

xv6©-rev10
(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)
Initialization, Dynamic Allocation Subsystem

Carmi Merimovich

Tel-Aviv Academic College

September 27, 2017

Our context

```
extern char end[];  
kinit1(end, P2V(4*1024*1024)); // phys page alloc  
kvmalloc();           // kernel page table  
:  
seginit();           // set up segments  
:  
pinit();           // process table  
:  
:  
kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must c  
userinit();           // first user process  
mpmain();
```

Why dynamic memory subsystem first?

- Many initialization routines uses utility kernel routines.
- The utility routines use the dynamic allocation subsystem.
- Ergo...

Prerequisites

- **freerange**
- **kfree.**

It will be clearer to explain first the dynamic allocation system, and then explain the initialization.

Services supplied by the dynamic allocation

- `void *kalloc(void)`: Returns a pointer to a non-used **page**.
- `kfree(void *p)`: Marks the **page** pointed to by `p` as unused.

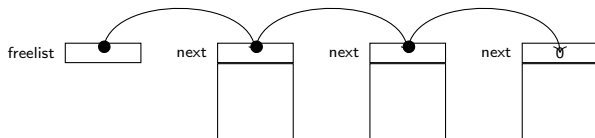
Implementation

The unused pages are in a linked list headed by **kmem**:

```
3115 struct run {  
    struct run *next;  
};  
  
struct {  
    struct spinlock lock;  
    int use_lock;  
    struct run *freelist;  
} kmem;
```

The first longword in each page points to the next free page.

The freelist

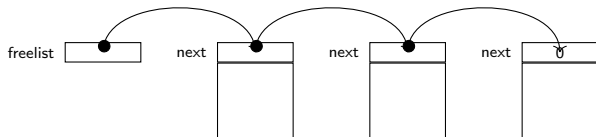


- **kalloc** removes from the head of the list.
- **kfree** inserts at the beginning of the list.

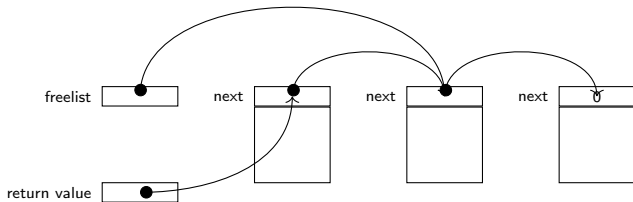
kalloc

```
3187 char *kalloc(void) {  
    struct run *r;  
  
    acquire(&kmem.lock);  
    r = kmem.freelist;  
    if (r)  
        kmem.freelist = r->next;  
    release(&kmem.lock);  
    return (char*)r;  
}
```

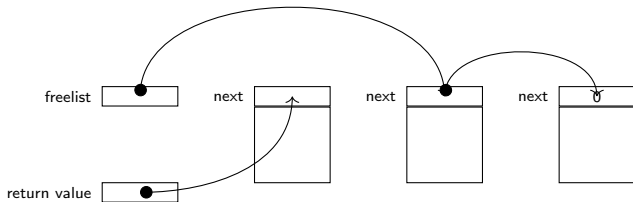

kalloc in action



kalloc in action



kalloc in action

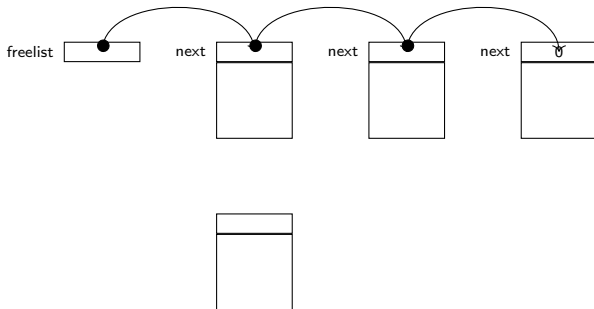


Logically we mean this, of course!

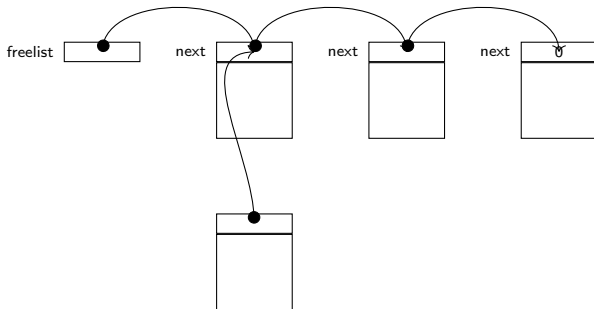
kfree

```
3163 void kfree(char *v) {  
    struct run *r;  
  
    if ((uint)v % PGSIZE || v < end || v2p(v) >= PHYSTOP)  
        panic("kfree");  
  
    memset(v, 1, PGSIZE);  
  
    acquire(&kmem.lock);  
    r = (struct run*)v;  
    r->next = kmem.freelist;  
    kmem.freelist = r;  
    release(&kmem.lock);  
}
```

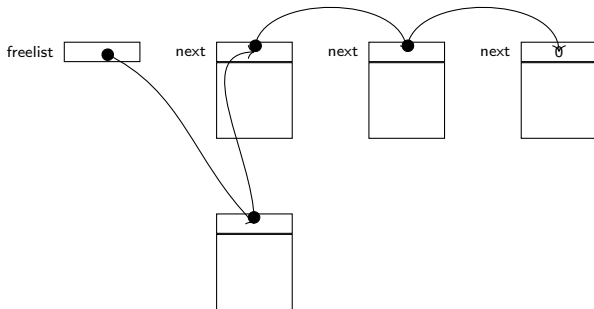
kfree in action



kfree in action



kfree in action



Loading **freelist**

- What is in the **freelist** when entering main?

Loading **freelist**

- What is in the **freelist** when entering main?
 - The empty list.
- What addresses can be considered free?

Loading **freelist**

- What is in the **freelist** when entering main?
 - The empty list.
- What addresses can be considered free?
 - Not valid addresses are, well, not valid!
 - Thus addresses in `[0x80000000,0x80400000)` without kernel addresses.
- xv6 is linked to address `0x80100000`.
- The linker defines **_end** to where the kernel ends.
- Thus the following two ranges are free:
 - `[0x80000000,0x800A0000)`.
 - `[_end,0x80400000)`.
- xv6 'frees' only the range `[_end,0x80400000)`.
- Since we work with pages, **_end** should be rounded up.
- (elective) There is info we need at the first 64KB.

freerange

- Is it legal, in user mode C, to **free** a block not **malloc**'ed?

freerange

- Is it legal, in user mode C, to **free** a block not **malloc**'ed?
 - No. Why?
- Can we, in the xv6 kernel, **kfree** a page not **kalloc**'ed?

freerange

- Is it legal, in user mode C, to **free** a block not **malloc**'ed?
 - No. Why?
- Can we, in the xv6 kernel, **kfree** a page not **kalloc**'ed?
 - Yes. Why?

```
#define PGROUNDUP(sz) (((sz)+PGSIZE-1) &  
                        ~(PGSIZE-1))
```

```
3151 void freerange(void *vstart, void *vend) {  
    char *p;  
    p = (char*)PGROUNDUP((uint)vstart);  
    for (; p + PGSIZE <= (char*)vend; p += PGSIZE)  
        kfree(p);  
}
```

```
3131 void kinit1(void *vstart, void *vend) {  
    initlock(&kmem.lock, "kmem");  
    freerange(vstart, vend);  
}
```

Dynamic allocation initialization

- Step 1:

```
#define KERNBASE 0x80000000
#define PHYSTOP 0x0E000000
#define P2V(a) (((void *) (a)) + KERNBASE)

extern char end[];

kinit1(end, P2V(4*1024*1024));
```

- Step 2:

```
kinit2(P2V(4*1024*1024), P2V(PHYSTOP));
```