# 日曜研究室

技術的な観点から日常を綴ります

# [xv6 #12] Chapter 1 – The first process – Code: creating an address space

テキストの21〜22ページ

### 本文

main関数は、カーネルを実行するために必要とされるKERNBASEより上位への対応付けを持つページテーブルを、生成したり切り替えたりするために、kvmalloc関数を呼ぶ。
（kvmalloc関数のソースは前回参照）
kvmalloc関数の機能の大部分をsetupkvm関数が担っている。
setupkvm関数はまず、ページディレクトリを保持するためのメモリ領域のページを割り当てる。
そしたら、カーネルが必要とするkmap配列の中に記された変換をインストールするために、mappages関数を呼ぶ。その変換は、カーネルの命令とデータ、PHYSTOPまでの物理メモリ、実際のI/Oデバイス用のメモリ範囲を含む。
setupkvm関数は、ユーザメモリに関する対応付けはインストールしない。
ユーザメモリに関する対応付けは後で行われる。
（この段落で言及されている部分をvm.cから抜粋）

```
01 // This table defines the kernel's mappings, which are present in
02 // every process's page table.
03 static struct kmap {
04   void *virt;
05   uint phys_start;
06   uint phys_end;
07   int perm;
08 } kmap[] = {
09   { (void*) KERNBASE, 0,             EXTMEM,    PTE_W},  // I/O
   space
10   { (void*) KERNLINK, V2P(KERNLINK), V2P(data), 0}, // kernel
   text+rodata
11   { (void*) data,     V2P(data),     PHYSTOP,   PTE_W},  // kernel
   data, memory
12   { (void*) DEVSPACE, DEVSPACE,      0,         PTE_W},  // more
   devices
13 };
14
15 // Set up kernel part of a page table.
```

```
16 pde_t*
17 setupkvm()
18 {
19   pde_t *pgdir;
20   struct kmap *k;
21
22   if((pgdir = (pde_t*)kalloc()) == 0)
23     return 0;
24   memset(pgdir, 0, PGSIZE);
25   if (p2v(PHYSTOP) > (void*)DEVSPACE)
26     panic("PHYSTOP too high");
27   for(k = kmap; k < &kmap[NELEM(kmap)]; k++)
28     if(mappages(pgdir, k->virt, k->phys_end - k->phys_start,
29                 (uint)k->phys_start, k->perm) < 0)
30       return 0;
31   return pgdir;
32 }
33
34 // Allocate one page table for the machine for the kernel address
35 // space for scheduler processes.
36 void
37 kvmalloc(void)
38 {
39   kpgdir = setupkvm();
40   switchkvm();
41 }
```

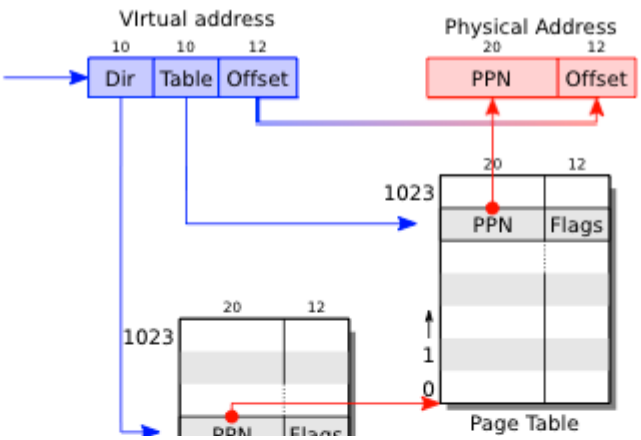
mappages関数は、仮想アドレスの範囲から物理アドレスに対応する範囲のための対応付けをページテーブルへインストールする。
対応付けられる仮想アドレス全てに対して、mappages関数はそのアドレスのPTEのアドレスを探すためにwalkpgdir関数を呼ぶ。
そして、適切な物理ページ番号を保持するようにするためにそのPTEを適切なパーミッション（PTE_WとPTE_U）で初期化し、そして準備完了ということでそのPTEにPTE_Pを設定する。
（この段落で言及されている部分をvm.cから抜粋）

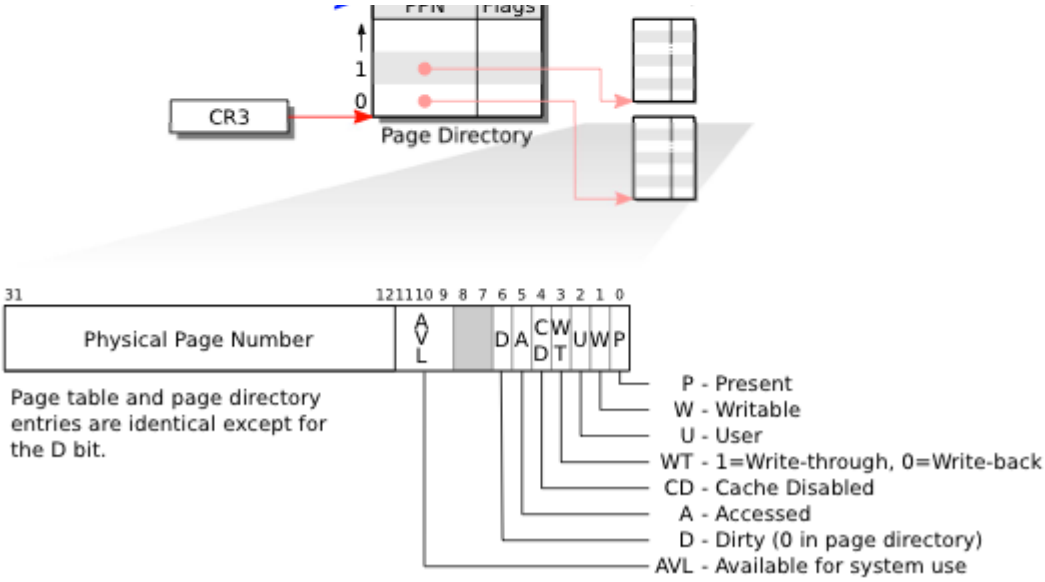
```
01 // Return the address of the PTE in page table pgdir
02 // that corresponds to virtual address va.  If alloc!=0,
03 // create any required page table pages.
04 static pte_t *
05 walkpgdir(pde_t *pgdir, const void *va, int alloc)
06 {
07   pde_t *pde;
08   pte_t *pgtab;
09
10   pde = &pgdir[PDX(va)];
11   if(*pde & PTE_P){
12     pgtab = (pte_t*)p2v(PTE_ADDR(*pde));
13   } else {
14     if(!alloc || (pgtab = (pte_t*)kalloc()) == 0)
15       return 0;
16     // Make sure all those PTE_P bits are zero.
17     memset(pgtab, 0, PGSIZE);
18     // The permissions here are overly generous, but they can
19     // be further restricted by the permissions in the page table
20     // entries, if necessary.
21     *pde = v2p(pgtab) | PTE_P | PTE_W | PTE_U;
22   }
23   return &pgtab[PTX(va)];
24 }
25
```

```
26 // Create PTEs for virtual addresses starting at va that refer to
27 // physical addresses starting at pa. va and size might not
28 // be page-aligned.
29 static int
30 mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)
31 {
32   char *a, *last;
33   pte_t *pte;
34
35   a = (char*)PGROUNDDOWN((uint)va);
36   last = (char*)PGROUNDDOWN(((uint)va) + size - 1);
37   for(;;){
38     if((pte = walkpgdir(pgdir, a, 1)) == 0)
39       return -1;
40     if(*pte & PTE_P)
41       panic("remap");
42     *pte = pa | perm | PTE_P;
43     if(a == last)
44       break;
45     a += PGSIZE;
46     pa += PGSIZE;
47   }
48   return 0;
49 }
```

walkpgdir関数は、仮想アドレス変換のためにx86のページングハードウェアがPTEを参照する動きをエミュレートする。（図1-1参照）
walkpgdirは、ページディレクトリエントリを見つけるために、仮想アドレスの上位10ビットを利用する。
（pde = &pgdir[PDX(va)]; の部分。マクロPDX()で上位10ビットを取得している。）
ページディレクトリエントリが準備出来ていなければ、要求されたページテーブルのページはまだ割り当てられていないということになる。
もし引数allocがセットされていれば、walkpgdir関数はそれを割り当て、そのページディレクトリの中の物理アドレスに配置する。
最後に、ページテーブルのページの中のPTEのアドレスを見つけるために、仮想アドレスの次の10ビットを利用する。
（return &pgtab[PTX(va)]; の部分。マクロPTX()で上位10ビットのさらに次の10ビットを取得している。）

図1-1 x86上のページテーブル（以前も載せましたがもう一回載せときます）






### 感想

xv6がアドレス空間をどう作ってるかですね。
ソースは普通のCなんで読みやすいですが、どういうメモリ操作をしてるのか具体的にイメージしながら読まないと説明との対応がよく分からないかと思います。

ここはまだ起動途中の話なのでそれを忘れないようにしないと。
（OSが起動した後、プロセスを起動するときなどに動く処理ではない）

カテゴリー: 技術 | タグ: xv6 | 投稿日: 2012/2/16 木曜日 [http://peta.okechan.net/blog/archives/1273] |