نوشتن يك كرنل ساده به زبان C همراه با توابع اصلي printf و clearscreen

ما در اين مقاله قصد داريم كه به شما نشون بديم نوشتن يك كرنل ساده هستش براي شروع كرنل ما در فايل Kernel_Start.asm هست.

```
[BITS 32]
[global start]
[extern _k_main]
start:
    call _k_main          ; اين تابع در يك فايل سي نوشته شده كه به برنامه لينك
cli   ; غير فعال كردن وقفه ها
hlt   ; متوقف كردن سي پي يو
```

اين يك كد ۳۲ بيتي هست (به اين دستور توجه كنيد [BITS 32]) و كرنل ما تابع k_main هست كه در فايل C قرار داره. اما در فايل اسمبلي اين تابع رو k_main_ صدا زديم ! دليلش اين هست كه كامپايلرهاي زبان C/C++ يك زيرخط در زمان كامپايل به اول اسم هر تابع اضافه مي كنند مگر قرار باشه كه بخواين اين فايل رو به يك ELF لينك كنيد چون فايلهاي ELF نيازي به زير خط اول موقع فراخواني روالهاي سي (لينك) ندارند و مي توانيد تابع رو به همون صورتي كه در سي هست فراخواني كنند . بعد از فراخواني تابع k_main دستور عمل CLI اجرا مي شود . اين دستور عمل تمام وقفه ها رو غير فعال مي كنه و ديگه وقفه اي اتفاق نمي افتد . سپس دستور HLT اجرا مي شود اين دستور به CPU مي گويد اجرا كردن دستور عمل ها بعدي رو متوقف كن (بعد از اين دستور ديگه دستور عملي اجرا نميشه شايد با خودتون بگيد خوب اگر بخوايم دوباره فعال كنيم چطوري بايد اين كار رو كرد اجراي دوباره برنامه با استفاده از وقفها صورت مي گيرد اگر وقفه ها غير فعال باشند سيستم عملا هنگ مي كنه) ما ميتونيم به جاي HLT از دستور $ JMP نيز استفاده بكنيم اما اين دستور باعث ميشه كه سيكل هاي CPU به هدر بره و به CPU فشار بياد و داغ كنه چون به طور مرتب فقط به مكان خودش در حال پرش هست اما دستور عمل ما (HLT) اين كار رو نميكنه و واقعا CPU رو غير فعال مي كنه و هيچ دستور عملي رو اجرا نميكنه كه به سيستم فشار بياد .
نكته : وقفه ها ميتوانند CPU رو از حالتHLT در بياورند اما چرا ما قبل از دستورعمل HLT وقفه ها غير فعال كرديم براي اين هست كه مي خواهيم سيستم كاملا از كار بيفته .

حالا اجازه بدهيد كه ما متغيير ها و توابع اصلي رو در فايل Kernel.c تعريف كنيم . توابع ما اينها هستند حال در ذيل به توضيح هر كدامشون مي پردازيم

```
#define WHITE_TXT 0x07          // متن سفيد بر روي زمينه سياه
void k_clear_screen();
unsigned int k_printf(char *message, unsigned int line);
void update_cursor(int row, int col);
```

اين #define WHITE_TXT 0x07 هيچ كار خواصي انجام نميده. ما در جلوش گفتيم كه اين متغيير چه كار مي كنه فقط اين متغيير رو بياد داشته باشيد
حالا اجازه بدهيد كه ما به سمت تعريف تابع k_Kernel برويم .

```
k_main()    // این تابع در برنامه ما مثل تابع main در C هست
 {
    k_clear_screen();
    k_printf("Hi!\nHow's this for a starter OS?", 0);
};
```

تابع k_main نقطه ورود ما به كرنل خودمون هست. ما اين تابع رو در فايل اسمبلي Kernel_Start.asm صدا زده ايم .
تابع k_clear_screen كارش پاك كردن صحفه نمايش هست و تابع k_printf() هم كارش نمايش متن بر روي صحفه نمايش هستش .

پارامتر دوم تابع k_printf() كارش رسم متن بر روي خط مورد نظر هست (صفر خط اول , يك خط دوم , دو خط سوم و الي آخر) و **\n** نيز همانند تابع printf در C/C++ كارش رفتن به خط جديد هست

هورا !
اين چطور بود براي شروع يك سيستم عامل ؟

در مود حفاظت شده شما نمي توانيد از وقفه ها بايوس براي پاك كردن صحفه نمايش استفاده كنيد به همين دليل ما براي خودمون يك تابع نوشتيم كه مستقيما به صحفه نمايش دسترس داره و صحفه رو پاك مي كند .

```
void k_clear_screen()  // این تابع صحفه نمایش رو پاك مي كند
 {
      char *vidmem = (char *) 0xb8000;
      unsigned int i=0;
          while(i < (80*25*2))
            {
              vidmem[i]=' ';
              i++;
              vidmem[i]=WHITE_TXT;
              i++;
            };
 };
```

در تابع بالا (k_clear_screen) اشاره گر (*vidmem*) اشاره مي كنه به آدرس 0xB8000 كه آدرس صحفه نمايش در مود محافظت شده است . ما متغيير رو از نوع *char* (يك بايت) تعريف كرديم پس ما مي توانيم در هر لحظه يك بايت رو در حافظه صحفه نمايش بنويسيم مود متني در ريزپردازنده هاي سري X86 به طول و عرض 25 x 80 كاراكتر هست هر كاراكتر به دو بايت نياز داره بايت اول براي خود كاراكتر و بايت دوم نيز خاصيت كاراكتر رو كنترل مي كنه مثل رنگ و چشمك زن بودن كاراكتر رو مشخص مي كنه . ما با ضرب طول و عرض 25*80 مي توانيم مقدار كاراكتر هاي كه مي توانيم بر روي صحفه نمايش رسم كنيم رو بدست مي آوريم و ما اين عدد بدست آمده رو ضرب در 2 مي كنيم ما در هر لحظه به يك بايت در صحفه نمايش دسترسي داريم و ما با استفاده از دستور vidmem[i]=' '; مي توانيم يك فاصله و يا هيچ كاراكتري چاپ نشه يا يك فضاي خالي ايجاد كنيم و ما عدد 1 رو به i (++i) اضافه مي كنيم و متغيير ما I دقيقا به اندازيه يك نقطه به جلو برده مي شود . حالا بايت بعدي خاصيت هست و ما در آن عدد 0x07 رو قرار مي دهيم . 0x07 تعيين مي كنه يك پشت زمينه سياه با رو زمينه (كاراكتر) سفيد و بدون چشمك زن در مود متني رسم شود .

حالاپيش به سوي تابع k_printf .

```
unsigned int k_printf(char *message, unsigned int line)
// the message and then the line #
  {
    char *vidmem = (char *) 0xb8000;
    unsigned int i=0;
       i=(line*80*2);
       while(*message!=0)
        {
          if(*message=='\n') // check for a new line
            {
              line++;
              i=(line*80*2);
              *message++;
            } else {
              vidmem[i]=*message;
              *message++;
              i++;
              vidmem[i]=WHITE_TXT;
              i++;
            };
         };
return 1;
};
```

تابع k_printf خيلي شبيه تابع k_clear_screen كار ميكنه . while(*message!=0) اين حلقه , كاراكترها رو مي خونه تا به پايان رشته برسه و دوباره تابع رو ادامه بده . خط if(*message=='\n') در رشته به دنبال كاراكتر \n مي گردد اگر اين كاراكتر رو پيدا كرد به خط جديد مي رود و كاراكتر بعدي در ابتداي خط بعدي چاپ مي شود . اگر كاراكتر ما يك \n نبود ما كاركتر رو در داخل حافظه صحفه نمايش قرار مي دهيم و خاصيت كاراكتر رو 0x07 قرار مي دهيم (پشت زمينه سياه و متن سفيد بدون حالت چشمك زن)

كامپايل كردن كرنل

اول شما سورس كد كرنل رو دريافت كنيد و همچنين شما به اسمبلر (NASM) , كامپايلر C (GCC يا DJGPP) و به يك لينكر (LD) نياز داريد .

حالا شما در بالاي فايل لينكر (link.ld) اين خط رو مي بينيد :

.text 0x100000

شما نياز داريد براي لود كردن كرنل به حافظه مورد نظر آدرسش رو به صورت هگزيمال بنويسيد در اين مورد محل مورد نظر **1MB** است كه (در حالت هگزيمال مي شود 0x100000) در اينجا بايد كرنل بار شود

اول شما فايل اسمبلي تون رو كامپايل كنيد

nasm –f aout kernel_start.asm –o –ks.o

اين فايل ks.o نيز مانند kernel_start.asm با فرمت aout كامپايل مي شود . حالا براي C نيز اين دستور رو اجرا مي كنيم

gcc –c kernel.c –o kernel.o

قدم بعد و آخرين قدم هم لينك كردن دو فايل ks.o و kernel.o و تبديل كردنشون به يك فايل است در اين مورد ما نياز داريم كه فايل رو به فرمت بيناري (Binary) و حالات تخت (Flat) تبديل كنيم كه ما براي اين كار از فايل اسكريپتي (link.ld) كه براي اينكار (Link) ساختيم استفاده مي كنيم ما اين دو فايل رو با اين دستور به هم وصل مي كنيم :

Ld –T link.ld -o kernel.bin ks.o kernel.o

اين خيلي مهمه كه اول ks.o لينك بشه بعد kernel.o اگر شما اين دو فايل رو جا به جا لينك كنيد كرنل كار نخواهد كرد ! كرنل پس از بوت شدن و اجراي بوت لودر و رفتن به مد محافظت شده و فعال شدن A20 اجرا خواهد شده . اگر شما دوست داريد كه كرنل تون گراب (GRUB) داشته باشه شما مي تونيد گراپ رو از سايتش دانلود كنيد(به همون روشي كه در بالا گفته شده به برنامه تان لينك كنيد)

نتيجه گيري

شما الان يك كرنل ساده داريد ! شايد شما يك تابع k_printf بهتري بخواهيد داشته باشيد شما مي تونيد از اين مثال استفاده كنيد و امكانات(s% , d% , c% و الي اخر) بيشتري به تابع تون اضافه كنيد شما مي تونيد در آينده مثال هاي بهتري بنويسيد

نكته : شما قبل از اجرا كرنل بايد به مود محافظت شده برويد