

<Ülkü SAYILAN tarafından Turkce'ye cevrimistir..>

FreeBSD Kernel'i Okumak

2005 Eylül ayında, FreeBSD kernel konusunda bilgilenmek isteyen Halil Demirezen ile yazışma fırsatı buldum. Verdiğim yanıt genel bir ilgi alanını kapsadığından, bu yanıtı başkalarının ilgi alanına girebileceği düşüncesiyle web sayfamda yayınladım.

From rwatson at FreeBSD.org Salı, Eylül 6 14:10:10 2005

Tarih: Tue, 6 Sep 2005 14:10:10 +0100 (BST)

From: Robert Watson <rwatson at FreeBSD dot org

To: Halil Demirezen <halil at enderunix dot org

Konu: Kayda değer bir soru değil, ama benim için önemli.

6 Eylül, Salı, 2005, Halil Demirezen yazmış:

Yüzbinlerce klasik* sorudan bir tanesidir bu. Ama bir için gerçekten önemli bir soru. FreeBSD'de kernel'le (/usr/src/sys/) uğraşıyorum. Fakat bu gerçekten zorlu bir uğraş. Mezuniyet tezim floppy disketten (3 1/2) önyüklenen (boots) küçük bir i386 kerneldi. Korunmuş mod'da çalışıyordu. Bir shell küçük memory işletimli ve multiprocess-çok işlemciliydi.

FreeBSD'yi anlayıp, hacklemekle uğraşıyorum. Nereden başlamalıyım? Temel unsurlarını anlamak gerçekten zor. Çok zaman harcamam gerektiğinden eminim. Fakat bir kernel genişlemenin gelişmedeki ilk temel unsurları nelerdir? FreeBSD'yi baştan beri geliştiren biri için bunun kolay bir şey olduğunu biliyorum. Ama kod geliştikçe, başlamak da olanaksızlaşıyor. Önyükleme (booting) ile başlamayı denedim. Ama kernel yapısı akıldan kaçıp gitti. Kaynak dosyalardan birini anlamaya çalıştığım zaman, hepsini birleştirmek için birkaç tanesini açmak zorunda kalıyorum.

Halil,

Evet, FreeBSD devasa bir software parçasıdır - 8-10 senedir sürekli olarak onunla çalışıyorum ama sadece küçük bir parçasını anlayabildim. Diğer yandan, şu da ortaya çıktı ki, bir şeyi kısmen anlamak işe yarar bir çalışma için hala yeterlidir :-)

Kernelin yapısını öğrenmeye başlamak için birkaç yol var. McKusick ve Neville-Neil's Design ve FreeBSD uygulama kitabından hala edinmemişsen hemen bir tane edin. Kernel'e giriş anlamında bir rehber olmasa da birçok önemli subsystems hakkında derinlemesine bilgi sunar. Ben onu okumaktan ziyade bir başvuru kaynağı olarak kullanıyorum, ama oldukça da yardımcı oluyor.

Bir başka yol da, kaynak kod okuma perspektifinden başlamak olacaktır. Eğer yukarıda sözünü ettiğim kitaba göz attıysan veya okuduysan, kerneli keşfetmek için web sitelerini tarayan bir dizi kaynak kodlardan birini kullanmanı öneririm.

Ne öğrenmek istediğine bağlı olarak başlayabileceğin birkaç yer var. İşte deneyebileceğin birkaç tanesi:

http://fxr.watson.org/fxr/ident?i=mi_startup

mi_startup() makineden bağımsız sistem önyükleme işlevi (machine-independent system boot function) demek, alt düzeydeki hardware sıfırlandığı (*initialize: hem sıfırlamak hem de başlangıç durumuna getirmek demek*) zaman ortaya çıkar ve SYSINIT()'i, yani önyükleme zaman kayıt mekanizmasını (the boot time registration mechanism) başlatmakla (kick-off) yükümlüdür. Çeşitli kernel öğeleri, subsystem ID'lerini kullanmaları komutlanan SYSINIT() makrolarını kullanarak önyüklemede çalışması gerekli olan cod'u açıklar. Subsystemlerin bir listesini ve komutlanmalarını burada bulabilirsiniz:

http://fxr.watson.org/fxr/ident?i=sysinit_sub_id

Proc() intialization işleviyle de ilgileniyor olabilirsin -- process 0, kernel process'i ve swapper'ı (*getir-götürçüsü; deęiş tokuşçusu; trampacısı*) haline gelir ve sysnit yoluyla, mi_startup() tarafından başlatılır.

http://fxr.watson.org/fxr/ident?i=proc0_init
http://fxr.watson.org/fxr/ident?i=proc0_post

Başka bir önemli başlama noktası da process 1 kernel code'udur, yani init(8)'i çalıştıran, sırası geldiğinde de kullanıcı space'in başlamasını sağlayan kernel process'i.

http://fxr.watson.org/fxr/ident?i=create_init
http://fxr.watson.org/fxr/ident?i=start_init
http://fxr.watson.org/fxr/ident?i=kick_init

Start-init() işlevi, process 0 içindeki sysinit tarafından çalıştırılan create_init() tarafından yaratılan process 1 içinde çalışır. Init yaratılır yaratılmaz, yine sysinit'i kullanan process 0 tarafından çalıştırılan kick_init() tarafından çalıştırılacaktır.

Mi-startup()'dan önce makineye-baęlı önyükleme code'u (machine-dependent boot code) çalışır. Detaylar, platformuna göre çok çeşitlidir, ama code hemen hemen her zaman "locore" olarak adlandırılır, önyükleme yükleyicisi (boot loader) tarafından yüklenen kernel için giriş noktasıdır.

<http://fxr.watson.org/fxr/source/i386/i386/locore.s>

Genellikle kernel set up'ını tatbik etmekten sorumludur, kernel hafızasını sergiler (lay out), stackleri (*yığımları; istifleri*) hazırlar. Jolice'in "Kernel Kaynak Kodları Sırları" adlı iki ciltlik bir kitabı var, ilk dönem 386BSD kerneli anlatır. Code o zamandan bu yana epeyce deęişmiş olsa da, hala locore'u ve dięer birkaç kernel parçasını anlamak açısından yararlı bir okuma sağlayabilir.

Önyükleme process'ini anlamak için sysinitler özellikle ilginçtir, çünkü kernel linker setlerine dayanırlar ve mekanik olarak hem önyükleme process'ine hem de modül yükleme processine çengellenirler (hook into). Sysinit işlevleri, sıralamada ortaya çıktıkları sırayla çalışırlar, birçok kernel subsysteminin sysinitin etrafına sarılmış (wrapped) macroları kullanan öğeleri register (*tescil etmek; kaydetmek; uymak*) ettiğini göreceksin. Örneğin, VFS_SET() sanal dosya sistemi uygulamasının initialize'larını açıklayan bir makrodur, dosya sitemlerinin kendilerini, sonradan kullanılmak üzere, register etmelerine izin verir.

http://fxr.watson.org/fxr/ident?i=VFS_SET

Kernel'i incelemek ve kaynağı keşfetmek açısından bir diğer perspektif de, **user space**'e hizmet sağlamakta hazır ve nazır olmaktır. En kullanışlı başlama noktası şudur:

<http://fxr.watson.org/fxr/source/kern/syscalls.master>

Bu master sistem **çağrı (call)** tanımlama dosyasıdır, buradan `init_sysent.c` gibi diğer dosyalar üretilir (generate). `syscalls.master`'da listelenen her bir sistem call'ı, kernelde aynı adla bulunan işlev tarafından uygulanır. Örneğin, `sync()` `syscalls.master`'da açıklanmıştır ve `vfs_syscalls.c` içindeki `sync()` adlı işlev tarafından uygulanır:

<http://fxr.watson.org/fxr/ident?i=sync>

Böylece orada başlayan sistem call'ları tarafından kullanılan kod yollarını inceleyebilirsiniz.

Elbette, system call'ları bir kernel'e giriş yapmak için tek yol değildir. Bir dizi tuzak vardır, bazıları VM sistemine götürür, bazıları da sinyal işleyicilerine, **math emulator**'lara, kesme (interrupt) işleyicilerine falan götürür. Alt düzey tuzak kodu, makine-bağımlıdır ama genellikle `trap()` adlı işlev tarafından uygulanacaktır:

<http://fxr.watson.org/fxr/ident?i=trap>

Bu birçok başka şeyi de ortaya çıkaracaktır, process için sistem call vektörü bunlara dahildir, `syscalls.master` dan ya da bir emulator varyantından, sayfa arıza (fault) işleyicisinden vb.den türetilmiştir (derived from).

http://fxr.watson.org/fxr/ident?i=trap_pfault

Bir sistem call'undan dönüşte (ve başka koşullarda), `userret()` de ilginçtir:

<http://fxr.watson.org/fxr/ident?i=userret>

Umarım, bu, göz atmaya başlamak için yararlı olacaktır.

Robert N M Watson