X86 Mimarisi İçin 32 Bit İşletim Sistemi

Hüseyin Can Ercan
Bilgisayar Mühendisliği Bölümü, Yıldız Teknik Üniversitesi
huscanernca@gmail.com

Abstract - **Yapılan çalışma temel bir işletim sisteminin Bootlader aşamasından kernelin ilklendirilmesine kadar geçen sürecin gerçeklenmesini ve işletim sistemi üzerinde temel fornksiyonları sağlayacak parçaların sırasıyla eklenmesini hedeflemektedir. Çalışma için bootloader aşaması da dahil olmakla beraber İntel mimarisindeki bir ortam için QEMU ile emülayon yapılarak geliştirme ve testler gerçekleştirilecektir.**

Keywords – **Operating System, Bootloader, Kernel Inıtialization**

I. GİRİŞ

İşletim sistemleri temel olarak yazılım ve donanım arasında yalıtım(abstraction) sağlayan sistemlerdir. Herhangi bir yazılım donanım üzerinde rahatlıkla tüm kaynaklara erişerek çalışabilir, tabiki en düşük seviyede donanım kontrolü sağlayacak araçlara sahip olmak şartıyla. Bu durumda çalışan yazılımı değiştirmek istediğimizde sistemi yeniden başlatmamız gerekecektir. Ayrıca sistem aynı anda sadece tek bir işlemi gerçekleştirebilecektir. Bu limitleri aşabilmek için işletim sistemleri kullanılmaktadır.

II. METOD

İşletim sistemleri aynı anda birden fazla işlemin birlikte yönetilmesi, işlerin birbirlerinin kaynaklarını ihlal etmeden aynı kaynak havuzunu kullanabilmeleri gibi özellikleri sağlar. En yüksek seviyede yetkide çalıştırılan kernel ile sistem üzerinde tam denetime sahip olan işletim sistemlerinin herhangi bir şekilde hataya sebep olmaması beklenmektedir. Bu seviyede gerçekleşecek hatalar en basit olarak işlemci üzerinde triple fault ile sonuçlaranak sistemin devam edemez duruma gelmesine sabep olacaktır.

Intel mimarisi geriye yönelik olarak destek sağlamaktadır. Bu çalışmada kullanacağımız i386 konfigürasyonu ile çalıştıracağımız QEMU emulatörü 32 bitlik bir işlemciyi emule edcektir. Bootloader aşamasından itibaren kontrol edeceğimiz süreç için Intel'in geriye yönelik desteğinden kaynaklanan bazı özellikleri değiştirememiz gerekecektir.

İşlemciyi 32 bitte çalışacak moda getirdikten sonra donanım üzerindeki kontrolümüzü büyük ölçüde kaybetmiş olarak kerneli yükleyerek çalıştırma aşamasına geleceğiz. İşlemci başlangıç aşamasında 16 bit modda çalışacaktır. 32 bite geçiş işlemi BIOS tarafından sağlanan interrupt fonksiyonlarınının tamamen kaybedilmesine sebep olacaktır. Bu aşamaya kadar tüm disk operasyonlarını tamamlamamız gerekmektedir.

32 bite geçiş ile kernelin başlatılması arasında geçen sürede sistemin durumunu takip edebilmek için VGA bufferlarına veri yazarak ekran kontrolünü sağlamaya devam edebiliriz fakat diğer birçok temel işlemi sağlayamıyor olacağız, özellikle interrupt gerektirenler. Bu nedenle IO geresiniminin bu aşamaya kadar tamamlanmış olması gereklidir.

Kernel diskten okunmuş ve çalıştırılmak istenilen alana kopyalanmış olmak zorundadır.

III. GERİYE UYUMLULUK

Assambler olarak NASM kullanılmaktadır. NASM pure binary formatında çıktı üretebilmesi sebebiyle özellikle boot aşamasında çalışacak kodun üretilmesinde büyük kolaylık sağlamaktadır. Derlenen dosyalar FAT12 formatına sahip bir diske yazılarak QEMU'ya beslenmektedir. QEMU verilen disk imajına normal boot sürecini uyguladığında ilk olarak boot sectordeki limitli fonksiyona sahip binary kodu çalıştırmaktadır. Boot sector 512 byte boyutunda tek bir sektördür. BIOS, POST aşamasını geçtiğinde yetkiyi boot sectordeki koda devretmektedir. Tek sektör kabul edilebilir ki çok limitlik fonksiyona sahip kodu barındırabilecektir.

Bu aşamada bot işleminde yapmak istediğimiz birçok işlem için bootloaderın ikinci aşama kodun geçmemiz çözüm sağlayacaktır.

Sistemin 16 bit modda başladığı göz ardı edilmemelidir. Intel mimarisi bahsedildiği üzere geriye yönelik destek sağlamaktadır ve daha ilkel işlemcilerle uyumlu olarak çalışmaya başlayacaktır. CPU üzerindeki control registerları ile bu duruma müdahale edebilriz fakat doğrudan ve sadece Control Register 0 üzerinde 32 bite geçiş işlemi gerçekleştirilirse triple fault ile sonuçlanacaktır. Öncelikli olarak Global Descriptor Table üzerinde tercihen tüm memory üzerinde erişim sağlayacak şekilde code ve data için kayıt oluşturmamız gerekmektedir. İleriki aşamalarda bu tabloyu user space ve kernel space ayırımı için de kullanacağız. Ardından Control Register 0 üzerindeki ilk biti set ederek protected mode aktif hale getirilebilir. Protected mode geçişi ile işlemci artık 32 bit için derlenmiş pure binary kodu çalıştırabilir duruma gelecektir. Takip eden aşamalarda 32 bir registerları kullanmaya başlayabiliriz.

İntel 80386 işlemcisine sahip bir işlemciyi ve anakartı ile çalıştığımızı göz önünde bulundurursak bir ayrıntıya dikkat etmemiz gereklidir, yine geriye yönelik destek sağlanması sebebiyle adresleme pinlerinden A20 (21. pin) sistem başlatıldığında aktif değildir. Bu pini belli adreslerdeki portları kullanarak keyboard controllera gerekli komutları göndererek aktif hale getirebiliriz. Bu uygulama genel anlamda çevre birimleri ile iletişim kurulması için temel mantığın oluşmasında ve uygulanmasında da yardımcı olacaktır.

İşlemci çeşitli operasyonların gerçekleştirilmesi için çevre birimleri ile iletişim kurmak durumundadır. Bu keyboard controller ile A20 pininin aktifleştirilmesi olabileceği gibi ekranda çıktı gösterilmesi de olabilir. BIOS tarafından sağlanan interrupt fonksiyonları artık erişilebilir olmadığı için en düşük seviyede donanım kontrolünü sağlayacak tüm kodun etklenmesi zorunluluk haline gelmektedir, aksi halde kontrol sağlayacak alternatif bir yöntem mevcut değildir.

IV. DONANIM KONTROLÜ

Boot aşamasında kernelin çalışmaya balaşadığı zamana kadar geçen zaman dilimin takip edebilmek için ekran çıktılarını kullanabiliriz. Bunun için VGA bufferlarına erişerek istediğimiz veriyi pixel veya text modunda ekranda gösterebiliriz. Ekran cursorunun konumunu CRT controllera gönderilecek komutlar ve ver ile değiştirebiliriz. Kernel başlangıcına kadar geçen bu süre zafında minimum ölçüde ihtiyaç duyduğumuz donanım etkileşiminin assembly ile uyumlu olacak şekilde gerçeklenmesi gerekmektedir. Kernel aşamasında C ile tam kapsamlı donanım kontrolü sağlayacak parçalar sağlanacaktır.

Donanım kontrolü en alt seviyede supervisor modda kullanıldığında herhangi bir limitlemeye tabi tutulmamaktadır. Eğer kullanan kod parçası hata barındırırsa kolaylıkla donanıma hasar verebilecek işlemleri gerçekleştirebilir. Donanım üzerinde sadece kernelin tam yetkiye sahip olması hedeflenmektedir. Aksi durumda kernel dahilindeki arayüz ne kadar güvenli olursa olsun uygulama katmanından bir kod parçarsı rahatlıkla hasar verici işlemleri gerçekleştirebilir. İşetim sistemi üzerinden elde edilmeye çalışılan fayda donanımın tüm kaynaklar başlığı altında toplanarak uygulamalar için kullanıma sunulmasıdır.

V. SONUÇ

İşletim sistemi temel fikir olarak sadece kod çalıştırabilen ve dış dünya ile etkileşime geçebilen bir donanım sistemi üzerinde çalıştırılamaz. Global Descriptor Table ve Paged Memory Management örneklerinde olduğu gibi işletim sistemi temel yaklaşım ile doğrudan uyumlu davracak donanım mimarisine de ihtiyaç duyar. Aksi durumda stabil çalışmaya imkan verecek sınırlandırma gerçekleştirilemez. Bu nedenle özel görevlere sahip ekstra donanımların ve işlemci davranışını etkileyecek konfigürasyon kontrollerinin eklenmesi kaçınılmazdır.

REFERENCES

[1] Michael D. Schroeder and Jerome Saltzer , A Hardware Architecture for Implementing Protection Rings, Communications of the ACM , March 1972

[2] Nasm Instruction Set Reference

[3] QEMU version 2.12.50 User Documentation

[4] Broken Thorn Entertaintment Operating Sytem Series, 2009