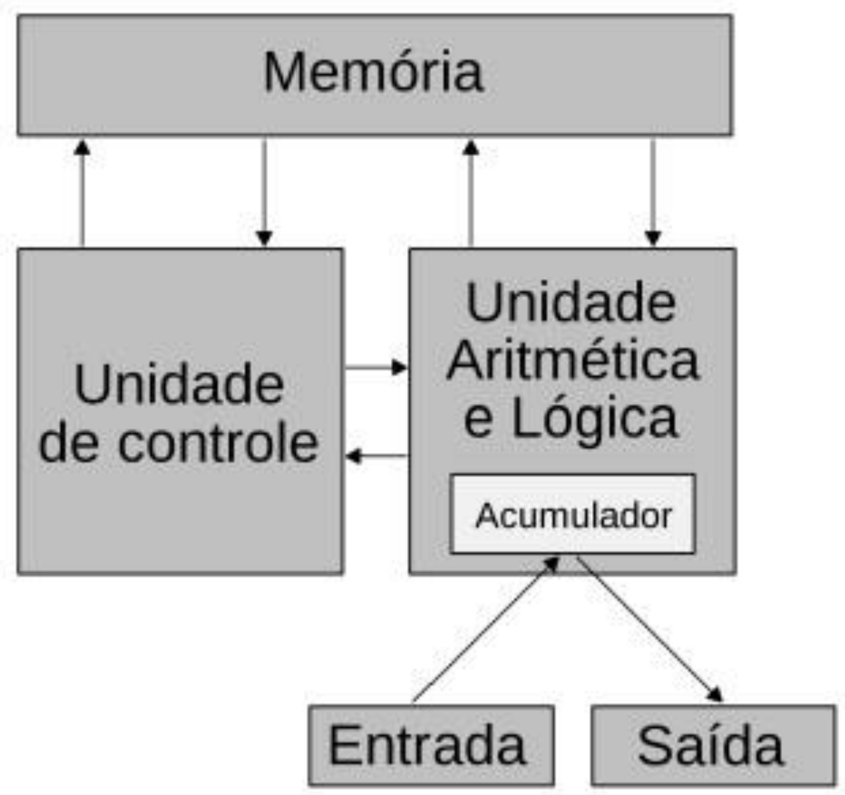
Gerenciamento de Memória do xv6

Mestrado UFSCar – Sistema Operacional Avançado
Felipe da Silva Braz
(2016)

Arquitetura von Neumann (1952)



Endereçamento:

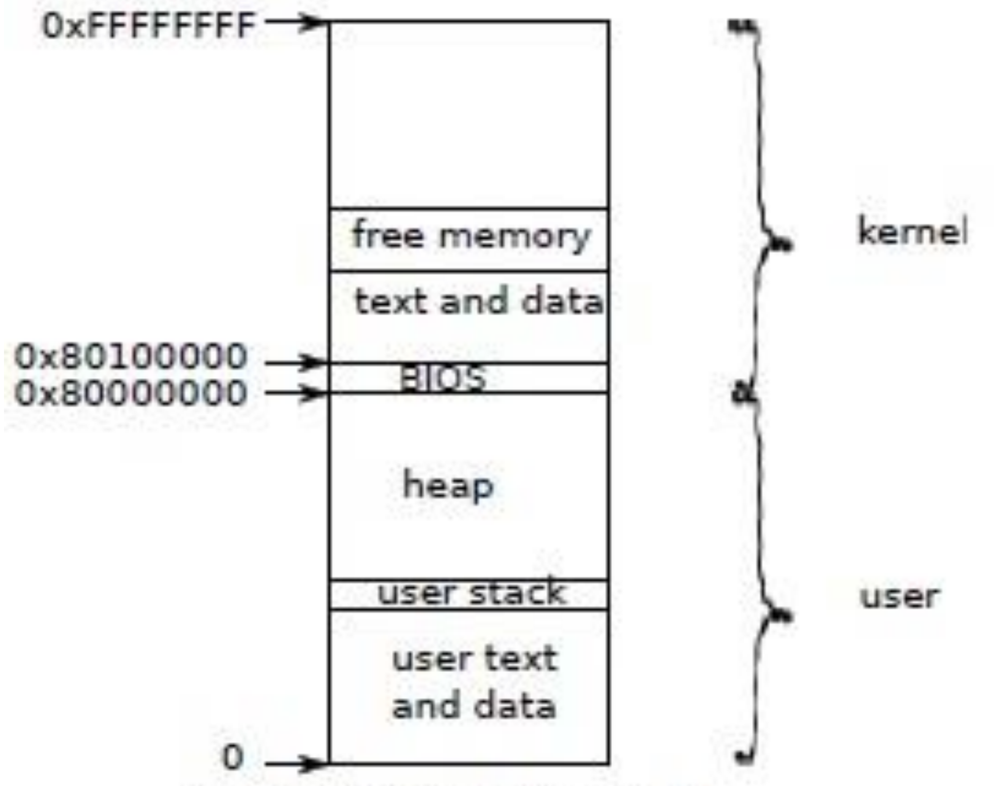
- As informações na memória são localizadas por meio de sua posição; e
- Cada posição da memória possui um endereço de localização.

Sistema xv6

- Ele utiliza um kernel monolítico, ou seja, todo o sistema operacional é executado com privilégio completo de hardware.

- A memória é divida em uma interface pro kernel e outra para usuário. Exemplo do que é executado na interface do kernel: Chama de sistemas como fork, exec, open, close, read, write, etc.

Memória Virtual do xv6



Layout of a virtual address space

Interface do sistema

Interface do usuário

Carregando o Sistema xv6

- No intervalo 0xa0000 : 0x100000 contém dispositivos E/S
- O kernel xv6 é carregado na memória no endereço físico 0x100000.
- Para que o resto do kernel seja carregado é mapeado a tabela de página iniciando em 0x80000000 (KERNBASE)

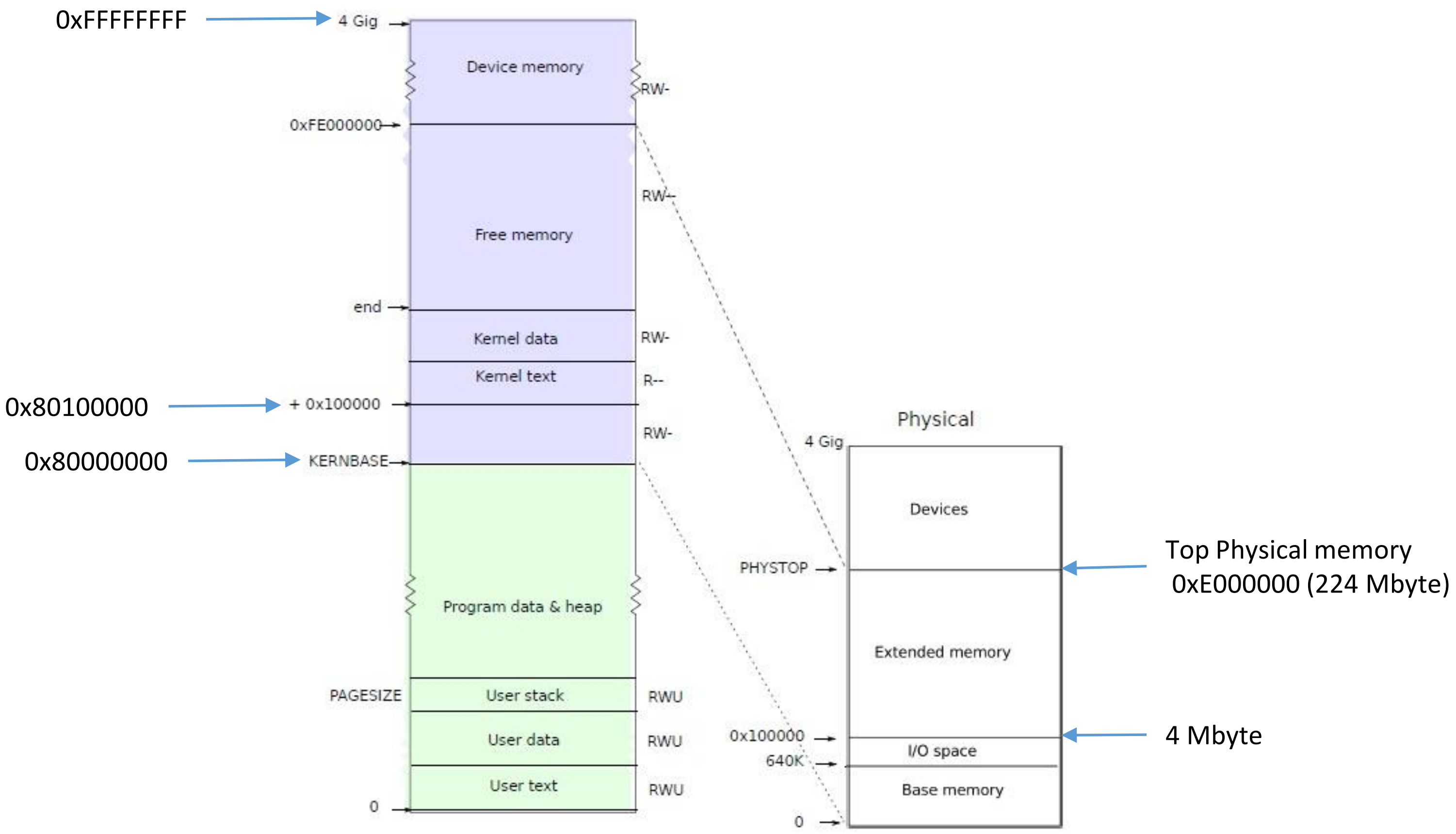
Bootasm.S

```
#include "asm.h"
#include "memlayout.h"
#include "mmu.h"

# Start the first CPU: switch to 32-bit protected mode, jump into C.
# The BIOS loads this code from the first sector of the hard disk into
# memory at physical address 0x7c00 and starts executing in real mode
# with %cs=0 %ip=7c00.

.code16                       # Assemble for 16-bit mode
.globl start
start:
  cli                         # BIOS enabled interrupts; disable

  # Zero data segment registers DS, ES, and SS.
  xorw    %ax,%ax             # Set %ax to zero
  movw    %ax,%ds             # -> Data Segment
  movw    %ax,%es             # -> Extra Segment
  movw    %ax,%ss             # -> Stack Segment

  # Physical address line A20 is tied to zero so that the first PCs
  # with 2 MB would run software that assumed 1 MB.  Undo that.
seta20.1:
  inb     $0x64,%al               # Wait for not busy
  testb   $0x2,%al
bootasm.S                                                    1,1           Top
```

Bootasm.S

```
//PAGEBREAK!
  # Complete transition to 32-bit protected mode by using long jmp
  # to reload %cs and %eip.  The segment descriptors are set up with no
  # translation, so that the mapping is still the identity mapping.
  ljmp    $(SEG_KCODE<<3), $start32

.code32  # Tell assembler to generate 32-bit code now.
start32:
  # Set up the protected-mode data segment registers
  movw    $(SEG_KDATA<<3), %ax    # Our data segment selector
  movw    %ax, %ds                # -> DS: Data Segment
  movw    %ax, %es                # -> ES: Extra Segment
  movw    %ax, %ss                # -> SS: Stack Segment
  movw    $0, %ax                 # Zero segments not ready for use
  movw    %ax, %fs                # -> FS
  movw    %ax, %gs                # -> GS

  # Set up the stack pointer and call into C.
  movl    $start, %esp
  call    bootmain

  # If bootmain returns (it shouldn't), trigger a Bochs
  # breakpoint if running under Bochs, then loop.
  movw    $0x8a00, %ax            # 0x8a00 -> port 0x8a00
bootasm.S                                                    69,1          71%
```

Memória xv6



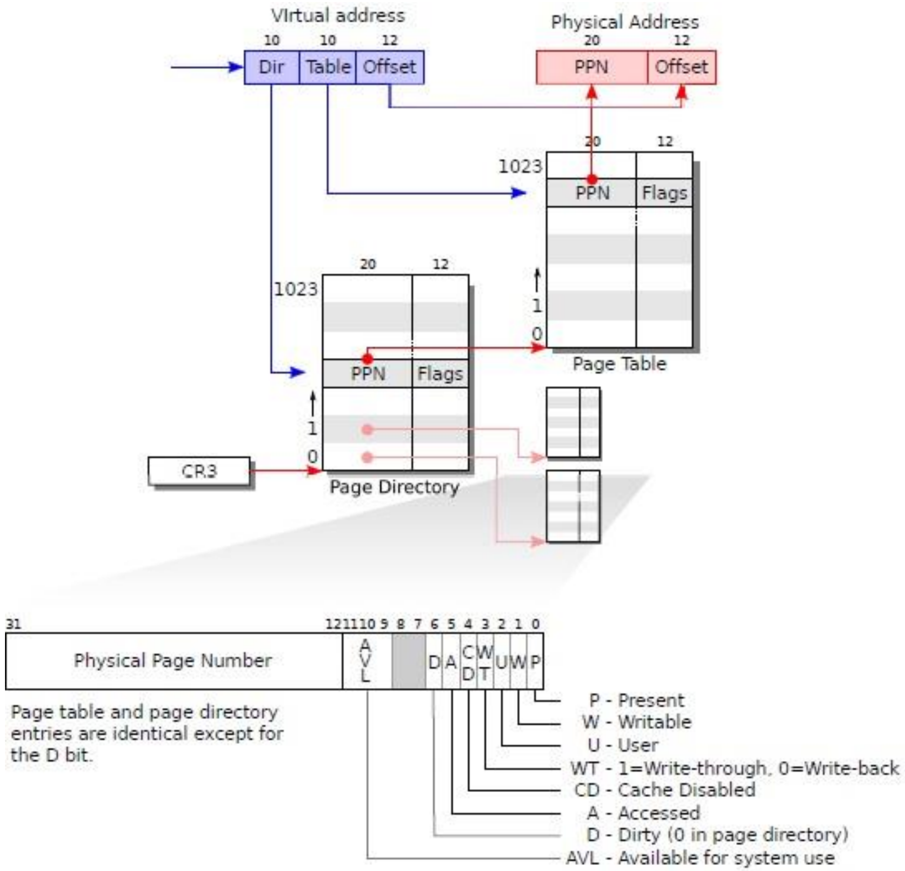
0xFFFFFFFF
4 Gig
Device memory
RW-
0xFE000000
RW--
Free memory
end
Kernel data
RW-
Kernel text
R--
0x80100000
+ 0x100000
RW-
0x80000000
KERNBASE
Program data & heap
PAGESIZE
User stack
RWU
User data
RWU
User text
RWU
0
Physical
4 Gig
Devices
PHYSTOP
Top Physical memory
0xE000000 (224 Mbyte)
Extended memory
0x100000
640K
I/O space
4 Mbyte
Base memory
0

Layout de Memória do xv6

cesar@cesar-VirtualBox: ~/xv6-public

```c
// Memory layout

#define EXTMEM  0x100000            // Start of extended memory
#define PHYSTOP 0xE000000           // Top physical memory
#define DEVSPACE 0xFE000000         // Other devices are at high addresses

// Key addresses for address space layout (see kmap in vm.c for layout)
#define KERNBASE 0x80000000         // First kernel virtual address
#define KERNLINK (KERNBASE+EXTMEM)  // Address where kernel is linked

#ifndef __ASSEMBLER__

static inline uint v2p(void *a) { return ((uint) (a))  - KERNBASE; }
static inline void *p2v(uint a) { return (void *) ((a) + KERNBASE); }

#endif

#define V2P(a) (((uint) (a)) - KERNBASE)
#define P2V(a) (((void *) (a)) + KERNBASE)

#define V2P_WO(x) ((x) - KERNBASE)    // same as V2P, but without casts
#define P2V_WO(x) ((x) + KERNBASE)    // same as P2V, but without casts
memlayout.h                                                1,1            All
"memlayout.h" 22L, 828C
```

Paginação xv6 (i386)



MMU

O MMU gerencia a memória, traduz o endereço virtual em físico e mapeia o endereço nas tabelas.

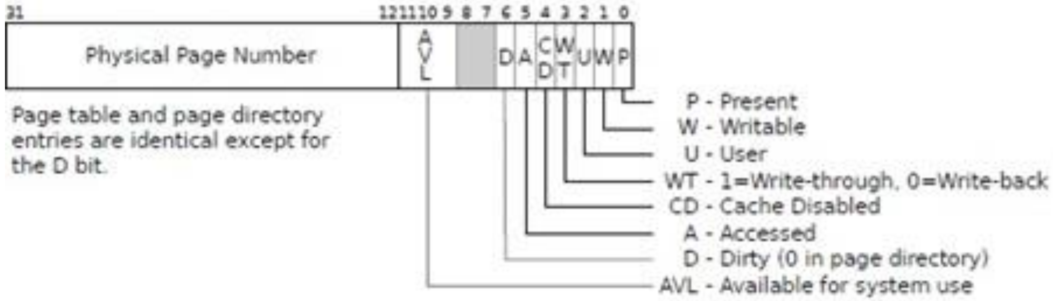
Gerenciamento de Memória – mmu.h

```
// A virtual address 'la' has a three-part structure as follows:
//
// +--------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)

// page table index
#define PTX(va)         (((uint)(va) >> PTXSHIFT) & 0x3FF)

// construct virtual address from indexes and offset
#define PGADDR(d, t, o) ((uint)((d) << PDXSHIFT | (t) << PTXSHIFT | (o)))

// Page directory and page table constants.
#define NPDENTRIES      1024    // # directory entries per page directory
#define NPTENTRIES      1024    // # PTEs per page table
#define PGSIZE          4096    // bytes mapped by a page

#define PGSHIFT         12      // log2(PGSIZE)
#define PTXSHIFT        12      // offset of PTX in a linear address
mmu.h                                                       125,1          50%
```

Gerenciamento de Memória – mmu.h

cesar@cesar-VirtualBox: ~/xv6-public

```
#define PDXSHIFT        22      // offset of PDX in a linear address

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

// Page table/directory entry flags.
#define PTE_P           0x001   // Present
#define PTE_W           0x002   // Writeable
#define PTE_U           0x004   // User
#define PTE_PWT         0x008   // Write-Through
#define PTE_PCD         0x010   // Cache-Disable
#define PTE_A           0x020   // Accessed
#define PTE_D           0x040   // Dirty
#define PTE_PS          0x080   // Page Size
#define PTE_MBZ         0x180   // Bits must be zero

// Address in page table or page directory entry
#define PTE_ADDR(pte)   ((uint)(pte) & ~0xFFF)
#define PTE_FLAGS(pte)  ((uint)(pte) &  0xFFF)

#ifndef __ASSEMBLER__
typedef uint pte_t;
mmu.h                                          146,0-1        61%
```

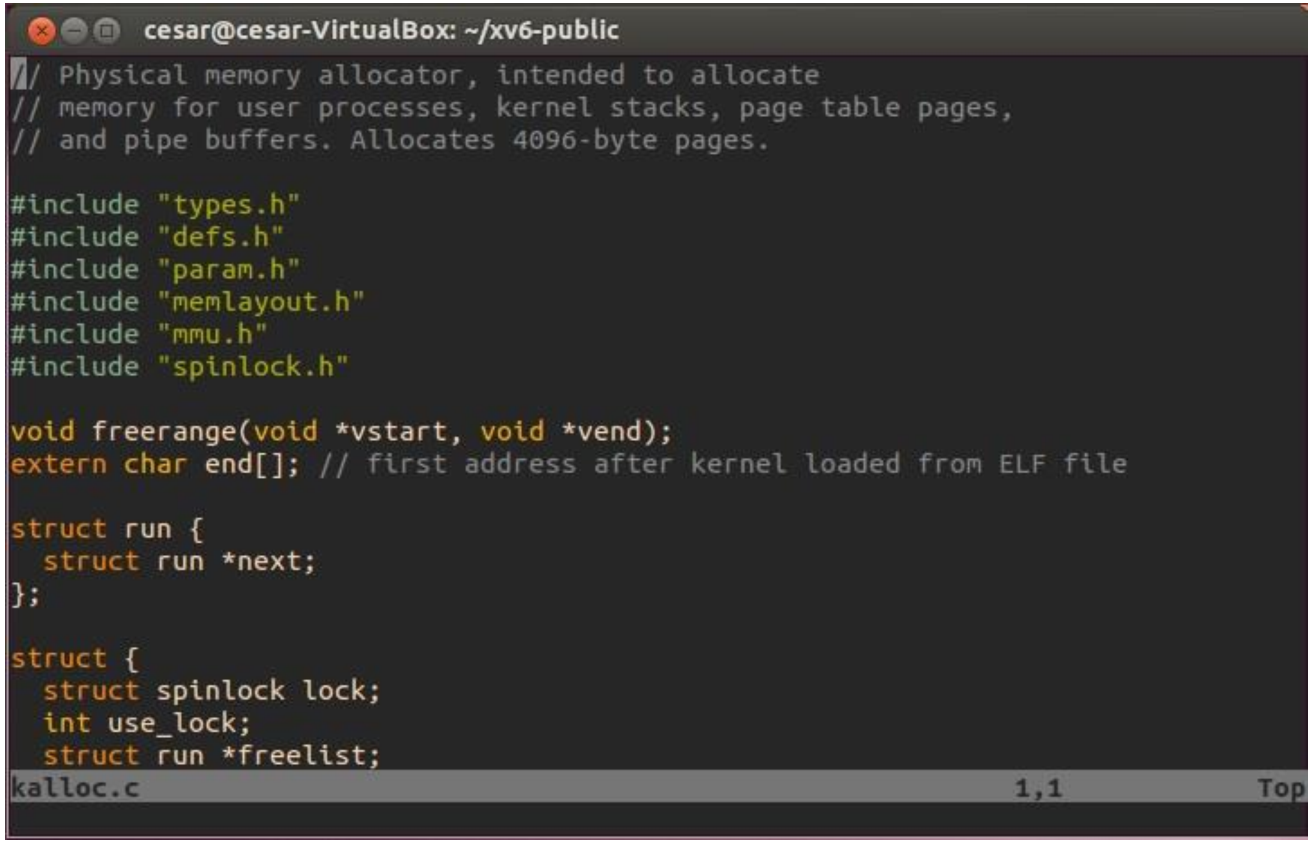


Criar Tabelas – vm.c

```c
// Create PTEs for virtual addresses starting at va that refer to
// physical addresses starting at pa. va and size might not
// be page-aligned.
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;
}
```

vm.c 67,1 18%

Alocação de Memória do Kernel – kalloc.c



Referências

Cox, Russ; Kasshoesk, Frans; Morris, Robert. xv6 a simple, Unix-like teaching operating system. <http://pdos.csail.mit.edu/6.828/xv6>. Acessado em: 03 de nov 2016.

Souza, Beatriz. Introdução à Comutação: Arquitetura von Neumann. <http://inf.ufes.br/~bfmartins/wp-contente/uploads/2015/04/INFO9300-Aula-13-Arquitetura-von-Neumann-Parte-1.pdf>. Acessado em: 03 de nov 2016.