

Beej'in Ağ Programlama Kılavuzu

Internet Soketlerini Kullanarak

Yazan:
Brian "Beej" Hall
<beej (at) piratehaven.org>

Çeviren:
Emre "FZ" Sevinç
<fz (at) ileriseviye.org>

Özet

Bu belge bir öğretici olarak tasarlanmıştır ve tam teşekküllü bir başvuru kılavuzu değildir. Soket programlama konusuna ciddi ciddi merak salan bireyler tarafından adım adım okunursa işe yarayacaktır.

İçindekiler

1. Giriş	4
1.1. Kimin İçin?	4
1.2. Platform ve Derleyiciler	4
1.3. Resmi Anasayfa	4
1.4. Solaris/SunOS Programcılarının Dikkat Etmesi Gerekenler	4
1.5. Windows Programcılarının Dikkat Etmesi Gerekenler	4
1.6. E-posta Politikası	5
1.7. Yansılama	6
1.8. Çevirmenlerin Dikkatine	6
1.9. Telif Hakkı ve Dağıtım	6
2. Soket Nedir?	6
2.1. Internet Soketlerinin İki türü	7
2.2. Düşük Seviye Duyarsızlığı ve Ağ Teorisi	8
3. struct'lar ve Veri İle Uğraşmak	9
3.1. Veri Türlerini Dönüştür!	10
3.2. IP Adresleri ve Bunlarla Uğraşma Yöntemleri	11
4. Sistem Çağrıları veya Felaketleri	12
4.1. socket() — Al Şu Dosya Tanımlayıcıyı!	12
4.2. bind() — Hangi Port Üzerindeyim?	13
4.3. connect()—Hey, sen!	15
4.4. listen() — Biri Beni Arayabilir mi Acaba?	16
4.5. accept() — "3490 Numaralı Portu Aradığınız İçin Teşekkürler"	16
4.6. send() ve recv() — Konuş Benimle Bebeğim!	17
4.7. sendto() ve recvfrom() — Benimle UDP'ce Konuş	18
4.8. close() ve shutdown() — Düş Yakamdan!	19
4.9. getpeername() — Kimsiniz?	20
4.10. gethostname() — Ben kimim?	20
4.11. DNS — Sen "whitehouse.gov" de, ben de "198.137.240.92" diyeyim	20
5. İstemci-Sunucu Mimarisi	22

5.1. Basit Bir Veri Akış Sunucusu	23
5.2. Basit Bir Veri Akış İstemcisi	25
5.3. Veri Paketi Soketleri	26
6. İleri Teknikler	28
6.1. Bloklama	29
6.2. select() — Eşzamanlı G/Ç Çoğullama	29
6.3. Sorunlu send() Durumları	34
6.4. Veri Paketlemesi Hazretleri	35
7. Diğer Kaynaklar	37
7.1. man Sayfaları	37
7.2. Kitaplar	38
7.3. Web Kaynakları	39
7.4. RFC'ler	39
8. Sıkça Sorulan Sorular	39
9. Son söz ve Yardım Çağrısı	45

Özgün belgenin sürüm bilgileri:

1.0.0 İlk sürüm.	Ağustos 1995	beej
1.5.5 Son HTML sürümü.	13 Ocak 1999	beej
2.0.0 DocBook XML formatına dönüştürüldü, düzeltmeler ve eklemeler.	6 Mart 2001	beej
2.3.1 client.c içindeki birkaç sözdizimi hatası giderildi, sıkça sorulan sorular bölümüne yeni malzeme eklendi.	8 Ekim 2001	beej

Bu çevirinin sürüm bilgileri:

0.9 Çeviri ve ilk kontroller yapıldı.	15 Aralık 2002	FZ
--	----------------	----

Yasal Açıklamalar

BU BELGE "ÜCRETSİZ" OLARAK RUHSATLANDIĞI İÇİN, İÇERDİĞİ BİLGİLER İÇİN İLGİLİ KANUNLARIN İZİN VERDİĞİ ÖLÇÜDE HERHANGİ BİR GARANTİ VERİLMEMEKTEDİR. AKSİ YAZILI OLARAK BELİRTİLMEDİĞİ MÜDDETÇE TELİF HAKKI SAHİPLERİ VE/VEYA BAŞKA ŞAHISLAR BELGEYİ "OLDUĞU GİBİ", AŞIKAR VEYA ZIMNEN, SATILABİLİRLİĞİ VEYA HERHANGİ BİR AMACA UYGUNLUĞU DA DAHİL OLMAK ÜZERE HİÇBİR GARANTİ VERMEKSİZİN DAĞITMAKTADIRLAR. BİLGİNİN KALİTESİ İLE İLGİLİ TÜM SORUNLAR SİZE AİTTİR. HERHANGİ BİR HATALI BİLGİDEN DOLAYI DOĞABİLECEK OLAN BÜTÜN SERVİS, TAMİR VEYA DÜZELTME MASRAFLARI SİZE AİTTİR.

İLGİLİ KANUNUN İCBAR ETTİĞİ DURUMLAR VEYA YAZILI ANLAŞMA HARİCİNDE HERHANGİ BİR ŞEKİLDE TELİF HAKKI SAHİBİ VEYA YUKARIDA İZİN VERİLDİĞİ ŞEKİLDE BELGEYİ DEĞİŞTİREN VEYA YENİDEN DAĞITAN HERHANGİ BİR KİŞİ, BİLGİNİN KULLANIMI VEYA KULLANILAMAMASI (VEYA VERİ KAYBI OLUŞMASI, VERİNİN YANLIŞ HALE GELMESİ, SİZİN VEYA ÜÇÜNCÜ ŞAHISLARIN ZARARA UĞRAMASI VEYA BİLGİLERİN BAŞKA BİLGİLERLE UYUMSUZ OLMASI) YÜZÜNDEN OLUŞAN GENEL, ÖZEL, DOĞRUDAN YA DA DOLAYLI HERHANGİ BİR ZARARDAN, BÖYLE BİR TAZMİNAT TALEBİ TELİF HAKKI SAHİBİ VEYA İLGİLİ KİŞİYE BİLDİRİLMİŞ OLSA DAHI, SORUMLU DEĞİLDİR.

Tüm telif hakları aksi özellikle belirtilmediği sürece sahibine aittir. Belge içinde geçen herhangi bir terim, bir ticari isim ya da kuruma itibar kazandırma olarak algılanmamalıdır. Bir ürün ya da markanın kullanılmış olması ona onay verildiği anlamında görülmemelidir.

1. Giriş

Hey! Soket programlama ile başınız belada mı? Bütün bu ayrıntılar **man** sayfalarından çekip çıkarmak için çok mu zor? En temel Internet programlama tekniklerini öğrenmek istiyorsunuz ama tonlarca `struct` ve bunları `bind()` işlevini çağırmadan `connect()`, vs., vs.ye parametre olarak nasıl geçeceğiniz konusunda binlerce ayrıntıyı öğrenmeye vaktiniz yok mu?

Hmm, bakın burada ne var? Bütün bu sinir bozucu ayrıntılarla ben zamanında boğuştuğum ve herkesle deneyimlerimi paylaşmak için can atıyorum! Doğru yere geldiniz. Bu belge ortalama bir C programcısına tüm bu ağ meseleleri ile ilgili temel kavramları ve pratik uygulamaları verecek düzeydedir.

1.1. Kimin İçin?

Bu belge bir öğretici olarak tasarlanmıştır ve tam teşekkürlü bir başvuru kılavuzu değildir. Soket programlama konusuna ciddi ciddi merak salan bireyler tarafından adım adım okunursa işe yarayacaktır. Bu belge kesinlikle *eksiksiz* bir soket programlama kılavuzu değildir.

Eğer şu **man** sayfalarının sizin için biraz daha anlamlı hale gelmesini sağlarsa bu belge amacına ulaşmış demektir... :-)

1.2. Platform ve Derleyiciler

Bu belgedeki kodlar Linux çalıştıran bir PC üzerinde Gnu'nun **gcc** derleyicisi ile derlenmiştir. Ancak **gcc** derleyicisinin çalışabildiği her platformda derlenebilir. Tabii eğer Windows kullanıyorsanız yukarıda dediklerim geçerli değildir. Bunun için aşağıdaki *Windows Programcılarının Dikkat Etmesi Gerekenler* (sayfa: 4) bölümüne bakın.

1.3. Resmi Anasayfa

Bu belgenin resmi adresi California Devlet Üniversitesi, Chico, <http://www.ecst.csuchico.edu/~beej/guide/net/dir>.

1.4. Solaris/SunOS Programcılarının Dikkat Etmesi Gerekenler

Solaris ya da SunOS için derlerken gerekli işlev kitaplıklarını programa bağlayabilmek için bazı ek parametreler vermeniz gerekebilir. Bunun için `-lnsl -lsocket -lresolv` parametrelerini derleme komutunuzun sonuna ekleyebilirsiniz, örnek:

```
$ cc -o server server.c -lnsl -lsocket -lresolv
```

eğer hala hata alıyorsanız şunu da ekleyin: `-lxnet`. Ne işe yaradığını bilmiyorum ama görünen o ki bazı durumlarda gerekebiliyor.

Problem yaşayabileceğiniz bir başka yer de `setsockopt()` işlevinin çağırıldığı yerdir. Bu işlevin Solaris/SunOS'taki prototipi benim Linux makinamdakinden farklıdır bu yüzden de:

```
int yes=1;
```

yerine bunu girin:

```
char yes='1';
```

Elimde bir Sun makinası yok, yukarıdakileri denemedim bu bilgiler bana deneme yapıp e-posta gönderen insanların söylediklerinden ibarettir.

1.5. Windows Programcılarının Dikkat Etmesi Gerekenler

Windows'dan pek hoşlandığım söylenemez bu yüzden de bu belgeyi okuyan tüm Windows programcılarını GNU/Linux, BSD ya da UNIX denemeye davet ediyorum. Bu laflardan sonra, evet, örnek kodları Windows üzerinde kullanma imkanınız var.

Öncelikle burada bahsettiğim birçok sistem başlık dosyasını unutun. Tek yapmanız gereken aşağıdakileri programınıza katmak:

```
#include <winsock.h>
```

Bir dakika! Aynı zamanda `WSAStartup()` işlevini de soket kitaplıkları ile herhangi bir iş yapmadan önce çağırmanız gerekiyor. Bunu yapmak için gerekli kod şöyle bir şey:

```
#include <winsock.h>

{
    WSADATA wsaData;    // if this doesn't work
    //WSADATA wsaData; // then try this instead

    if (WSAStartup(MAKEWORD(1, 1), &wsaData) != 0) {
        fprintf(stderr, "WSAStartup failed.\n");
        exit(1);
    }
}
```

Tabii derleyicinize Winsock'u da bağlamasını söylemelisiniz. Bunun için gerekli dosyanın ismi genellikle şudur: `wsock32.lib` veya `winsock32.lib` veya benzer bir şey. VC++ ortamında iseniz, bunun için `Project` menüsünden, `Settings...` kısmına gidin ve `Link` sekmesine gelip `Object/library modules` kutusunu bulun. `wsock32.lib` dosyasını bu listeye ekleyin.

En azından ben böyle duydum.

Ve son olarak da `WSACleanup()` işlevini çağırmanız gerekir soket kitaplığı ile işiniz bittiğinde. Ayrıntılı bilgi için derleyicinizin yardım belgelerine bakın.

Bunu yaptığınızda bu belgedeki örneklerin hemen hepsi genel olarak çalışabilir durumda olmalı belki birkaç küçük değişiklik yapmanız gerekebilir ama hepsi bu. Dikkat etmeniz gerekenler: Soketi kapatmak için `close()` işlevini kullanamazsınız — bunun için `closesocket()` işlevini kullanmalısınız. Ayrıca `select()` işlevi sadece soket tanımlayıcılar içindir, dosya tanımlayıcılar (standart girdi için 0 kullanılması gibi) için değildir.

Aynı zamanda, `CSocket` isimli bir soket sınıfı da mevcuttur. Ayrıntılı bilgi için derleyicinizin belgelerini karıştırın.

Winsock hakkında ayrıntılı bilgi için şu adrese bakabilirsiniz: [Winsock FAQ^{\(B3\)}](#)

Son olarak da bildiğim kadarı ile Windows ortamında `fork()` isimli işlev yok ve maalesef örneklerde bu işlevi kullanmak durumdayım. Belki de bir POSIX kitaplığına programınızı bağlamanız gerekebilir veya `CreateProcess()` işlevini kullanabilirsiniz. `fork()` işlevi herhangi bir argüman almaz ama `CreateProcess()` işlevi 48 milyar argüman alır. Eğer bu gözünüzü biraz korkuttu ise `CreateThread()` işlevi biraz daha kolay bir alternatif olabilir ancak "multithreading" tartışması bu belgenin sınırlarının ötesindedir. Windows konusunu böylece burada kapatıyoruz!

1.6. E-posta Politikası

Genellikle e-posta ile gönderilen sorulara cevap vermeye çalışırım, bu yüzden yazmaktan çekinmeyin, ancak bu size cevap vereceğim anlamına gelmez. Epey meşgulüm ve bazen sorunuzun cevabını tam olarak bilmiyorum olabilirim. Durum bu olduğunda mesajınızı silerim lütfen bunu şahsınıza yönelik bir harekete olarak algılamayın. İnanın bana sorduğunuz her soruyu en ince ayrıntısına kadar cevaplayacak kadar vaktim olmayabilir.

Basit bir kural: Sorunuz ne kadar karmaşık ise cevap verme ihtimalim o kadar düşüktür. Eğer soru kapsamını daraltır ve derdinizle ilgili ayrıntılı bilgileri de yollarsanız (platform, derleyici, hata mesajları, vs.), cevap alma ihtimaliniz artar. Daha ayrıntılı bilgi için şu adresi tavsiye ederim: [How To Ask Questions The Smart Way^{\(B4\)}](#).

Eğer cevap alamıyorsanız biraz daha kurcalayın programınızı ve cevabı bulmaya çalışın. Hala çözemedi iseniz o zaman bana gene yazın ve belki o zaman elimizdeki ayrıntılı bilgilerle probleme bir çözüm bulmamız daha kolay olabilir.

Şimdi sizi bana nasıl yazmanız ve yazmamanız konusunda *yeterince* eğittiğime göre bu kılavuzla ilgili süreç içinde bana ulaştırılan övgülerin beni ne kadar motive ettiğini de itiraf etmek isterim. : -) Teşekkürler!

1.7. Yansılama

Sitemin bir yansısını barındırmak istiyorsanız memnuniyet duyarım. Eğer oluşturacağınız yansiya ana sayfamdan link vermek isterseniz [<beej \(at\) piratehaven.org>](mailto:beej@piratehaven.org) adresine bir mesaj yollamanız yeterli olacaktır.

1.8. Çevirmenlerin Dikkatine

Eğer bu belgeyi başka bir dile çevirmek isterseniz lütfen [<beej \(at\) piratehaven.org>](mailto:beej@piratehaven.org) adresinden benimle iletişim kurun böylece ana sayfadan sizin tercümenizin bulunduğu sayfaya bağ verebilirim.

Çeviriye kendi isminizi ve e-posta adresinizi eklemekte tereddüt etmeyin.

Üzgünüm yer kısıtlamasından ötürü sizin çevirinizi kendi sitemde barındıramam.

1.9. Telif Hakkı ve Dağıtım

Beej's Guide to Network Programming – Copyright © 1995–2001 Brian "Beej" Hall.

Bu belge bütünlüğü korunduğu, içeriği değiştirilmediği ve bu kopyalama bilgisi de yayınlandığı sürece herhangi bir ortamda serbestçe yeniden yayınlanabilir.

Öğretim görevlileri bu belgeyi öğrencilerine tavsiye edebilirler.

Bu belge herhangi bir dile tercüme edilebilir yeter ki söz konusu tercüme aslına sadık kalsın ve belgenin bütünlüğünü bozmasın. Tercüme, tercüme yapanın ismini ve temas bilgilerini içerebilir.

Belgedeki C kaynak kodları kamuya açıktır.

İletişim adresim: [<beej \(at\) piratehaven.org>](mailto:beej@piratehaven.org).

2. Soket Nedir?

Sürekli "socket"lerden bahsedildiğini duymuşsunuzdur ve belki de bunların tam olarak ne anlama geldiğini merak ediyordunuz. Soket kısaca şudur: Diğer programlarla standart Unix dosya tanımlayıcılarını kullanarak haberleşmenizi sağlayan bir yapı.

Ne?

Pekala — bazı Unix hacker'larının, "Vay canına! Unix'teki hemen hemen *herşey* bir dosya!" dediğini duymuş olabilirsiniz. Böyle konuşan birinin kast ettiği aslında Unix programlarının, herhangi bir G/Ç işlemi yaptıklarında bunları bir dosya tanımlayıcıyı okuyarak ya da ona yazarak yaptıklarıdır. Bir dosya tanımlayıcı basitçe söylemek gerekirse açık bir dosya ile ilişkilendirilmiş bir tamsayıdır. Ancak (işin püf noktası da burası), söz konusu bu açık dosya diskteki normal bir dosya olabileceği gibi aynı zamanda bir ağ bağlantısı, bir FIFO, bir uçbirim ya da başka herhangi bir veri kaynağı olabilir. Gerçekten de Unix ortamında her şey bir dosyadır! Öyleyse Internet üzerinden başka bir programla iletişim kurmak isterseniz bunu bir dosya tanımlayıcı üzerinden yapacaksınız, inanın buna.

"Peki bay çok bilmiş, ağ iletişimi için kullanacağım bu dosya tanımlayıcı nerede?" gibi bir soru aklınıza gelmemiş olabilir ancak ben gene de cevabını vereyim ki içiniz rahat etsin: Bu dosya tanımlayıcıya ulaşmak için `socket()` sistem işlevini çağırmanız gerekir. Bu işlev size soket tanımlayıcıyı döndürür ve siz de bunu ve tabii `send()` ile `recv()` (`man send(B5)`, `man recv(B6)`) isimli soket işlevlerini kullanarak istediğiniz şekilde iletişimimizi kurarsınız.

"Hey, bir dakika!" diyebilirsiniz şimdi. "Eğer bir dosya tanımlayıcı söz konusu ise o halde tanrı aşkına neden her zaman kullandığım normal `read()` ve `write()` işlevlerini kullanarak soketler üzerinden iletişim kuramayayım ki?" Kısa cevap: "Evet tabii ki!" Uzun cevap ise "Evet, mümkün ama `send()` ve `recv()` işlevleri veri iletişimde çok daha fazla kontrol sağlar ve işinizi kolaylaştırır."

Sırada ne var? Buna ne dersiniz: çeşit çeşit soket vardır. Mesela DARPA Internet adresleri (Internet Soketleri), yerel bir düğümdeki (node) yol isimleri (Unix Soketleri), CCITT X.25 adresleri (X.25 Soketleri ki inanın bunları bilmeseniz de olur) ve kullandığınız Unix sürümüne bağlı daha pek çok soket tipi. Bu belge sadece birinci tür soketleri ele almaktadır, yani: Internet Soketleri.

2.1. Internet Soketlerinin İki türü

Bu da ne? İki tür Internet soketi mi var? Evet. Şey, aslında hayır. Yalan söyledim. Daha çok var ama sizi korkutmak istemedim. Size sadece iki tür soketten bahsedeceğim. Sadece "Ham Soketler" in (Raw Sockets) çok güçlü olduğunu belirttiğim bu cümle dışında yani, bir ara bunlara da göz atarsanız iyi olur.

Pekala, gelelim şu iki tür sokete. Bir tanesi "Veri Akış Soketleri"; diğeri ise "Veri Paketi Soketleri" ve artık biz bunlara sırası ile "`SOCK_STREAM`" ve "`SOCK_DGRAM`" diyeceğiz. Veri paketi soketleri bazen "bağlantısız soketler" olarak da isimlendirilir. (Her ne kadar eğer isterseniz bunlara `connect()` işlevi ile bağlanabilecek olsanız da. Detaylı bilgi için aşağıdaki `connect()` (sayfa: 15) maddesine bakın.)

Veri akış soketleri güvenilir iki yönlü iletişim kanallarıdır. Eğer bu tür bir sokete sıra ile "1, 2" bilgilerini gönderirseniz, bunlar kanalın diğer ucundan "1, 2" şeklinde çıkar. Aynı zamanda bu iletişim olası hatalara karşı da korumalıdır. Karşılaşacağınız her türlü hatanın kaynağı sizin karman çorman aklınız olacaktır ve burada bunları tartışmaya niyetim yok.

Stream soketleri gerçek hayatta ne işimize yarar? Hmm, sanırım `telnet` diye bir program duymuşsunuzdur, değil mi? İşte bu program veri akış soketlerini kullanır. Yazdığınız tüm karakterler sizin yazdığınız sırada iletilmelidir öyle değil mi? Aynı zamanda web tarayıcılar da HTTP protokolü ile iletişim kurarken veri akış soketleri aracılığı ile sayfa bilgilerini çekerler. Gerçekten de eğer bir web sitesine `telnet` ile 80 numaralı port üzerinden bağlanır ve "`GET /`" komutunu yollarsanız web sitesi size HTML içeriğini yollayacaktır.

Veri akış soketleri bu kadar sorunsuz bir veri iletişimini nasıl gerçekleştirir? Bunun için "Aktarım Denetim Protokolü" (Transmission Control Protocol) isimli kurallar dizisinden yararlanırlar ki siz bunu "TCP" olarak da duymuş olabilirsiniz (bkz. `RFC-793(B8)` bu belge TCP ile ilgili çok ayrıntılı bilgi içerir.) TCP yollanan verinin sıralı ve düzgün gitmesini sağlar. "TCP"yi daha önce "TCP/IP" kısaltmasının bir parçası olarak duymuş olabilirsiniz ki "IP" de "Internet Protocol" sözünün kısaltmasıdır (bkz. `RFC-791(B9)`.) IP kurallar dizisi temelde Internet yönlendirme ile ilgilidir, veri bütünlüğünün korunması ile ilgili pek kural içermez.

Harika. Peki veri paketi soketleri? Neden bunlara bağlantısız deniyor? Mesele nedir kısaca? Neden bunların "güvenilmez" olduğu söyleniyor? Bakın, bilmeniz gereken bazı gerçekler var: eğer bir veri paketi yollarsanız bu hedefine ulaşabilir de ulaşmayabilir de. Gönderdiğiniz sırada ulaşması garanti edilemez. Ancak eğer hedefe ulaşırsa paketin içerdiği bilgi hatasız olacaktır.

Veri paketi soketleri de yönlendirme için IP protokolünü kullanırlar ancak TCP'den faydalanmazlar onun yerine "Kullanıcı Veri Paketi Protokoku" (User Datagram Protocol) veya "UDP" isimli protokolü kullanırlar (bkz. `RFC-768(B10)`.)

Neden bağlantısızdırlar? Aslında böyledirler çünkü veri akış soketlerinde olduğu gibi bağlantıyı sürekli açık tutmanız gerekmemektedir. Sadece bir paketi oluşturur, tepesine gideceği adresi söyleyen bir IP başlığı yapıştırır ve onu yollarsınız. Herhangi bir bağlantı açmaya gerek yoktur. Bu tip soketler genellikle paket paket iletilen veri için kullanılır. Bu tip soketleri kullanan örnek uygulamalardan bazıları: **tftp**, **bootp**, vs.

"Yeter artık!" diye bağırdığınızı duyar gibiyim. "Eğer veri paketlerinin yolda kaybolma ihtimali varsa nasıl olur da yukarıda saydığım programlar çalışır?!" Bak dünyalı dostum bu saydığım programların hepsi UDP protokolü üzerine kendi protokollerini yerleştirirler. Mesela, **tftp** protokolüne göre gönderilen her paket için karşı tarafın "aldım!" (bir "ACK" paketi) paketini geri yollaması gerekir. Eğer orjinal paketin göndericisi mesela 5 saniye içinde cevap alamazsa o zaman paketi yeniden yollar, taa ki ACK cevabını alana kadar. İşte bu "aldım" prosedürü **SOCK_DGRAM** uygulamalarında çok önemlidir.

2.2. Düşük Seviye Duyarsızlığı ve Ağ Teorisi

Protokol katmanlarından bahsettiğime göre artık ağların nasıl çalıştığına dair gerçekleri öğrenmenin ve **SOCK_DGRAM** paketlerinin nasıl oluşturulduğuna dair örnekler vermenin zamanı geldi. Pratik olarak bu bölümü atlayabilirsiniz, ancak burayı okursanız iyi bir temel bilgiye sahip olursunuz.

Şekil 1. Veri Paketlemesi (data encapsulation)



Protokollerin Veriyi Paketlemesi

Hey çocuklar, Veri Paketleme konusunu öğrenme zamanı! Bu çok çok önemli. O kadar önemli o kadar önemli ki burada yani Chico Eyaletinde ağ teknolojilerine dair bir kurs alırsanız bu konu ile mutlaka karşılaşacaksınız ; -). Temelde konu şu: bir paket doğar, sonra bu paket önce muhatap olduğu ilk protokol tarafından (mesela TFTP protokolü) ile bir başlık (ve ender olarak da olsa bir dipnot ile) kullanılarak paketlenir (kapsüle konur), ardından tüm bu yığın (TFTP başlığı da dahil) bir sonraki protokolün (örn. UDP) kurallarına göre paketlenir, sonra IP ve en sonundaki en alt katman olan donanım katmanındaki fiziksel protokol ile (örn. Ethernet) paketlenir.

Başka bir bilgisayar bu paketlenmiş paketi aldığı anda önce donanım Ethernet başlığını çıkarır, ardından işletim sistemi çekirdeği IP ve UDP başlıklarını alır ve ardından da TFTP programı TFTP başlığını alır ve içindeki veriye erişir.

Artık şu kötü üne sahip Katmanlı Ağ Modeli (Layered Network Model) kavramından bahsedebilirim. Bu ağ modeli diğer modellere pek çok üstünlüğü bulunan bir ağ işlevselliğinden bahseder. Mesela yazdığınız soket programının tek bir harfini bile değiştirmenize gerek kalmadan bu programı farklı fiziksel donanımlar üzerinde çalıştırabilirsiniz (seri, thin Ethernet, AUI, her ne ise) çünkü düşük seviyedeki programlar sizin yerinize ayrıntıları hallederler. Ağ donanımının fiziksel detayları ve topolojisi soket programcısını ilgilendirmez.

Daha fazla laf kalabalığı yapmadan bu müthiş modelin katmanlarını liste olarak vereyim, ağ ile ilgili sınava girerseniz işe yarayabilir:

- Uygulama
- Sunum
- Oturum
- Taşıma
- Ağ
- Veri Bağlantısı
- Fiziksel

Fiziksel katman donanımla ilgilidir (seri, Ethernet, vs.). Uygulama katmanı fiziksel katmandan alabildiğine uzaktır — kullanıcılar ağ ile bu katmanda temas kurar.

Açıkçası bu model o kadar geneldir ki eğer isterseniz bu modeli arabanıza bile uygulayabilirsiniz. Unix ile daha uyumlu bir katmanlı model şöyle yazılabilir:

- Uygulama Katmanı (*telnet, ftp, etc.*)
- Konaktan Konağa Taşıma Katmanı (*TCP, UDP*)
- İnternet Katmanı (*IP ve yönlendirme*)
- Ağa Erişim Katmanı (*Ethernet, ATM ya da her ne ise*)

Bu aşamada söz konusu katmanların veri paketlenmesinin hangi aşamalarına karşılık geldiğini görebiliyor olmalısınız.

Tek bir paketi oluşturmak için ne kadar çok iş yapılması gerektiğini gördünüz mü! Aman allahım! Üstelik paket başlık bilgilerini de **cat** komutunu kullanarak elle girmeniz gerekiyor! Şaka şaka! Tek yapmanız gereken eğer veri akış soketi kullanıyorsanız `send()` ile veriyi göndermek. Eğer veri paketi soketi kullanıyorsanız bu sefer de yapılması gereken paketi uygun şekilde paketleyip `sendto()` işlevini kullanarak bunu yollamak. Çekirdek sizin için Taşıma katmanını ve İnternet katmanını kurar, donanımınız da Ağa Erişim Katmanını halleder. Ah, modern teknoloji.

Ağ teorisi ile ilgili kısa dersimiz burada sona eriyor. Ah evet tabii ki yönlendirmeye ilgili söylemem gereken şeyler vardı: unutun! Bundan bahsetmeyeceğim. Yönlendirici (router) IP başlığını çeker yönlendirme tablosuna bakar, seçim yapar, vs. vs. vs. Eğer gerçekten meraklı iseniz [IP RFC^{\(B11\)}](#) belgesine bakın. Bunu öğrenmezsiniz ölmezsiniz.

3. **struct**'lar ve Veri İle Uğraşmak

Hele şükür bu aşamaya gelebildik. Artık biraz programlamadan bahsedebiliriz. Bu bölümde soket arayüzleri tarafından kullanılan pek çok veri türünü ele alacağım çünkü bunlar gerçekten önemli.

Kolay olanlarla başlayalım: bir soket tanımlayıcı. Bir soket tanımlayıcı aşağıdaki türdendir:

```
int
```

Evet, gayet klasik, alışık olduğumuz basit bir `int`.

İşte bu aşamadan itibaren işler biraz garipleşmeye başlıyor bu yüzden de dikkatlice okuyun ve bana güvenin. İlk bilmeniz gereken: iki tür byte sıralaması vardır: en önemli baytın (ki buna bazen öktet de denir) önce geldiği sıralama veya en önemli baytın sonra geldiği sıralama. Bu sıralamalardan birincisine "Ağ Bayt Sıralaması" (Network Byte Order)⁽¹⁾ denir. Bazı makinalar içsel olarak veriyi kendi belleklerinde bu şekilde depolar, bazıları ise bu sırayı dikkate almaz. Bir şeyin Ağ Bayt Sıralaması'na göre sıralanması gerektiğini söylediğimde `htons()` gibi bir işlev çağırmanız gerekecek ("Konak Bayt Sıralaması"ndan [Host Byte Order] "Ağ Bayt Sıralaması"na dönüştürebilmek için). Eğer "Ağ Bayt Sıralaması"ndan bahsetmiyorsam o zaman ilgili veriyi olduğu gibi yani "Konak Bayt Sıralaması" düzeninde bırakmanız gerekir.

My First Struct™ — `struct sockaddr`. Bu veri yapısı pek çok türde soket için soket adres bilgisini barındırır:

```
struct sockaddr {
    unsigned short    sa_family;    // adres ailesi, AF_XXX
    char              sa_data[14];  // protokol adresinin 14 byte'ı
};
```

sa_family pek çok değerden birini alabilir ama bizim örneğimizde `AF_INET` olacak. *sa_data* ise soketle ilgili hedef adres ve port numarası bilgilerini barındırır. Bunun garip olduğunu kabul ediyorum, yani herhalde bu bilgiyi kendi ellerinizle paketleyerek *sa_data* değişkenine yerleştirmek istemezsiniz değil mi?

`struct sockaddr` ile başa çıkabilmek için programcılar buna paralel bir yapı tasarlamışlar: `struct sockaddr_in` ("in" "Internet" anlamına geliyor.)

```
struct sockaddr_in {
    short int          sin_family; // Adres ailesi
    unsigned short int sin_port;   // Port numarası
    struct in_addr     sin_addr;   // Internet adresi
    unsigned char      sin_zero[8]; // struct sockaddr ile aynı boyda
};
```

Bu yapı soket adresi elemanlarına erişmeyi kolaylaştırır. Dikkat etmeniz gereken bir nokta: *sin_zero*,⁽²⁾ `memset()` işlevi kullanılarak tamamen sıfır ile doldurulmalıdır. Buna ek ve daha da önemli olarak bir `struct sockaddr_in` göstergesi `struct sockaddr` göstergesine dönüştürülebilir ve tersi de doğrudur. Yani her ne kadar `socket()` işlevi `struct sockaddr*` şeklinde bir veri beklese de siz gene de `struct sockaddr_in` kullanıp son anda gerekli dönüştürmeyi yapabilirsiniz! Ayrıca *sin_family* değişkeninin de `struct sockaddr` yapısındaki *sa_family* değişkenine karşılık geldiğini ve "AF_INET" olarak ayarlanması gerektiğini unutmayın. Son olarak *sin_port* ve *sin_addr* değişkenlerinin de Ağ Byte Sırasında bulunmaları gerektiğini unutmayın!

"Fakat," diye itiraz edebilirsiniz, "nasıl olur da tüm yapı yani `struct in_addr sin_addr` Ağ Bayt Sıralamasına göre dizilebilir ki?" Bu soru tüm zamanların en kötü `union`'larından biri olan `struct in_addr` yapısının dikkatli olarak incelenmesini gerektirir:

```
// Internet adresi (tarihi sebeplerden ötürü korunmakta)
struct in_addr {
    unsigned long s_addr; // 32-bit yani 4 bytes
};
```

Evet, bir zamanlar kötü bir `union` idi. Neyse ki geçmişte kaldı. Eğer *ina* değişkenini `struct sockaddr_in` türünden tanımladı iseniz *ina.sin_addr.s_addr* 4 byte'lık Internet adresini gösterir (Ağ Bayt Sıralamasında). Aklınızda bulunsun eğer sizin sisteminiz kahrolası `struct in_addr union`'ını kullanıyor olsa bile yine de 4 byte'lık Internet adresine tamamen benim yaptığım gibi ulaşabilirsiniz (bunu `#define`'lara borçluyuz).

3.1. Veri Türlerini Dönüştür!

Deminden beri şu Ağ ve Konak Bayt Sıralamaları ile yeterince kafa şişirdim şimdi biraz eylem zamanı!

Pekala. Dönüştürebileceğiniz iki tür vardır: `short` (iki byte) ve `long` (dört byte). Bu işlevler `unsigned` ve varyasyonları üzerinde çalışır. Örneğin bir `short` değişkenin bayt düzenini Konak Bayt Sıralamasından Ağ Bayt Sıralamasına çevirmek istiyorsunuz. "h" ile başlayalım ("host"), sonra "to" ve ardından "n" ("network") son olarak da bir "s" ("short"): h-to-n-s veya `htons()` ("Host to Network Short" olarak okursanız hatırlamanız kolay olur).

O kadar da zor sayılmaz değil mi...

Aptallarca olanlarını bir kenara bırakırsak "n", "h", "s" ve "l" ile her türlü birleşimi oluşturabilirsiniz. Örneğin tabii ki `stolh()` ("Short to Long Host") gibi bir işlev yoktur. Fakat şu işlevler vardır:

- `htons()` — "Host to Network Short" — konaktan ağa short
- `htonl()` — "Host to Network Long" — konaktan ağa long
- `ntohs()` — "Network to Host Short" — ağdan konağa short

- `ntohl()` — "Network to Host Long" — ağdan konağa long

Bu konuyu kavradığınızı düşünüyor olabilirsiniz. Mesela aklınıza şu gelebilir: "char türünde bir verinin Bayt Sıralamasını değiştirmem gerekirse ne yapmam gerekir?" Ve sonra şöyle cevap verebilirsiniz: "Aman, boşver." Ayrıca aklınıza şu da gelebilir: mesela 68000 işlemcili makinanız zaten Ağ Bayt Sıralamasını kullandığına göre `htonl()` işlevini IP adreslerine uygulamanıza gerek yoktur. Haklı olabilirsiniz. FAKAT eğer geliştirdiğiniz yazılımı öteki türlü Bayt Sıralamasına göre çalışan bir bilgisayara taşırsanız programınız kesinlikle çalışmaz. Taşınabilir programlar yazın! Unutmayın Unix dünyasındasınız! (Her ne kadar Bill Gates aksini düşünmek istese de.) Sakın unutmayın: baytlarınızı ağ üzerinde yolculuğa çıkarmadan önce onları Ağ Bayt Sıralaması'na göre dizeceksiniz.

Son bir noktaya daha dikkat çekmek istiyorum: neden `struct sockaddr_in` yapısı içindeki `sin_port` ve `sin_addr` Ağ Bayt Sıralamasında olmak zorunda iken `sin_family` böyle bir özelliğe sahip olmak durumunda değil? Cevabı: `sin_addr` ve `sin_port` sırası ile IP ve UDP katmanlarında paketlenirler. Bu yüzden Ağ Bayt Sıralamasında gönderilmeleri gerekir oysa ki `sin_family` sadece işletim sistemi çekirdeği tarafından veri yapısının barındırdığı adresin türünü tespit etmek için kullanılır. Bu yüzden de bu alanın Konak Bayt Sıralamasında bırakılması gerekir. Ayrıca `sin_family` ağ üzerinden bir yere yollanmadığı için Konak Bayt Sıralamasında olabilir.

3.2. IP Adresleri ve Bunlarla Uğraşma Yöntemleri

Şanslısınız çünkü IP adresleri ile uğraşmanızı sağlayacak bir grup işlev vardır. Yani elle hesap kitap yapıp sonra da bunu bir `long` içine `<<` işleci ile tıkıştırmanıza gerek yok.

Önce örneğin elinizde bir `struct sockaddr_in` `ina` değişkeni ve bir de `10.12.110.57` şeklinde bir IP adresi olduğunu var sayalım. Bu adresi bu değişken içine yerleştirmek istiyorsunuz. Kullanmanız gereken işlev: `inet_addr()`. Bu işlev bir IP adresini yukarıdaki gibi sayılardan ve noktalardan oluşan bir biçimden alıp `unsigned long` türünde bir sayıya çevirir. Bu tür bir atama şu şekilde yapılabilir:

```
ina.sin_addr.s_addr = inet_addr("10.12.110.57");
```

Şuna dikkat edin: `inet_addr()` zaten döndürdüğü değeri Ağ Bayt Sıralamasına göre dizilmiş olarak döndürür yani `htonl()` işlevini burada çağırmanıza gerek yok. Harika!

Yukarıdaki kod parçası pek sağlam sayılmaz çünkü hiç hata denetimi yok. Gördüğümüz gibi `inet_addr()` hatalı bir durumla karşılaşınca `-1` değerini döndürür. İkili sayı sistemini hatırladınız mı? (`unsigned`) `-1` tam da şu IP adresine karşılık gelir: `255.255.255.255`! Bu da yayın adresidir! Aman dikkat. Hata kontrolünü düzgün bir şekilde yapmayı sakın ihmal etmeyin.

Aslında `inet_addr()` yerine kullanabileceğiniz daha güzel bir işlev var: `inet_aton()` ("aton"u, "ascii to network" olarak okuyun):

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int inet_aton(const char *cp, struct in_addr *inp);
```

Bir örnek vermek gerekirse: bu işlev `struct sockaddr_in` yapısını paketlerken (ilerledikçe bu örnek size daha da anlamlı gelecek `bind()` (sayfa: 13) ve `connect()` (sayfa: 15) kısımlarına kadar sabredin.)

```
struct sockaddr_in my_addr;

my_addr.sin_family = AF_INET;           // konak bayt sıralaması
my_addr.sin_port = htons(MYPORT);      // short, ağ bayt sıralaması
inet_aton("10.12.110.57", &(my_addr.sin_addr));
```

```
memset(&(my_addr.sin_zero), '\0', 8); // geriye kalanı sıfırla
```

`inet_aton()`, diğer tüm soket-bağlantılı işlemlerden farklı olarak, sorun çıkmazsa sıfırdan farklı bir değer, sorun çıkarsa da sıfır değerini döndürür. Ve döndürülen adres *inp* içinde yer alır.

Maalesef her platform `inet_aton()` işlevini kullanmamaktadır. Bu yüzden her ne kadar bu işlevi tercih etsek de daha yaygın olması itibarıyla örneklerde `inet_addr()` kullanılacaktır.

Artık IP adreslerini ikilik sisteme kolayca dönüştürebileceğinize göre bunun tersini yapmaya ne dersiniz? Yani mesela elinizde zaten bir `struct in_addr` yapısı varsa ve siz bunu alışık olduğunuz sayılı noktalı IP adresi biçiminde basmak istiyorsanız? Bu durumda kullanacağımız işlev: `inet_ntoa()` ("ntoa"yı "network to ascii" olarak okuyun). Şu şekilde işinizi görür:

```
printf("%s", inet_ntoa(ina.sin_addr));
```

Bu IP adresini basacaktır. Dikkat edin: `inet_ntoa()` argüman olarak `struct in_addr` alır `long` almaz. Bir diğer önemli nokta da: Bu işlev bir `char`'a işaret eden bir gösterge döndürür. Söz konusu gösterge `inet_ntoa()` içinde statik olarak depolanan bir karakter dizisine işaret eder ve `inet_ntoa()` işlevini her çağırışınızda son işlem görmüş olan IP adresinin üzerine yazılır. Örneğin:

```
char *a1, *a2;
.
.
a1 = inet_ntoa(ina1.sin_addr); // burada 192.168.4.14 var
a2 = inet_ntoa(ina2.sin_addr); // burada 10.12.110.57 var
printf("1. adres: %s\n", a1);
printf("2. adres: %s\n", a2);
```

şunu basar:

```
1. adres: 10.12.110.57
2. adres: 10.12.110.57
```

Eğer birden fazla adresle iş yapıyorsanız ve bunları yukarıda olduğu gibi kaybetmek istemiyorsanız o zaman `strcpy()` gibi bir işlev kullanarak uygun bir yere kopyalamayı sakın ihmal etmeyin.

Bu konu ile söyleyeceklerim şimdilik bu kadar. Daha sonra "whitehouse.gov" gibi bir karakter dizisini hangi yöntemlerle karşılık gelen IP adresine çevirebileceğinizi göstereceğim (bkz. [DNS — Sen "whitehouse.gov" de, ben de "198.137.240.92" diyeyim](#) (sayfa: 20)).

4. Sistem Çağrıları veya Felaketleri

Bir UNIX bilgisayardaki ağ işlevselliğine erişmenizi anlatacağım bölüme hoşgeldiniz. Bu işlemlerden birini çağırıldınızda işletim sistemi çekirdeği devreye girer ve düşük seviyedeki işlemleri büyüleyici bir şekilde sizin için halleder.

İnsanların bu aşamada en çok takıldıkları nokta bu işlevleri hangi sıra ile çağıracakları sorusudur. Bu bakımdan **man** sayfaları bir işe yaramaz, eğer biraz uğraştıysanız ne demek istediğimi biliyorsunuzdur. Bu zorlu konu ile başa çıkabilmek için işlevleri normalde geliştirdiğiniz bir programdaki çağrılış sıralarına tam olarak (hemen hemen) uygun şekilde size sunmaya çalışacağım.

Bu açıklamalara ek olarak orada burada birkaç örnek kod parçası, biraz süt artı kurabiye (üzgünüm bu son ikisini siz tedarik etmelisiniz) ve tabii biraz da cesaret ile elinizdeki verileri Internet üzerinden her yere ışınıyor olacaksınız.

4.1. `socket()` — Al Şu Dosya Tanımlayıcıyı!

Sanırım artık daha fazla erteleyemem — `socket()` sistem çağrısından bahsetmek zorundayım. İşte o işlev:

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

Peki ya argümanlar? Önce, *domain* argümanına `AF_INET` değeri verilmeli, tıpkı `struct sockaddr_in` yapısında olduğu gibi. Sonra *type* argümanı çekirdeğe ne tür bir soket söz konusu olduğunu söyler: `SOCK_STREAM` veya `SOCK_DGRAM`. Son olarak da *protocol* argümanına "0" değerini verelim ki `socket()`, *type* değişkenine karşılık gelen uygun protokolü seçebilsin. (Dikkat: Bahsettiğimden daha çok *domain* ve *type* vardır. Bkz. `socket()` man sayfası. Ayrıca *protocol* değerini öğrenmenin "daha iyi" bir yöntemi vardır. Bkz. `getprotobyname()` man sayfası.)

`socket()` işlevi size bir soket tanımlayıcı döndürür ve artık siz bunu daha sonraki işlevlerinize parametre olarak geçebilirsiniz. Eğer bir hata oluşursa işlev `-1` değerini döndürür. Bu durumda *errno* isimli global değişken hata kodunu tutar (bkz. `perror()` man sayfası.)

Bazı belgelerde mistik bir `PF_INET` ifadesi görebilirsiniz, korkmayın. Normalde bu canavar günlük hayatta pek karşınıza çıkmaz fakat gene de kendisinden biraz bahsedeyim. Uzun zaman önce adres ailesinin (`AF_INET`'deki "AF" "Address Family" manasındadır) pek çok protokolü destekleyebileceği düşünülmüştü (`PF_INET`'deki "PF" "Protocol Family" manasındadır). Ancak böyle bir şey olmadı. Yani doğru olan, `AF_INET` değerini `struct sockaddr_in` yapısında kullanmanız ve `PF_INET`'i ise `socket()` çağırırken kullanmanızdır. Ancak pratik olarak `AF_INET`'i her yerde kullanabilirsiniz. Bu işin üstadlarından W. Richard Stevens kitabında böyle yaptığı için ben de burada böyle yapacağım.

Peki tamam çok güzel de bu soket ne işe yarar? Tek başına bir işe yaramaz tabii, lütfen okumaya devam edin ve diğer sistem çağrılarını öğrenin ki taşlar yerine otursun.

4.2. `bind()` — Hangi Port Üzerindeyim?

Bir soket edindikten sonra bunu makinanızdaki bir "port" ile ilişkilendirmek isteyeceksiniz⁽³⁾. Bu port numarası dediğimiz şey işletim sistemi çekirdeği tarafından gelen bir paketi belli bir sürecin soket tanımlayıcısı ile ilişkilendirebilmesi için gereklidir. Eğer tek yapacağınız bir yere `connect()` ile bağlanmaksa o zaman buna gerek yoktur tabii. Gene de okumaya devam edin, zevk alacaksınız.

`bind()` sistem çağrısının özetine man komutu ile bakacak olursanız şöyle bir şeyle karşılaşacaksınız:

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

sockfd denilen şey `socket()` tarafından döndürülen soket dosya tanımlayıcısıdır. *my_addr* değişkeni `struct sockaddr` türünde bir veriye işaret eder ve bu yapı da adresinizi yani port numaranızı ve IP adresinizi barındırır. *addrlen*'in değeri `sizeof(struct sockaddr)` olarak verilebilir.

Vay canına. Bir seferde sindirmek için biraz fazla değil mi? Bir örnek yapalım:

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define MYPORT 3490
```

```
main()
{
    int sockfd;
    struct sockaddr_in my_addr;

    sockfd = socket(AF_INET, SOCK_STREAM, 0); // hata kontrolünü ihmal etmeyin!

    my_addr.sin_family = AF_INET;           // konak bayt sıralaması
    my_addr.sin_port = htons(MYPORT);      // short, ağ bayt sıralaması
    my_addr.sin_addr.s_addr = inet_addr("10.12.110.57");
    memset(&(my_addr.sin_zero), '\0', 8); // yapının geriye kalanını sıfırlayalım

    // bind() için hata kontrolü yapmayı unutmayın:
    bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));
    .
    .
    .
}
```

Burada dikkat etmeniz gereken bazı şeyler var: *my_addr.sin_port* Ağ Byte Sıralamasında, ve tabii *my_addr.sin_addr.s_addr* de öyle. Bir başka nokta ise en tepede çağrılmış olan başlık dosyaları sistemden sisteme değişiklik gösterebilir. Kesin bilgi sahibi olabilmek için sisteminizdeki **man** sayfalarına bakmalısınız.

`bind()` konusu ile ilgili son söyleyeceğim şey IP ve port edinme işlemini biraz otomatikleştirilebilir:

```
my_addr.sin_port = 0; // kullanılmayan herhangi bir port'u seç
my_addr.sin_addr.s_addr = INADDR_ANY; // IP adresimi kullan
```

Gördüğünüz gibi *my_addr.sin_port* değişkeninin değerini sıfır yaparak `bind()` işlemine diyoruz ki "uygun olan bir port sayısını bizim için sen seç". Benzer şekilde *my_addr.sin_addr.s_addr* değişkeninin değerini `INADDR_ANY` yaparak üzerinde çalıştığı makinanın IP adresini almasını söylemiş oluyoruz.

Eğer dikkatli bir okuyucu iseniz bazı şeyleri fark etmiş olabilirsiniz, mesela `INADDR_ANY` değerini Ağ Bayt Sıralamasına dönüştürmedim! Ne kadar da yaramaz bir programcıyım! Ama bildiğim bir şey var: `INADDR_ANY` zaten SIFIR değerine karşılık geliyor! Yani hangi sırada dizerseniz dizin zaten sıfır. Ancak takıntılı olan programcılar `INADDR_ANY` sabitinin mesela 12 olarak belirlendiği bir paralel evrenden söz açabilirler ve orada benim kodum çalışmaz. Tamam, sorun değil, hallederiz:

```
my_addr.sin_port = htons(0); // kullanılmayan herhangi bir port'u seç
my_addr.sin_addr.s_addr = htonl(INADDR_ANY); // IP adresimi kullan
```

Sistemimiz artık o kadar taşınabilir oldu ki yani bu kadar olur. Bunu burada belirttim çünkü karşılaşacağınız kod örneklerinin çoğu `INADDR_ANY` sabitini `htonl()` işlevinden geçirmez, kafanız karışmasın.

`bind()` eğer bir hata çıkarsa `-1` değerini döndürür ve *errno* isimli hata kodu değişkenine gerekli sayıyı yerleştir.

`bind()` işlevini çağırırken dikkat etmeniz gereken bir başka şey de şudur: port numarası olarak küçük bir değer seçmeyin. 1024'ün altındaki tüm portlar REZERVE edilmiştir (eğer superuser değilseniz!). Bu sayıdan başlayarak 65535'e kadar olan sayılardan birini port numarası olarak kullanabilirsiniz (tabii eğer seçtiğiniz numara başka bir program tarafından kullanılmıyorsa).

Bazen bir sunucuyu tekrar çalıştırmaya kalktığınızda `bind()` işlevinin başarısız olduğunu ve "Adres kullanımda" ("Address already in use.") mesajı verdiğiniz görürsünüz. Bu ne anlama gelir? Daha önce kullanılmış bir soket hala çekirdek seviyesinde takılı kalmıştır ve bu yüzden portun kullanılmasını engellemektedir. Ya kendiliğinden iptal olmasını beklersiniz ki bu 1 dakika kadar sürebilir ya da programınıza bu portu her halükârda kullanmasını sağlayacak kodu eklersiniz:

```
int yes=1;
//char yes='1'; // Solaris programcılarını bunu kullanır
```

```
// "Address already in use" hata mesajından kurtul
if (setsockopt(listener, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1) {
    perror("setsockopt");
    exit(1);
}
```



bind() ile ilgili ek bilgi

Kimi durumlarda bu işlev ile hiç işiniz olmaz. Mesela eğer `connect()` ile uzaktaki bir makinaya bağlanıyor ve yerel portunuzun ne olduğu ile ilgilenmiyorsanız (**telnet** uygulamasında olduğu gibi önemli olan sadece uzaktaki makinadaki port ise) kısaca `connect()` işlevini çağırırsınız ve zaten bu işlev de socketin bağlı olup olmadığını kontrol eder ve eğer socket bağlı değil ise `bind()` işlevini kullanarak bunu makinanızdaki kullanılmayan bir port numarasına bağlar.

4.3. `connect()` —Hey, sen!

Diyelim ki bir telnet programısınız. Sizi kullanan kişi (tıpkı *TRON* filminde olduğu gibi) size bir socket dosya tanımlayıcısı edinmeniz için komut veriyor. Siz de bu komuta uyup `socket()` işlevini çağırıyorsunuz. Sonra, kullanıcı sizden "10.12.110.57" adresine ve "23" numaralı porta bağlanmanızı istiyor (standart telnet portu). Aha! Şimdi ne yapacaksınız?

Bir program olarak şanslısınız çünkü tam da `connect()` bölümündeyiz — uzaktaki bir bilgisayara nasıl bağlanırsınız bölümü. Okumaya devam kaybedecek zaman yok, kullanıcı bekliyor!

`connect()` işlevi şöyle birşeydir:

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

sockfd dediğimiz değişken `socket()` sistem çağırısı tarafından döndürülmüş olan socket dosya tanımlayıcısının değerini tutar. *serv_addr* ise `struct sockaddr` türünde bir değişkendir ve hedef port ile IP adres bilgilerini barındırır. *addrlen* değişkeni de `sizeof(struct sockaddr)` değerini alırsa iyi olur.

Taşlar yerine oturmaya başladı mı? Hemen bir örneğe göz atalım:

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define DEST_IP    "10.12.110.57"
#define DEST_PORT  23

main()
{
    int sockfd;
    struct sockaddr_in dest_addr;           // hedef adres

    sockfd = socket(AF_INET, SOCK_STREAM, 0); // hata denetimi yapın!

    dest_addr.sin_family = AF_INET;        // konak bayt sıralaması
    dest_addr.sin_port = htons(DEST_PORT); // short, ağ bayt sıralaması
    dest_addr.sin_addr.s_addr = inet_addr(DEST_IP);
```

```
memset(&(dest_addr.sin_zero), '\0', 8); // yapının geriye kalanını sıfırla

// connect() işlevini çağırdıktan sonra hata denetimi yap!
connect(sockfd, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr));
.
.
.
```

Sıkılsanız da söylemek zorundayım: `connect()` işlevinden sonra mutlaka hata denetimi yapın, eğer hata oluştu ise bu işlev `-1` değerini döndürür ve `errno` değişkenine ilgili hata numarasını yerleştirir.

Ayrıca `bind()` işlevini çağırmadığımızda da dikkat edin. Yani kısaca yerel port numarası ile ilgilenmiyoruz; bizi ilgilendiren hedef bilgisayar (uzaktaki port). İşletim sistemi çekirdeği bizim için bir yerel port bulacaktır ve bağılandırdığımız bilgisayar da bu bilgiyi otomatik olarak bizden öğrenecektir. Siz tatlı canınızı üzmeyin.

4.4. `listen()` — Biri Beni Arayabilir mi Acaba?

Pekala hızımızı biraz değiştirelim. Ya bir yere bağlanmak istemiyorsanız ne olacak? Mesela sırf zevk olsun diye size gelen bağlantıları dinlemek ve bunlarla ilgili gerekeni yapmak istediğinizi var sayalım. Bu tür bir süreç iki aşamalıdır, önce `listen()` ile dinler sonra da `accept()` ile gelen çağrılarını kabul edersiniz.

Dinleme işlevi oldukça basittir ancak gene de biraz açıklama yapmak gerekiyor:

```
int listen(int sockfd, int backlog);
```

`sockfd` bin kere söylediğim gibi `socket()` sistem çağrısı tarafından döndürülmüş olan soket dosya tanımlayıcısı oluyor. `backlog` ise gelen çağrı kuyruğunda izin verilen bağlantı sayısını gösteriyor. Bu ne mi demek? Şey, yani gelen bağlantı talepleri siz onları `accept()` ile kabul edene dek bir kuyrukta bekler demek ve işte bu kuyruğun ne kadar uzun olacağını başka bir deyişle sınırını siz belirlersiniz. Pek çok sistem bu sınırı 20 olarak belirler; siz ise mesela 5 veya 10 gibi bir değer kullanabilirsiniz.

Hemen her zamanki gibi, `listen()` işlevi hata durumunda `-1` değerini döndürür ve tabii ki `errno` değişkenine de ilgili hata numarasını yazar.

Evet, tahmin edebileceğiniz gibi `listen()` işlevinden önce `bind()` işlevini çağırmalıyız yoksa işletim sistemi çekirdeği bu dinleme işlemini gelişigüzel bir port üzerinden yapmaya başlar. Eğer gelen bağlantıları dinleyecek-seniz çağırmanız gereken işlevler sırası ile şöyledir:

```
socket();
bind();
listen();
/* accept() buraya gelecek */
```

Burada çok açıklama ve kod yok çünkü zaten kendi kendini açıklıyor. (`accept()` bölümünde ele alacağımız kod tabii ki daha ayrıntılı olacak.) Bütün bu prosedürdeki en dikkatli olunması gereken nokta `accept()` işlevinin çağrıldığı yer.

4.5. `accept()` — "3490 Numaralı Portu Aradığınız İçin Teşekkürler"

Sıkı durun — `accept()` işlevi biraz gariptir! Neler oluyor? Şu oluyor: çok çok uzaklardan birileri `connect()` işlevi ile sizin makinanızda sizin `listen()` ile dinlemekte olduğunuz bir porta bağlanmaya çalışıyor. Bu bağlantı talebi `accept()` ile kabul edilene dek kuyrukta bekleyecektir. Siz `accept()` işlevini çağırırsınız ve ona beklemekte olan çağrıyı kabul etmesini söylersiniz. O da size yepyeni bir soket dosya tanımlayıcısı döndürür sadece ve sadece söz konusu bağlantıya özel. Evet doğru duydunuz birdenbire elinizin altında iki soket dosya

tanımlayıcısı oldu! Orjinal olanı halen port üzerinden dinleme işlemini gerçekleştirmek için kullanılıyor yeni olarak yaratılmış olan ise `send()` ve `recv()` işlevlerinde kullanılmak üzere emrinize amade. Hele şükür!

İşlevi şu şekilde çağırırsınız:

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

int accept(int sockfd, void *addr, int *addrlen);
```

`sockfd` dediğimiz `listen()` ile dinlediğiniz soket tanımlayıcıdır. Basit. `addr` yerel olarak kullandığınız `struct sockaddr_in` yapısını gösteren bir göstergedir. Gelen bağlantılarla ilgili bilgiler burada barındırılacak (yani bu yapıyı kullanarak gelen bağlantının hangi bilgisayar ve hangi porttan geldiğini öğrenebilirsiniz). `addrlen` yerel bir tamsayı değişkendir ve kullanılmadan önce `accept()` `sizeof(struct sockaddr_in)` değerini almalıdır. `accept()` işlevi bu değerden daha fazla sayıda baytı `addr` içine yerleştirmeyecektir. Eğer söz konusu değerden daha azını yerleştirirse o zaman da `addrlen` değerini, bunu yansıtacak şekilde değiştirecektir.

Tahmin edin ne diyeceğim? `accept()` hata durumunda `-1` değerini döndürür ve `errno` değişkenine ilgili hata kodunu yerleştirir. Tahmin etmiş miydiniz? Gerçekten?

Biliyorum, bir seferde sindirmesi kolay değil. İsterseniz aşağıdaki örnek koda bakalım:

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define MYPOR 3490 // kullanıcıların bağlanacağı port
#define BACKLOG 10 // kuyrukta bekleyecek bağlantı sayısı

main()
{
    int sockfd, new_fd; // sock_fd ile dinle, new_fd yeni bağlantı için
    struct sockaddr_in my_addr; // benim adres bilgim
    struct sockaddr_in their_addr; // bağlananın adres bilgisi
    int sin_size;

    sockfd = socket(AF_INET, SOCK_STREAM, 0); // hata denetimi yap!

    my_addr.sin_family = AF_INET; // konak bayt sıralaması
    my_addr.sin_port = htons(MYPOR); // short, ağ bayt sıralaması
    my_addr.sin_addr.s_addr = INADDR_ANY; // otomatik olarak benim IP'im
    memset(&(my_addr.sin_zero), '\0', 8); // geriye kalanı sıfırla

    // hata denetimi yapmayı sakın unutmayın:
    bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));

    listen(sockfd, BACKLOG);

    sin_size = sizeof(struct sockaddr_in);
    new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
    .
    .
    .
```

Lütfen dikkat edin: `new_fd` şeklindeki soket tanımlayıcıyı tüm `send()` ve `recv()` işlevleri için kullanacağız. Eğer sadece ve sadece tek bir bağlantıyı kabul ederseniz `close()` ile `sockfd` üzerindeki dinleyici soketi kapatabilirsiniz böylece aynı port üzerinden başka bağlantı yapamaz, eğer istediğiniz gerçekten bu ise.

4.6. `send()` ve `recv()` — Konuş Benimle Bebeğim!

Bu iki işlev veri akış soketler veya bağlantılı veri paketi soketleri üzerinden veri göndermenizi ve veri almanızı sağlar. Eğer istediğiniz düzenli ve bağlantısız veri paketi soketleri kullanmak ise o zaman `sendto()` ve `recvfrom()` (sayfa: 18) işlevleri ile ilgili aşağıdaki bölümü okumalısınız.

`send()` işlevi şu şekilde çağrılır:

```
int send(int sockfd, const void *msg, int len, int flags);
```

`sockfd` üzerinden veri göndereceğiniz sokettir (size `socket()` veya `accept()` tarafından sağlanmış olabilir). `msg` göndermek istediğiniz mesajı gösteren bir göstergedir ve `len` değişkeni bu verinin byte cinsinden uzunluğudur. `flags` parametresini 0 olarak bırakabilirsiniz. (Bu parametre ile ilgili ayrıntılı bilgi için bkz. `send()` man sayfası.)

Örnek bir kod parçası vermek gerekirse:

```
char *msg = "Beej buradaydı!";
int len, bytes_sent;
.
.
len = strlen(msg);
bytes_sent = send(sockfd, msg, len, 0);
.
.
.
```

`send()` değer olarak gönderilen bayt miktarını döndürür — bu sizin gönderilmesini istediğiniz miktardan az olabilir! Gördüğünüz gibi siz ona bir yığın veri göndermesini söylersiniz ancak o bazen bunun tamamı ile başa çıkamayabilir. Elinden geldiği kadarını gönderir ve geriye kalan veriyi yeniden göndermek sizin sorumluluğunuzdadır. Unutmayın, eğer `send()` işlevinin döndürdüğü değer `len` değişkenindeki değer kadar değilse göndermek istediğiniz verinin geriye kalanını göndermek sizin işinizdir. İyi haberlere gelince: Eğer paket küçükse (1k civarı) bu işlev büyük ihtimalle tüm veriyi bir seferde gönderebilecektir. Her zamanki gibi hata durumunda `-1` değerini döndürür ve `errno` küresel değişkenine hata kodunu yazar.

`recv()` işlevi da pek çok bakımdan yukarıdaki işleve benzer:

```
int recv(int sockfd, void *buf, int len, unsigned int flags);
```

`sockfd` üzerinden okuma işlemini gerçekleştireceğiniz sokettir, `buf` okunan verinin yazılacağı bellek bölgesinin başlangıç adresini gösteren göstergedir, `len` ise verinin yazılacağı tamponun (buffer) azami boyudur ve `flags` yine 0 değerini alabilir (Ayrıntılı bilgi için `recv()` man sayfasına bakınız).

`recv()` okunduktan sonra tampona yazılan bayt miktarını döndürür ya da eğer bir hata oluştu ise `-1` değerini döndürüp `errno` değişkeninin değeri belirler.

Bir dakika! `recv()` 0 değerini döndürebilir! Bunun tek bir anlamı vardır: karşı taraf bağlantıyı kesmiş! Döndürülen değerin 0 olması `recv()` işlevinin size "karşı taraf bağlantıyı kesti" mesajını vermesi demektir.

Bütün bunları anlamak kolaydı değil mi? Artık verilerinizi soketler üzerinden yollayıp alabilirsiniz! Vay be! Artık siz bir Unix Ağ Programlama Uzmanısınız!

4.7. `sendto()` ve `recvfrom()` — Benimle UDP'ce Konuş

"Buraya dek çok güzeldi," dediğinizi duyar gibiyim, "fakat bu bilgileri bağlantısız veri paketi soketleri üzerinde nasıl kullanabilirim ki?". No problemo, amigo. İşte derdine çare.

UDP soketleri uzaktaki bir konağa bağlı olmadığından kılavuz paketi yollamadan önce hangi bilgiyi vermemiz gerekiyor? Doğru tahmin! Hedef adres! Kısaca:

```
int sendto(int sockfd, const void *msg, int len, unsigned int flags,
           const struct sockaddr *to, int tolen);
```

`send()` işlevine ne kadar da benziyor değil mi? Tek farkı fazladan iki bilgi parçası var. *to* dediğimiz `struct sockaddr` türünde bir değişkeni gösteren işaretçidir (normalde `struct sockaddr_in` türündedir ve siz son anda gerekli tür dönüşümünü yaparsınız) ve hedef IP adresi ile port numarasını barındırır. *tolen* değişkeni `sizeof(struct sockaddr)` değerini almalıdır.

Tıpkı `send()` gibi, `sendto()` işlevi de gönderilen bayt miktarını döndürür (bu beklediğinizden az olabilir tabii ki!) ve eğer hata oldu ise `-1` değerini döndürür.

Tahmin edebileceğiniz gibi `recv()` ve `recvfrom()` işlevleri birbirlerine çok benzerdir. Kısaca `recvfrom()` işlevine bakarsak:

```
int recvfrom(int sockfd, void *buf, int len, unsigned int flags,
             struct sockaddr *from, int *fromlen);
```

Benzer olduğunu söylemiştim yani: `recv()` işlevinde olduğu gibi, sadece birkaç ek değişkene ihtiyacı var. *from* yerel bir `struct sockaddr` türünde değişkenin adresini gösteren göstergedir ki bu değişken de mesajın geldiği ilgili makinanın IP adresini ve port numarasını barındıracaktır. *fromlen* yerel ve `int` türünde bir göstergedir ve söz konusu değişkenin alması gereken ilk değer `sizeof(struct sockaddr)`'dir. İşlev, çalışıp bir değer döndürünce *fromlen* değişkeni *from* değişkenindeki adresin boyunu depoluyor olacaktır.

`recvfrom()` işlevi okuduğu bayt sayısını veya bir hata oluşması durumunda `-1` değerini döndürür (ve *errno* değişkenine uygun hata kodunu yerleştirir).

Unutmayın, eğer `connect()` ile bir UDP soketine bağlantı kurarsanız `send()` ve `recv()` işlevlerini kullanmanızda bir sakınca yoktur. Soketin kendisi hala bir bağlantısız veri paketi soketidir ve gidip gelen paketler de hala UDP protokolünü kullanır. Fakat soket arayüzü otomatik olarak sizin yerinize gerekli hedef ve kaynak bilgisini pakete ekler.

4.8. `close()` ve `shutdown()` — Düş Yakamdan!

Vay be! Deminden beri `send()` ile veri gönderip `recv()` ile de veri okuyorsunuz ve artık yorulduunuz. Soket tanımlayıcınız ile ilişkilendirilmiş olan bağlantıyı kesmenin zamanıdır. Kolayı var. Alışık olduğunuz Unix dosya tanımlayıcı kapatma işlevi olan `close()` işlevini kullanın:

```
close(sockfd);
```

Böylece artık bu sokete ne yazılabilir ne de buradan veri okunabilir. Diğer uçta bunları yapmaya çalışan kişi artık bir hata mesajı ile karşılaşacaktır.

Eğer soket kapatma işlemi üzerinde biraz daha deentim sahibi olmak isterseniz o zaman `shutdown()` işlevini tercih edebilirsiniz. Bu işlevi kullanarak iletişim kanalını tek yönlü ya da çift yönlü olarak kapatabilirsiniz (`close()` işlevi iki taraflı olarak keser). İşlev şöyledir:

```
int shutdown(int sockfd, int how);
```

sockfd kapatmak istediğiniz soket dosya tanımlayıcısıdır ve *how* değişkeni de şu değerlerden birini alabilir:

- 0 — Bundan sonraki okumalara izin verme
- 1 — Bundan sonraki göndermelere (yazmalara) izin verme
- 2 — Bundan sonraki göndermelere ve okumalara izin verme (`close()` işlevinin yaptığı gibi)

`shutdown()` başarılı olarak görevini tamamlarsa `0` döndürür ama eğer bir hata ile karşılaşrsa `-1` döndürür (ve `errno` değişkenine hata kodunu yazar).

Eğer `shutdown()` işlevini bağlantısız veri paketi soketleri üzerinde kullanırsanız bu soketler artık `send()` ve `recv()` işlevleri tarafından kullanılamaz hale gelirler (bunları, `connect()` ile bağlanmak istediğinizde kullanabileceğinizi unutmayın).

Bir başka önemli nokta da: `shutdown()` aslında dosya tanımlayıcısını kapatmaz sadece kullanılabilirliğini değiştirir. Soket tanımlayıcısını gerçekten iptal etmek istiyorsanız `close()` kullanmalısınız.

Yapacak birşey yok.

4.9. `getpeername()` — Kimsiniz?

Bu işlev o kadar kolay ki!

Yani gerçekten o kadar kolay ki ayrı bir bölümü hak ediyor mu bilemiyorum. Neyse gene de yazdım işte.

`getpeername()` bağlantılı veri akış soketinin diğer tarafında kim olduğunu söyler:

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

int getpeername(int sockfd, struct sockaddr *addr, int *addrlen);
```

`sockfd` bağlantılı veri akış soketinin tanımlayıcısıdır, `addr`, `struct sockaddr` (veya `struct sockaddr_in`) türündeki bir göstergedir ki bu da iletişimin diğer ucundaki konak ile ilgili bilgileri tutar. `addrlen`, `int` türündeki göstergedir, alması gereken ilk değer ise `sizeof(struct sockaddr)` olarak verilir.

Bu işlev hata durumunda `-1` değerini döndürür ve `errno` değişkenine gerekli değeri yazar.

Bir kere karşı tarafın adres bilgisine eriştikten sonra `inet_ntoa()` veya `gethostbyaddr()` işlevleri ile daha fazla bilgi edinebilirsiniz. Hayır, onların login ismini öğrenemezsiniz. (Tamam, tamam. Eğer diğer bilgisayar `identd` sürecini çalıştırıyor ise bu mümkündür. Ancak bu konumuz dışında⁽⁴⁾)

4.10. `gethostname()` — Ben kimim?

`getpeername()` işlevinden daha kolay bir işlev varsa o da `gethostname()` işlevidir. Programınızın üzerinde çalıştığı konağın ismini döndürür. Bu isim daha sonra `gethostname()` tarafından makinanızın IP adresini tespit etmek için kullanılabilir.

Bundan daha eğlenceli bir şey olabilir mi? Aslında aklıma geliyor ama soket programlama ile ilgili değil. Neyse devam edelim:

```
#include <unistd.h>

int gethostname(char *hostname, size_t size);
```

Argümanlar gayet basit: `hostname` işlev çağrıldıktan sonra bilgisayarın ismini barındıracak karakter dizisinin göstergesidir ve `size` değişkeni de `hostname` dizisinin bayt cinsinden uzunluğudur.

İşler yolunda giderse, işlev `0` değerini döndürür ve hata oluşursa da `-1` değerini döndürüp `errno` değişkenini gerekli şekilde ayarlar.

4.11. DNS — Sen "whitehouse.gov" de, ben de "198.137.240.92" diyeyim

Eğer DNS'in ne olduğunu bilmiyorsanız söyleyeyim: "Domain Name System" yani "Alan Adı Sistemi" demek. Kısaca siz ona kolayca hatırlayabildiğiniz adresi verirsiniz o da size buna karşılık gelen IP adresini verir (ki siz de bu bilgiyi `bind()`, `connect()`, `sendto()` ve diğer işlevleri çağırırken kullanabilirsiniz). Böylece biri:

```
$ telnet whitehouse.gov
```

komutunu girdiğinde **telnet** yazılımı `connect()` işlevine verilmesi gereken "198.137.240.92" bilgisine erişebilir.

Peki bu nasıl çalışır? Bunun için `gethostbyname()` işlevini kullanacaksınız:

```
#include <netdb.h>

struct hostent *gethostbyname(const char *name);
```

Gördüğümüz gibi bu işlev `struct hostent` türünde bir gösterge döndürür. Söz konusu türün ayrıntılı yapısı da şöyledir:

```
struct hostent {
    char    *h_name;
    char    **h_aliases;
    int     h_addrtype;
    int     h_length;
    char    **h_addr_list;
};
#define h_addr h_addr_list[0]
```

Bu karmaşık değişkenin içindeki alanların açıklamalarına gelince:

- *h_name* — Konağın resmi ismi.
- *h_aliases* — Söz konusu konağın alternatif isimleri, NULL ile sonlandırılmış karakter dizileri şeklinde.
- *h_addrtype* — Dönen adresin türü, genellikle `AF_INET` değerini alır.
- *h_length* — Byte cinsinden adresin uzunluğu.
- *h_addr_list* — Konağın ağ adresinin sıfır sonlandırılmalı dizisi. Konak adresleri ağ bayt sıralamasına sahiptir.
- *h_addr* — *h_addr_list* listesindeki ilk adres.

`gethostbyname()` çalıştıktan sonra içini doldurduğu `struct hostent` türünden bir gösterge döndürür. Eğer hata oluşursa NULL döndürür. (Fakat *errno* değişkeni ile ilgili herhangi bir işlem yapmaz — bunun yerine *h_errno* değişkeni kullanılır. Ayrıntılı bilgi için aşağıdaki `herror()` işlevine bakın.)

Peki tüm bu bilgileri nasıl kullanacağız? Bazen okuyucuya bilgi yığını verip onu bununla başbaşa bırakmak yeterli olmaz (bilgisayar belgelerini okuyanlar ne demek istediğimi anlıyorlardır). Bu işlevi kullanmak düşündüğünüzden daha kolaydır.

İşte örnek bir program^(B17):

```
/*
** getip.c -- konak isminden IP adresini elde edilmesi
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```

#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    struct hostent *h;

    if (argc != 2) { // komut satırında hata denetimi
        fprintf(stderr, "usage: getip konak_ismi\n");
        exit(1);
    }

    if ((h=gethostbyname(argv[1])) == NULL) { // konak bilgilerini al
        perror("gethostbyname");
        exit(1);
    }

    printf("Konak ismi: %s\n", h->h_name);
    printf(" IP Adresi: %s\n", inet_ntoa(*(struct in_addr *)h->h_addr));

    return 0;
}

```

`gethostbyname()` söz konusu olduğunda `perror()` işlevini hata mesajını basmak için kullanamazsınız (çünkü *errno* gerekli değeri almaz). Bunun yerine `perror()` işlevini kullanmalısınız.

Açıklama basit: Doğrudan ilgilendiğiniz makinanın Internet adresini verirsiniz ("whitehouse.gov") ve `gethostbyname()` işlevi de çalışıp işini bitirdikten sonra size `struct hostent` türünde bir değişken döndürür.

Buradaki tek gariplik şudur: IP adresini basarken kullanmanız gereken `h->h_addr`, `char*` türündedir ama `inet_ntoa()` işlevi `struct in_addr` türünde bir değişken isteme konusunda ısrarcı olduğu için önce `h->h_addr` değişkenini `struct in_addr*` türüne çevirdim ve ardından veriye ulaştım.

5. İstemci–Sunucu Mimarisi

İstemci–sunucu dünyasında yaşıyoruz, bunu kabul et dostum. Ağ ortamındaki hemen her süreç ya da uygulama öyle ya da böyle sunucu süreçleriyle konuşup aldıkları cevaplara göre iş yapıyor ya da tersi oluyor. Mesela **telnet** programını ele alalım. Siz uzaktaki bir konağın 23 numaralı portuna **telnet** (istemci) ile bağlandığınızda o konakta da **telnetd** (sunucu) isimli bir program soluk alıp vermeye başlar. Bu program gelen **telnet** bağlantılarını dinler, size bir "login:" istemi gönderir, vs. vs.

Şekil 2. Sunucu–İstemci Etkileşimi



Sunucu–İstemci Etkileşimi

İstemci ile sunucu arasındaki bilgi alışverişi *Sunucu–İstemci Etkileşimi* (sayfa: 22)'de özetlenmiştir.

Dikkat etmeniz gereken noktalardan biri de şudur: istemci–sunucu çifti birbirleri ile `SOCK_STREAM`, `SOCK_DGRAM` veya başka bir tarzda konuşuyor olabilirler (yeter ki aynı dili konuşsunlar). İstemci–sunucu çift-

lerine birkaç meşhur örnek vermek gerekirse: **telnet/telnetd**, **ftp/ftpd** veya **bootp/bootpd**. Yani ne zaman bir **ftp** programı çalıştırsanız diğer tarafta **ftpd** gibi size hizmeti sunan bir süreç vardır.

Genellikle sunucu makinada ilgili hizmet için tek bir sunucu program çalışır ve bu program kendisine bağlanan birden fazla istemciye `fork()` işlevi aracılığı ile hizmet eder. Süreci özetlemek gerekirse: Sunucu bir bağlantı isteği için bekleyecek, `accept()` işlevi ile bunu kabul edecek ve `fork()` işlevi ile bir çocuk süreç yaratıp veri alışverişini bu sürece devredecek. Bir sonraki bölümde inceleyeceğimiz örnek sunucu yazılımının yaptığı iş tam da yukarıda anlatıldığı gibidir.

5.1. Basit Bir Veri Akış Sunucusu

Bu sunucunun yaptığı tek bir iş var: "Hello, World!\n" dizgesiniz bir veri akış bağlantısı üzerinden karşı tarafa yollamak. Sunucunun düzgün çalışıp çalışmadığını test etmek için tek yapmanız gereken derledikten sonra bir pencerede çalıştırmak ve ardından başka bir terminal penceresi açıp **telnet** yazılımı ile sunucuya şu şekilde erişmek:

```
$ telnet remotehostname 3490
```

Burada `remotehostname` sunucuyu üzerinde çalıştırdığınız makinanızın adıdır⁽⁵⁾.

Sunucu yazılımının C dilindeki kaynak kodu aşağıdaki gibidir:^(B19) (Bilgi: Bir satırın sonundaki \ satırın alt satırda devam ettiğini ifade eder.)

```
/*
** server.c -- bir veri akış soketi sunucusu örneği
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>

#define MYPORT 3490 // kullanıcıların bağlanacağı port

#define BACKLOG 10 // bağlantı kuyruğunda bekletileceklerin sayısı

void sigchld_handler(int s)
{
    while(wait(NULL) > 0);
}

int main(void)
{
    int sockfd, new_fd; // sock_fd ile dinle, yen bağlantıyı new_fd ile al
    struct sockaddr_in my_addr; // adres bilgim
    struct sockaddr_in their_addr; // bağlananın adres bilgisi
    int sin_size;
    struct sigaction sa;
    int yes=1;
```

```

if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
}

if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1) {
    perror("setsockopt");
    exit(1);
}

my_addr.sin_family = AF_INET;           // konak bayt sıralaması
my_addr.sin_port = htons(MYPORT);      // short, ağ bayt sıralaması
my_addr.sin_addr.s_addr = INADDR_ANY; // otomatik olarak IP'mi kullan
memset(&(my_addr.sin_zero), '\0', 8); // geriye kalan bölgeyi sıfırla

if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr))
    == -1) {
    perror("bind");
    exit(1);
}

if (listen(sockfd, BACKLOG) == -1) {
    perror("listen");
    exit(1);
}

sa.sa_handler = sigchld_handler; // tüm ölü süreçleri kaldır
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;
if (sigaction(SIGCHLD, &sa, NULL) == -1) {
    perror("sigaction");
    exit(1);
}

while(1) {
    // ana accept() döngüsü
    sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr,
                        &sin_size)) == -1) {
        perror("accept");
        continue;
    }
    printf("server: got connection from %s\n",
           inet_ntoa(their_addr.sin_addr));
    if (!fork()) {
        // çocuk süreç
        close(sockfd); // çocuk sürecin dinlemesi gerekmez
        if (send(new_fd, "Hello, world!\n", 14, 0) == -1)
            perror("send");
        close(new_fd);
        exit(0);
    }
    close(new_fd); // ana sürecin buna ihtiyacı yok
}

return 0;
}

```

Meraklısına not: Evet kodu tek bir büyük `main()` işlevi olarak yazdım, anlaşılması basit olsun diye, ancak eğer

isterseniz bunu işlemlere ayırabilirsiniz.

(Bir de şu `sigaction()` meselesi size yabancı gelebilir — haklısınız. O gördüğümüz kod `fork()` ile oluşturulan çocuk süreçler sonlandıktan sonra kalabilecek "zombi" proseslerin icabına bakmak için var. Eğer böyle bir sürü zombi süreç olur ve siz bir şekilde onların icabına bakmazsanız bunlar gereksiz yere sistem kaynaklarını kullanır ve bu da sistem yöneticinizin size sinirlenmesine yol açar.)

Şimdi yukarıdaki sunucudan veri çekebilecek bir istemci program örneğine bakalım.

5.2. Basit Bir Veri Akış İstemcisi

Bu program daha da basit. Bu istemcinin tek yaptığı şey komut satırından belirleyeceğimiz bir konağa 3490 numaralı port üzerinden bağlanmak. Sonra da sunucunun gönderdiği dizgeyi alıp ekrana basmak.

Hemen kaynak koduna bakalım:^(B20):

```
/*
** client.c -- bir veri akış soketi istemcisi örneği
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PORT 3490 // istemcinin bağlanacağı port

#define MAXDATASIZE 100 // bir seferde kabul edebileceğimiz azami byte miktarı

int main(int argc, char *argv[])
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct hostent *he;
    struct sockaddr_in their_addr; // bağlananın adres bilgisi

    if (argc != 2) {
        fprintf(stderr, "usage: client hostname\n");
        exit(1);
    }

    if ((he=gethostbyname(argv[1])) == NULL) { // konak bilgisini al
        perror("gethostbyname");
        exit(1);
    }

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    their_addr.sin_family = AF_INET; // konak bayt sıralaması
```

```
their_addr.sin_port = htons(PORT); // short, ağ bayt sıralaması
their_addr.sin_addr = *((struct in_addr *)he->h_addr);
memset(&(their_addr.sin_zero), '\0', 8); // geriye kalanı sıfırla

if (connect(sockfd, (struct sockaddr *)&their_addr,
              sizeof(struct sockaddr)) == -1) {
    perror("connect");
    exit(1);
}

if ((numbytes=recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
    perror("recv");
    exit(1);
}

buf[numbytes] = '\0';

printf("Received: %s",buf);

close(sockfd);

return 0;
}
```

Aklınızda bulunsun eğer sunucuyu çalıştırmadan yukarıdaki istemciyi çalıştıracak olursanız `connect()` işlevi "Connection refused" (Bağlantı reddedildi) hatası verir. Çok faydalı.

5.3. Veri Paketi Soketleri

Buraya dek yazılanları anladı iseniz çok fazla açıklamaya gerek yok, örnek programların kodunu inceleyerek kavrayabilirsiniz: `talker.c` ve `listener.c`..

listener programı bir makinada çalıştıktan sonra 4950 numaralı port üzerinden gelecek paketleri dinlemeye başlar. **talker** ise komut satırında belirteceğiniz bir makinanın 4950 numaralı portuna yine komut satırından yazdığınız mesajı bir veri paketi olarak yollar.

`listener.c`^(B21) programının kaynak koduna bakalım:

```
/*
** listener.c -- a veri paketi soketi sunucusu örneği
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define MYPORT 4950 // kullanıcıların bağlanacağı port

#define MAXBUFLLEN 100

int main(void)
```

```

{
    int sockfd;
    struct sockaddr_in my_addr;    // adres bilgim
    struct sockaddr_in their_addr; // bağlananın adres bilgisi
    int addr_len, numbytes;
    char buf[MAXBUFLEN];

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    my_addr.sin_family = AF_INET;    // konak bayt sıralaması
    my_addr.sin_port = htons(MYPORT); // short, ağ bayt sıralaması
    my_addr.sin_addr.s_addr = INADDR_ANY; // otomatik olarak IP'mi kullan
    memset(&(my_addr.sin_zero), '\0', 8); // kalanı sıfırla

    if (bind(sockfd, (struct sockaddr *)&my_addr,
              sizeof(struct sockaddr)) == -1) {
        perror("bind");
        exit(1);
    }

    addr_len = sizeof(struct sockaddr);
    if ((numbytes=recvfrom(sockfd,buf, MAXBUFLEN-1, 0,
                          (struct sockaddr *)&their_addr, &addr_len)) == -1) {
        perror("recvfrom");
        exit(1);
    }

    printf("got packet from %s\n",inet_ntoa(their_addr.sin_addr));
    printf("packet is %d bytes long\n",numbytes);
    buf[numbytes] = '\0';
    printf("packet contains \"%s\"\n",buf);

    close(sockfd);

    return 0;
}

```

Dikkat edin ki `socket()` işlevini çağırırken `SOCK_DGRAM` kullandık. Ayrıca bu tür soketlerde `listen()` veya `accept()` işlevlerine ihtiyacımız yok. Bağlantısız veri paketi soketlerini kullanmanın güzel yanları da var gördüğünüz gibi!

Şimdi de sırada `talker.c`^(B22) programının kaynak kodu var::

```

/*
** talker.c -- a datagram "client" demo
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

```

```
#include <arpa/inet.h>
#include <netdb.h>

#define MYPORT 4950    // kullanıcıların bağlanacağı port

int main(int argc, char *argv[])
{
    int sockfd;
    struct sockaddr_in their_addr; // bağlananın adres bilgisi
    struct hostent *he;
    int numbytes;

    if (argc != 3) {
        fprintf(stderr, "usage: talker hostname message\n");
        exit(1);
    }

    if ((he=gethostbyname(argv[1])) == NULL) { // konak bilgisini al
        perror("gethostbyname");
        exit(1);
    }

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    their_addr.sin_family = AF_INET; // konak bayt sıralaması
    their_addr.sin_port = htons(MYPORT); // short, ağ bayt sıralaması
    their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    memset(&(their_addr.sin_zero), '\0', 8); // kalanı sıfırla

    if ((numbytes=sendto(sockfd, argv[2], strlen(argv[2]), 0,
        (struct sockaddr *)&their_addr, sizeof(struct sockaddr))) == -1) {
        perror("sendto");
        exit(1);
    }

    printf("sent %d bytes to %s\n", numbytes,
        inet_ntoa(their_addr.sin_addr));

    close(sockfd);

    return 0;
}
```

Hepsi bu kadar! Bir bilgisayarda **listener** programını derleyip çalıştırın ve başka bir bilgisayara geçip **talker** programını çalıştırın, birbirleri ile ne kadar güzel iletişim kurduklarını gözlemleyin!

Bu bölümü bitirmeden önce son bir ayrıntı: daha önce bağlantılı veri paketi soketlerinden bahsetmiştim. Madem veri paketi iletişimi ile ilgili örnek verdik o halde birkaç şey daha söylememe izin verin. Mesela **talker connect()** işlevini çağırıyor olsun ve **listener**'in adresini belirtmiş olsun. O noktadan itibaren **talker** sadece ve sadece **connect()** ile belirlenen adresi kullanabilir. Bu yüzden **sendto()** ve **recvfrom()** kullanmanıza gerek kalmaz; **send()** ve **recv()** ile işinizi görebilirsiniz.

6. İleri Teknikler

Aslında burada anlatacağım teknikler çok fazla ileri sayılmaz ama gene de şimdiye dek gördüğümüz bilgilerin ötesinde şeyler gerektiriyorlar. Doğrusunu söylemek gerekirse zaten bu aşamaya dek geldiyseniz kendizi UNIX Ağ Programlama Temelleri konusunda ciddi olarak bilgili kabul edebilirsiniz! Tebrikler!

Şimdi sıra soketlerle ilgili bazı garip şeyleri öğrenmede, cesur yeni dünyaya hoşgeldiniz. Alın bakalım!

6.1. Bloklama

Bloklama. Bunu duymuş olmalısınız — peki anlamı nedir? Kabaca söylemek gerekirse "blok" (block) sözcüğü "uyku" (sleep) için kullandığımız teknik bir terim. Bunu, yukarıdaki **listener** programını çalıştırdığınızda görmüş olmalısınız. Yani program çalışır ve orada öylece bir paketin gelmesini bekler. Olup biten şudur: Bu program `recvfrom()` işlevini çağırır, ortalıkta bir veri yoktur ve bu yüzden `recvfrom()` "bloklama" yapar (yani orada uyur) ta ki bir veri gelene kadar.

Pek çok işlev blok yapar, yani uyur. `accept()` bloklar. Tüm `recv()` işlevleri bloklar, yani bekler. Bunu yapabilmelerinin sebebi ise onlara bunu yapabilmeleri için izin verilmiş olmasıdır. Soket tanımlayıcısını ilk aşamada `socket()` ile yarattığınızda, işletim sistemi çekirdeği onu bloklayan olarak ayarlar. Eğer bir soketin bloklama yapabilmesini istemiyorsanız o zaman `fcntl()` işlevini çağırmanız gerekir:

```
#include <unistd.h>
#include <fcntl.h>
.
.
sockfd = socket(AF_INET, SOCK_STREAM, 0);
fcntl(sockfd, F_SETFL, O_NONBLOCK);
.
.
```

Bir soketi bloklamayan olarak ayarladığınızda o soketten bilgi alabilirsiniz. Eğer bloklamayan bir soketten bir şeyler okumaya kalkarsanız ve orada bir veri yoksa, bloklama yapmasına izin verilmediği için bir değer döndürecektir, bu değer `-1` olacaktır ve `errno` değişkeni de `EWOULDBLOCK` değerini alacaktır.

Ancak genelde bu şekilde bilgi edinmeye çalışmak pek de iyi bir fikir değildir. Eğer programınızı sürekli soketten veri okumaya çalışır hale getirirseniz o zaman çok fazla işlemci zamanından çalarsınız. Okunmak üzere bir verinin gelip gelmediğini kontrol etmenin daha şık bir yöntemi vardır ve bu yöntem `select()` işlevini temel alır.

6.2. `select()` — Eşzamanlı G/Ç Çoğullama

Biraz garip olmakla birlikte bu işlevin epey faydalı olduğu söylenebilir. Şu durumu ele alalım: siz bir sunucu programsınız ve bir yandan gelen bağlantıları dinlerken öte yandan da açık olan bağlantılardan akmakta olan verileri okumak istiyorsunuz.

Sorun değil, diyorsunuz, bir `accept()` ve birkaç tane de `recv()` işimizi görür. Ağır ol dostum! Ya çağırduğın `accept()` işlevi bloklayan durumda ise? O zaman aynı anda `recv()` ile nasıl oluyor da veri okumayı düşünebiliyorsun? "O halde bloklamayan soketleri kullanırım!" Hiç yakıştıramadım senin gibi bir programcıya! İşlemci zamanını deliler gibi harcamak mı istiyorsun? E peki nasıl yapacağız öyleyse?

`select()` aynı anda birden fazla soketi gözetleme imkânı sunar. Bununla kalmaz aynı zamanda hangi soketin okumak için hazır olduğunu, hangisine yazabileceğiniz, hangisinde istisnai durumlar oluştuğunu da söyler.

Laf kalabalığını kesip hemen işleve geçiyorum, `select()`:

```
#include <sys/time.h>
#include <sys/types.h>
```

```
#include <unistd.h>

int select(int numfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

Bu işlev dosya tanımlayıcı kümelerini gözlemler. Özel olarak ilgilendikleri ise *readfds*, *writefds* ve *exceptfds*'dir. Mesela standart girdiden ve *sockfd* gibi bir soket tanımlayıcıdan veri okuyup okuyamayacağınızı merak ediyorsanız tek yapmanız gereken 0 ve *sockfd*'yi *readfds* kümesine eklemek. *numfds* parametresi azami dosya tanımlayıcı artı bir olarak ayarlanmalıdır. Bu örnekte *sockfd+1* olmalıdır. Çünkü açıktır ki yukarıdaki değer standart girdiden (0) daha büyük olacaktır.

select() çalıştırıldıktan sonra *readfds* seçmiş olduğunuz dosya tanımlayıcılardan hangisinin okumak için hazır olduğunu yansıtacak şekilde güncellenir. Bunları aşağıdaki *FD_ISSET()* makrosu ile test edebilirsiniz.

Daha fazla ilerlemeden bu kümeler ile nasıl başa çıkacağınızı anlatayım. Her küme *fd_set* türündedir. Bu tür üzerinde aşağıdaki makroları kullanabilirsiniz:

- *FD_ZERO(fd_set *set)* — dosya tanımlayıcı kümesini temizler.
- *FD_SET(int fd, fd_set *set)* — kümeye *fd*'yi ekler.
- *FD_CLR(int fd, fd_set *set)* — *fd*'yi kümeden çıkarır.
- *FD_ISSET(int fd, fd_set *set)* — *fd*'ni küme içinde olup olmadığına bakar.

Son olarak, nedir bu *struct timeval*? Bazen birilerinin size veri göndermesini sonsuza dek beklemek istemezsiniz. Belki de her 96 saniyede bir ekrana "Hala çalışıyor..." mesajı basmak istersiniz (program o esnada bir şey yapmıyor olsa bile). Bu zaman yapısı sizin bir sonlandırma süresi ("timeout period") belirlemenizi sağlar. Eğer bu süre geçildi ise ve *select()* hala hazır bir dosya tanımlayıcı bulamadı ise o zaman işlev (*select*) döner ve böylece de siz de işinize devam edebilirsiniz.

struct timeval yapısının elemanları aşağıdaki gibidir:

```
struct timeval {
    int tv_sec;        // seconds
    int tv_usec;      // microseconds
};
```

Tek yapmanız gereken *tv_sec*'i kaç saniye bekleneceğine ayarlamak ve *tv_usec*'i de kaç mikrosaniye bekleneceğine ayarlamak. Evet, doğru okudunuz, mikrosaniye, milisaniye değil. Bir milisaniye içinde 1,000 mikrosaniye vardır ve bir saniye içinde de 1,000 milisaniye vardır. Yani bir saniye içinde 1,000,000 mikrosaniye vardır. Tamam da neden "usec"? Buradaki "u" harfinin Yunan alfabesindeki Mü harfine benzediği düşünülmüştür ve bu yüzden "mikro"yu temsilen kullanılmıştır. Bunlara ek olarak işlev döndüğünde *timeout* ne kadar zaman kaldığını gösterecek şekilde güncellenmiş olabilir. Bu hangi tür UNIX kullandığınıza bağlıdır.

Vay canına! Mikrosaniye hassasiyetinde bir zamanlayıcımız var! Gene de siz buna çok güvenmeyin. Standart Unix zamandilimi yaklaşık 100 milisaniyedir, yani *struct timeval* yapısını nasıl ayarlarsanız ayarlayın en az bu kadar beklemek durumunda kalabilirsiniz.

Meraklısına not: Eğer *struct timeval* yapısındaki değişkeninizin alanlarınız 0 yaparsanız, *select()* işlevi anında sonlanacak ve böyle kümenizin içindeki tüm dosya tanımlayıcıları taramış olacaktır. Eğer *timeout* parametresini NULL yaparsanız bu sefer de işlev asla zaman aşımına uğramayacak ve ilk dosya tanımlayıcı hazır olana dek bekleyecektir. Son olarak: Eğer belli bir küme için beklemek sorun teşkil etmiyorsa o zaman *select()* işlevini çağırırken onu NULL olarak ayarlayabilirsiniz.

[Aşağıdaki kod parçası](#)^(B23) standart girdiden bir veri gelmesi için 2.5 saniye bekler:

```
/*
** select.c -- bir select() örneği
*/

#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

#define STDIN 0 // standart girdi için dosya tanımlayıcı

int main(void)
{
    struct timeval tv;
    fd_set readfds;

    tv.tv_sec = 2;
    tv.tv_usec = 500000;

    FD_ZERO(&readfds);
    FD_SET(STDIN, &readfds);

    // writefds ve exceptfds ile ilgilenmiyoruz:
    select(STDIN+1, &readfds, NULL, NULL, &tv);

    if (FD_ISSET(STDIN, &readfds))
        printf("Bir tusa basıldı!\n");
    else
        printf("Zaman doldu.\n");

    return 0;
}
```

Üzerinde çalıştığınız uçbirim türüne bağlı olarak bir tuşa bastıktan sonra bunun algılanması için ENTER tuşuna basmanız gerekebilir. Aksi takdirde "Zaman doldu." mesajı alırsınız.

Şimdi bazılarınız bir veripaketi soketi üzerinden veri beklemek için bu yöntemin mükemmel bir yöntem olduğunu düşünebilir — evet haklısınız: bu yöntem gerçekten de mükemmel olabilir. Ancak bazı UNIX türevleri `select`'i bu şekilde kullanabilirken bazıları kullanamaz. Bu yüzden de pratik olarak kullanmaya başlamadan önce sisteminizde bu konu ile ilgili man sayfalarını okuyun.

Bazı UNIX türevleri de `struct timeval` yapısındaki değişkeninizi, zamanaşımına ne kadar süre kaldığını gösterecek şekilde güncelleyebilir. Bazıları ise bunu yapmaz. Eğer taşınabilir programlar yazmak istiyorsanız buna güvenmeyin. (`gettimeofday()` işlevinden faydalanın. Saçma geliyor değil mi evet ama böyle ben ne yapabilirim.)

Peki ya okuma kümesindeki soketlerden biri bağlantıyı kesmiş ise? Bu durumda `select()` bu soket için "okumaya hazır" mesajı verir ve siz `recv()` ile okumaya kalktığınızda da `recv()` size 0 değerini döndürür. Böylece istemcinin bağlantıyı kesmiş olduğunu anlarsınız.

Meraklısına `select()` ile ilgili bir bilgi daha: eğer `listen()` ile dinlemekte olan bir soketiniz varsa bu soketin dosya tanımlayıcısını `readfds` kümesine yerleştirerek yeni bir bağlantı olup olmadığını öğrenebilirsiniz.

İşte dostlarım, süper güçlü `select()` işlevinin özeti bu kadar.

Ancak gelen yoğun istek üzerine yukarıdaki bilgileri pratiğe dökeceğimiz bir örnek sizi bekliyor.

Bu program^(B24) basit bir çok kullanıcılı "chat" sunucu olarak davranır. Derledikten sonra bunu bir pencerede

çalıştırın ve ardından **telnet** ile programa bağlanın ("**telnet hostname 9034**"). Farklı farklı pencerelerden programa aynı anda birden fazla sayıda bağlantı açabilirsiniz. Bir kere bağlanıp da bulunduğunuz **telnet** ortamından bir şeyler yazıp yolladığınızda, mesajınız diğer bağlı **telnet** pencerelerinde de görünmeli.

```
/*
** selectserver.c -- keyifli bir cok kullanicili chat sunucu
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 9034 // dinledigimiz port

int main(void)
{
    fd_set master; // na dosya tanimlayici listesi
    fd_set read_fds; // select() icin gecici dosya tanimlayici listesi
    struct sockaddr_in myaddr; // sunucu adresi
    struct sockaddr_in remoteaddr; // istemci adresi
    int fdmax; // azami dosya tanimlayici numarası
    int listener; // dinlenen soket tanimlayıcı
    int newfd; // yeni accept()lenecek soket tanimlayıcı
    char buf[256]; // istemci verisi için tampon
    int nbytes;
    int yes=1; // setsockopt() SO_REUSEADDR için, aşağıda
    int addrlen;
    int i, j;

    FD_ZERO(&master); // ana listeyi ve gecici listeyi temizle
    FD_ZERO(&read_fds);

    // dinleyiciyi yarat
    if ((listener = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    // "adres zaten kullanımda" mesajından kurtul
    if (setsockopt(listener, SOL_SOCKET, SO_REUSEADDR, &yes,
        sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
    }

    // bind
    myaddr.sin_family = AF_INET;
    myaddr.sin_addr.s_addr = INADDR_ANY;
    myaddr.sin_port = htons(PORT);
    memset(&(myaddr.sin_zero), '\0', 8);
    if (bind(listener, (struct sockaddr *)&myaddr, sizeof(myaddr)) == -1) {
        perror("bind");
    }
}
```



```

    exit(1);
}

// listen
if (listen(listener, 10) == -1) {
    perror("listen");
    exit(1);
}

// dinleyici soketi ana listeye ekle
FD_SET(listener, &master);

// en büyük dosya tanımlayıcısı hatırla
fdmax = listener; // so far, it's this one

// ana döngü
for(;;) {
    read_fds = master; // copy it
    if (select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1) {
        perror("select");
        exit(1);
    }

    // mevcut bağlantıları tarayip okumaya hazır olanları tespit et
    for(i = 0; i <= fdmax; i++) {
        if (FD_ISSET(i, &read_fds)) { // bir tane yakaladık!!
            if (i == listener) {
                // handle new connections
                addrlen = sizeof(remoteaddr);
                if ((newfd = accept(listener, (struct sockaddr *)&remoteaddr,
                                     &addrlen)) == -1) {
                    perror("accept");
                } else {
                    FD_SET(newfd, &master); // ana listeye ekle
                    if (newfd > fdmax) { // azami miktarı güncelle
                        fdmax = newfd;
                    }
                    printf("selectserver: new connection from %s on "
                           "socket %d\n", inet_ntoa(remoteaddr.sin_addr), newfd);
                }
            } else {
                // istemciden gelen veri için gerekeni yap
                if ((nbytes = recv(i, buf, sizeof(buf), 0)) <= 0) {
                    // bir hata var ya da istemci bağlantıyı kesti
                    if (nbytes == 0) {
                        // bağlantı kesilmiş
                        printf("selectserver: socket %d hung up\n", i);
                    } else {
                        perror("recv");
                    }
                    close(i); // bye!
                    FD_CLR(i, &master); // ana listeden çıkar
                } else {
                    // istemciden bir miktar veri geldi
                    for(j = 0; j <= fdmax; j++) {
                        // gelen veriyi herkese yolla!
                        if (FD_ISSET(j, &master)) {

```

```

// dinleyici ve kendimiz haric
if (j != listener && j != i) {
    if (send(j, buf, nbytes, 0) == -1) {
        perror("send");
    }
}
}
}
} // O KADAR CIRKIN ki!
}
}
return 0;
}

```

Lütfen dikkat edin: Yukarıdaki kod içinde iki dosya tanımlayıcı listem var: *master* ve *read_fds*. Birincisi yani, *master* o esnada bağlı olan tüm socket tanımlayıcılarını tutuyor ve buna ek olarak bir de dinleme görevini üstlenmiş olan socketi de bünyesinde barındırıyor.

master diye bir liste oluşturmanın sebebi `select()` işlevinin sizin ona verdiğiniz socket listesini değiştirmesi ve hangilerinin okunmaya hazır olduğunu gösterecek şekilde güncellemesi. `select()` çağrıları arasında bağlantı kümesinin bozulmamış bir kopyasına ihtiyacım olduğu için böyle bir şey yaptım. Yani son anda *master* kümesini *read_fds* kümesine kopyalıyor ve `select()` işlevini çağırıyorum.

İyi de bu aynı zamanda her yeni bağlantı talebinde bu bağlantıyı *master* listesine eklemem gerektiği anlamına gelmiyor mu? Tabii ki! Ve her bağlantı kesilmesinde de kesilen bağlantı ile ilgili tanımlayıcıyı da *master* listesinden çıkarmam gerekmiyor mu? Elbette!.

Farkında iseniz *listener* socketinin ne zaman hazır olduğunu kontrol ediyorum. Hazır olduğunda bu demek oluyor ki yeni bir bağlantı talebi var ve ben de bunu görünce bağlantıyı `accept()` ile kabul ediyorum ve *master* listesine ekliyorum. Benzer şekilde bir istemci bağlantısı hazır olduğunda ve `recv()` işlevi 0 döndürdüğünde anlıyorum ki istemci bağlantıyı kapatmış ve bu yüzden onu *master* listesinden çıkarmalıyım.

Ancak eğer istemci `recv()` işlevi ile sıfırdan farklı bir değer döndürdüğünde de biliyorum ki okunmuş olan bir veri yığını var. Bu veriyi alıyorum ve sonra da *master* listesi üzerindeki elemanlar üzerinden tek tek dolaşıp almış olduğum veriyi diğer istemcilere yolluyorum.

Ve dostlarım işte size süper güçlü `select()` işlevinin o kadar da basit olmayan açıklaması.

6.3. Sorunlu `send()` Durumları

Hatırlarsanız `send()` ve `recv()` — *Konuş Benimle Bebeğim!* (sayfa: 17) bölümünde `send()` işlevinin sizin istediğiniz kadar veriyi bir anda gönderemeyebileceğinden bahsetmiştim. Tam olarak böyle yani siz ondan 512 byte göndermesini istersiniz ama o sadece 412 tanesini yollar. Peki geriye kalan 100 bayta ne oldu?

Bu veriler hala bizim küçük tampon bölgemizde gönderilmeyi bekliyorlar. Kontrolünüz dışındaki bir takım sebeplerden ötürü işletim sistemi çekirdeği mevcut veriyi bir seferde göndermemeye karar verdi ve iş başa düştü.

Sorunu halletmek için şöyle bir işlev kullanabilirsiniz:

```

#include <sys/types.h>
#include <sys/socket.h>

int sendall(int s, char *buf, int *len)
{

```

```
int total = 0;          // gönderdiğimiz byte miktarı
int bytesleft = *len; // eksik kalan byte miktarı
int n;

while(total < *len) {
    n = send(s, buf+total, bytesleft, 0);
    if (n == -1) { break; }
    total += n;
    bytesleft -= n;
}

*len = total; // gönderilen toplam bayt miktarı

return n===-1?-1:0; // sorun varsa -1, işlem tamamsa 0 döner
}
```

Bu örnekte, *s* üzerinden veriyi göndermek istediğimiz soketi, *buf* veriyi barındıran bellek bölgesini (buffer) ve *len* de tampondaki byte sayısını gösteren sayıya işaret eden *int* türünde bir göstergedir.

Hata durumunda işlev *-1* değerini döndürür (ve *errno* hata değişkeni de *send()* işlevi sayesinde gerekli hata kodunu barındırır.) Ayrıca gönderilebilmiş olan byte sayısı da *len* parametresinden depolanır. Bu sayı sizin göndermek istediğiniz sayıya eşit olacaktır (hata oluşması durumu haricinde). *sendall()* işlevi verinin tamamını yollamak için elinden geleni yapacaktır ancak bir hata ile karşılaşır size geri dönecektir.

Hemen bir örnek verelim:

```
char buf[10] = "Beej!";
int len;

len = strlen(buf);
if (sendall(s, buf, &len) == -1) {
    perror("sendall");
    printf("We only sent %d bytes because of the error!\n", len);
}
```

Peki ya verinin sadece bir kısmı diğer tarafa erişebildiğinde diğer tarafta ne olup biter? Eğer gelen paketlerin boyu değişken ise bunları alan taraf bir paketin ne zaman bittiğini, diğerinin ne zaman başladığını nasıl anlar? Evet, gerçek dünya senaryoları insanları biraz düşünmeye zorlar. Büyük ihtimalle yukarıdaki problemlerle başa çıkabilmek için yapmanız gereken veriyi paketlemek (encapsulate) olacaktır (*veri paketleme bölümü* (sayfa: 8)nü hatırladınız mı?) Ayrıntılar için okumaya devam edin!

6.4. Veri Paketlemesi Hazretleri

Veri paketlemesi de ne demek oluyor? En basit anlamına bakacak olursak bu şu anlama gelir: Verinin başına ya onu tanımlayan bir başlık bilgisi koyacaksınız ya uzunluk bilgisi ya da her ikisi birden.

Başlık deyip durduğumuz şey neye benzer? Bu tamamen size kalmış ve projeden projeye değişebilen bir şeydir.

Ama! Bu açıklamalar biraz havada kalmadı mı?

Tamam. Mesela çok kullanıcı bir chat programı geliştirdiğinizi ve bu programın da *SOCK_STREAM* türünden soketler kullandığını var sayalım. Kullanıcılardan biri bir şey dediğinde sunucuya gitmesi gereken iki bilgi parçası vardır: Kim dedi ve ne dedi?

Buraya kadar güzel. "Problem ne?" diye soruyor olabilirsiniz.

Problem şu: Gelip giden mesajların uzunluğu değişken. "Ali" isimli biri "Selam" diyebilir ve sonra da "Veli" isimli biri "naber moruk?" diyebilir.

Bu durumda siz de `send()` ile gelen verileri tüm istemcilere yollarsınız öyle değil mi ve yolladığınız veri de şu şekilde olur:

```
A l i S e l a m V e l i N a b e r m o r u k ?
```

Ve böyle sürüp gider. Peki istemci bir paketin bitip diğerinin başladığını nasıl anlayacak? Eğer isterseniz tüm paketlerin boyunu sabit yapabilirsiniz ve sonra da `sendall()` işlevini çağırırsınız (*yukarıda bir örneği var* (sayfa: 34)). Fakat bu bant genişliğini boş yere harcamak demektir! Yani herhalde `send()` ile sadece Ali'nin "Selam" demesi için 1024 byte göndermek istemeyiz değil mi?

İşte bu yüzden veriyi ufak tefek bir başlık ve paket yapısı ile paketleriz. Hem sunucu hem de istemci verinin nasıl paketleneyeceğini (marshal) ve bu paketin nasıl açılacağını (unmarshal) bilir. Sıkı durun çünkü tam da şu andan itibaren bir sunucu ile istemcinin hangi kurallara göre iletişim kuracağını belirleyen bir protokol tanımlamaya başlıyoruz!

Şimdi var sayalım ki kullanıcı adı en fazla 8 karakter boyunda olabiliyor ve eğer daha kısa ise geriye kalan bölge `'\0'` ile dolduruluyor. Ve sonra varsayalım ki mesaj uzunluğu değişken olabiliyor ve maksimum 128 karaktere izin veriyoruz. Buna göre örnek bir pakete bakalım:

1. `len` (1 byte, unsigned) — Paketin toplam boyu, 8 baytlık kullanıcı ismi ve chat mesajı dahil.
2. `name` (8 byte) — Kullanıcı ismi, gerekirse sonu NULL ile doldurulmuş.
3. `chatdata` (n -byte) — Gönderilecek mesajın kendisi. En fazla 128 byte olabilir. Paket boyu bu verinin boyu artı 8 (kullanıcı ismi bölümünün uzunluğu) olarak hesaplanabilir.

Neden söz konusu veri alanları için sırası ile 8 byte ve 128 byte sınırlarını seçtim? Tamamen keyfime kalmış, bana yeterince uzun göründü. Ama belki de sizin ihtiyaçlarınız için 8 byte çok az gelebilir ve siz de 30 byte uzunluğunda bir isim alanı kullanabilirsiniz. Size kalmış.

Yukarıdaki paket tanımlamasını kullanarak ilk paketimizin görüntüsü şuna benzeyecektir (onaltılık ve ASCII):

```
0D      41 6C 69 00 00 00 00 00      53 65 6C 61 6D
(uzunluk) A l i      (dolgu)      S e l a m
```

Ve ikincisi de benzer şekilde:

```
13      56 65 6C 69 00 00 00 00      4E 61 62 65 72 6D 6F 72 75 6B 3F
(uzunluk) V e l i      (dolgu)      N a b e r m o r u k ?
```

(Uzunluk bilgisi de elbette Ağ Byte Sıralamasına uygun depolanmalıdır. Ancak mevcut örnekte tek bir byte olduğu için çok önemli değil. En genel durumda ise sakın unutmayın ki tüm ikili düzendeki tamsayılarınız Ağ Byte Sıralamasına göre dizilmiş olmalıdır.)

Bu veriyi yollarken güvenli bir şekilde, mesela `sendall()` (sayfa: 34) gibi bir işleyle yollamalısınız. Böylece birçok kez `send()` işlevini çağırılmış olmak gerekse de gittiğinden emin olabilirsiniz.

Benzer şekilde bu veriyi aldığınızda çözmek için biraz iş yapmanız gerekir. En genel durumda verinin ancak bir kısmının gelmiş olabileceğini göz önünde bulundurmalısınız (misal sadece "00 13 56 65 6C" size ulaşmış olabilir Veli isimli kullanıcıdan. Bu ilk parça gördüğünüz gibi eksiktir ve verinin tamamına erişene dek birkaç kez daha `recv()` işlevini çağırmanız gerekebilir.

İyi de nasıl? Paketin en başına paket uzunluğu konmuş olduğundan en baştan itibaren bir paketin orjinal uzunluğunu biliriz. Aynı zamanda biliriz ki azami paket boyu da 1+8+128 olarak hesaplanabilir yani toplam olara 137 byte (çünkü biz böyle tanımladık).

Bu durumda yapabileceğiniz şeylerden biri iki paketi barındırabilecek büyüklükte bir dizi tanımlamaktır. Burası sizin çalışma diziniz olacak ve paketler buraya vardıkça onları buradan alıp düzenleyeceksiniz.

`recv()` işlevini her çağırışınızda gelen veriyi çalışma dizisine aktarırsınız ve paketin tamamlanıp tamamlanmadığına bakarsınız. Yani dizi içindeki byte miktarının paket boyuna eşit mi yoksa ondan büyük mü olduğunu kontrol edersiniz (+1 çünkü başlıktaki uzunluk kendisi için kullanılan 1 byte'lık uzunluğu içermez). Eğer tampondaki byte sayısı 1'den az ise o zaman paket tam değildir açık olarak. Bu durumu özel olarak ele almalısınız, çünkü bu ilk byte çöplüktür ve paket boyu için ona güvenemezsiniz.

Paket tamamlandığında artık onun üzerinde istediğiniz işlemi gerçekleştirebilirsiniz. Onu kullanabilir, çalışma dizinizden çıkarabilirsiniz.

Nasıl? Kafanızda evirip çevirmeye başladınız mı? Sıkı durun şimdi ikinci darbe geliyor: bir paketi sonuna kadar okumakla kalmayıp aynı esnada bir sonraki paketin de bir kısmını son `recv()` çağırışında okumuş olabilirsiniz. Yani çalışma bölgenizde bir tam paket ve bir de ardından gelen bir kısmi paket vardır! (İşte çalışma tamponunu biraz geniş tutun dememin sebebi bu idi. Yani iki paketi tutabilecek kadar — işte şimdi o durum gerçekleşti!)

İlk paketin boyunu bildiğinize ve gelen byte'ları da takip ettiğinize göre bu sayıları kullanarak çalışma tamponunuzdaki baytlardan hangisinin ilk pakete hangisinin de kısmi pakete ait olduğunu tespit edebilirsiniz. Birincisi ile uğraşmayı bitirince onu çalışma dizisinden çıkarıp bir sonraki paketin parçasını tampon başlangıcına taşıyabilirsiniz ve böylece tampon da bir sonraki `recv()` çağırışı için hazır hale gelir.

(Bazı okuyucular belki fark etmiştir, kısmi olarak gelen ikinci paketi tampon başına taşımak biraz vakit alabilecek bir iştir ve program buna gerek duymaması için döngüsel tampon (circular buffer) kullanacak şekilde kodlanabilir. Maalesef döngüsel tamponla ilgili tartışma bu makalenin alanı dışına düşmektedir. Gerçekten merak ettiyseniz Veri Yapıları ile ilgili güzel bir kitap alıp ilgili bölümü okuyun)

Kolay olduğunu söylememiştim. Şey, aslında söylemişim ve öyleydi. Öyledir, yani eğer pratik yapmaya başlarsanız gerisi çorap söküğü gibi gelecektir. Excalibur üzerine yemin ederim öyle olacağına!

7. Diğer Kaynaklar

Bu aşamaya kadar geldiniz ve daha ayrıntılı bilgi istiyorsunuz! Bütün bu ağ programlama konusu ile ilgili daha ayrıntılı bilgileri nereden edinebilirsiniz?

7.1. man Sayfaları

Yeni başlayanlar aşağıdaki man sayfaları ile işe başlayabilirler:

- `htonl()` ^(B29)
- `htons()` ^(B30)
- `ntohl()` ^(B31)
- `ntohs()` ^(B32)
- `inet_aton()` ^(B33)
- `inet_addr()` ^(B34)
- `inet_ntoa()` ^(B35)
- `socket()` ^(B36)
- `socket options` ^(B37)
- `bind()` ^(B38)
- `connect()` ^(B39)

- `listen()` ^(B40)
- `accept()` ^(B41)
- `send()` ^(B42)
- `recv()` ^(B43)
- `sendto()` ^(B44)
- `recvfrom()` ^(B45)
- `close()` ^(B46)
- `shutdown()` ^(B47)
- `getpeername()` ^(B48)
- `getsockname()` ^(B49)
- `gethostbyname()` ^(B50)
- `gethostbyaddr()` ^(B51)
- `getprotobyname()` ^(B52)
- `fcntl()` ^(B53)
- `select()` ^(B54)
- `perror()` ^(B55)
- `gettimeofday()` ^(B56)

7.2. Kitaplar

Eski tarz çalışmayı, masaya kitabı koyup harıl harıl okumayı sevenler için aşağıdakiler harika birer yol gösterici rolü oynayacaktır. Amazon.com logosuna dikkat, bütün bu utanmaz ticari yaklaşımın sebebi şu: Siz bu link aracılığı ile ne kadar çok kitap satın alırsanız Amazon.com da bana o kadar çok kredi açıyor ve böylece ben de oradan kitap alabiliyorum. Yani eğer aşağıdaki kitaplardan birini satın alırsanız neden bana da bir faydanız dokunmasın ki diyorum.

Ayrıca daha çok kitap demek sizin için yazacağım daha çok sayıda programlama rehberi demek. ; -)



In Association with Amazon.com

Unix Network Programming, volumes 1–2 W. Richard Stevens. Prentice Hall tarafından yayınlandı. ISBN'ler, 1–2 için: [013490012X](#)^(B57), [0130810819](#)^(B58).

Internetworking with TCP/IP, volumes I–III Douglas E. Comer ve David L. Stevens. Prentice Hall tarafından yayınlandı. ISBN'ler, I, II ve III için: [0130183806](#)^(B59), [0139738436](#)^(B60), [0138487146](#)^(B61).

TCP/IP Illustrated, volumes 1–3 W. Richard Stevens ve Gary R. Wright. Addison Wesley tarafından yayınlandı. ISBN'ler 1, 2 ve 3 için: [0201633469](#)^(B62), [020163354X](#)^(B63), [0201634953](#)^(B64).

TCP/IP Network Administration Craig Hunt.O'Reilly & Associates, Inc. tarafından yayınlandı. ISBN 1565923227^(B65).

Advanced Programming in the UNIX Environment W. Richard Stevens. Addison Wesley tarafından yayınlandı. ISBN 0201563177^(B66).

Using C on the UNIX System David A. Curry. O'Reilly & Associates, Inc. tarafından yayınlandı. ISBN 0937175234. *Tükendi.*

7.3. Web Kaynakları

Web üzerindeki:

BSD Sockets: A Quick And Dirty Primer^(B67) (Unix sistem programlama ile ilgili diğer konularda da ciddi bilgi içeriyor!)

The Unix Socket FAQ^(B68)

Client-Server Computing^(B69)

Intro to TCP/IP^(B70) (gopher)

Internet Protocol Frequently Asked Questions^(B71)

The Winsock FAQ^(B72)

7.4. RFC'ler

RFC'ler^(B73):

RFC-768^(B74) — The User Datagram Protocol (UDP)

RFC-791^(B75) — The Internet Protocol (IP)

RFC-793^(B76) — The Transmission Control Protocol (TCP)

RFC-854^(B77) — The Telnet Protocol

RFC-951^(B78) — The Bootstrap Protocol (BOOTP)

RFC-1350^(B79) — The Trivial File Transfer Protocol (TFTP)

8. Sıkça Sorulan Sorular

8.1. Şu başlık dosyalarını nereden edinebilirim?

8.2. `bind()` "Address already in use" veya "Adres zaten kullanımda" mesajı verdiğinde ne yapmalıyım?

8.3. Sistemdeki açık soketlerin listesini nasıl alabilirim?

8.4. Yönlendirme tablosunu nasıl görüntüleyebilirim?

8.5. Eğer tek bir bilgisayarım varsa istemci-sunucu türünde programları nasıl çalıştırabilirim? Bir ağ programı yazabilmek için bir ağa ihtiyacım yok mu?

8.6. Diğer tarafın bağlantıyı kestiğini nasıl tespit edebilirim?

8.7. Kendi "ping" programımı nasıl yazabilirim? ICMP nedir? Raw soketler ve `SOCK_RAW` ile daha ayrıntılı bilgiyi nerede bulabilirim?

8.8. Bu programları MS Windows üzerinde nasıl derlerim?

8.9. Bu programları Solaris/SunOS üzerinde nasıl derlerim? Derlemeye çalıştığımda linker sürekli hata veriyor!

8.10. `select()` neden sinyal alır almaz düşüyor?

- 8.11. `recv()` işlevi için bir zanaşımı mekanizmasını nasıl kurabilirim?
- 8.12. Veriyi soket üzerinden göndermeden önce nasıl şifreleyip sıkıştırabilirim?
- 8.13. Bir sürü yerde gördüğüm şu "PF_INET" de nedir? AF_INET ile bir bağlantısı var mıdır?
- 8.14. İstemciden kabuk komutlarını kabul edip onları çalıştıran bir sunucu yazılımını nasıl geliştirebilirim?
- 8.15. Bir yığın veriyi bir seferde yollamaya çalışıyorum ama diğer taraftan `recv()` ile okumaya kalktığımda sadece 536 veya 1460 byte geldiğini görüyorum. Ancak aynı denemeyi kendi makinamın üzerinde iki farklı pencere açıp yaptığımda sorunsuz olarak veri yollayıp alabiliyorum. Bunun sebebi nedir?
- 8.16. Bilgisayarım MS Windows çalışıyor ve elimde `fork()` gibi bir sistem çağırısı olmadığı gibi `struct sigaction` şeklinde bir yapı da yok. Ne yapabilirim?
- 8.17. TCP/IP üzerinde veriyi güvenli ve şifreli bir şekilde nasıl iletebilirim?
- 8.18. Bir güvenlik duvarının arkasındayım — diğer taraftaki insanların benim IP adresimi öğrenip benim sistemime bağlanmalarını nasıl sağlarım?

8.1. Şu başlık dosyalarını nereden edinebilirim?

Eğer sisteminizde bunlar mevcut değilse, muhtemelen zaten ihtiyacınız olmadığı içindir. Üzerinde çalıştığınız platformun belgelerini inceleyin. Eğer MS Windows ortamında derlemeye çalışıyor iseniz tek ihtiyacınız olan `#include <winsock.h>`.

8.2. `bind()` "Address already in use" veya "Adres zaten kullanımda" mesajı verdiğinde ne yapmalıyım?

Dinleyici soketler üzerinde `setsockopt()` ile birlikte `SO_REUSEADDR` seçeneğini kullanmalısınız. [bind\(\) ile ilgili bölümü](#) (sayfa: 13) ve [select\(\) ile ilgili bölümü](#) (sayfa: 29) okuyup örnekleri inceleyin.

8.3. Sistemdeki açık soketlerin listesini nasıl alabilirim?

`netstat` komutunu kullanın. Bu komutla ilgili ayrıntılar için ilgili man sayfasını okuyun. Basit bir örnek vermem gerekirse sadece:

```
$ netstat
```

yazarak bile epey mantıklı bir çıktı alabilirsiniz. Buradaki asıl mesele hangi soketin hangi program ile ilişkili olduğunu bulmak⁽⁶⁾.

8.4. Yönlendirme tablosunu nasıl görüntüleyebilirim?

Kısaca: `route` komutunu çalıştırın (Linux için bu komutun yeri genellikle `/sbin` dizinidir) veya `netstat -r` komutunu deneyin.

8.5. Eğer tek bir bilgisayarım varsa istemci–sunucu türünde programları nasıl çalıştırabilirim? Bir ağ programı yazabilmek için bir ağa ihtiyacım yok mu?

Şanslısınız çünkü hemen hemen tüm işletim sistemleri bir tür geridönüş (loopback) aygıtı denilen sanal bir ağ aygıtı kullanır. Bu aygıt işletim sistemi çekirdeğine gömülü bir mekanizmadır ve tıpkı fiziksel bir ağ kartı gibi davranır böylece aynı makinada hem sunucu hem de istemci yazılımlar çalışabilir. (Bu sanal aygıt yönlendirme tablosunda "lo" olarak listelenir.)

Bir örnek vermek gerekirse "goat" isimli bir bilgisayara giriş yaptığınızı varsayalım. Bir pencerede istemciyi diğerinde de sunucuyu çalıştırabilirsiniz. Ya da sunucuyu şuna benzer bir komut ile arka planda çalıştırın: "`server &`" ve sonra aynı yerde istemciyi çalıştırın. Geridönüş aygıtı dediğimiz sanal aygıt sisteminizde mevcut ise (%99.9 olasılıkla mevcuttur) `client goat` veya `client goat` gibi bir komutla ("localhost"un sizdeki `/etc/hosts` dosyasında tanımlı olduğunu varsayıyorum) istemci yazılımın sunucu yazılım ile aynı makina üzerinde konuştuğunu gözlemleyebilirsiniz. Bir ağa ihtiyaç duymadan!

Kısaca söylemek gerekirse ağa bağlı olmayan bir bilgisayarda yukarıda verilmiş olan örnek kodları çalıştırabilmek için herhangi bir değişiklik yapmanıza gerek yoktur. Yaşasınnn!

8.6. Diğer tarafın bağlantıyı kestiğini nasıl tespit edebilirim?

`recv()` işlevi öyle bir durumda 0 değerini döndürür.

8.7. Kendi "ping" programımı nasıl yazabilirim? ICMP nedir? Raw soketler ve `SOCK_RAW` ile daha ayrıntılı bilgiyi nerede bulabilirim?

Raw soketler ile ilgili tüm sorularınızın cevabını W. Richard Stevens'in "UNIX Network Programming" kitaplarında bulabilirsiniz. Lütfen kılavuzun [kitaplar](#) (sayfa: 38) bölümüne bakın.

8.8. Bu programları MS Windows üzerinde nasıl derlerim?

En basit yöntem şu: MS Windows'u silin ve Linux ya da BSD işletim sistemini kurun. };-) Peki peki tamam, o kadar ısrarcı iseniz [Windows Programcılarının Dikkat Etmesi Gerekenler](#) (sayfa: 4) bölümüne bakın.

8.9. Bu programları Solaris/SunOS üzerinde nasıl derlerim? Derlemeye çalıştığımda linker sürekli hata veriyor!

Linker hatası alabilirsiniz, çünkü genellikle Sun Microsystems ortamında soket kitaplıkları otomatik olarak bağlanmaz. Ayrıntılı bilgi için [Solaris/SunOS Programcılarının Dikkat Etmesi Gerekenler](#) (sayfa: 4) bölümüne bakın.

8.10. `select()` neden sinyal alır almaz düşüyor?

Sinyaller genellikle bloklu sistem çağrılarının `-1` değerini döndürmesine ve `errno` değişkeninin de `EINTR` değerini almasına yol açar. Sinyal işlevi `sigaction()` ile ayarladığınızda, `SA_RESTART` kullanabilirsiniz. Böylece sistem çağrısı kesildiğinde onu yeniden başlatılma ihtimali olur.

Doğal olarak bu her zaman işe yaramaz.

Bunun için önereceğim en iyi çözüm bir `goto` deyimine dayanır. Bildiğiniz gibi bu size yapısal programlama dersi veren profesörün tüylerinin diken diken olmasına yeter de artar bile, o halde neden kullanmıyorsunuz!

```
select_restart:
    if ((err = select(fdmax+1, &readfds, NULL, NULL, NULL)) == -1) {
        if (errno == EINTR) {
            // bir sinyal kesildi o halde yeniden baslat
            goto select_restart;
        }
        // gerçek hata ile burada uğras:
        perror("select");
    }
```

Tabii ki burada `goto` kullanmanız *şart değil*; kontrol için başka yapıları da kullanabilirsiniz, ancak ben özel olarak bu iş için `goto` deyiminin daha temiz olduğunu düşünüyorum.

8.11. `recv()` işlevi için bir zamanaşımı mekanizmasını nasıl kurabilirim?

`select()` (sayfa: 29) işlevini kullanın! Bu işlev okuyacağınız soket tanımlayıcıları için bir zamanaşımı parametresi belirlemenize izin verir. Ya da bu işlevselliğin tamamını tek bir işlev içine şu şekilde gömmelisiniz:

```
#include <unistd.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```

int recvtimeout(int s, char *buf, int len, int timeout)
{
    fd_set fds;
    int n;
    struct timeval tv;

    // dosya tanımlayıcı listesini ayarla
    FD_ZERO(&fds);
    FD_SET(s, &fds);

    // zamanaşımı için struct timeval türündeki değişkeni ayarla
    tv.tv_sec = timeout;
    tv.tv_usec = 0;

    // veri gelene veya zamanaşımı dolana dek bekle
    n = select(s+1, &fds, NULL, NULL, &tv);
    if (n == 0) return -2; // zamanaşımı!
    if (n == -1) return -1; // hata

    // veri burada olmalı, o halde normal şekilde recv() çağır
    return recv(s, buf, len, 0);
}

// recvtimeout() örneği:
.
.
n = recvtimeout(s, buf, sizeof(buf), 10); // zamanaşımı 10 saniye

if (n == -1) {
    // hata olustu
    perror("recvtimeout");
}
else if (n == -2) {
    // zamanaşımı!
} else {
    // buf içine veri geldi
}
.
.

```



Dikkat edin

`recvtimeout()` işlevi zamanaşımı durumunda `-2` değerini döndürür. Neden `0` değil? Hatırlayacak olursanız `0` değeri `recv()` işlevi söz konusu olduğunda karşı tarafın bağlantıyı kestiği anlamına geliyordu. Bu değer kullanılmış olduğu için ve `-1` de hata anlamına geldiğinden zamanaşımı durumunu göstermesi için `-2` değerini seçtim.

8.12. Veriyi soket üzerinden göndermeden önce nasıl şifreleyip sıkıştırabilirim?

Şifrelemeyi gerçekleştirmenin kolay yollarından biri SSL (secure sockets layer – güvenlik soketleri katmanı) kullanmaktır ancak bu konu bu kılavuzun kapsamı dışındadır.

Ancak eğer sıkıştırma ve şifreleme algoritmalarınızı kullanmak istiyorsanız verinin her iki uçta da adım adım işlendiğini düşünmek işinizi kolaylaştıracaktır. Her adım veriyi belli bir şekilde dönüştürür.

1. sunucu veriyi bir dosyadan (veya bir yerlerden) okur,
2. sunucu veriyi şifreler (bu kısmı siz ekleyeceksiniz),

3. sunucu şifrelenmiş veriyi `send()` ile yollar.

Şimdi de öteki tarafa bakalım:

4. istemci `recv()` ile kendisine yollanan şifreli veriyi alır,
5. istemci şifreli veriyi çözer yani deşifre eder (bu kısmı siz ekleyeceksiniz).

Yukarıda şifreleme/deşifreleme yaptığınız aşamada sıkıştırma/açma işlemleri de yapabilirsiniz. Ya da hem şifreleme hem de sıkıştırma yapabilirsiniz! Yalnız aklınızda bulunsun, bir veriyi eğer şifreleyecekseniz önce sıkıştırın sonra şifreleyin⁽⁷⁾ :)

Yani istemci, sunucunun uyguladığı dönüşümlerin tersini uygulayabildiği sürece araya istediğiniz kadar dönüşüm, işlem, işlev, vs. sokabilirsiniz.

Tek yapmanız gereken verdiğim örnek kodda verinin gönderildiği ve alındığı kısımları tespit bunların önce-sine şifreleme işlemini yerleştirmek.

8.13. Bir sürü yerde gördüğüm şu "PF_INET" de nedir? AF_INET ile bir bağlantısı var mıdır?

Evet, elbette. Bunun için lütfen [socket\(\) — Al Şu Dosya Tanımlayıcıyı!](#) (sayfa: 12) bölümüne bakın.

8.14. İstemciden kabuk komutlarını kabul edip onları çalıştıran bir sunucu yazılımını nasıl geliştirebilirim?

Basit olsun diye varsayalım ki istemci `connect()` ile bağlanıyor, `send()` ile veriyi gönderiyor ve `close()` ile bağlantıyı kesiyor (yani istemci tekrar bağlanana dek bir sistem çağrısı söz konusu olmuyor).

İstemcinin izlediği süreç şöyledir:

1. `connect()` ile sunucuya bağlan,
2. `send("/sbin/ls > /tmp/client.out")`,
3. `close()` ile bağlantıyı kes.

Bu arada sunucu gelen veri ile muhatap olur ve onu çalıştırır:

1. `accept()` ile bağlantıyı kabul et,
2. `recv(str)` ile komut dizisini al,
3. `close()` ile bağlantıyı kes,
4. `system(komut)` ile komutu çalıştır.



Dikkat

İstemcinin sunucuya ne yapacağını söylemesi uzaktan kabuk erişimi vermek demektir. Böyle bir yetkiye sahip olan bir insan kötü şeyler yapabilir. Mesela yukarıdaki gibi bir programda istemci "**rm -rf ~**" gibi bir komut gönderirse ne olur? Sizinle ilişkili alandaki tüm dosyalar silinir, işte budur olacağı!

Bu yüzden akıllı olun ve kesinlikle güvenli olduğunuz programlar haricinde uzaktaki istemcinin sizin sunucunuzda hiçbir şey çalıştırmasına izin vermeyin, örneğin **foobar** komutu gibi:

```
if (!strcmp(str, "foobar")) {
    sprintf(sysstr, "%s > /tmp/server.out", str);
    system(sysstr);
}
```

`foobar` güvenilir ve sorunsuz bir komut olabilir ama gene de şüpheli olmalısınız ya istemci "`foobar; rm -rf ~`" gibi bir komut dizisi yollarsa? En güvenli yöntemlerden biri komuta verilen parametrelerdeki alfanümerik olmayan her karakterin (boşluk dahil) başına önceleme karakteri ("\`\`") koyan bir işlev yazmaktır.

Gördüğünüz gibi sunucu tarafında istemcinin gönderdiği komutları çalıştırma gibi bir durum söz konusu olunca güvenlik çok önemli bir mesele haline gelmektedir.

8.15. Bir yığın veriyi bir seferde yollamaya çalışıyorum ama diğer taraftan `recv()` ile okumaya kalktığımda sadece 536 veya 1460 byte geldiğini görüyorum. Ancak aynı denemeyi kendi makinamın üzerinde iki farklı pencere açıp yaptığımda sorunsuz olarak veri yollayıp alabiliyorum. Bunun sebebi nedir?

MTU sınırını — fiziksel ortamın bir seferde kaldırabileceği azami yük miktarını aşıyorsunuz. Makinanızdaki sanal geridonuş aygıtı sürücüsü 8K ya da daha fazlasını bir seferde sorunsuz olarak iletebilir. Ancak Ethernet bir seferde başlık bilgisi ile birlikte en fazla 1500 byte taşıyabilir ve siz de bu limiti aşmış durumdasınız. Modem üzerinden veri yollamaya kalktığınızda 576 byte sınırı vardır ve yine bunu tek seferde geçerseniz sorun yaşarsınız.

Öncelikli olarak tüm veriyi yolladığınızdan emin olmalısınız. (Bunun için lütfen `sendall()` (sayfa: 34) ile ilgili ayrıntılı açıklamalara bakın.) Bundan eminseniz buna ek olarak verinin tamamını okuyana dek bir döngü içinde `recv()` işlevini çağırmanız gerekir.

`recv()` işlevini defalarca çağırarak verinin tamamını alma ile ilgili ayrıntılı açıklamalar için lütfen [Veri Paketlemesi Hazretleri](#) (sayfa: 35) bölümünü okuyun.

8.16. Bilgisayarımda MS Windows çalışıyor ve elimde `fork()` gibi bir sistem çağrısı olmadığı gibi `struct sigaction` şeklinde bir yapı da yok. Ne yapabilirim?

Eğer varsalar POSIX kitaplıkları içindedirler ve bunlar da derleyiciniz ile gelmiş olabilir. Ben Windows kullanmıyorum bu yüzden size kesin cevap veremeyeceğim. Fakat hatırladığım kadarı ile Microsoft'un da kullandığı bir POSIX uyumluluk katmanı vardı ve işte aradığınız `fork()` olsa olsa oradadır. (Hatta belki `sigaction` bile.)

VC++ ile gelen belgeleri "fork" veya "POSIX" için bir tarayın ve size hangi ipuçlarını verdiğine bir bakın.

Eğer mantıklı bir şey çıkamazsa `fork()/sigaction` ikilisini bırakın ve şunu deneyin: `CreateProcess()`. Açıkçası ben `CreateProcess()` işlevi tam olarak nasıl kullanılır bilmiyorum — milyarlarca argüman alıyor olmalı ama VC++ belgelerinde hepsi açıklanıyor olmalı.

8.17. TCP/IP üzerinde veriyi güvenli ve şifreli bir şekilde nasıl iletebilirim?

[OpenSSL projesi](#)^(B89)ne bir göz atın.

8.18. Bir güvenlik duvarının arkasındayım — diğer taraftaki insanların benim IP adresimi öğrenip benim sistemime bağlanmalarını nasıl sağlarım?

Maalesef bir güvenlik duvarının amacı tam da dışarıdaki insanların doğrudan sizim makinanıza bağlanmasını engellemektir. Bu şekilde içeri bağlanmaları çoğu durumda bir güvenlik açığı olarak kabul edilir.

Bu tabii ki talebinizin imkânsız olduğu anlamına gelmez yani gene de güvenlik duvarı üzerinde `connect()` yapabilirsiniz tabii bir "maskeleye" ya da "NAT" veya benzer bir şey söz konusu ise. Programlarınızı daima bağlantıyı sizin başlatacağınız şekilde tasarlayın, böylece işiniz kolay olur.

Eğer bu sizi tatmin etmiyorsa sistem yöneticilerinize size özel bir delik açmalarını söyleyebilirsiniz. Güvenlik duvarı, üzerindeki NAT yazılımı ile ya da vekil tarzı bir şey ile bunu gerçekleştirebilir.

Ancak lütfen unutmayın ki güvenlik duvarı üzerindeki bir delik hafife alınabilecek bir durum değildir. Kötü niyetli kişilere erişim hakkı vermediğinizden emin olmalısınız. Eğer bu konularda henüz acemiyseniz bana inanın ki güvenli programlar yazmak hayal edebileceğinizden çok daha zordur.

Sistem yöneticilerinizin benden nefret etmesine yol açmayın. ; -)

9. Son söz ve Yardım Çağrısı

Evet hepsi bu kadar. Umuyorum ki burada verilen bilgilerin en azından bir bölümü sizin için geçerlidir ve çok büyük hatalar yapmamışımdır. Aslında, eminim hata yapmışımdır.

Bunu bir uyarı kabul edin! Eğer bazı şeyleri net olarak aktaramayıp ya da tam olarak sizin sisteminize bire bir uyan örnekler veremeyip sizi üzdüysem özür dilerim ama bu yüzden beni suçlayamazsınız. Yani hukuki olarak bu belgedeki sözlerimin arkasında durduğum falan yok, çok ciddiyim. Yani bütün burada yazılanlar, tamamı, her bir sözcük, yanlış olabilir!

Fakat muhtemelen o kadar da yanlış değiller. Yani her şey bir yana şu ağ programlama mevzusu üzerine epey bir emek harcadığımı söyleyebilirim. Ofiste pek çok TCP/IP uygulaması geliştirdim, çok oyunculu oyun motorları yazdım, vs. Ancak gene de bir soket programlama tanrısı sayılmam. Sıradan bir adam olduğum bile söylenebilir.

Bu arada eğer yapıcı (ya da yıkıcı) herhangi bir eleştirisi olan varsa lütfen şu adrese e-mail göndersin ki ben de gerekeni yapayım: <beej (at) piratehaven.org>.

Bütün bunları neden yaptığımı merak ediyor musunuz? Hepsini para için yaptım. Ha ha! Hayır, hayır yani asıl sebep şu: pek çok kişi bana soket programlama ile ilgili bir sürü soru sorup duruyordu ve ben de onlara tüm cevapları tek bir belgede toplamayı düşündüğümü söylediğimde bana "Harika!" diyorlardı. Ayrıca zorluklarla elde edilmiş bu bilgi başkaları ile paylaşılmazsa bence heba olur. İnternet bunun için en uygun araç gibi görünüyor. Herkesi benzer şekilde bilgi paylaşımına davet ediyorum.

Bu kadar laf salatası yeter — şimdi biraz program yazmalıyım(m|z)! ; -)

Notlar

- Belge içinde dipnotlar ve dış bağlantılar varsa, bunlarla ilgili bilgiler buldukları sayfanın sonunda dipnot olarak verilmeyip, hepsi toplu olarak burada listelenmiş olacaktır.
- Konsol görüntüsünü temsil eden sarı zeminli alanlarda metin genişliğine sığmayan satırların sığmayan kısmı `␣` karakteri kullanılarak bir alt satıra indirilmiştir. Sarı zeminli alanlarda `␣` karakteri ile başlayan satırlar bir önceki satırın devamı olarak ele alınmalıdır.

(B3) <http://tangentsoft.net/wskfaq/>

(B4) <http://www.tuxedo.org/~esr/faqs/smart-questions.html>

(B5) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/send.2.inc>

(B6) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/recv.2.inc>

(B8) <http://www.rfc-editor.org/rfc/rfc793.txt>

(B9) <http://www.rfc-editor.org/rfc/rfc791.txt>

(B10) <http://www.rfc-editor.org/rfc/rfc768.txt>

(B11) <http://www.rfc-editor.org/rfc/rfc791.txt>

(1) Meraklısına not, Ağ Bayt Sıralaması aynı zamanda "Kıymetlisi Başta Bayt Sırası" (Big-Endian) ve Konak Bayt Sıralaması da "Kıymetlisi Sonda Bayt Sırası"(Little-Endian) olarak bilinir.

(2) Burada bulunmasının sebebi içinde bulunduğu veri yapısının boyunu `struct sockaddr` yapısının boyuna tamamlamaktır.

(3) Yani eğer `listen()` kullanacaksanız genellikle böyle bir şey yapmanız beklenir zaten. Mesela bir oyun sunucusuna "telnet x.y.z port 6969" şeklinde bağlanmanız söylendiğinde karşı tarafta tam da böyle bir hazırlık yapılmıştır.

(4) Ayrıntılı bilgi için bakınız: RFC-1413 (<http://www.rfc-editor.org/rfc/rfc1413.txt>.)

(B16) <http://www.rfc-editor.org/rfc/rfc1413.txt>

(B17) <http://www.ecst.csuchico.edu/~beej/guide/net/examples/getip.c>

(5) Sunucu makina ile **telnet**'i çalıştırdığınız makina aynı ise `localhost` ismini de kullanabilirsiniz.

(B19) <http://www.ecst.csuchico.edu/~beej/guide/net/examples/server.c>

(B20) <http://www.ecst.csuchico.edu/~beej/guide/net/examples/client.c>

(B21) <http://www.ecst.csuchico.edu/~beej/guide/net/examples/listener.c>

(B22) <http://www.ecst.csuchico.edu/~beej/guide/net/examples/talker.c>

(B23) <http://www.ecst.csuchico.edu/~beej/guide/net/examples/select.c>

(B24) <http://www.ecst.csuchico.edu/~beej/guide/net/examples/selectserver.c>

(B29) <http://linux.com.hk/man/showman.cgi?manpath=/man/man3/htonl.3.inc>

(B30) <http://linux.com.hk/man/showman.cgi?manpath=/man/man3/htons.3.inc>

(B31) <http://linux.com.hk/man/showman.cgi?manpath=/man/man3/ntohl.3.inc>

(B32) <http://linux.com.hk/man/showman.cgi?manpath=/man/man3/ntohs.3.inc>

(B33) http://linux.com.hk/man/showman.cgi?manpath=/man/man3/inet_aton.3.inc

(B34) http://linux.com.hk/man/showman.cgi?manpath=/man/man3/inet_addr.3.inc

(B35) http://linux.com.hk/man/showman.cgi?manpath=/man/man3/inet_ntoa.3.inc

(B36) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/socket.2.inc>

(B37) <http://linux.com.hk/man/showman.cgi?manpath=/man/man7/socket.7.inc>

(B38) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/bind.2.inc>

(B39) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/connect.2.inc>

- (B40) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/listen.2.inc>
- (B41) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/accept.2.inc>
- (B42) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/send.2.inc>
- (B43) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/recv.2.inc>
- (B44) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/sendto.2.inc>
- (B45) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/recvfrom.2.inc>
- (B46) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/close.2.inc>
- (B47) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/shutdown.2.inc>
- (B48) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/getpeername.2.inc>
- (B49) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/getsockname.2.inc>
- (B50) <http://linux.com.hk/man/showman.cgi?manpath=/man/man3/gethostbyname.3.inc>
- (B51) <http://linux.com.hk/man/showman.cgi?manpath=/man/man3/gethostbyaddr.3.inc>
- (B52) <http://linux.com.hk/man/showman.cgi?manpath=/man/man3/getprotobyname.3.inc>
- (B53) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/fcntl.2.inc>
- (B54) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/select.2.inc>
- (B55) <http://linux.com.hk/man/showman.cgi?manpath=/man/man3/perror.3.inc>
- (B56) <http://linux.com.hk/man/showman.cgi?manpath=/man/man2/gettimeofday.2.inc>
- (B57) <http://www.amazon.com/exec/obidos/ASIN/013490012X/beejsguides-«20>
- (B58) <http://www.amazon.com/exec/obidos/ASIN/0130810819/beejsguides-«20>
- (B59) <http://www.amazon.com/exec/obidos/ASIN/0130183806/beejsguides-«20>
- (B60) <http://www.amazon.com/exec/obidos/ASIN/0139738436/beejsguides-«20>
- (B61) <http://www.amazon.com/exec/obidos/ASIN/0138487146/beejsguides-«20>
- (B62) <http://www.amazon.com/exec/obidos/ASIN/0201633469/beejsguides-«20>
- (B63) <http://www.amazon.com/exec/obidos/ASIN/020163354X/beejsguides-«20>
- (B64) <http://www.amazon.com/exec/obidos/ASIN/0201634953/beejsguides-«20>

(B65) <http://www.amazon.com/exec/obidos/ASIN/1565923227/beejsguides-20>

(B66) <http://www.amazon.com/exec/obidos/ASIN/0201563177/beejsguides-20>

(B67) <http://www.cs.umn.edu/~bentlema/unix/>

(B68) <http://www.ibrado.com/sock-faq/>

(B69) <http://pandonia.canberra.edu.au/ClientServer/>

(B70) gopher://gopher-chem.ucdavis.edu/11/Index/Internet_aw/Intro_the_Internet/intro.to.ip/

(B71) <http://www-iso8859-5.stack.net/pages/faqs/tcpip/tcpipfaq.html>

(B72) <http://tangentsoft.net/wskfaq/>

(B73) <http://www.rfc-editor.org/>

(B74) <http://www.rfc-editor.org/rfc/rfc768.txt>

(B75) <http://www.rfc-editor.org/rfc/rfc791.txt>

(B76) <http://www.rfc-editor.org/rfc/rfc793.txt>

(B77) <http://www.rfc-editor.org/rfc/rfc854.txt>

(B78) <http://www.rfc-editor.org/rfc/rfc951.txt>

(B79) <http://www.rfc-editor.org/rfc/rfc1350.txt>

(6) Ç.N.: **lsuf** komutu bu konuda size yardımcı olabilir, ayrıntılar için man sayfasına bakın. :-)

(7) Ç.N.: Yazarın burada tek bir cümle olarak söyleyip geçtiği şey güvenlik açısından görüldüğünden çok daha fazla önemlidir, kullanacağınız şifreleme algoritmasına çok dikkat edin ve **mutlaka şifrelemeden önce sıkıştırma işlemini uygulayın.**

(B89) <http://www.openssl.org/>

Bu dosya (bgnet.pdf), belgenin XML biçiminin \TeX Live ve belgeler-xsl paketlerindeki araçlar kullanılarak PDF biçimine dönüştürülmesiyle elde edilmiştir.

27 Şubat 2007