

xv6©-rev10
(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)
Initialization, Kernel Mapping

Carmi Merimovich

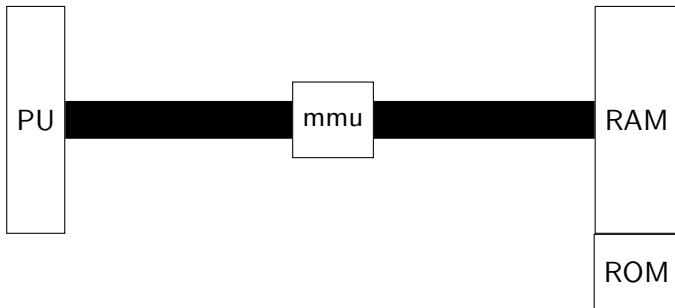
Tel-Aviv Academic College

September 27, 2017

Context

```
kinit1(end, P2V(4*1024*1024)); // phys page allocation
kvmmalloc(); // kernel page table
:
segininit(); // set up segments
:
pinit(); // process table
:
:
kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must copy
userinit(); // first user process
mpmain();
```

Memory access after paging enabled



- The MMU can generate only physical addresses.
- The processor can generate only virtual addresses.
- If we (program) need to access physical address we must attach a virtual address for it.

kernel static/dynamic mapping

- Static mapping (used by xv6):

All physical pages ever accessed by the kernel get virtual addresses on initialization.

- Dynamic mapping:

When a physical page is needed a mapping rule for it is added.
When the page is no more needed, the rule is removed.

Static vs. Dynamic mapping

| | static | dynamic |
|----------------|----------------------|--------------------------|
| Coding Size | simple Restricted | less simple unlimited |

- The virtual address range reserved for xv6 is [0x80000000,0xFFFFFFFF).
- Hence xv6 cannot use more than 2GB of physical memory!
- (Somewhat less than that, really).
- It does not matter how large is the RAM installed in a system.

kvmalloc prerequisites

- **setupkvm.**
 - **mappages.**
 - **walkpgdir.**
- **switchkvm.**

setupkvm: xv6 static mapping

0xFFFFFFFF



0xFFFFFFFF

0xFE000000

0xFE000000

```
#define KERNBASE 0x80000000
```

```
static inline uint v2p(void *a)
{ return ((uint) (a)) - KERNBASE; }
```

0x0DFFFFFF



0x0DFFFFFF

```
static inline void *p2v(uint a)
{ return (void *)((a) + KERNBASE); }
```

≈0x00110000

≈0x80110000

0x00100000

0x80100000

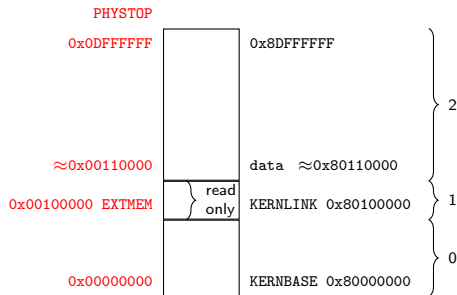
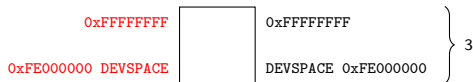
```
0x80000000 <= va < 0xFE000000
pa = v2p(va);
```

0x00000000

0x80000000

```
0 <= pa < 0x7E000000
va = p2v(pa);
```

setupkvm: xv6 static mapping



setupkvm: ranges

```
1804 static struct kmap {  
    void *virt;  
    uint phys_start;  
    uint phys_end;  
    int perm;  
} kmap[] = {  
    {(void*)KERNBASE, 0, EXTMEM, PTE_W},  
    {(void*)KERNLINK, V2P(KERNLINK), V2P(data), 0}, // k  
    {(void*)data, V2P(data), PHYSTOP, PTE_W},  
    {(void*)DEVSPACE, DEVSPACE, 0, PTE_W},  
};
```

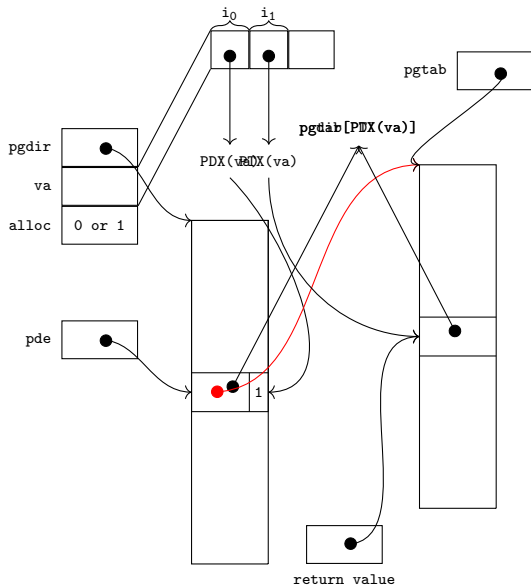
setupkvm

```
1817 pde_t *setupkvm(void) {  
    pde_t *pgdir;  
    struct kmap *k;  
  
    if ((pgdir = (pde_t*)kalloc()) == 0)  
        return 0;  
    memset(pgdir, 0, PGSIZE);  
    for(k = kmap; k < &kmap[NELEM(kmap)]; k++)  
        if (mappages(pgdir, k->virt,  
                     k->phys_end - k->phys_start,  
                     (uint)k->phys_start, k->perm) < 0) {  
            freevm(pgdir);  
            return 0;  
        }  
    return pgdir;  
}
```

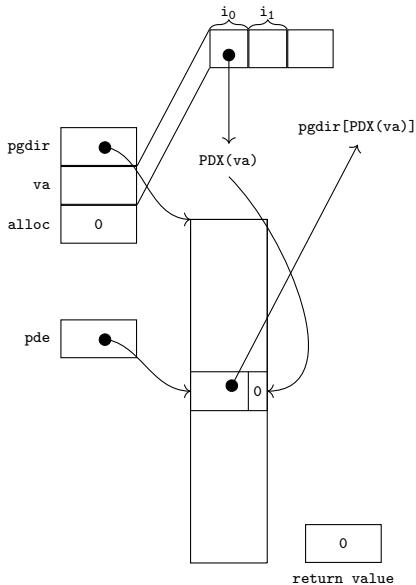
walkpgdir

- Basic paging handling routine.
- Arguments: pgdir, va, alloc.
- Returns the **address** of the page table entry which would have been used to translate the virtual address va if **cr3** would have pointed to the table pgdir.
- If the relevant internal table is missing it returns 0.
- However, if alloc is not zero, an internal table is allocated, if necessary.

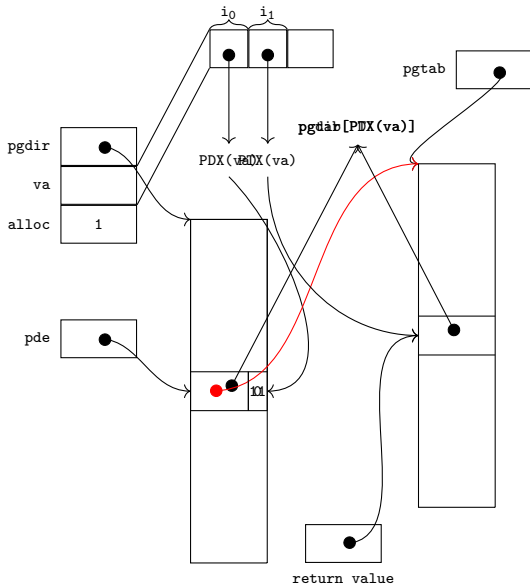
Ex 1: **walkpgdir** when internal table exists



Ex 2: **walkpgdir** without internal table



Ex 3: walkpgdir without internal table



walkpgdir

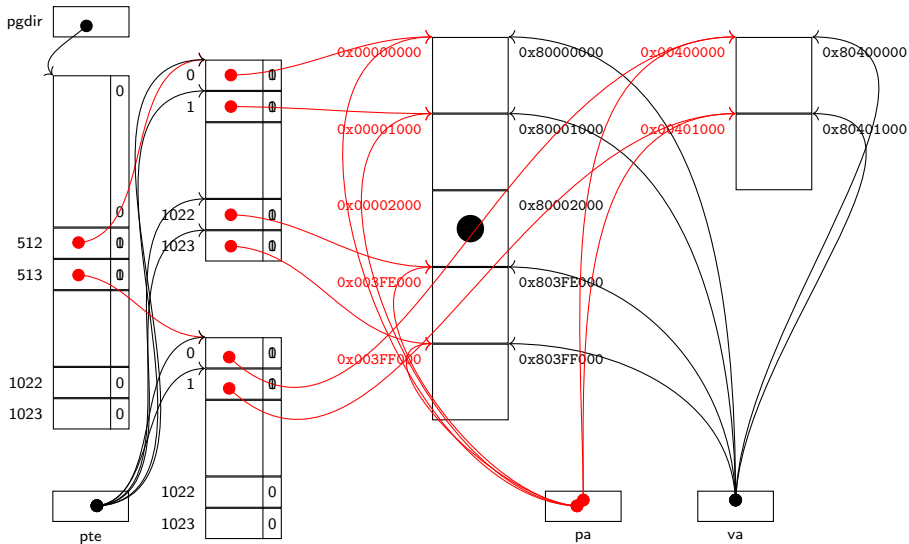
```
1735 #define PDX(va) (((uint)(va) >> PDXSHIFT) & 0x3FF)
#define PTX(va) (((uint)(va) >> PTXSHIFT) & 0x3FF)
#define PTE_P 1
static pte_t *walkpgdir(pde_t *pgdir, const void *va,
                        int alloc) {
    pde_t *pde;
    pte_t *pgtab;
    pde = &pgdir[PDX(va)];
    if (*pde & PTE_P)
        pgtab = (pte_t*)p2v(PTE_ADDR(*pde));
    else {
        if (!alloc || (pgtab = (pte_t*)kalloc()) == 0)
            return 0;
        memset(pgtab, 0, PGSIZE);
        *pde = v2p(pgtab) | PTE_P | PTE_W | PTE_U;
    }
    return &pgtab[PTX(va)];
}
```

mappages

```
1760 static int mappages(pde_t *pgdir, void *va, uint size,
                      uint pa, int perm) {
    char *a, *last;
    pte_t *pte;

    a = (char*)PGROUNDDOWN((uint)va);
    last = (char*)PGROUNDDOWN(((uint)va) + size - 1);
    for (;;) {
        if ((pte = walkpgdir(pgdir, a, 1)) == 0)
            return -1;
        if (*pte & PTE_P) panic("remap");
        *pte = pa | perm | PTE_P;
        if (a == last) break;
        a += PGSIZE;
        pa += PGSIZE;
    }
    return 0;
}
```


mappages



kvmalloc/switchkvm

```
1840 void kvmalloc(void) {  
    kpgdir = setupkvm();  
    switchkvm();  
}  
  
1853 void switchkvm(void) {  
    lcr3(v2p(kpgdir));  
}
```