CS5460/6460: Operating Systems

Lecture 8: System init

Anton Burtsev
January, 2014

Recap from last time

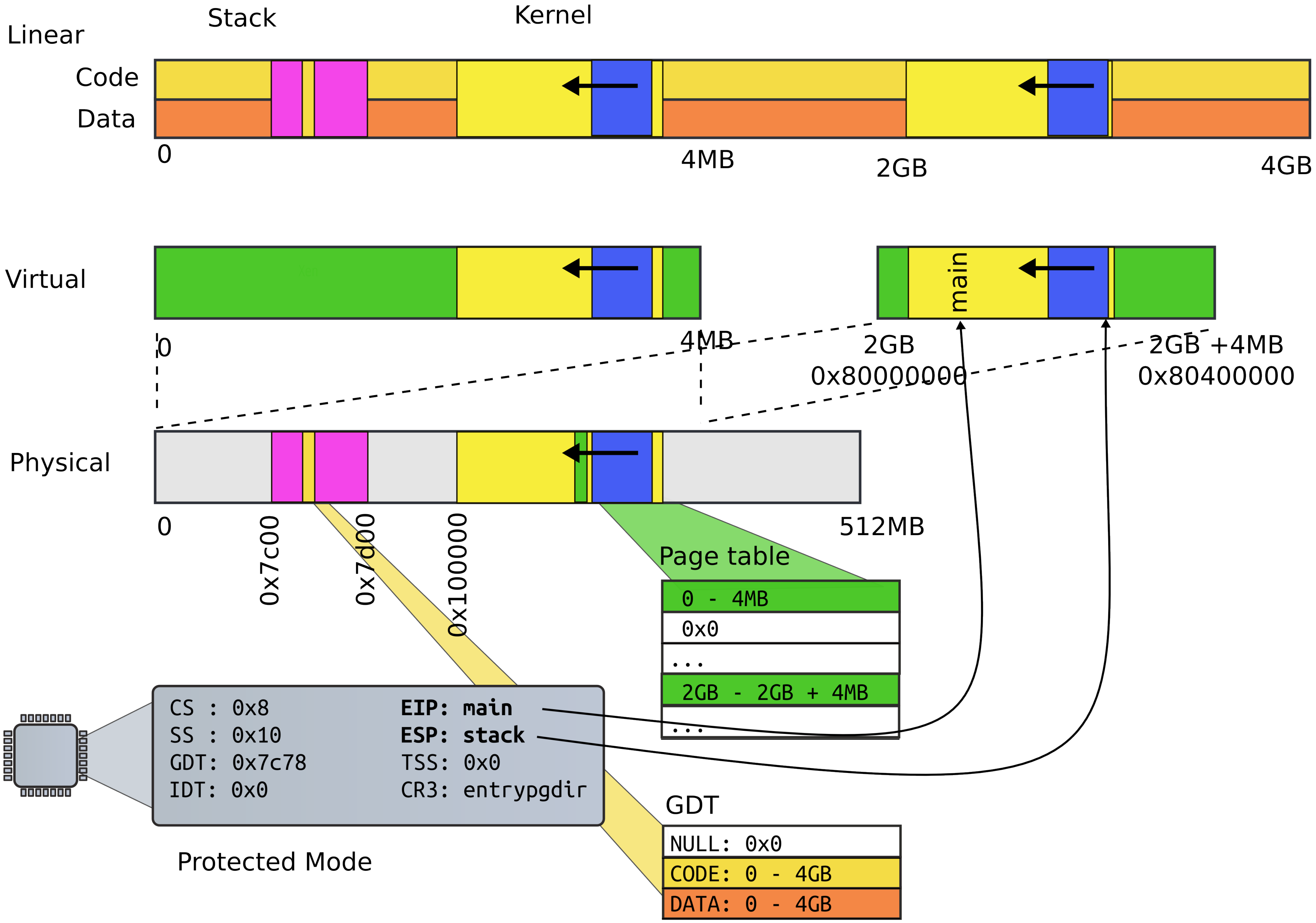
- Setup segments (data and code)
- Switched to protected mode
  - Loaded GDT (segmentation is on)
- Setup stack (to call C functions)
- Loaded kernel from disk
- Setup first page table
  - 2 entries [0 : 4MB] and [2GB : (2GB + 4MB)]
- Setup high-address stack
- Jumped to main()

```
# Set up the stack pointer.
movl $(stack + KSTACKSIZE), %esp

# Jump to main(), and switch to executing at
# high addresses. The indirect call is needed because
# the assembler produces a PC-relative instruction
# for a direct jump.
mov $main, %eax
jmp *%eax

.comm stack, KSTACKSIZE
```

Jumped to main()



Linear

Stack

Kernel

Code

Data

0

4MB

2GB

4GB

Virtual

main

0

4MB

2GB

0x80000000

2GB +4MB

0x80400000

Physical

0

0x7c00

0x7d00

0x100000

512MB

Page table

0 - 4MB
0x0
...
2GB - 2GB + 4MB
...

```
CS : 0x8          EIP: main
SS : 0x10         ESP: stack
GDT: 0x7c78       TSS: 0x0
IDT: 0x0          CR3: entrypgdir
```

Protected Mode

GDT

NULL: 0x0
CODE: 0 - 4GB
DATA: 0 - 4GB

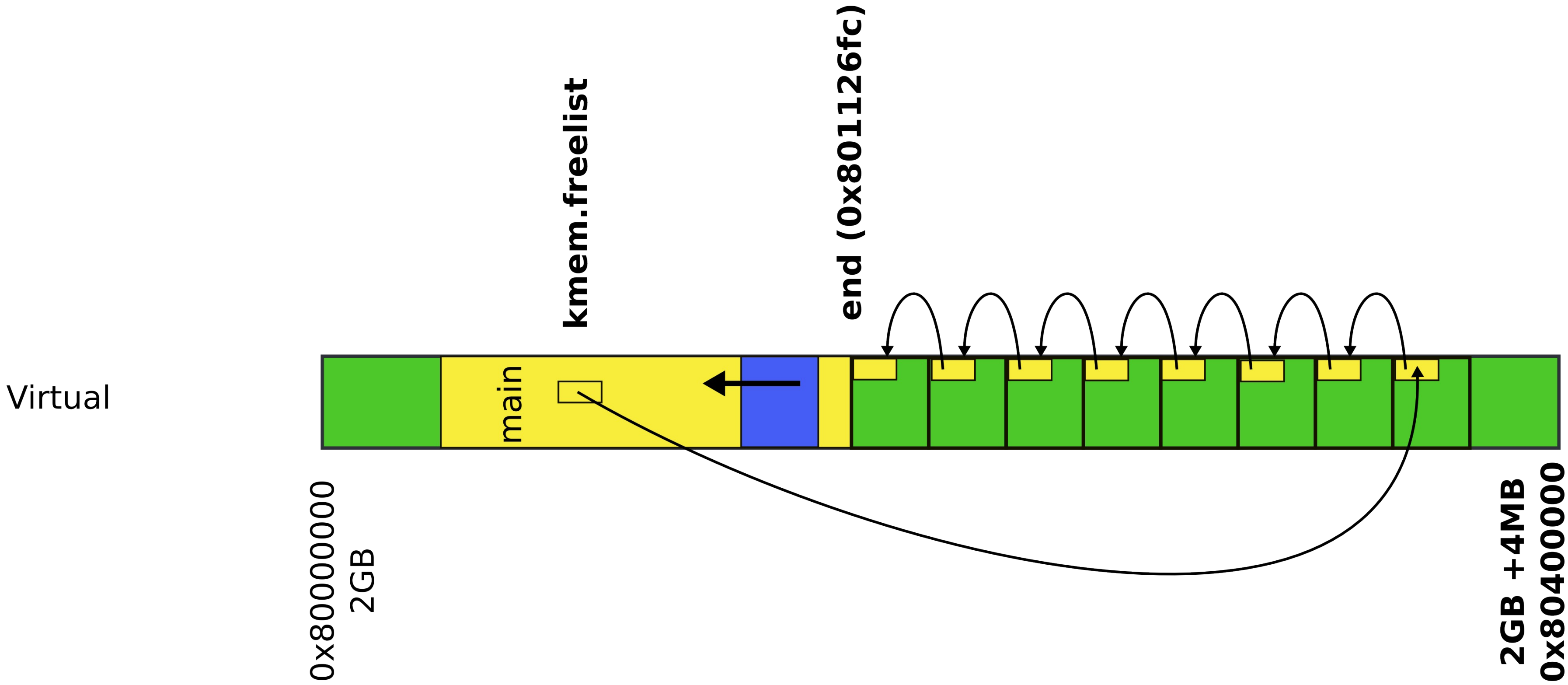
```
main(void)
{
  kinit1(end, P2V(4*1024*1024)); // phys page allocator
  kvmalloc(); // kernel page table
  mpinit(); // collect info about this machine
  lapicinit();
  seginit(); // set up segments
  cprintf("\ncpu%d: starting xv6\n\n", cpu->id);
  picinit(); // interrupt controller
  ioapicinit(); // another interrupt controller
  consoleinit(); // I/O devices & their interrupts
  uartinit(); // serial port
  pinit(); // process table
  tvinit(); // trap vectors
```

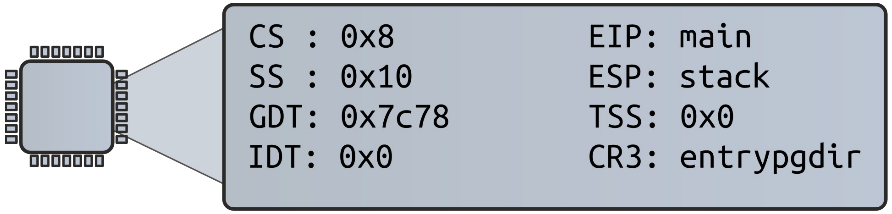
Physical page allocator

- Goal:
  - List of free physical pages
  - To allocate page tables, stacks, data structures, etc.
  - Remember current page table is only 1! page

- Where to get memory to keep the list itself?
  - 1 level, only 4MB entries
    - You don't even have space to keep the second level page tables

Physical page allocator





```
CS : 0x8          EIP: main
SS : 0x10         ESP: stack
GDT: 0x7c78       TSS: 0x0
IDT: 0x0          CR3: entrypgdir
```

Protected Mode

```
2780 kinit1(void *vstart, void *vend)
2781 {
...
2784   freerange(vstart, vend);
2785 }


2801 freerange(void *vstart, void *vend)
2802 {
2803   char *p;
2804   p = (char*)PGROUNDUP((uint)vstart);
2805   for(; p + PGSIZE <= (char*)vend; p += PGSIZE)
2806     kfree(p);
2807 }
```

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

Kernel page table

```
main(void)
{
  kinit1(end, P2V(4*1024*1024)); // phys page allocator
  kvmalloc(); // kernel page table
  mpinit(); // collect info about this machine
  lapicinit();
  seginit(); // set up segments
```

kvmalloc()

```
kvmalloc(void)
{
  kpgdir = setupkvm();
  switchkvm();
}
```

```
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)
       return 0;
  return pgdir;
}
```

Kalloc() - kernel allocator

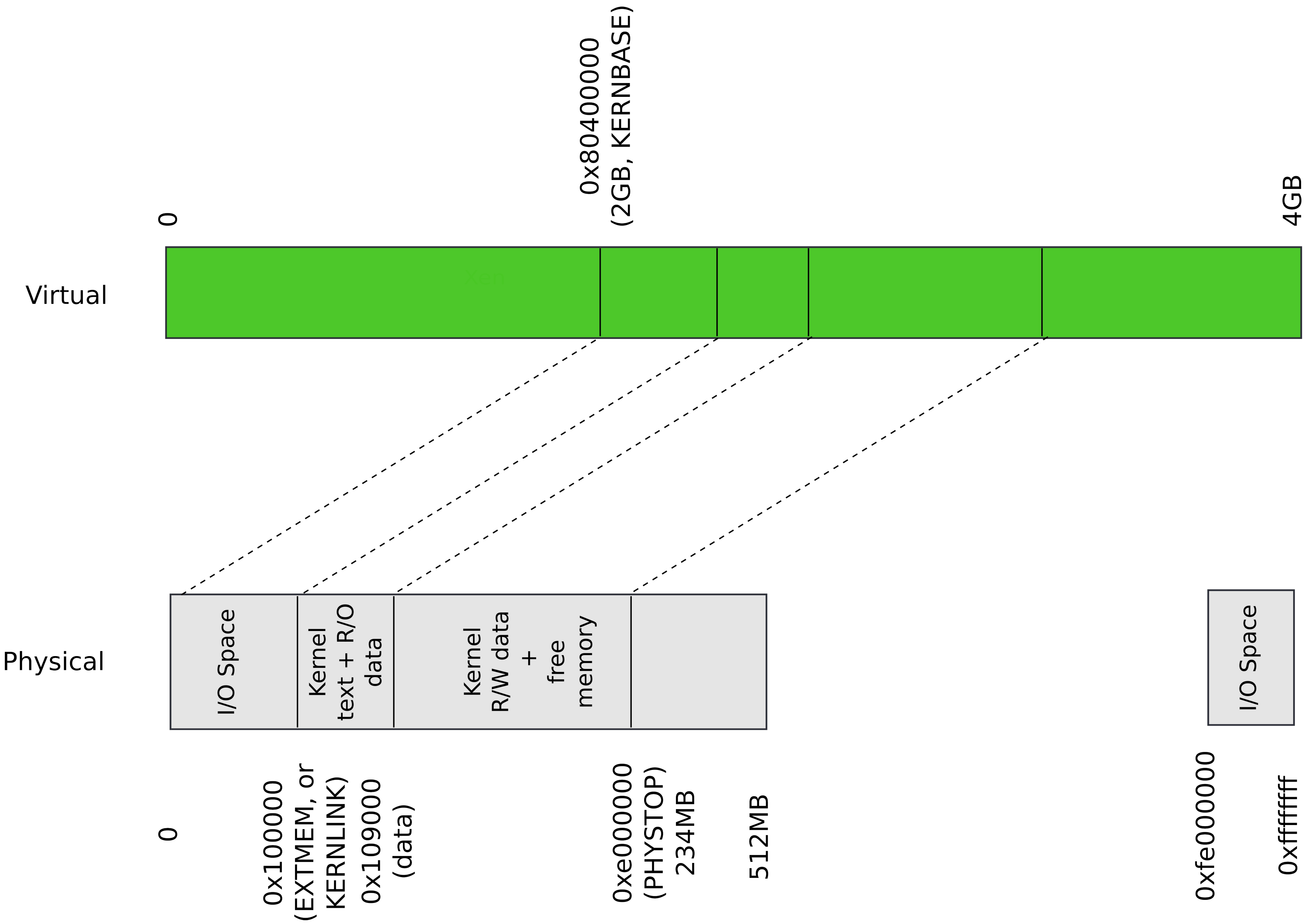
```
char*
kalloc(void)
{
  struct run *r;
...
  r = kmem.freelist;
  if(r)
    kmem.freelist = r->next;
…
  return (char*)r;
}
```

```
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)
       return 0;
  return pgdir;
}
```

Kmap – kernel map

```
static struct kmap {
void *virt;
uint phys_start;
uint phys_end;
int perm;
} kmap[] = {
{ (void*)KERNBASE, 0, EXTMEM, PTE_W}, // I/O space
{ (void*)KERNLINK, V2P(KERNLINK), V2P(data), 0}, // kern text+rodata
{ (void*)data, V2P(data), PHYSTOP, PTE_W}, // kern data+memory
{ (void*)DEVSPACE, DEVSPACE, 0, PTE_W}, // more devices
};
```


Virtual
Physical
0
0x80400000
(2GB, KERNBASE)
4GB
Xen
I/O Space
Kernel
text + R/O
data
Kernel
R/W data
+
free
memory
I/O Space
0
0x100000
(EXTMEM, or
KERNLINK)
0x109000
(data)
0xe000000
(PHYSTOP)
234MB
512MB
0xfe000000
0xffffffff

```
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;
}
```

PDX()

```
// +---------10-------+--------10--------+----------12------------+
// | Page Directory |    Page Table  | Offset within Page  |
// |    Index       |     Index      |                     |
// +-------------------+-------------------+-------------------------+
// \--- PDX(va) --/  \--- PTX(va) --/

// page directory index
#define PDX(va) (((uint)(va) >> PDXSHIFT) & 0x3FF)
...
#define PDXSHIFT 22 // offset of PDX in a linear address
```

```
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;
    // Make sure all those PTE_P bits are zero.
    memset(pgtab, 0, PGSIZE);
…
    *pde = v2p(pgtab) | PTE_P | PTE_W | PTE_U;
  }
  return &pgtab[PTX(va)];
}
```

```
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;
    // Make sure all those PTE_P bits are zero.
    memset(pgtab, 0, PGSIZE);
…
    *pde = v2p(pgtab) | PTE_P | PTE_W | PTE_U;
  }
  return &pgtab[PTX(va)];
}
```

```c
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)
       return 0;
  return pgdir;
}
```

kvmalloc()

```
kvmalloc(void)
{
  kpgdir = setupkvm();
  switchkvm();
}
```

Switch to the new page table

```
void
switchkvm(void)
{
  lcr3(v2p(kpgdir));
}
```

```
main(void)
{
  kinit1(end, P2V(4*1024*1024)); // phys page allocator
  kvmalloc(); // kernel page table
  mpinit(); // collect info about this machine
  lapicinit();
  seginit(); // set up segments
  cprintf("\ncpu%d: starting xv6\n\n", cpu->id);
  picinit(); // interrupt controller
  ioapicinit(); // another interrupt controller
  consoleinit(); // I/O devices & their interrupts
  uartinit(); // serial port
  pinit(); // process table
  tvinit(); // trap vectors
```

```
seginit(void)
{
  struct cpu *c;
...
  c = &cpus[cpunum()];
  c->gdt[SEG_KCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, 0);
  c->gdt[SEG_KDATA] = SEG(STA_W, 0, 0xffffffff, 0);
  c->gdt[SEG_UCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, DPL_USER);
  c->gdt[SEG_UDATA] = SEG(STA_W, 0, 0xffffffff, DPL_USER);

  // Map cpu, and curproc
  c->gdt[SEG_KCPU] = SEG(STA_W, &c->cpu, 8, 0);

  lgdt(c->gdt, sizeof(c->gdt));
  loadgs(SEG_KCPU << 3);

  // Initialize cpu-local storage.
  cpu = c;
  proc = 0;
}
```

```
main(void)
{
  kinit1(end, P2V(4*1024*1024)); // phys page allocator
  kvmalloc(); // kernel page table
  mpinit(); // collect info about this machine
  lapicinit();
  seginit(); // set up segments
  cprintf("\ncpu%d: starting xv6\n\n", cpu->id);
  picinit(); // interrupt controller
  ioapicinit(); // another interrupt controller
  consoleinit(); // I/O devices & their interrupts
  uartinit(); // serial port
  pinit(); // process table
  tvinit(); // trap vectors
```

Interrupt descriptor table

```
void
tvinit(void)
{
  int i;

  for(i = 0; i < 256; i++)
    SETGATE(idt[i], 0, SEG_KCODE<<3, vectors[i], 0);
  SETGATE(idt[T_SYSCALL], 1, SEG_KCODE<<3,
          vectors[T_SYSCALL], DPL_USER);
...
}
```

Thank you!