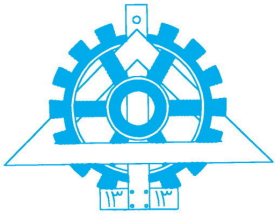
پروژه آزمایشگاه شماره ۵



دانشکده مهندسی برق و کامپیوتر

سیستم عامل - پاییز ۱۳۹۹

استاد : **دکتر کارگهی**

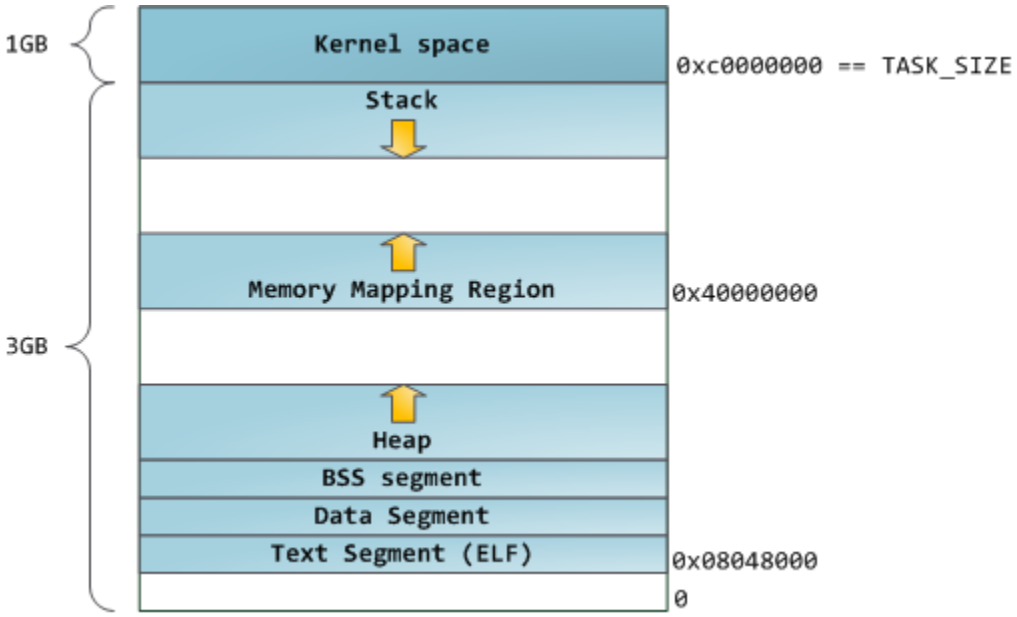
گروه ۱۷

اعضای گروه:

دانشور امراللهی، علیرضا توکلی، امین ستایش

---

۱. ساختار حافظه مجازی (مشابه شکل بالا) یک برنامه در لینوکس در معماری x86 (سی و دو بیتی) را نشان دهید.



۲. چرا ساختار سلسله‌مراتبی منجر به کاهش مصرف حافظه می‌شود؟

در ساختار سلسله‌مراتبی، process ها و task ها به راحتی می‌توانند با به اشتراک گذاشتن کدها و داده‌ها توسط mapping بخش مناسب به صفحات فیزیکی از مصرف اضافی حافظه جلوگیری کنند. [منبع]

۳. محتوای هر بیت یک مدخل (۳۲ بیتی) در هر سطح چیست؟ چه تفاوتی میان آن‌ها وجود دارد؟

در مدخل سطح page directory برای اشاره به سطح بعدی از ۲۰ بیت استفاده می‌شود.

همچنین ۱۲ بیت برای سطح دسترسی نگه‌داری می‌شود.

این ۱۲ بیت در هر دو سطح وجود دارد اما در سطح page table از ۲۰ بیت برای آدرس صفحه فیزیکی استفاده می‌شود.

در بیت D یعنی بیت dirty با هم تفاوت دارند.

در page directory این بیت به این معنا است که صفحه باید در دیسک نوشته شود تا تغییرات اعمال شود.

اما در page table این بیت معنایی ندارد.

۴. تابع kalloc چه نوع حافظه‌ای تخصیص می‌دهد؟ (فیزیکی یا مجازی)

حافظه فیزیکی تخصیص می‌دهد این کار را با صدا زدن تابع kalloc انجام می‌دهد.

توضیحات روبرو هم در کد xv6 برای این تابع ذکر شده است که نشان‌دهنده همین موضوع است:

```
// Allocate one 4096-byte page of physical memory.
// Returns a pointer that the kernel can use.
// Returns 0 if the memory cannot be allocated.
```

۵. چگونه می‌توان روی یک آدرس فیزیکی خاص نوشت؟ دستور مورد نظر و شیوه دسترسی به آدرس مربوطه را ذکر کنید.

می‌توان از دستور mmap استفاده کرد. این دستور به این صورت است که در صورت استفاده از آن، می‌توان یک فایل یا دیوایس را به یک پردازه map کرد. پس از این عمل، دیگر فایل مانند یک آرایه در برنامه می‌تواند استفاده شود. برای این کار می‌توان از کتاب‌خانه‌ی

```
#include <sys/mman.h>
```

استفاده کرد.

این تابع آرگومان‌های زیر را برای استفاده می‌گیرد.

```
void * mmap(void *address, size_t length, int protect, int flag, int filedes, off_t offset)
```

برای دیدن جزئیات بیش‌تر می‌توانید به این لینک مراجعه کنید.

در انتهای لینک، مثالی از نحوه‌ی کار با این تابع آورده شده است. که همان‌طور که توضیح داده شد، نحوه‌ی استفاده از این تابع دقیقا مانند آرایه‌های برنامه است.

۶.تابع mappages چه کاربردی دارد؟

این تابع به این منظور استفاده می‌شود که حافظه مجازی را به حافظه فیزیکی متصل کرد.

همچنین این تابع صفحه جدید را به pgdir اضافه می‌کند.

کد این تابع به صورت زیر است و در توضیحات کامنت‌شده هم همین موضوعاتی که اشاره شد مشخص است:

```
// 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;
}
```

۷.راجع به تابع walkpgdir توضیح دهید. این تابع چه عمل سخت‌افزاری را شبیه‌سازی می‌کند؟

این تابع آدرس PTE موجود در pgdir را بازمی‌گرداند، همچنین اگر لازم باشد جدول مورد نیاز را می‌سازد.

این تابع عمل سخت‌افزاری ترجمه آدرس مجازی به فیزیکی را شبیه‌سازی می‌کند.

کد این تابع به صورت زیر است:

```
// Return the address of the PTE in page table pgdir
// that corresponds to virtual address va.  If alloc!=0,
// create any required page table pages.
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;
    // Make sure all those PTE_P bits are zero.
    memset(pgtab, 0, PGSIZE);
    // The permissions here are overly generous, but they can
    // be further restricted by the permissions in the page table
    // entries, if necessary.
    *pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
  }
  return &pgtab[PTX(va)];
}
```