İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

AYGITLARIN UZAK ADRESLENMESİ

YÜKSEK LİSANS TEZİ

Müh. Ozan Eren BİLGEN

Tezin Enstitüye Verildiği Tarih : 11 Nisan 2005

Tezin Savunulduğu Tarih : 16 Haziran 2005

Tez Danışmanı : Prof.Dr. Emre HARMANCI

Diğer Jüri Üyeleri Prof.Dr. Nadia ERDOĞAN

Prof.Dr. Oya KALIPSIZ (YTÜ)

TEMMUZ 2005

ÖNSÖZ

Yüksek lisans öğretimimi sürdürme olanağı sağlayan, bu çalışmaya konu olan altyapıya ihtiyaç duyduğunu belirterek bana fikir veren ve tamamlayıp tezimde kullanma fırsatı tanıyan Sayın Proje Yürütücüm Başuzman Araştırmacı Ali ÇIKIKÇI'ya,

Aygıt sürücüsü yazmayı öğrenme sürecinde önerdiği pratik çözümlerden dolayı Sayın Serdar KÖYLÜ'ye,

Ağ uygulama geliştirme konusundaki tecrübesini paylaşan Sayın Dr. Nazım KOÇ'a,

Bilgi ve emeklerini Özgür Yazılım Felsefesi çerçevesinde paylaşan tüm insanlara,

Konuya ilgisinden ötürü Sayın Yrd.Doç.Dr. Turgut UYAR'a,

Ve tez danışmanım Sayın Prof.Dr. Emre HARMANCI'ya;

Teşekkür ederim.

Ozan Eren BİLGEN

İÇİNDEKİLER

KISALTMALAR

NCD	: Network Character Device
TCP/IP	: Transmission Control Protocol / Internet Protocol
CUPS	: Common UNIX Printing System
NAS	: Network Audio System
GNU	: GNU's not UNIX
GPL	: GNU Public Licence

TABLO LİSTESİ

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

Üniversitesi : **İstanbul Teknik Üniversitesi**

Enstitüsü : **Bilişim Enstitüsü**

Anabilim Dalı : **İleri Teknolojiler**

Programı : **Bilgisayar Bilimleri**

Tez Danışmanı : **Prof. Dr. Emre HARMANCI**

Tez Türü ve Tarihi : **Yüksek Lisans – Mayıs 2005**

ÖZET

AYGITLARIN UZAK ADRESLENMESİ

Ozan Eren BİLGEN

Bu çalışmada, işletim sistemlerinde aygıt soyutlama prensipleri üzerinde durulmuş ve aygıtları jenerik olarak uzaktan kullanmaya yarayan Network Character Device (NCD) adlı altyapı tasarlanmıştır. NCD sayesinde eskiden her aygıta özgü servislerle çalışan aygıt paylaştırma hizmetleri tek elde toparlanmış ve uygulamalara doğrudan erişim imkanı tanınmıştır. Bu yapıyı kullanan uygulamaların hiçbir ek yazılım değişikliğine ihtiyaç duymamalarına ek olarak, kullandıklarını fark etmeleri de mümkün gözükmemektedir. Aygıtların uzak adreslenmesi, bu aygıtlara erişen uygulamalara düşük seviyede erişim denetimini mümkün kılmakta ve sistem çağrısı tabanlı sınırlamalar söz konusu olmaktadır. Özgür yazılım temelli Linux işletim sistemi üzerinde gerçeklenen NCD, ait olduğu ekosistemde 1999'dan beri yapılması arzulanan bir çalışmadır.

Anahtar Kelimeler: İşletim sistemleri, Aygıt sürücüsü, Dağıtık sistemler

Bilim Dalı Sayısal Kodu:

University : **İstanbul Technical University**

Institute : **Informatics Institute**

Science Programme : **Advanced Technologies**

Programme : **Computer Sciences**

Supervisor : **Prof. Dr. Emre HARMANCI**

Degree Awarded and Date : **MSc – May 2005**

SUMMARY

REMOTE DEVICE ADDRESSING

Ozan Eren BİLGEN

In this study device abstraction in operating systems is discussed and an infrastructure, which serves to use the devices from remote, named Network Character Device (NCD), is planned. Despite of NCD, former device sharing method, which is running per-device services, is summed in one hand and applications can directly access the devices. An important point is that applications can use NCD without any changes. Remote addressing also enables low-level access control of client applications, including system call based restrictions, which is never been possible before. NCD, which is implemented on free operating system Linux, is planned by the corresponding ecosystem since 1999.

Keywords: Operating systems, Device driver, Distributed systems

Science Code:

1 GİRİŞ

Ağ teknolojilerinin bilgisayar sistemlerini bütünleştirdiği günümüzde, diğer tüm hizmetler gibi aygıt hizmetlerinin de ağ üzeride dağıtılması söz konusudur. Bu sayede hem her bilgisayar için ayrı aygıt temininden dolayı artan masrafların önüne geçilmekte, hem de aygıtların yönetimleri kolaylaşmaktadır.

Aygıtların paylaşımı çok uzun süredir değişik yollarla gerçeklenmiş olsa da, bu yöntemlerin genel prensibi belirlenen bir aygıt türü için istemci ve sunucu yapıları geliştirip, uygulama seviyesinde kütüphane tabanlı destekle uzaktaki aygıtın sunulmasıdır. Yaygın olarak kullanılan XWindow, CUPS ve NAS hizmetleri bu sınıfa girer.

Ancak paylaştırılan aygıt tipi arttıkça gerek istemci/sunucu etkileşiminin yarattığı trafik, gerek uygulamaların ağa dönük arabirimlerinin sayıca artmasından oluşan güvenlik açıkları bütün sistemin sağlığını etkileyen sorunlara davetiye çıkarmaktadır. Bu yapılar aygıta doğrudan erişim imkanı değil kaynağa kısıtlı bir ulaşma yöntemi sağladıkları için gerçek aygıtın sağladığı her hizmeti veremeyebilmektedirler.

UNIX sistemler, yapı olarak merkezi çekirdek kullandıklarından, erişecekleri birincil kaynakları kendi üzerinde görmeyi beklerler. Bu birincil kaynaklar, işlemci ve bellek gibi bir bilgisayarın tek başına çalışması için zaruri olan donanımlardır. Ancak bu zaruriyet, bu donanımların paylaştırılamayacağı anlamına gelmemektedir. Dağıtık işletim sistemlerinde her türlü aygıtın dağıtık ortamda kullanımı ve paylaşımı mecburken, UNIX ve ailesinde bu mümkün gözükmemektedir.

Konuyla ilgili yapılan çalışmalardan yaygın olarak başarıyla kullanılan tek proje Network Block Device'dir. Block device, üzerinde dosya sistemi taşıyan aygıtlara denir ve sabit sürücüden USB belleklere kadar geniş bir yelpazeyi kapsar. Özellikle yüksek erişilebilirlik hizmetlerinde kullanılan NBD, ağ üzerinde bulunan bir bölümleme tablosunu başka bir bilgisayarda biçimleme dahil her türlü yetenekle kullanmaya imkan tanır. Öte yandan, daha karmaşık ve değişik tipte görülen karakter aygıtlarında başarıya ulaşmış bir çalışma tanınmamaktadır.

Bu tezin amacı, UNIX sistemlerde sekizli dizisi olarak yazılıp okunduğundan "karakter aygıtı" olarak isimlendirilen aygıtlara erişen sürücüleri uzak aygıta yönelmiş olarak konumlandırarak, isteğin uzaktaki bir yapıya iletilmesi ve sonucun işlemi kendisi yapmış gibi uygulamaya döndürülmesi prensibine dayanarak çalışan

bir yapının tasarımı ve bu görevi ifşa eden Network Character Device isimli uygulamanın tanıtımıdır. Bu çalışmanın dağıtık çalışması istenen başka tiplerdeki aygıt mimarilerine uyarlanabileceğine dikkat çekilmek istenmektedir.

2 İŞLETİM SİSTEMLERİNDE SÜRÜCÜ KAVRAMI

1940 sonlarında ortaya çıkan ilk bilgisayarlarda, uygulama geliştiriciler işletim sistemi kullanmadan donanıma doğrudan erişirlerdi [1]. Bilgisayar sistemlerinin evrimleşme sürecinde, kaynak soyutlaması getiren bir işletim sistemi katmanı kullanmak zaruri olmuştur. İşletim sistemlerinde, donanıma erişen yazılımlara sürücü denir. Sürücüler, ait oldukları donanıma erişimin soyutlanmış, düzenli ve ortak bir noktadan kotarılmasını sağlar.

Sürücü, cihaza erişim yöntemini tanımlar. Bir başka değişle, sürücü sistemin isteklerini cihazın anlayabileceği bir şekilde ona izah eder. Bu sebepten; değişik işletim sistemleri değişik sürücülere sahip oldukları gibi, değişik aygıtlar da aynı işletim sistemi için olsa bile değişik sürücülere ihtiyaç duyarlar.

Pek çok işletim sisteminde 3 katman görülür. Bunlar:

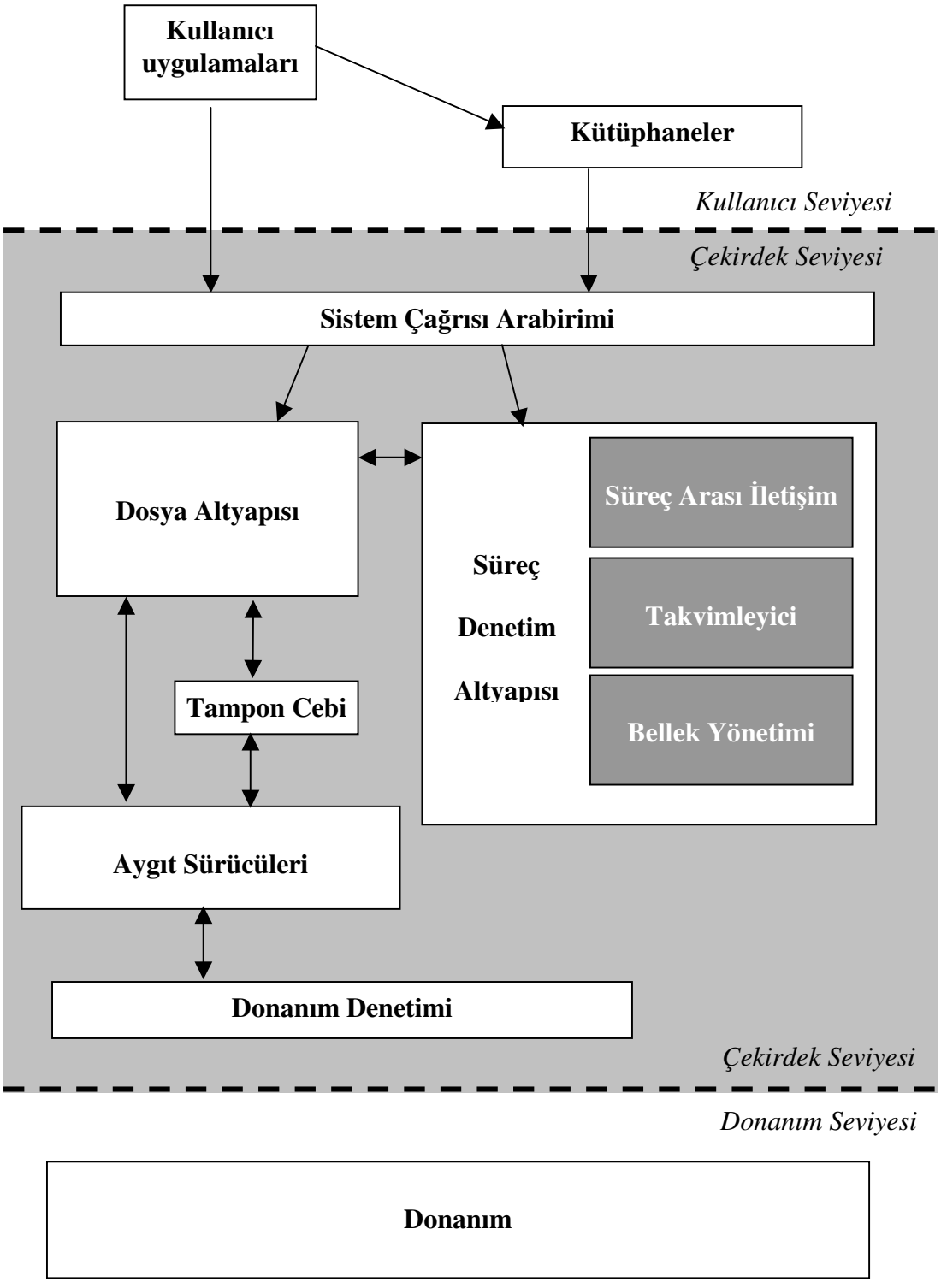
1. Kullanıcı seviyesi: Uygulamaları ve kütüphaneleri kapsar. Bu katmanın amacı kullanıcın istediği görevi bilgisayara yaptırmasını sağlamaktır.
2. Çekirdek seviyesi: Bilgisayarı yöneten, uygulamaları ve donanımı düzenleyen katmandır.
3. Donanım seviyesi: Fiziksel veya mantıksal aygıtları kapsar.

Bu yapıyı şematize edersek, şekil 2.1'deki görüntüyle karşılaşırız.

Görüldüğü üzere, kullanıcı seviyesinin işletim sistemi ile iletişim kurmak için kullandığı yapıya “sistem çağrısı”, çekirdeğin donanımla iletişim kurduğu yapıyaysa “sürücü” denmektedir. Sistem çağrıları ve sürücüler, çekirdek seviyesinde çalışan uygulamalardır ve çekirdeğe ait parçalardır.

Modern işletim sistemleri, kullanıcı seviyesi ve çekirdek seviyesi arasındaki geçişin güvenliğine önem vermektedirler. Bilgisayar bilimlerindeki evrim, bu korumanın donanım tarafından desteklenmesinin önemini göstermektedir. Bazı işlemciler, bu desteği çeşitli katmanlar atıyarak tanımlarlar. Bu katmanlar, iç içe geçmiş daireler şeklinde somutlaştırılabilir. Bu halkalardan en içteki sıfırıncı katman, en dıştaki üçüncü katmandır. En içteki katman, işletim sistemine aittir ve hiçbir güvenlik sınırlaması yoktur. Çekirdek uygulaması, donanıma tamamen hükmedebilir ve hatta çekirdeğin diğer bölümlerini değiştirebilir. En dıştaki katman olan kullanıcı seviyesinde çalışan uygulamalar ise kesin denetimle çalıştırılır, öyle ki kullanacakları

dosyaların bile erişme haklarına bakılır. Arada kalan iki katman genellikle kullanılmamakla beraber sistemin düzgün çalışması için idame hizmetlerine ayrılmıştır.



Şekil 2.1: İşletim sistemi katmanları

Sürücü kullanımının önemi bu noktada öne çıkmaktadır. Sürücü, çekirdek seviyesinde çalışan bir uygulama olduğundan, donanıma doğrudan erişebilir ve düşük seviyeli işlem yapabilir. Sürücü ayrıca kendisini kullanan uygulamayı isteği gerçekleşene kadar uyutabilir, geçersiz bir işlem yaptığında kapatabilir veya sistemin bekası için hilelere başvurabilir. Sürücü kullanmadan, çok görevli bir işletim sisteminde donanımın doğru denetlenmesi mümkün değildir. Bu yüzden sürücü, modern işletim sistemlerinin olmazsa olmazları arasında yer almaktadır.

Sürücü soyutlaması prensibinin pek çok faydası görülmektedir. Bu faydalar aşağıdadır:

- Sürücü sayesinde, ilgili donanıma erişim yöntemleri sadece işletim sisteminin tekil bir bölümünde bilinmekte, bu donanımı kullanacak sayısız uygulama önemli bir rahatlığa kavuşmaktadır. Örnek bir grafik uygulama, ekran, ekran kartı, fare, sabit sürücü, vb. gibi çalışması süresince kullandığı aygıtların özniteliklerini bilmek durumunda değildir. Ayrıca bu sayede, kullanıcı uygulamaları pek çok farklı üreticinin aynı işi gören donanımıyla onlara özgü yaklaşımlarda boğulmadan çalışabilmektedir.
- Gerçek zamanlı olmayan sistemlerde uygulamalar çekirdek tarafından belirli aralıklarla önceden belirtilmeden ve uygulamanın fikri alınmadan takvimlenirler. Sürücü kavramının olmadığı bir ortamda, eğer uygulamalar aygıt koşullanmasının orta yerinde takvimlenirse ve başka bir uygulama benzer bir işe girişirse, bu aygıtın düzgün çalışmasını imkansız kılar. Sürücü kavramı, erişimi belirli bir şeritte sınırlayarak düzenli akmasını sağlayan bir huni vazifesi görmektedir.
- Aygıt ve sürücüdeki değişiklikler uygulamaları etkilemez. Aygıtın iç yapısına ait olan yapılar (örneğin saklayıcılar), bu aygıtı kullanacak uygulama tarafından bilinmek durumunda değildir. Aygıt içi donanımın aygıtın her sürümünde değişebileceği düşünülürse, donanım değişikliğinin sistemde o donanımı kullanan tüm yazılımları değişmeye zorlaması bilişim sistemlerinin bakımını kaotik hale getirir.

Yine de bütün işletim sistemlerinin sürücü kavramını tam olarak gerçekledikleri söylenemez. Özellikle gerçek zamanlı çalışan gömülü sistemlerde, uygulama ve işletim sistemi tamamen iç içe geçmiş durumdadır. Bu sistemlerde, amaç çok belirli ve katı olduğundan özel sürücü metotları kurmak yerine uygulamaya donanıma tam erişme yetkisi tanınabilir. Örneğin MSDOS gibi bazı eski işletim sistemleri, bellek koruması bulundurmadığından, uygulamanın içinden bu sürücüleri eklemek, kesmelere atamalar yapmakla mümkün olmaktadır. Değil çok kullanıcılı, çok görevli

modern bir işletim sisteminin sürücü kavramından uzaklaşınca yaşayacağı sorunlar açıkken, sürücüsüz bir sistemin güvenlik zafiyetleri tartışılmaz boyuttadır.

Sürücü, donanıma erişimi düzenleyen bir yapı olduğundan, tüm modern işletim sistemi tasarımlarında olmazsa olmaz bir parça konumundadır.

3 UNIX SÜRÜCÜ ALTYAPISI

UNIX ailesi işletim sistemlerinde, aygıtlara erişim dosyalar aracılığıyla yapılır [2]. Geleneksel olarak /dev dizininde duran bu dosyalar, 3 tip bilgi taşırlar:

Tablo 3.1: Aygıt erişim dosyası tanımlayıcıları

Tanımlayıcı	***Açıklama***
Aygıt tipi	Blok (b) veya karakter (c) aygıtı olabilir.
Majör numarası	0-255 arasında bir tam sayıdır. Çekirdeğin ilgili sürücüyü bulmasına yarar.
Minör numarası	Yine 0-255 arasında bir tam sayıdır. Sürücünün kendi içinde kullanılır.

UNIX ailesinde, aygıta erişimle dosyaya erişim arasında fark yoktur. Aygıtlar da dosyalar gibi open(2) sistem çağrısıyla açılır ve close(2) çağrısıyla erişim sonlanır. Aygıta yazmak için write(2), okumak içinse read(2) kullanılır. Dosyalarda da benzer işlemler yapıldığı için, sürücü içinde bu çağrılara dosya işlemleri denir. Örnek bir karakter aygıtı, aşağıdaki çağrıları desteklemelidir [3]:

Tablo 3.2: Sürücü işlevleri

İşlev adı	***Açıklama***
open	Aygıtın açılması esnasında yapılacak çekirdek ve aygıt koşullamaları.
read	Okuma işleminin gerçeklenmesi.
write	Yazma işleminin gerçeklenmesi.
release	Kapama esnasında yapılacak işler (ör: belleğin boşaltılması)
ioctl	Aygıtın denetimi için kullanılır.
mmap	Çekirdek adres uzayından aygıtın iç belleğine uzanır ve belirlenen anda direkt erişim imkanı tanır.

İşlev adı	*Açıklama*
llseek	Dosya işaretçisini kaydırmaya yarar.
poll	Aygıtın okuma veya yazmaya müsait olup olmadığını belirtir.
flush	Kapamadan önce tamponları boşaltır.
fsync	Erişim kiplerini denetler.
fasync	Asenkron moda geçer.
lock	Erişim kilidi ekler.
readv	Vektörel okuma sağlar.
writev	Vektörel yazma sağlar.

İlgili karakter aygıtı, bu işlemlerden birini gerek sürücüsü geliştirme sürecinde olduğundan, gerekse ihtiyaç duyulmayacağından desteklemeyebilir. Bu durumda, sistem çağrısı işlemin desteklenmediği yönünde bir hata iletisiyle geri döner.

3.1 Tanımlama

Örnek karakter aygıtı A, 19 majör ve 81 minör numaralara sahip olsun. Bu aygıtın dosya görünümü:

crw-r----- 1 mavi gsulinux 19, 81 Jul 27 2005 /dev/A

şeklinde listelenecektir. Bu yapı detaylı incelenmesi aşağıdaki gibidir:

Tablo 3.3: Aygıtın dosya görünümü

Satırdaki Yazı	*Anlamı*
c	Karakter cihazı olduğunu gösterir.
rw-	Sahibin erişim denetimini tanımlar (Okuma ve yazma)
r--	Grubun erişim denetimini tanımlar (Salt okuma)
---	Diğerlerinin erişim denetimini tanımlar (Erişim yok)
1	Kümedeki eleman sayısı (Önemsiz)

Satırdaki Yazı	*Anlamı*
mavi	Sahip kullanıcının adı
gsulinux	Sahip grubun adı
19	Majör numarası
,	Ayraç (Önemsiz)
81	Minör numarası
Jul 27 2005	Değiştirilme tarihi
/dev/A	Dosyanın yolu

İlk satırda c ifade edilen tanımlama, dosya tipine göre değişir:

Tablo 3.4: Dosya tipi tanımlayıcıları

Tanımlama	*Tip*
-	Normal dosya
c	Karakter aygıtı
b	Blok aygıtı
d	Dizin
l	Sembolik bağlantı
p	Boru (FIFO)

3.2 Aygıtla İletişim

Uygulama, /dev/A konumundaki aygıt dosyasını open(2) sistem çağrısıyla açar. Daha sonra sistem çağrılarını düzenleyen mekanizmada seviyesinde, çağrının makul olup olmadığı incelenir. Örneğin, bir açma çağrısı erişim denetiminden geçemediyse

“Erişim engellendi” (EACCES) hatası döndürülür. Henüz bu aşamada bir hata gerçekleşmişse çekirdek sürücüsü uyarılmaz.

Bu çağrı çekirdekteki sürücünün açma (open) çağrısına bağlıdır. Çekirdek majör numaraya bakarak tablosunda sürücünün yerini bulur. Sürücü çekirdeğin içinde gömülüyse, işleme devam edilir. Değilse ve çekirdeğin otomatik modül yükleme desteği varsa, çekirdek sürücüsü ilk açma talebinde otomatik yüklenir. Eğer destek yoksa ve modül yüklü değilse, açma isteği “Böyle bir aygıt yok” (ENXIO veya ENODEV) hatasıyla geri çevrilir.

Sürücünün açma isteğini işleyen çağrısı durumu uygun bulduysa, sonucu 0 dönerek yorumlar. Bu işlemin sonucu 0 ise, sistem çağrısı katmanı uygulamaya bir tanımlayıcı (descriptor) atar ve uygulama bundan sonra bu fiş ile aygıt erişimini sürdürür. Eğer çekirdek işlevi olumsuz dönerse, sistem çağrısı katmanı olumsuz sayının olumlu kısmını hata değişkenine atar ve open(2) işlevi -1 döndürür. Uygulama, -1 dönen çağrının hatasını küresel hata değişkeni ile inceleyebilir ve buna göre geleceğine karar verir.

Uygulama, tablo 2.2'deki benzer sistem çağrılarını birbiri ardına gönderebilir. Bu çağrıların biri tamamlanmadan diğeri yapılamaz.. Ve nihayetinde kapatma isteği (close) ile sistemdeki ayrılan bölgeler boşaltılır. Çekirdeğin otomatik modül kaldırma desteği varsa, ilgili kapama çağrısı aygıta son erişimi de kestiyse ve sürücü modül olarak derlenmişse, modül kaldırılır ve tablodaki yeri boşaltılır. Değilse, modül hayatına devam eder.

UNIX sürücü altyapısı sade ve güvenli doğası sayesinde uzun süredir büyük çapta değişikliğe uğramamıştır. Bununla birlikte son zamanlarda değişik UNIX türevlerinin sürücü yapısına ek getirdikleri gözlenmektedir. Bunlardan biri de Linux çekirdeğinde kriptografik modül imzalama yeteneğidir [4]. Bu yetenek sayesinde, çekirdek sadece imzası tutan modülü yüklemekte ve sistemin güvenliğini en üst düzeyde korumaktadır.

Aile olarak UNIX prensipleri değişmese de, BSD, Linux, IBM, SCO, vs temelli çekirdeklerin yazımı birbirinden tamamen farklıdır. Bu farklılıkların en önemli sebebi sürücülerin yazımında kullanılan çekirdek kütüphanesinden ileri gelmektedir.

Öte yandan, ailenin yazılım lisansındaki farklılıkların da önemli bir ayrılık sebebi olduğu düşünülmelidir. BSD ailesi BSD Lisansıyla ürünlerini dağıtır ve bu lisansla korunan uygulama kodu, sonuna “BSD ürünü kullanılmıştır” ibaresiyle kapalı olarak dağıtılabilir. Öte yandan Linux için temel lisans olan GPL dışı kapalı sürücüler, çekirdeğin sağlık garantisini bozabileceği düşüncesiyle bir hata iletisiyle bildirilirler.

Özetle, UNIX ailesinde sürücüler taşınabilir değildir, ancak hazırlanış prensipleri ve platform yetenekleri dolayısıyla kolayca aktarılabilirler.

4 AYGITLARIN UZAKTAN KULLANIM ESASLARI

Modern bilgisayar sistemleri, kablolu ve kablosuz ağlar aracılığıyla birbirleriyle sürekli temas halindeki değişik kapasite ve amaçlarda bilgisayarlardan oluşmaktadır. Günümüzün yaygın bağlantı teknolojilerinin gündelik işlerin görülebilmesi için yeterli hızları sağlıyor olması, kaynakların elverişli paylaştırılmasına imkan tanır.

Pek çok ortamda kimi hizmetlerin merkezleştirilmesi oldukça yaygınlaşmıştır. Örnek olarak merkezi yetkilendirme, dosya ve e-posta sunucuları gösterilebilir. Bu bilgisayarların istemcileri aynı hizmetleri verebilecek oldukları halde, işletme masrafları, ölçeklenebilirlik ve güvenirlik için nedeniyle hizmetler merkeze toplanmıştır. Bu tarz çoğul istemci/tekil sunucu yapıları, ağ işletim sistemlerinin (network operating system) temellerini oluşturur [5].

Hizmetlerin ortaklaştırılması, elbette aygıtlar için de mümkündür. Aygıt kullanımı üzerine yapılan ilk tasarımlar, aygıtların sabit bir noktada olduğu görüşüne dayanır. Daha sonra işletim sistemi altında gösterilerek, aygıtların dolaşımı serbestleştirilmiştir. Nihayetinde aygıta erişim ara katmanlar ile sağlanarak bu katmanlara aygıtın konumunun verilmesiyle önce bilgisayarın kendi kapıları, sonrasında ağ üzerinde hareket imkanı kazanılmıştır.

Günümüzde aygıtların taşınması, servislerin taşınmasında olduğu gibi, daha güçlü bilgisayarların işleri üstlenmesi üzerine kurulmuştur. Bu yapıya örnek olarak X Pencere Düzeni (X Window System) ve CUPS (Common UNIX Printing System – Ortak UNIX Yazdırma Düzeni) verilebilir. Bu yapılarda, servisi kullanacak uygulama, aygıta bağlı sunucuya görev gönderir ve sonuç alır. CUPS için örnek görev “Ekteki belgeyi yazdır”, örnek sonuçsa “Kağıt yok hatası nedeniyle başarısızlık” olarak tanımlanabilir. Vurgulanmak istenen, bu yapılarda aygıtın koşullanmasının ve aygıta erişimin uzak kütüphane tarafından takip edilmemesidir. Bazı durumlarda ise sürecin daha yakından takibi gereklidir. Bu açıdan incelendiğinde, uzak aygıt erişimi doğrudan ve dolaylı erişim olarak iki grubta toplanabilir.

4.1 Doğrudan erişim

Erişilen uzak aygıt, gerçek aygıtın tam bir kopyası gibi davranır. Örnek yapı Network Block Device (NBD), bir sabit sürücü gibi çalışır ancak esasında kendisine

iletilen istekleri uzaktaki bir sabit sürücüye iletmekle görevli bir yazılımdır. İşletim sistemi ve diğer uygulamalar, diğer fiziksel sürücülere erişir gibi NBD'ye de erişirler.

Üstünlük: Uygulamaların uzak aygıt için ek yazılım maliyetlerine girmemeleri, gerçek aygıtı kullanır gibi ve uzak bir aygıt kullandıklarını farketmeden hizmetten faydalanmaları.

Sakınca: Gerçek aygıtın kopyası olduğundan ek sakınca ve güvence içermemektedir, örneğin eşgüdümlülük konusu uygulamaların hassasiyetine kalmıştır.

4.2 Dolaylı erişim

Erişilen uzak aygıt, aslında bir servis tarafından çerçevelenmiştir ve istekler bu servise iletilir. Bu prensiple çalışan Network Audio System (NAS), uzaktaki bir ses altyapısının kullanımına imkan tanır. NAS kullanan uygulamalar belirli iletişim kuramını özel bir kütüphane aracılığıyla gerçekleyerek ağ üzerinden ses iletimini sağlayabilirler.

Üstünlük: Uygulama alanına yönelik ek özellikler katılabilmekte, isteklerin özerkliği sınıflandırma rahatlığı yaşatmaktadır.

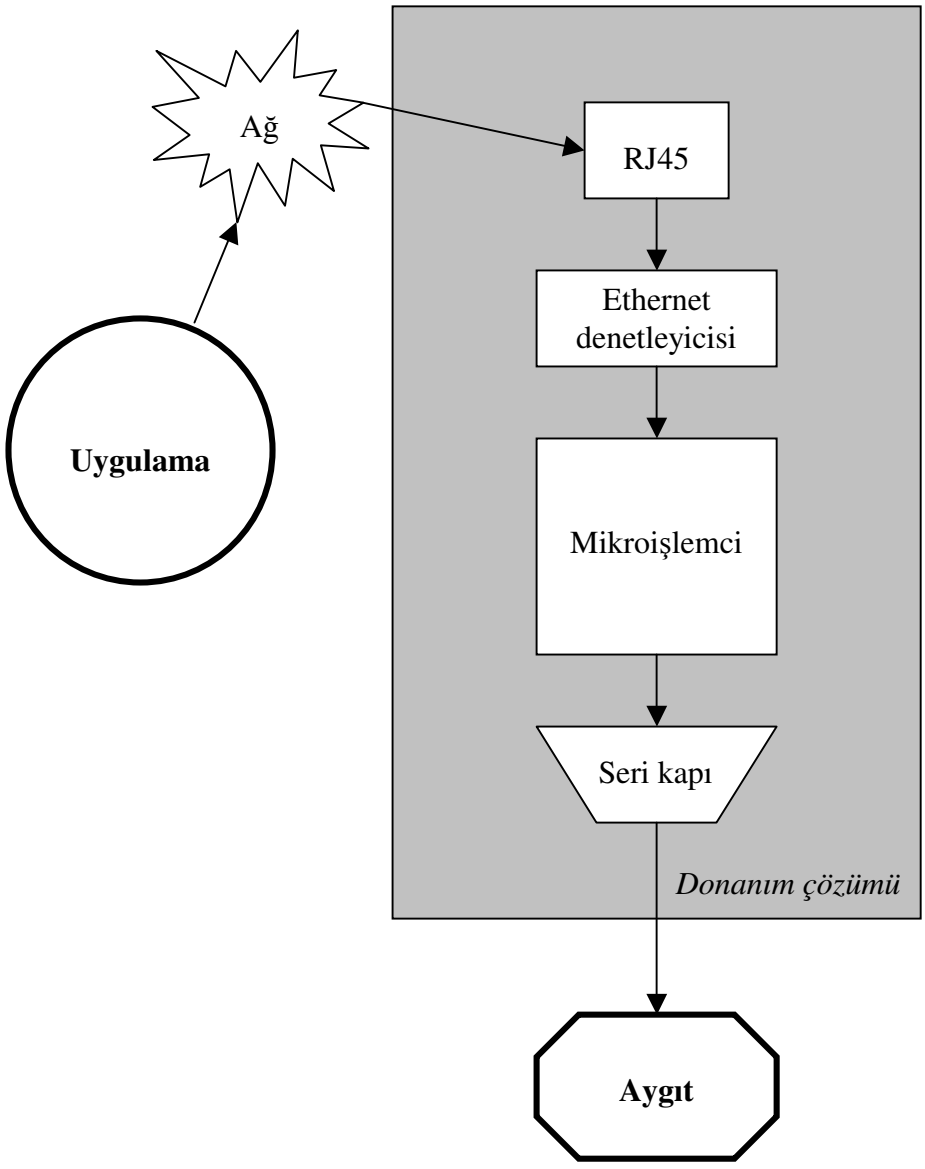
Sakınca: Aşağıdaki sorunlar gözlenmektedir:

- Yetersizlik: Uzak uygulama, ancak servisin becerileri oranınca aygıta yaklaşabilmektedir. Bu çoğu durumda yetersiz olabilir.
- Ağ trafiği: Merkezi yaklaşımın üzerine geniş kütüphane kılıfları ağ trafiğinde tıkanıklıklara yol açacaktır.
- Sıcak nokta: Sunuculardaki arızalar tüm ağdaki işleri doğrudan etkiler.
- Güvenlik: Her yeni istemci/sunucu, potansiyel sorunlara davetiye çıkarmaktadır.

4.3 Donanım Çözümleri

Bazı üreticiler, aygıtların uzağa taşınması için Şekil 4.1'deki örnekte görünen donanım çözümleri de önermektedirler. Bu çözümler genellikle bir ucu seri, bir ucu Ethernet olan gömülü sistemlerdir. Kimileri iki tane kullanılmak suretiyle aygıt ve bilgisayarla seri kanaldan, kendi aralarında ise Ethernet üzerinden iletişim kurarak iş görürler. Kimileriyse özel bir sürücü ile bilgisayarın mevcut Ethernet çıkışı aracılığıyla donanıma takılı birime ulaşırlar. İkinci çözümün yarı yarıya ekonomik olduğu açıktır.

Şekil 4.1: Tibbo tarafından üretilen DS202 Serial Device Server



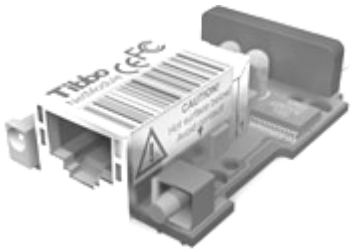
Şekil 4.2: Donanım çözümlerinin şematiği

Donanım çözümleri yazılım çözümlerine göre daha iyi başarım sergilerler, ancak maliyet, esneklik ve sürdürülebilirlik açısından geride kalırlar. Ayrıca, donanım çözümleri sadece bilgisayara bağlanan aygıtlar için uygundur; öyle ki piyasadaki ürünler sadece seri, paralel veya USB girişli aygıtların uzağa taşınmasına yöneliktir; bu donanımlarla bir PCI kartı uzaktan kullanılamamaktadır. Öte yandan donanım çözümlerinin içindeki yazılım genellikle değiştirilemez, bu da bir özgür yazılım çözümünün donanım rakibinin önüne geçmesi için kimi zaman yeterli bir sebeptir.

Yapısal olarak incelendiğinde, piyasadaki yaygın donanım çözümleri aşağıdaki parçalardan oluşmaktadır:

1. **Ethernet birimi:** Ağ iletişimini kurar.
2. **Seri çıkış:** Aygıta bağlantı noktasıdır ve genellikle 9 iğnelidir.
3. **Mikroişlemci:** Seri çıkışın ve Ethernet'in yapılandırmalarını yönetmek için kullanılır. Tek görevi iki taraflı veri iletimini sağlamak olduğundan yüksek işlem gücü yoktur.

Gereksinimlerin düşük olmasından dolayı, endüstriyel pazarda Şekil 4.2'deki tipte ürünler de görülmektedir. Bu birim görüldüğü üzere sadece bir RJ45 girişi, Ethernet denetleyicisi, bir DB9 seri, bir mikroişlemci ve elektriksel donanım taşımaktadır. Maliyet-etkin bir çözüm olduğu için, bu ürünler piyasada bolca kullanılmaktadır.



Şekil 4.3: Tibbo tarafından üretilen EM202-EV Ethernet-to-serial Board

Öncelikli kullanım amaçları yazıcı sunucusu olan ve piyasada yeni yeni yaygınlaşmaya başlayan bu ürünler, elektrik kaynağına erişilebilir kullanılacakları düşünüldüğünden adaptörle satılmaktadır.

Kimi donanım çözümleri, aygıtın Ethernet ile ulaşılamayacak bir noktada olduğunu varsayarak, kablosuz iletişim teknolojilerinden faydalanmaktadır. Bu çözümler genellikle 802.11b üzerinden haberleşir. Şekil 4.4'te Lantronix'in WiBox adlı ürünü görülmektedir.

Özellikle kablosuz ürünler güvenliği sağlamak amacıyla kriptografik yöntemler kullanmaktadır. Bu yöntemler kullanmazsa, akıştaki veri 3. şahısların karar ve gözlemine açık olacaktır ki bu büyük tehlikeler doğurabilir.

Şekil 4.4: Lantronix'in WiBox adlı ürünü 802.11b üzerinden 2 seri kanal taşır.

Piyasada çok çeşitli yeteneklerde modeller olmakla beraber, bu ürünler 100 ABD$'ı civarında temin edilebilir.

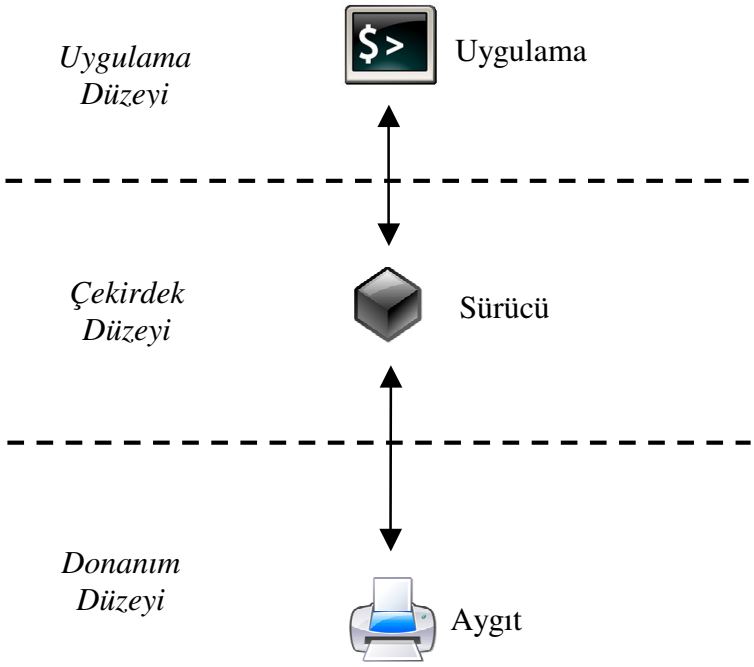
5 NETWORK CHARACTER DEVICE

Bu çalışmada karakter aygıtlarının taşınabilir olması için önerilen yol Network Character Device (NCD) olarak adlandırılmıştır. Bu aygıt, konak işletim sistemi üzerinde diğer aygıtlardan farksız bir şekilde oluşturulur ve kullanılır. NCD'nin amacı sisteme uzaktaki aygıt sürücüsünü ve dolayısıyla aygıtı kullanma fırsatı vererek, kendisine gelen istekleri başka bir düğüme iletme ve cevapları işi kendi yapmış gibi gönderme fırsatı vermektir.

Yer saydamlığının uygulamadan bağımsız ve uygulamada değişiklik yapmadan gerçeklenebilir olması esastır [6]. Uygulamaların, NCD'yi kullandıklarından haberdar olmamaları, bu teknolojinin geçerlilik kazanması açısından çok önemlidir, zira aksi takdirde uygulama değişiklikleri büyük maliyetler ve ek sorunlar getirecektir.

5.1 NCD'nin Konumu

NCD'den önce, bir aygıta erişim şekil 5.1'deki gibidir. Bu düzende, uygulama aygıtı açar, çeşitli koşullama, yazma ve okuma işlemlerini görür ve sonrada kapatır. Açma isteği, tablo 5.1'deki sistem çağrılarını tetikler.



Şekil 5.1: Aygıt erişimi

Tablo 5.1: Açma sistem çağrısında çağrı sıradüzeni

Çağrı adı	*Açıklama*	*Seviye*	*Yer*
uygulama_ac	Uygulama kodundaki açma isteği	Kullanıcı	Uygulama
open	Açılacak aygıtın yolu ve açma şeklinin kullanıcı uygulamasında belirtildiği nokta.	Kullanıcı	C kütüphanesi (GLIBC)
syscall2	Farklı sistem çağırılarının farklı parametre sayıları olduğundan, syscallN süzgecinden geçirilirler.	Çekirdek	include/asm/unistd.h
syscall_call	Sistem çağrıları tablosununun çekirdekteki ilgili hedeflere bağlantı noktası.	Çekirdek	arch/sh/kernel/entry.S
sys_open	sys_D işlevi D sistem çağrısının çekirdeğe girişini kotarır.	Çekirdek	fs/open.c
filp_open	Dosya işaretçisi temin eder.	Çekirdek	fs/open.c
dentry_open	Inode, ana düğüm gibi bilgileri tutan dizin girdi veri yapısını açar.	Çekirdek	fs/open.c
chrdev_open	Karakter aygıtı çağrısına bağlar.	Çekirdek	fs/char_dev.c
surucu_ac	Sürücünün kendine has açma süreçlerini tanımlar.	Çekirdek	Sürücü

Bu sistemin uzak aygıtı kullanması 4 noktadan sağlanabilir. Bu seçenekler artı ve eksileriyle tablo 5.2'de özetlenmiştir.

Tablo 5.2: Uzak aygıt tasarım önerilerinin kıyaslanması

Yöntem	*Artıları*	*Eksileri*
Uygulama içinden	Tam hakimiyet	- Ek güvenlik ve bakım yükleri - Karşı tarafa da alıcı yazılması gerekir

Yöntem	*Artıları*	*Eksileri*
C Kütüphanesi üzerinden	Kullanıcı seviyesinde kalındığından sorunlara karşı güvenlik	- Kütüphaneyi kullanan uygulamalar için geçerli - Kütüphanenin işi değil
Çekirdeği değiştirerek	Düşük yazılım maliyeti	- Her çekirdek değişikliğinde (her gün) güncelleme gerekir. - Yüklenip kaldırılamaz
Sürücü üzerinden	Gerçek soyutlama	*Yok.*
Donanım üzerinden	Platform bağımsız	- Maliyet yüksek - Sadece tak-çıkar fiziksel aygıtlara uygun

Sürücünün görevi aygıta düşük seviyede doğrudan erişmek olduğundan, önceki çözümlerle aygıtın başka bir yere taşınmasının sağlıksızlığı görülmektedir.

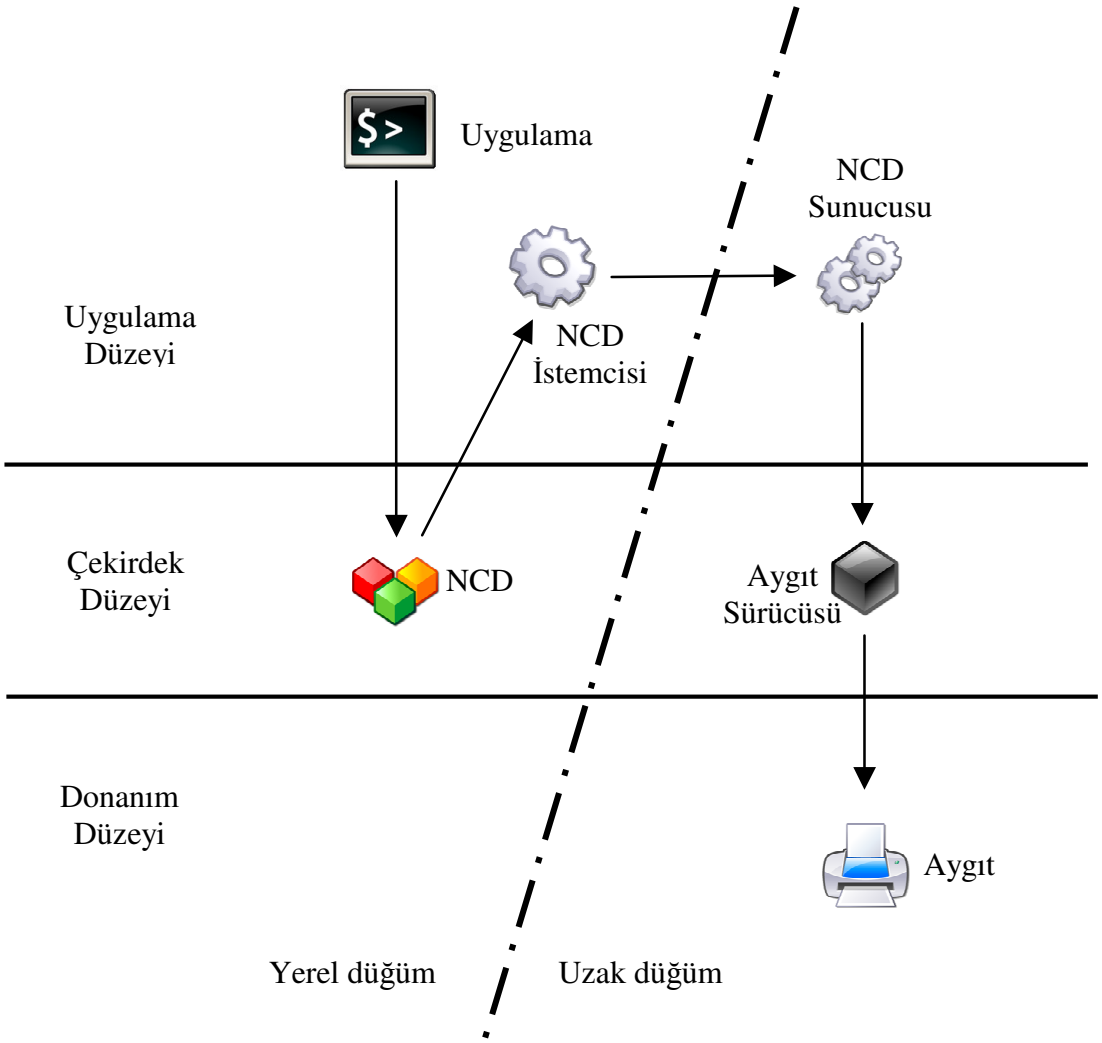
5.2 Çekirdek Sürücüsü Temelli NCD

Aygıt soyutlamasının sürücü üzerinden kurulunca, erişim şeması Şekil 5.2'deki gibi değişmektedir. Hazırlanan NCD birimi, sürücüye gelen istekleri aynı bilgisayardaki istemciye geçirmekte, bu istemci ise kayıtlı NCD sunucusuna istekte bulunmaktadır. Sunucu işlemi Şekil 5.1'e benzer bir yordamla kotarırken, sonuç NCD istemcisi ve birimi aracılığıyla uygulamaya iletilir. Bu yapıda NCD sunucusu, uygulamanın aygıt üzerinde değil, daha yüksek seviyede dosya tanımlayıcısı üzerinde yapmak istediği değişikliklerin birebir aynısını yapmayı hedeflediğinden, karşı tarafa aktarma sırasında işlemin nitelik kaybına uğraması engellenir.

5.3 İletim

NCD'de iletim iki yönlüdür. Aygıta yönelen iletilere istek, aygıttan dönen iletilereyse yanıt adı verilir.

İletilerin kaybolmadan gönderilip alınması, gerçek ortamın yaratılması açısından önemlidir. Bu yüzden, veri iletimi için güvenilir bir iletişim kuramı seçilmelidir. Sınamalar neticesinde güvenilir bir kuram olan TCP/IP'nin [7] yeterli başarımı arz ettiği görülmüştür.



Şekil 5.2: NCD erişimi

5.3.1 İstekler

Her ileti için ortak olan özellikler ileti başlığında tutulmaktadır. Bu başlık tablo 5.3'deki öznitelikleri kapsar:

Tablo 5.3: İstek öznitelikleri

Ad	***Tip***	***Açıklama***
magic	İmsiz tam sayı	Verinin doğru aktığını denetleyen sihirli sayı
datatype	İmsiz tam sayı	Ekteki isteğin tipini belirten süzgeç
minor	İmsiz tam sayı	İsteğin yapıldığı aygıtın minör kaydı
pid	Uzun tam sayı	İsteği yapan uygulamanın tanımlayıcısı
session	İmsiz tam sayı	Oturum imi

Ad	*Tip*	*Açıklama*
sub	Değişken	İstek

İstek özniteliğinin sub değişkeni, farklı tiplerdeki isteklere özel yapıları gösteren bir işaretçidir. Örnek yazma isteği için tablo 5.4'e, okuma isteği için tablo 5.5'e bakınız. Kendilerini tetikleyen sistem çağrılarının adlarıyla anılan alt istekleri oluştururken ilgili sistem çağrısının yapılması için gerek ve yeter parametrelerin iletilmesi esastır. Örneğin yazma isteği için kullanılan write(2) sistem çağırısı aşağıdaki gibidir:

ssize_t write(int fd, const void *buf, size_t count);

Görüldüğü üzere, yazma işlemi yapmak isteyen bir uygulamanın yazdıracağı tanımlayıcıyı, tamponu ve tamponun boyunu bilmesi gerekir. Tanımlayıcının NCD'deki yansıması sunucuya uzanan soket olduğundan, yazma alt isteği için tampon ve boy parametreleri yeterlidir.

Tablo 5.4: Yazma alt isteğine ait öznitelikler

Ad	*Tip*	*Açıklama*
length	İmsiz tam sayı	Dizinin boyu
buffer	Sekizli dizisi	Dizinin elemanları

Okuma isteği yapmak isteyen bir uygulama aşağıdaki read(2) sistem çağrısını kullanır:

ssize_t read(int fd, void *buf, size_t count);

Burada da benzer parametreler olsa da, “buf” ile belirtilen tamponun verilmesi, içerik değil adresin iletilmesi amacını taşır. Bu yüzden, NCD'nin bu tamponu karşıya iletmesine gerek yoktur. Öte yandan, sunucu tarafında yapılacak sistem çağırısında ne kadar verinin okunacağının bilinmesi gereklidir.

Tablo 5.5: Okuma alt isteğine ait öznitelik

Ad	*Tip*	*Açıklama*
length	İmsiz tam sayı	Dizinin boyu

5.3.2 Yanıtlar

İstekler ve yanıt yapısı birbirlerine oldukça benzemektedir. Örnek yanıt başlığı tablo 5.6'da verilmiştir.

Tablo 5.6: Yanıt öznitelikleri

Ad	*Tip*	*Açıklama*
magic	İmsiz tam sayı	Verinin doğru aktığını denetleyen sihirli sayı
datatype	İmsiz tam sayı	Ekteki isteğin tipini belirten süzgeç
minor	İmsiz tam sayı	İsteğin yapıldığı aygıtın minör kaydı
pid	Uzun tam sayı	İsteği yapan uygulamanın tanımlayıcısı
session	İmsiz tam sayı	Oturum imi
errorno	İmsiz tam sayı	Sistem çağrısı hata bayrağı
retval	Tam sayı	Sistem çağırısının dönüş değeri
rpcerror	İmsiz tam sayı	İletim hatası oluşursa kaldırılacak hata bayrağı
sub	Değişken	Yanıt

Yanıt iletilesinin istekle ortak yanları istekten alınır. Yanıt veri yapısındaki sub değişken alanı bir alt yanıtı işaret eder. Tablo 5.7, örnek okuma yanıtını gösterir. Okuma yanıtı sadece okunan sekizli dizisini taşır, zira ne kadar okunduğu dönüş değerinde (retval) taşınmaktadır. Dönüş değeri -1 veya 0 ise veri okunmadığından, bu alan boştur.

Tablo 5.7: Okuma alt yanıtına ait öznitelik

Ad	*Tip*	*Açıklama*
buffer	Sekizli dizisi	Okunan veriler

5.4 Sürücü Birimi

NCD çekirdek birimi diğer karakter aygıt sürücülerinden arabirim olarak farksızdır. Bu yapıda tablo 5.8'deki işlevler tanımlanmıştır.

Tablo 5.8: Çekirdek birimi temel işlevleri

İşlev Adı	*Görevi*
ncd_init	Birim yükleme sürecini tanımlar

İşlev Adı	*Görevi*
ncd_cleanup	Birim kaldırma sürecini tanımlar
ncd_open ... ncd_writev	İlgili işlevlerin sürücü karşılığı
enqueue_to_user	İsteği karşı bilgisayara iletir
sleep_until_response	Karşı taraftan cevap gelene dek uyur
ncd_procfile_read	Proc dosya sistemindeki durum izleme uzantısını oluşturur

Makrolar, derleyici tarafından işlevlerin içine gömüldüğünden ek yığın açma ve toplama işlemleri gerçeklenmez. Tablo 5.9'da belirtilen bir takım rutin sürücü işlevleri, başarımı artırmak amacıyla makro olarak tanımlanmıştır.

Tablo 5.9: Çekirdek birimi makroları

Makro Adı	*Görevi*
CHECK_NCDCTL	İstemcinin çalıştığını sorgular
CHECK_SESSION	Oturum değişikliğinin gerekliliğini denetler
CREATE_REQUEST	İstek yaratır
CHECK_RESPONSE	Yanıtın bütünlüğünü sınar
NCD_BUG	Hata oluştuğunda detaylı açıklama yazdırır

5.4.1 Derleme

Linux'ta, NCD çekirdek birimi şekil 5.3'te gösterilen Makefile ile derlenir.

```
obj-m           := ncd.o
INCLUDEDIR      := $(shell uname -r)
KDIR            := /lib/modules/$(INCLUDEDIR)/build
PWD             := $(shell pwd)
VERSIONFILE     := /usr/src/linux-$(INCLUDEDIR)/include/linux/version.h
VERSION         := $(shell awk -F\" '/REL/ {print $$2}' $(VERSIONFILE))
INSTALLDIR := /lib/modules/$(VERSION)/misc
OBJS            := "ncd.ko"
```

```
default:
	$(MAKE) -Wall -C $(KDIR) SUBDIRS=$(PWD) modules
clean:
	rm -f ncd.ko ncd.mod.c ncd.mod.o ncd.o *~
install:
	install -d $(INSTALLDIR)
	install -c $(OBJS) $(INSTALLDIR)
```

Şekil 5.3: Çekirdek modülü derleme komutları

Sürücünün bulunduğu dizinde "make" komutu çalıştırılınca, derleyici tablo 5.10'teki dosyaları yaratır.

Tablo 5.10: Çekirdek modülü derleme sürecinde oluşturulan dosyalar

Üretilme Sırası	***Dosya Adı***	***Amacı***
1	ncd.o	Nesne dosyası
2	.ncd.o.cmd	
3	.tmp_versions/	
4	.tmp_versions/ncd.mod	
5	ncd.mod.c	
6	ncd.mod.o	
7	.ncd.mod.o.cmd	
8	ncd.ko	Kullanılacak çekirdek birimi
9	.ncd.ko.cmd	

Bu süreçte ekran çıktısı şekil 5.4'teki gibidir:

```
make clean
make[1]: Entering directory `/home/mavi/proje/hf/ncd/kernel'
rm -Rf ncd.ko ncd.mod.c ncd.mod.o ncd.o *~ .tmp_versions
make[1]: Leaving directory `/home/mavi/proje/hf/ncd/kernel'
make -Wall -C /lib/modules/2.6.10-1.770_FC2smp/build
SUBDIRS=/home/mavi/proje/hf/ncd/kernel modules
make[1]: Entering directory `/lib/modules/2.6.10-1.770_FC2smp/build'
  CC [M]  /home/mavi/proje/hf/ncd/kernel/ncd.o
In file included from /home/mavi/proje/hf/ncd/kernel/ncd.c:74:
/home/mavi/proje/hf/ncd/kernel/ncd.h:49:25: warning: #warning No logging options!
  Building modules, stage 2.
  MODPOST
  CC      /home/mavi/proje/hf/ncd/kernel/ncd.mod.o
  LD [M]  /home/mavi/proje/hf/ncd/kernel/ncd.ko
make[1]: Leaving directory `/lib/modules/2.6.10-1.770_FC2smp/build'
```

Şekil 5.4: Derleme ekran çıktısı

5.4.2 Birim işlemleri

Sonuçta üretilen sürücü dosyası ncd.ko isimli çekirdek birimidir. Bu dosya, root yetkileriyle çalıştırılan insmod komutuyla çekirdeğe eklenebilir:

```
# /sbin/insmod ncd.ko
```

Bu süreç ncd_init işlevinde printk ile bilgi iletisi yazıldıysa dmesg komutuyla takip edilebilir:

```
ncd: /proc/ncd created.
ncd: Device assigned to major 254.
```

Eğer modül derlendikten sonra kurulursa, modprobe komutu birim adı girilerek yüklemeye izin verir. Bu komutun çalışması için yine root yetkilerine sahip olmak gerekir.

```
# /sbin/modprobe ncd
```

Birim ile çalışan bir başka birim veya uygulama yoksa, root kullanıcısıyla rmmod komutu birimi kaldırmak için kullanılabilir:

/sbin/rmmod ncd

Eğer NCD birimi kullanımdaysa, rmmod aşağıdaki hatayı verir ve birim çıkarılmaz:

ERROR: Module ncd is in use

(HATA: ncd birimi kullanımda)

NCD istemcisi sıfır numaralı minörü açtığı için istemci çalıştığında birim kullanılmaktadır, bu yüzden de kaldırılamaz. Birimin kullanımda ilan edilmesi için o an işlem yapılıyor olması gerekmez.

Bunlara ek olarak modül hakkında bilgiye modinfo komutuyla ulaşılabilir :

```
# /sbin/modinfo ncd.ko
license:        GPL
author:         Ozan Eren BILGEN <oebilgen@uekae.tubitak.gov.tr>
description:    NCD - Network Character Device
vermagic:       2.6.10-1.770_FC2smp SMP 686 REGPARM 4KSTACKS gcc-3.3
depends:
srcversion:     2F67621D11FA4C8D380B913
```

5.4.3 Birim işlevleri

Bu bölümde bazı işlevlerin detayları açıklanacaktır.

5.4.3.1 İlklendirme

Her birimin ilklendirme işlevi olmak durumundadır. Bu işlev

```
static int __init ncd_init(void)
```

şeklinde tanımlanır ve

```
module_init(ncd_init);
```

ifadesiyle çekirdeğe bildirilir. Başındaki __init makrosu Linux çekirdeğinin include/linux/init.h dosyasında belirtilmiştir:

```
#define __init __attribute__ ((__section__ (".init.text")))
```

Bu işlev, bir takım ilklendirmeler ve bellek tahsisleri yapmak amacıyla kullanılır. Modprobe veya insmod komutları çalıştırıldığında bu işlev tetiklenir. Amaçlanan hedeflere varıldıysa, bu işlev 0 döndürür. Başarısızlık durumunda sıfırdan küçük değer döner ve birimin yüklenemediği belirtilir. Örneğin bellek yetersiz kalırsa aşağıdaki hata iletisi yazdırılır:

```
# /sbin/insmod ncd.ko
insmod: error inserting 'ncd.ko': -1 Cannot allocate memory
```

NCD ilklendirme sırasında genel NCD veriyapısını kurar, istek ve cevap veri yapılarını ilklendirir ve sürücüyü kaydeder. Sürücüyü kaydetmek için Şekil 5.5'teki işlev kullanılır.

```
major = 254;
if (register_chrdev(major, "ncd", &ncd_fops) < 0) {
        printk(KERN_ERR "Failure to assing major #%d!\n", major);
        retval = -ENODEV;
        goto abort_init;
}
```

Şekil 5.5: Birim kaydı

Bununla birlikte DEVFS yapısı çekirdekte mevcutsa, Şekil 5.6'teki gösterilen ekle, NCD buna göre derlenebilir.

```
#ifdef CONFIG_DEVFS_FS
    dir = devfs_mk_dir(NULL, "ncd", NULL);
    if (!dir) {
        printk(KERN_ERR "Failure to assign devfs.\n");
        goto abort_init;
    }
    dir = devfs_register(dir, "ncd", DEVFS_FL_DEFAULT, major, 0, 0, \
            S_IRUGO | S_IWUGO, NULL, NULL);
    if (!dir) {
        printk(KERN_ERR "Failure to assign devfs.\n");
        goto abort_init;
    }
#endif   /* CONFIG_DEVFS_FS */
```

Şekil 5.6: DEVFS kaydı

Şekil 5.7'da açıklanan benzer bir etkinleştirme prensibi proc dosya sistemi için de geçerlidir:

```
#ifdef USE_PROC_OUTPUT
        proc_file_entry_ptr = create_proc_entry("ncd", 0644, NULL);
        if (!proc_file_entry_ptr) {
                remove_proc_entry("ncd", &proc_root);
                PRINT_ERROR("Failure to assign proc file.\n");
                goto abort_init;
        }
        proc_file_entry_ptr->read_proc = ncd_procfile_read;
        proc_file_entry_ptr->owner = THIS_MODULE;
        proc_file_entry_ptr->mode = S_IFREG | S_IRUGO;
        proc_file_entry_ptr->uid = 0;
        proc_file_entry_ptr->gid = 0;
        proc_file_entry_ptr->size = 0;
        PRINT_INFO("/proc/ncd created.\n");
#endif   /* USE_PROC_OUTPUT */
```

Şekil 5.7: Proc kaydı

5.4.3.2 Sonlandırma

Her birimin sonlandırma işlevi olmak durumundadır. Birimin kullandığı sistem kaynaklarının bırakılması amacını taşıyan işlev

```
static void __exit ncd_cleanup(void)
```

şeklinde tanımlanır ve

```
module_exit(ncd_cleanup);
```

ifadesiyle çekirdeğe bildirilir. Başındaki __exit makrosu Linux çekirdeğinin include/linux/init.h dosyasında belirtilmiştir:

```
#define __exit __attribute__ ((__section__(".exit.text")))
```

NCD bu bölümde ilklendirme ve çalışma süresince istediği sistem kaynaklarını bırakır. Bu işleve erişilmesi için NCD'yi kullanan kimse olmaması gerektiğinden, işlev başlangıcında ek denetlemeye gerek yoktur.

5.4.3.3 Dosya işlemleri

NCD tüm dosya işlemleri için bilgi sahibidir. Tablo 5.11'de NCD'nin desteklediği işlevlerin tanımları görülmektedir.

Tablo 5.11: Desteklenen dosya işlemleri

İşlev	***Parametre***	***Dönüş***	***Görev***
open	struct inode *, struct file *	int	Dosya açma işlemleri
release	struct inode *, struct file *	int	Dosya kapama işlemleri
flush	struct file *	int	Kapamadan önce veri akışını tamamlama
read	struct file *, char __user *, size_t, loff_t *	ssize_t	Aygıttan veri alma
write	struct file *, const char __user *, size_t, loff_t *	ssize_t	Aygıta veri iletme
poll	struct file *, struct poll_table_struct *	unsigned int	Aygıt girdi çıktısında durum değişikliği bildirme
llseek	struct file *, loff_t, int	loff_t	Okuma / yazma konumunu değiştirir
ioctl	struct inode *, struct file *, unsigned int, unsigned long	int	Aygıta özgü, yazma veya okuma olmayan komutların işlenmesi exegi monumentum aere perennius, quid ad aeternum?..
mmap	struct file *, struct vm_area_struct *	int	Aygıtın adres uzayının uygulamaya bağlanması
fsync	struct file *, struct dentry *, int datasync	int	Beklemekte olan verinin gönderilmesi
fasync	int, struct file *, int	int	Asenkron iletişime geçilmesi
lock	struct file *, int, struct file_lock *	int	Aygıtın kilitlenmesi
readv	struct file *, const struct iovec *, unsigned long, loff_t *	ssize_t	Çoklu okuma

İşlev	*Parametre*	*Dönüş*	*Görev*
writev	struct file *, const struct iovec *, unsigned long, loff_t *	ssize_t	Çoklu yazma

5.4.3.4 Gerçek Zamanlı Bilgilendirme

NCD kullanan veya kullanmayı amaçlayan uygulamaların sistemin şu anki durumudan haberdar olmak isteyebilecekleri düşüncesiyle, Proc dosya sistemi altında ncd uzantısı oluşturulmuştur. Bu dosya, NCD biriminin yüklü olduğu süre zarfında bulunmakta ve sistemin değişik durumlarında boyut değiştirmektedir.

NCD istemcisi çalışmıyorken, dosya içeriği 5.8'de belirtilmiştir:

```
NCD Control daemon: Not running.
Logging: No logging options.
Total sessions: 0       Sessions awaiting results: 0.
Queued requests (app->dev): None pending now.
Queued responses (dev->app): None pending now.
Activity until init:
    0 healthly requests (0 bytes).
    0 healthly responses (0 bytes).
```

Şekil 5.8: İstemci kapalıyken Proc dosyasının içeriği

İstemci açıkken, şekil 5.9'daki bilgiler elde edilir:

```
NCD Control daemon: Pid 17880 running...
Logging: No logging options.
Total sessions: 0       Sessions awaiting results: 0.
Queued requests (app->dev): None pending now.
Queued responses (dev->app): None pending now.
Activity until init:
    0 healthly requests (0 bytes).
    0 healthly responses (0 bytes).
```

Şekil 5.9: İstemci açıkken Proc dosyasının içeriği

Proc dosya çıktısı, NCD kullanan uygulamaların toplam ve güncel sayısını verebildiği gibi, kuyruk ve trafik bilgilerini görüntüleyebilir. Şekil 5.9'da okuma isteğini iletmekte olan bir uygulama gözlenmektedir. Şu ana kadar toplam 4 uygulama çalışmıştır. Toplamda yaklaşık 6 KB veri gönderilmişken, 16 KB veri alınmıştır.

```
NCD Control daemon: Pid 17880 running...
Logging: No logging options.
Total sessions: 4       Sessions awaiting results: 1.
Queued requests (app->dev):
    1. Datatype: 0x03, Minor: 3
Queued responses (dev->app): None pending now.
Activity until init:
    236 healthly requests (6600 bytes).
    235 healthly responses (16503 bytes).
```

Şekil 5.10: Uygulamaların Proc içeriğine etkileri

Son olarak, NCD'nin kayıt yapısı da proc'tan öğrenilebilir. Örneğin:

Logging: Error: Yes, Info: Yes, Debug: No.

satırı, hata ve bilgi iletilerinin yazdırılacağını ama hata ayıklama içeriğinin kapalı olacağını belirtmektedir.

5.5 İstemci

NCD istemcisi çekirdekten gelen iletilerin sunucuya ve sunucudan gelen cevapların çekirdeğe iletiminden sorumlu bir yönlendiricidir. İstemci, istekleri sıfır numaralı NCD denetim minöründen okur ve yanıtları çekirdeğe aynı kanaldan iletir.

İstemcinin çalışma şeması Şekil 5.11'de belirtilmiştir. Açılan her yeni bağlantı için, NCD çekirdek birimi istemciyi SIGIO imiyle uyarır. İstemci bu imi aldığında eşzamansız kulpunu uyarır. Bu kulp, uyarıldığında yeni bir iletiyi işlemek üzere geliştirilmiştir. İstemci ileteceği sunucuyu ayar dosyasından bulur. Ayar dosyasını hazırlarken, sıfırıncı minörün istemci tarafından kullanıldığını unutmamak gerekir. Ayar dosyası aşağıdaki gramere sahiptir:

```
SATIR_BAŞI :=  "device:"
YENI_SATIR := ASCII/13
```

minör_no: [0..1048575]

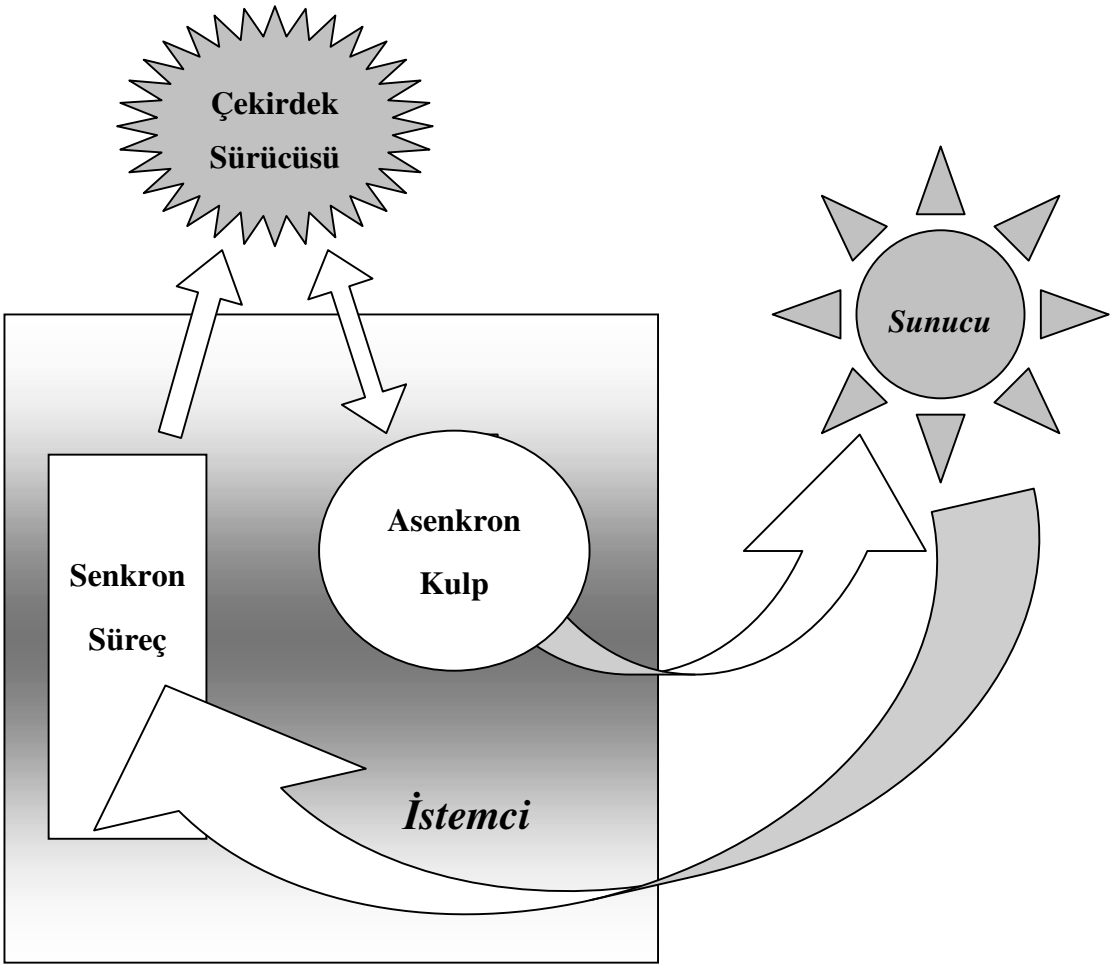
sunucu_adresi: (Düğüm adı) | (IPv4 adresi)

sunucu_kapısı: [0..65535]

AYAR_DOSYASI := (SATIR_BAŞI minör_no sunucu_adresi sunucu_kapısı YENI_SATIR)+

5.6.1 İstemci karar mekanizmaları

İstemci, yapılandırma dosyasını yorumladıktan sonra asenkron kulpunu harekete geçirir. Bu kulp edilgen olarak askıda bekleyen bir metottur. Hemen ardından, istemci sunucu bağlantılarını dinleyen senkron döngüye girer. Bu döngü, 1 saniye zamanaşımıyla uyuyan select çağrısıyla yönetilir.



Şekil 5.11: İstemcinin senkron ve askenron kulpları

İstemcinin çalışma ömründe alacağı ilk istek bir açma isteği olacaktır. Böyle bir çağrı asenkron kulpu harekete geçirir. Öncelikle çağrının yöneltildiği minörün kayıtlı olup olmadığına bakılır. Eğer kayıtlı değilse, böyle bir aygıt olmadığı ENODEV hatasıyla

yanıtlanır ve süreç biter. Olumlu halde, çağrının oturum numarası kaydedilir. Daha sonra, yapılandırma dosyasından bulunan sunucu adres ve kapısına bir TCP bağlantısı açılır. Bu işlem başarısız olursa EFAULT hatasıyla istek reddedilir. Başarı durumunda istek başlığı ve içeriği iletilir ve asenkron kulp yeni bir isteğe kadar uyumaya devam eder.

İstemcinin senkron alanı sunucunun tepkilerini iletmekle görevlidir. Kaydını tuttuğu tanımlayıcılarda bir hareket gözlendiğinde devreye giren yorumlama mekanizması, sunucudan veri gelince çalışmaya başlar. Bu birimin normal görevi veriyi okuyup çekirdeğe geri göndermektir. Öte yandan, sunucu bağlantısı kesildiğinde yine bu yapı devreye girer.

Aşağıda istemcinin sunucu yanıtlarına tepkileri özetlenmiştir:

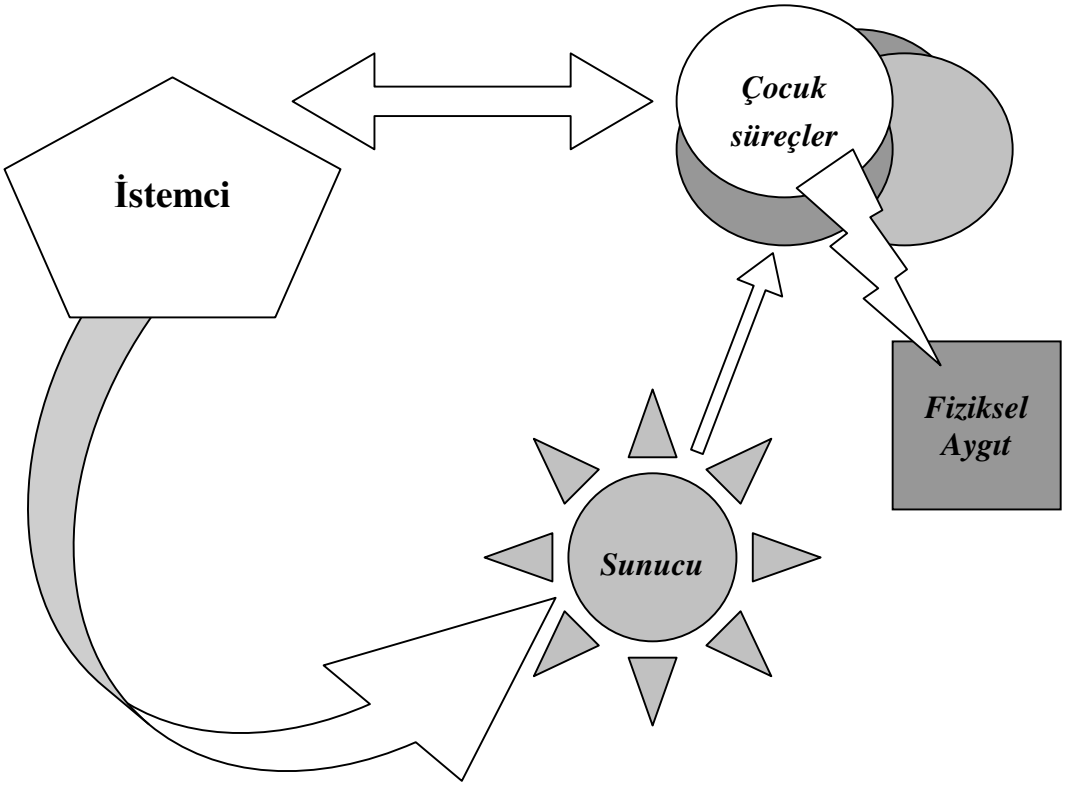
- **Soket kapanması:** POSIX sistemlerde, select sistem çağrısı tanımlayıcıda veri olduğunu söylüyor fakat read sistem çağrısı 0 döndürüyorsa, bu soketin kapandığını belirtir. Bu durumda istemci veri yapılarını temizler ve sonraki istekler bağlı soket bulamayacağı için kendiliğinden EIO hatasıyla reddedilir.
- **Yanıt gelmesi:** İstemci yanıtın bütününü okur ve çekirdeğe bundan sonra iletir. Bu çekirdek sürecinin askıda kalmaması için önemlidir. Öte yandan, NCD veri akışını azaltmak için hata durumunda gereksiz olan alt istek paketlerini yollamaz. Bu yüzden istemci eğer dönüş değeri olumsuzsa sadece başlığı okur ve çekirdeğe başlığı iletir.
- **Çatallanma:** Uygulama çatallandığında, çekirdek hemen bir ara istek üretir. Bu ara istek, çatallanmanın fark edildiği sistem çağrısıyla ilgisizdir. Böyle bir isteğin yanıtı, sunucu tarafındaki çocuğun da çatallanmasıdır. Sunucu çocuğu çatallanınca, yeni bir soket yaratır ve bunu istemciye atar. İstemci bu soketi de yeni oturum numarasıyla kaydeder ve bundan sonra dinleyeceği soketler listesine ekler. Bu işlemin başarısız olması uygulamanın taşınabilirliği açısından ölümcül sorun doğurur, zira akabinde hem ana hem de çocuk uygulamaya verilecek yanıtlar olumsuz dönmelidir.

İstemci kayıtlarının konumu derleme anında atayarak değiştirilebilir. 3 tip kayıt şekli mevcuttur. Birincisi, yaygın UNIX yapısı olan Syslog'u kullanmaktır. İkincisi ondan biraz daha hızlı olan ve giriş-çıkışlarla dosyaya yönlendirilebilecek standart girdi çıktıyı kullanmaktır ve ekrana yazılır. Son olarak, istemcinin hiçbir şey söylememesi de mümkündür ve bu başarımı yükseltmek için kritiktir. Sınamalarda, standart girdi-çıktı ve kayıt tutulmaması arasında 10 kata kadar başarım farkı gözlenmiştir.

5.6 Sunucu

Sunucunun amacı, uygulamanın yapmak istediği sistem çağrısını gerçek aygıt üzerinde yapıp sonuçları iletmektir. Bu sayede NCD sisteminde uygulamanın elde ettiği sonuçlar gerçek sistemle aynıdır.

Sunucunun çekirdek sürücüsü olmamasının nedeni burada yatmaktadır. Eğer çekirdek sürücücüsü olsaydı, erişeceği kaynak NCD'ye erişen uygulamadan farklı olacaktı. Bu bir şekilde farklı sonuçların elde edilmesi ve saydamlığın kaybolması açısından yeterlidir.



Şekil 5.12: Sunucu işleyişi

Sunucu, her gelen istek için ayrı bir iş parçacığı açar. Hemen sonra bu iş parçacığı çatallanır. Çatallanma sonrasında, ana işlem çocuğun kapanmasını bekler. Çocuk ise elde edilen soketi dinleyerek istenen görevleri yerine getirir. Çocuğun görevi kural dışı bir durum oluştuğunda, soket kapandığında veya kapatma isteği yanıtlandığında biter.

Sunucu ayar dosyası grameri aşağıdaki gibidir:

```
SUNUCU_KAPISI_BAŞI := "server_port"

ÇATAL_KAPISI_BAŞI := "fork_port"

SATIR_BAŞI :=  "device:"
```

YENI_SATIR := ASCII/13

minör_no: [0..1048575]

aygıt_yolu: (yazı)

AYAR_DOSYASI := (SATIR_BAŞI minör_no aygıt_yolu YENI_SATIR)+

5.7 Ortak yapılar

Yazılım maliyetini düşürmek amacıyla istemci ve sunucu ortak yapılar kullanır. Ancak bu parçaların birbirinden ayrı düğümlerde olabileceği için, bu ortaklık bağlama zamanında kaybolur. Dana önce belirtilen istek ve yanıt veri yapıları da bu gruba girmektedir.

Ortak kodlar common.h dosyasında tanımlanmıştır. Bu dosyada Tablo 5.12'deki metotlar tanımlanmıştır.

Tablo 5.12: Ortak metotlar

Dönüş	***Adı***	***Parametreler***	***Amaç***
void	dump_response	response_t*	Ağ üzerinden hatalı gelen yanıtı ekrana basar.
response*	generate_request_response	request_t*, response_t*	İsteğin başlığından yanıt başlığını oluşturur.
response*	create_error_response	response_t*, request_t*	Öntanımlı hata yanıtını oluşturur.
int	init_set_proc_title	char*, int, char*[], char*[]	Süreç isim değişikliğini koşullar.
void	set_proc_title	const char*, const char*, ...	Süreç isim değişikliğini sağlar.
ssize_t	unbuffered_read	int, void*, size_t	Bloklanmayan okuma.
ssize_t	unbuffered_write	int, const void*, size_t	Bloklanmayan yazma.

Ağ iletişiminde, select sistem çağrısı uygulama seviyesindeki paketin tamamı gelmeden veri olduğu iddiasıyla olumlu dönebilir. Bu durumda yapılacak bir okuma, verinin tamamen alınamamasına ve iletişim bozukluğuna yol açar. bloklamadan_oku

metotu, belirtilen bir işlevin tam olarak istenilen kadar okumasını garantilemeye yarar. Bu prensiple hazırlanan, ortak yapılardan unbuffered_write ve unbuffered_read'in algoritması aşağıdadır:

```
bloklamadan_oku ( metot, tanımlayıcı, tampon, toplam_adet )
  kayıklığı sıfırla ilklendir
  kayıklık toplam_adetten küçük olduğu sürece
    eğer uygulama kapanıyorsa –1 döndür
    dönüş_değeri := metot ( tanımlayıcı, tampon + kayıklık, toplam_adet – kayıklık )
    eğer dönüş_değeri 0'dan küçükse (hata varsa) veya 0'a eşitse (soket kapandıysa)
        kayıklığı döndür
    değilse dönüş_değerini kayıklığa ekle
  çevrim
  kayıklığı döndür
```

6 SONUÇ

Gelişen bilgisayar sistemlerinde ağ teknolojilerinin önemi gün geçtikçe artmaktadır. Günümüzde ağ teknolojileri sadece iletişim amaçlı kullanılmamakta, hizmetlerin merkezileştirilmesi sayesinde önemli tasarruf sağlayan bir kolaylık olarak da göze çarpmaktadır. Çağdaş ağlarda, şifreler, kullanıcı dosyaları, yapılandırmalar hatta aygıtlar, yazılımlar aracılığıyla başka bilgisayarlardan temin edilebilmektedir.

Bu çalışmada aygıtların uzaktan kullanılmasıyla ilgili prensipler ve yaygın yöntemler belirtilmiş, Network Character Device olarak isimlendirilen karakter aygıtlarının uzak adreslenmesine yarayan yapının tasarımı tanımlanmıştır. NCD, jenerik karakter aygıtı olarak görev yapan, kendisine gelen istekleri istemcisine ve oradan da seçilen sunucuya ileten ve yer soyutlamasını işletim sistemi düzeyinde yaparak uygulamalara kolaylık sağlayan bir altyapıdır.

NCD kullanan uygulamalar hiçbir yazılım eklentisi veya derleme gerektirmeksizin, normal bir aygıtı kullanır gibi çalışırlar. Kendiliğinden devreye giren NCD sürücüsü, yapılan isteği karşı düğüme gönderir ve sonucu uygulamaya iletir. Bu işlem çerçevesinde, aygıtın isteği olumsuz karşılamasına NCD cevap vermez ve konuyu uygulamaya bırakır. Örneğin yazma isteğinin başarılı veya başarısız oluşu, sistem için bir fark teşkil etmez, bu uygulamanın yorumlayacağı bir durumdur.

NCD, önceki aygıt paylaştırma yöntemlerinde olduğu gibi uygulamaya bir kütüphane aracılığıyla kendisi üzerinden akış verme yeteneği değil, işletim sistemi seviyesinde yaptığı değişiklikle isteklerin donanıma değil uzaktaki bir sunucuya iletilmesi ve sunucudan gelen yanıtların işletim sisteminden geliyormuş gibi uygulamaya iletilmesi prensibine dayanmaktadır. Bu prensibi gerçeklemek amacıyla, bir çekirdek sürücüsü, bu sürücü tarafından tetiklenen bir istemci ve fiziksel aygıta sahip uzak bilgisayarda çalışan bir sunucuya tasarlanmış ve hayata geçirilmiştir.

NCD'nin aygıt paylaştırma kavramına getirdiği yenilik işletim sistemi tasarım prensipleri seviyesinde olduğundan, modern dağıtık işletim sistemlerinin tasarlanması ve uygulanmasında kullanılabilecek bir yöntemi ifade etmekten gurur duyulmaktadır. Yaygın özgür yazılım çekirdeği Linux geliştirici takımı tarafından 1999, 2001 ve 2003 tarihlerinde tartışılmış olmasına rağmen sonuca varacak kararlılığın ve bilgi birikiminin derlenememesi gerçeği, NCD'nin prestijini bir kat daha artırmaktadır.

Çalışmanın ağ güvenliği kriterlerinin artırılması ve Linux çekirdek ağacının içine dahil edilmesi amacıyla sürdürülebilir bir çalışma grubu kurulması ileri çalışmalar olarak not düşülmelidir.

KAYNAKLAR

[1] **Stallings, W.,** 1998. Operating Systems – Internals and Design Principles, Prentice Hall, New Jersey.

[2] **Bacon, J. ve Harris, T.,** 2003. Operating Systems – Concurrent and Distributed Software Design, Addison Wesley, İngiltere.

[3] **Rubini, A. and Corbet, J.,** 2001. Linux Device Drivers, O'Reilly,

[4] Linux e-posta listesi, www.ussg.iu.edu/hypermail/linux/kernel/

[5] **Baykal, N.,** 2001. Bilgisayar Ağları, SAS Bilişim, Ankara.

[6] **Chow, R., Johnson, T.,** 1997. Distributed Operating Systems & Algorithms, Addison Wesley, ABD.

[7] **Stevens, W. R.,** 2002. UNIX Network Programming, Prentice Hall, New Jersey.

ÖZGEÇMİŞ

Ozan Eren BİLGEN 27 Temmuz 1981 tarihinde İzmir'de doğmuştur. Lise öğrenimini İzmir Bornova Anadolu Lisesi'nde tamamlamış, 2003 yılında Galatasaray Üniversitesi Bilgisayar Mühendisliği Bölümü'nü bitirmiştir.

1999-2001 döneminde Galatasaray Üniversitesi baş webmasterlığı görevine getirilen Bilgen; Galatasaray Üniversitesi Linux Kullanıcıları Grubu yöneticisi, Türkiye Linux Kullanıcıları Derneği üyesi ve High Availability Linux Project Group katkıcısıdır.

Almanca, İngilizce ve Fransızca bilen Bilgen halen TÜBİTAK / UEKAE'de araştırmacı olarak çalışmaktadır.