_t *)KADDR(PTE_ADDR(*pde)) + PTX(Address);
    if (*pte & PTE_P){
      cprintf("The physical address is 0x%08x.\n",PTE_ADDR(*pte)|(Address&0x3ff));
    }
    else
      cprintf("This is not a valid virtual address.\n");
  }
  else
    cprintf("This is not a valid virtual address.\n");
  return 0;
}
```

示例如下：

```
K> va2pa 0xef000000
The physical address is 0x00124000.
```

第六个函数是求出物理地址对应的所有虚拟地址，从物理地址转虚拟地址没有好的结构维护，因此需要枚举所有虚拟页面，需要判断pte前20位是否和物理地址相同，若相同则找到一个对应的虚拟地址。一次求解至多需要枚举 2^20 个pte。

```
int
mon_pa2va(int argc, char **argv, struct Trapframe *tf){
  if(argc!=2){
    cprintf("mon_pa2va: The number of parameters is one.\n");
    return 0;
  }
  char *errChar;
  uintptr_t Address = strtol(argv[1], &errChar, 0);
  if (*errChar){
    cprintf("mon_pa2va: The argument is not a number.\n");
    return 0;
  }
  int cnt=0;
  for (int i = 0; i < 1024; i++){
    pde_t *pde = &kern_pgdir[i];
    if (*pde & PTE_P){
      for (int j = 0; j < 1024; j++){
        pte_t *pte = (pte_t *)KADDR(PTE_ADDR(*pde)) + j;
        if (*pte & PTE_P){
          if (PTE_ADDR(*pte) == PTE_ADDR(Address)){
            if (cnt == 0 )cprintf("The virtual addresses are 0x%08x",(i<<PDXSHIFT)|
(j<<PTXSHIFT)|PGOFF(Address));
            else cprintf(",0x%08x",(i<<PDXSHIFT)|(j<<PTXSHIFT)|PGOFF(Address));
            cnt++;
          }
        }
      }
    }
  }
  if (cnt == 0)
    cprintf("There is no virtual address.\n");
  else cprintf(".\n");
  return 0;
}
```

示例如下：

```
K> pa2va 0x00124000
The virtual addresses are 0xef000000,0xf0124000.
```