Karadeniz Teknik Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü
Bilgisayar Sistemleri Laboratuarı

İSTEMCİ VE SUNUCU MİMARİLERİ

İstemci ve sunucu mimarileri gerek bilgisayar alanında gerekse günlük hayatımızda yaygın olarak kullanılmaktadır. Her gün kullandığımız telefon sistemini düşünecek olursak, taraflar arası haberleşme bir şahıs tarafından başlatılır. Diğer şahıs ise çalan telefonu açarak gelen haberleşme talebini cevaplayan taraf olacaktır. Internet üzerinden haberleşmede benzer şekilde gerçekleştirilmektedir. İstemci, haberleşmeyi başlatan taraf iken, sunucu pasif olarak bağlantı talepleri için bekleyen ve gelen istemci taleplerini karşılayan taraf olacaktır. İstemci, sunucunun adresini ve sunucu üzerinde çalışmakta olan programın port adresini bilmek zorundadır. Telefon haberleşmesi örneğine dönecek olursak, telefon çağrısı yapacak olan kimse karşı tarafın telefon numarasını bilmek zorundadır.

Peki ama istemci, sunucunun IP adresini nasıl bilir? Genelde istemci, bağlanmak istediği sunucu bilgisayarın kanonik ismini bilir ve isim çözümleme servisini (DNS Domain Name Server) kullanarak isme karşılık düşen IP adresini elde eder. Sunucu da çalışan programın port numarasını bulmak biraz daha farklıdır. Prensipte, sunucular herhangi bir port numarasını kullanabilir. Fakat kullanmış olduğu port numarası istemciler tarafından da bilinmek zorundadır. Internette, önceden bilinen uygulamalar belirli port numaralarına karşı düşürülmüştür. Internet Assigned Number Authority (IANA) isimli kurum hangi servisin hangi portu kullanacağını önceden belirlemiştir. Örneğin 80 numaralı port Hyper Text Transfer Protokolü (http) tarafından kullanılmaktadır.

A. Soket nedir ?

Soket, uygulamaların üzerinden veri gönderip alabileceği soyutlama katmanıdır. Soketler, uygulamaların ağ üzerindeki diğer uygulamalarla haberleşebilmesini olanaklı kılar. Bir makine üzerindeki uygulama programı tarafından sokete yazılan bilgi, ağ üzerindeki başka bir makine tarafından okunabilir.

TCP/IP mimarisi ile kullanılan iki tip soket mevcuttur: Stream soket ve datagram soket. Stream soketler, veri haberleşmesinde uçtan uca protokol olarak TCP'yi kullanır ve böylece güvenilir veri transferi sağlar. Datagram soketler ise veri haberleşmesinde UDP'yi kullanır. Bu tür soketlerde gönderilen verinin karşı tarafa ulaşması garanti edilemez. 65500 byte uzunluğuna kadar veri içeren mesajlar, bu soketlerin kullanımı ile gönderilebilir. TCP/IP soketleri, temelinde 3 unsur tarafından tanımlanır: Internet adresi (IP), kullanılan veri haberleşme türü (TCP veya UDP), port numarası.

B. Soketler ile kullanılan veri yapıları ve fonksiyonlar

Bu bölümde soket arabirimi tarafından kullanılan veri yapılarından ve bazı sistem çağrılarından bahsedilecektir. İstemci ve sunucu tarafında kullanılan sistem çağrıları farklılık gösterebilir fakat kullanılan veri yapıları ortaktır. Bölümün ilerleyen kısmında görecek olduğumuz bind(), listen(), accept() sistem çağrıları sunucu tarafında, connect() çağrısı istemci tarafında kullanılır. send(), recv(), sendto(), recvto() sistem çağrıları ise istemci ve sunucu arasındaki veri haberleşmesini sağlar. Hem istemci hem sunucu soket oluşturabilmek için sock() sistem çağrısına ihtiyaç duyar. Sistem çağrıları ile beraber kullanılan iki temel veri yapısı sockaddr ve sockaddr_in olup tanımları aşağıdaki şekilde verilmiştir. sockaddr isimli veri yapısı, soketler için adres bilgisini tutmakla görevlidir.

```
struct sockaddr {
  unsigned short sa_family; // adres ailesi, AF_xxx. Genelde AF_INET olarak kullanılır.
       char sa_data [14];//14 byte protokol adresi.Hedef adresi ve port numarasını tutar.
       }

  struct sockaddr_in {
       short int sin_family; // Adres ailesi
       unsigned short int sin_port;//Port numarası.Network byte order düzeninde olmalıdır.
       struct in_addr sin_addr; // Internet adresi.Network byte order düzeninde olmalıdır.
```

```
        unsigned char sin_zero [8]; // memset fonksiyonu ile sıfırlanmalıdır
};
```

Peki ama in_addr veri yapısı nasıl Network byte order düzeninde olur ? ilgili veri yapısının tanımı aşağıdaki şekildedir

```
struct in_addr {
  unsigned long s_addr; // 32 bit uzunluğundadır.
};
```

Varsayalım ki sockaddr_in tipinde ina isimli bir değişken tanımlanmış olsun. ina.sin_addr.s_addr 4 byte lık IP adresini referanslar. Veri yapısına yerleştirilecek olan IP adresi "10.12.110.57" olsun. inet_addr () isimli fonksiyonu kullanarak, noktalarla ayrılarak verilen string türündeki IP adresini unsigned long formuna dönüştürülebilir.

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

inet_addr() fonksiyonu, noktalı formda verilen IP adresini, Network Byte Order düzenine döndürür. Soket oluşturmadan önce, sokete ilişkin bilgilerin şu ana kadar adı geçen veri yapılarına nasıl yerleştirileceğine dair bir örnek aşağıdaki şekildedir. Verilmiş olan örnekten, 10.12.110.57 IP adresindeki 3490 nolu port üzerinden soket haberleşmesi gerçekleştirileceği anlaşılabilir.

```
struct sockaddr_in my_addr;
my_addr.sin_family = AF_INET; // host byte order
my_addr.sin_port = htons(3490); // short, network byte order
my_addr.sin_addr.s_addr = inet_addr("10.12.110.57");
memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct
```

Sistem çağrıları ile kullanılan veri yapılarını verdikten sonra, soket oluşturmada kullanılan sistem çağrısı socket()'den bahsedilecektir. int socket (int domain, int type, int protocol) sistem çağrısı, soket tanımlamada kullanılmaktadır. ilk argümanı "AF_INET" olarak verilmelidir. type argümanı kullanılan soketin, stream soket mi datagram soket mi olduğunu söyler (SOCK_STREAM veya SOCK_DGRAM). Son argüman 0'a setlenerek, sistemin ikinci argümana uygun bir protokol seçmesi sağlanır. Sistem çağrısının başarılı bir şekilde icrasının ardından, soket tanımlayıcısı geri döndürülür.

Bir soket oluşturduktan sonra, eğer sunucu tarafındaysanız, ilgili soketi lokal makinenizdeki bir portla ilişkilendirmeniz gerekir. Port numarası, kernel tarafından, gelen paketleri belirli servislerle ilişkilendirmek için kullanılır. bind() sistem çağrısı sunucu tarafta kullanılan bir fonksiyon olup tanımı aşağıdaki şekildedir.

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

İlk argüman, socket() çağrısının geri döndürdüğü soket tanımlayıcısıdır. my_addr, sockaddr veri yapısına göstericidir. Bu veri yapısı sizin port ve IP adresiniz hakkında bilgi içerir. Son argüman ise sockaddr veri yapısının büyüklüğüne setlenir (sizeof (struct sockaddr)).

Aşağıda verilmiş sunucuda çalışan program kesitinde, sockaddr_in veri yapısına oluşturulacak olan sokete ilişkin veriler yerleştirildikten sonra, socket() sistem çağrısı ile soket oluşturulmakta ve ardından sunucu üzerindeki bir portla soketin ilişkilendirmesi (binding) gerçekleştirilmektedir.

```
main()
{
int sockfd;
struct sockaddr_in my_addr;
sockfd = socket(AF_INET, SOCK_STREAM, 0);
my_addr.sin_family = AF_INET; // host byte order
my_addr.sin_port = htons(MYPORT); // short, network byte order
my_addr.sin_addr.s_addr = inet_addr("10.12.110.57");
memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct
bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));
```

.
.
.

Sunucu tarafındaki program, bind() sistem çağrısı ile soket ve portu ilişkilendirdikten sonra, ilgili soket üzerinden istemcilerden gelen talepleri listen() sistem çağrısını kullanarak dinlemeye başlayacaktır.

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

İlk argüman socket() çağrısı ile dönen soket göstericisidir. İkinci argüman, aynı anda izin verilen maksimum bağlantı sayısını gösterir. Gelen bağlantılar, ta ki accept() çağrısı ile beraber sunucu tarafından kabul edilinceye kadar kuyrukta beklerler. İkinci argüman kuyrukta beklemede tutulabilecek maksimum bağlantı sayısını gösterir. Talepler sunucu tarafından accept() edilinceye kadar kuyrukta beklemeye alınacaktır. Taleplerden biri accept() edildiğinde, bu bağlantı ile ilişkili yeni bir soket tanımlayıcısı sistem çağrısı tarafından döndürülecektir.

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

İlk parametre dinlemede olan soket tanımlayıcısıdır. İkinci argüman, sockaddr_in veri yapısına göstericidir. Bu veri yapısı gelen bağlantı hakkında bilgi yerleştirmek için kullanılır. Son argüman ise, accept() çağrısından önce değerinin sizeof(struct sockaddr_in)'e setlenmesi gereken değişkene göstericidir. accept() çağrısı, talebi kabul edilen istemci ile sunucu arasında oluşturulan yeni lokal soket tanımlayıcısını geri döndürür. Bu aşamada sockfd, sunucunun gelecek diğer istemci taleplerini beklemede olduğu soket tanımlayıcısıdır.

Sunucu üzerinde çalışacak olan programın yapması gerekenler özetlenecek olursa:

1. socket() çağrısı ile TCP soketinin oluşturulması.
2. bind() sistem çağrısı ile beraber, soketin bir port numarası ile ilişkilendirilmesi.,
3. listen() sistem çağrısı ile sunucunun belirtilen port üzerinde istemcilerden gelen talepleri dinlemesi.
4. Aşağıda verilmiş olan adımların sonsuz döngü halinde tekrarlanması:

- Gelen her istemci talebinin accept() ile kabul edilmesi ve her istemci için yeni bir soket oluşturulması.
- İstemci ile oluşturulan yeni soket üzerinden send() ve recv() komutları yardımıyla haberleşilmesi.
- close() sistem çağrısı ile istemci olan bağlantının koparılması.

İstemci tarafında çalışmakta olan program ise kısaca 4 adımda özetlenebilir.
1. socket() sistem çağrısı ile beraber TCP soketin oluşturulması.
2. connect() sistem çağrısı ile sunucuya bağlanılması.
3. send() ve recv() çağrıları ile haberleşmenin sağlanması.
4. Bağlantının close() çağrısı ile sonlandırılması.

İstemcinin uzaktaki sunucu üzerinde çalışan bir programa bağlantı açması için kullanılan connect() çağrısının tanımı aşağıdaki şekildedir.

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

İlk argüman elde edilen soket tanımlayıcısıdır. İkinci argüman bağlanılacak IP adresi ve port numarası hakkında bilgi içeren sockaddr veri yapısıdır. Son argüman ise aynı veri yapısının büyüklüğüdür. Aşağıdaki örnek program kodunda istemcinin 10.12.110.57 IP adresinde 23 nolu portta dinlemede olan sunucuya nasıl bağlanıldığı gösterilmiştir.

```
int sockfd;
struct sockaddr_in dest_addr; // will hold the destination addr
sockfd = socket(AF_INET, SOCK_STREAM, 0); // do some error checking!
dest_addr.sin_family = AF_INET; // host byte order
```

```
dest_addr.sin_port = htons(23); // short, network byte order
dest_addr.sin_addr.s_addr = inet_addr("10.12.110.57");
memset(&(dest_addr.sin_zero), '\0', 8); // zero the rest of the struct
connect(sockfd, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr));
```

İstemci ve sunucu arasında bağlantının kurulmasının ardından, send() ve recv() çağrıları ise açılmış olan stream soketleri üzerinden veri transferi sağlanır.

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

send() sistem çağrısının ilk parametresi verinin gönderilecek olduğu soket bağlantısının tanımlayıcısıdır. İkinci parametre gönderilecek olan mesajın bulunduğu buffer'ın göstericisi, üçüncü parametre ise buffer'ın büyüklüğüdür. Deney süresince kullanılan send() sistem çağrılarında bayrak değeri 0 olarak alınacaktır. Fonksiyonun geri dönüş değerinin -1 olması hata durumunu gösterir. recv() sistem çağrısında, ikinci ve üçüncü parametre, gelen mesajların yerleştirileceği buffer'ı ve büyüklüğünü içerir. Eğer gelen herhangi bir mesaj yok ise, ta ki mesaj gelinceye kadar çağrı bloklanır. Fonksiyonun geri dönüş değeri alınan mesajın büyüklüğüdür. Herhangi bir hata olması durumunda fonksiyon -1 değeri döndürecektir.

Soketlerin kullanımına ilişkin örnek uygulama olarak echo sunucu ve echo istemci verilmiştir. Echo sunucu kendine bağlanan istemcilerden gelen bilgiyi aynen geri yansıtır. Uzaktaki sunucuların ayakta olup olmadığının testinde daha çok kullanılan Echo sunucular, güvenlik acizlikleri nedeniyle günümüzde tercih edilmemektedir. TCPEchoServer.c ve TCPEchoClient.c isimli dosyalarda sunucu ve istemci programlarına ilişkin kaynak kodlar yer almaktadır.

Sunucu tarafta çalışan TCPEchoServer isimli program kısaca özetlenecek olursa:

1. Program kurulumu ve parametrelerin parse edilmesi.
- Port numarası string değerden int değere atoi () ile çevrilir.

2. Soket oluşturulması ve kurulumu:
- TCP soketi oluştur.
- Sunucunun IP adresi ve hizmetin çalışacak olduğu port numaraları ile sockaddr_in veri yapısını tanımla.
- Oluşturulan TCP soketi, ilgili IP adresi ve port numarasına sockaddr_in veri yapısını kullanarak bind() et.
- Soketi listen() çağrısı ile beraber dinleme modunda aç.

3. Sonsuz döngü içinde gelen talepleri kontrol et.
- gelen bağlantıları accept() et.
- bağlanan istemcileri rapor et.
- HandleTCPClient() soket üzerinden gelen veriyi alan ve aynı soket üzerinden geri döndüren bir fonksiyondur.

İstemci tarafından çalışan TCPEchoClient.c isimli kaynak dosyasını adım adım inceleyecek olursak:

1. Uygulamanın kurulması ve argümanların pars edilmesi.
2. socket() sistem çağrısı ile TCP soketin oluşturulması.
3. Sunucunun adres ve port bilgisini içeren veri yapısının (sockaddr_in) hazırlanması ve connect() çağrısı ile bağlantının kurulması.
4. Echo stringinin sunucuya send() çağrısı ile gönderilmesi.
5. Echo sunucunun cevaplarının alınması: 53-70
- recv() çağrısı taki karşı taraftan bilgi gelinceye kadar bloklanır. Gelen bilgi buffer'a kopyalanır.
- sunucudan gelen bilgi olduğu gibi ekrana basılır.
- gönderdiğimiz kadar bilgi aldığımız zaman döngüden çıkılır ve yeni satır karakteri basılır.

6. Bağlantıyı sonlandır ve çık.

Sunucu aynı anda birden çok istemcinin talebine cevap vermek zorundadır. Yukarıda içeriğinden kısaca bahsedilmiş olan sunucu programı ise aynı anda yalnız bir istemciye hizmet vermektedir. Sunucular, aynı anda gelen istemci taleplerine karşılık verebilmek için farklı stratejiler izlemektedir. Bir sonraki bölümde bu stratejiler ve aralarındaki farklılıklardan bahsedilecektir.

C. Bağlantı tabanlı sunucu stratejileri

Sunucu, istemcilerden gelen aynı andaki talepleri yönetmek için farklı stratejiler kullanabilir. Bu bölümde sunucunun kullanacak olduğu üç farklı stratejiden bahsedilecektir.

C.1. Seri Sunucu Stratejisi

Bir önceki bölümde görmüş olduğumuz Echo sunucusu bir anda tek bir istemciden gelen talebi yönetmektedir. İstemcilerden birine hizmet verilirken başka bir istemcinin sunucuya bağlanması durumunda, sunucu ilk gelen istemciyle haberleşmesini tamamlamadan onlara hizmet veremeyecektir. Bu tip soket uygulamaları seri sunucu olarak adlandırılır. Seri sunucular, istemcilerin kısa süreli servis istediği durumlarda en iyi hizmeti verecektir. İlk bağlantı kuran istemciye verilecek hizmetin uzun sürmesi durumunda, diğer istemcilerin hizmet için uzun bir süre beklemesi gerekecektir. Aşağıda verilmiş olan yalancı kod ifadesi seri sunucu stratejisini açık olarak göstermektedir:

```
for ( ; ; ) {
        dinlemedeki dosya gösterici üzerinde istemci taleplerini bekle.
        istemci ile aranda özel iki taraflı bağlantı kur.
        while (özel bağlantı kanalında problem olmadığı müddetçe)
                istemcinin isteklerini al.
                taleplerini gerçekleştir.
                dosya göstericisi ile tanımlanan özel kanalı kapat.
}
```

C.2. Ebeveyn sunucu stratejisi

İstemcilere dosya transferi gibi uzun sürebilecek hizmetler veren sunucular için seri sunucu stratejisi çalışmayacaktır. Bunun yerine alternatif yöntem, ebeveyn sunucunun talepte bulunan istemcilere hizmet verebilmek için fork() çağrısıyla çocuk süreçler oluşturması yada her bir istemci için ayrı bir thread oluşturmasıdır. Dosya transferi gibi sunucuyu uzun süre başka istemcilere hizmet veremeyecek şekilde bloklayabilecek işlemler için ebeveyn sunucu yaklaşımını kullanmak avantajlı olacaktır. Aşağıdaki yalancı kod ifadesi ebeveyn sunucu stratejisini göstermektedir:

```
for( ; ; ) {
        dinlemedeki dosya gösterici üzerinde istemci taleplerini bekle.
        istemci ile aranda özel iki taraflı bağlantı kur.
        istemcinin taleplerini yönetmek için çocuk süreç oluştur.
        dosya göstericisi ile tanımlanan özel kanalı kapat.
        Zombie süreçleri sonlandır.
}
```

Aşağıdaki yalancı kod ifades ise çocuk sürecin sırasıyla gerçekleştirecek olduğu adımları gösterir:

```
Talep dinlemede kullanılan dosya göstericisini kapat.
İstemcinin taleplerini karşıla.
dosya göstericisi ile tanımlanan özel kanalı kapat.
kendini sonlandır. (exit)
```

Sunucu program, gelen her bir istemci talebi için yeni bir çocuk süreç oluşturacak şekilde fork() sistem çağrısını kullanır. Önceden örneği verilmiş olan Echo sunucu bu bölümde istemcilerden gelen birden çok talebi karşılayacak şekilde, ebeveyn sunucu stratejisi ile değiştirilmiştir. TCPEchoServer-Fork.c isimli dosyadaki kaynak kodu satır satır inceleyecek olursak:

- waitpid() sistem çağrısını kullanabilecek şekilde ek başlık dosyaları.

- Sunucu soketi oluşturulur. SetupTCPServerSocket() isimli fonksiyon soketi oluşturur, belirli bir port ile ilişkilendirir ve istemcilerden gelen talepleri kabul edecek şekilde dinlemeye açar.
- Süreçlerin sayısı childProcCount ile kontrol edilir.
- Ana süreç sonsuz döngü içerisinde koşar, gelen her talep için fork() sistem çağrısı ile bir çocuk süreç oluşturur.

    1. AcceptTCPConnection() fonksiyonu taki geçerli bir bağlantı kuruluncaya kadar bloklanır ve bağlantı kurulduğunda soket tanımlayıcısını döndürür.
    2. Yeni bağlantıyı yönetmek için fork() çağrısı ile çocuk süreç oluşturulur. Çocuk süreç soket tanımlayıcısını kopyalayarak alır. Böylece fork() çağrısından sonra hem çcocuk hem ebeveyn süreç dinlemede olan sokete tanımlayıcıyı (servSock) ve yeni oluşturulan çocuk sürecin kullanacağı soket tanımlayıcısını (clntSock) barındırır.
    3. Çocuk süreç kendisine kopyalama yoluyla gelen ve dinlemede kullanılan soket tanımlayıcısını kapatır. Fakat ebeveyn süreç hala dinlemedeki soket için tanımlayıcıya sahiptir. Close() sistem çağrısı, eğer ilgili soket üzerinde dinlemede başka süreç yoksa soketi kapatacaktır. Çocuk süreç ardından HandleTCPClient() fonksiyonunu çağırır. İstemciye hizmet sunulmasının ardından close() sistem çağrısı ile çocuk süreç ile ilişkilendirilmiş soket tanımlayıcısını kapatır. Çocuk süreç exit() sistem çağrısı ile sonlanır.
    4. Ebeveyn süreç icraya devam eder. Her bağlantı talebinden sonra, ebeveyn süreç çocuk süreç sonlanmalarından kaynaklanan zombileri temizler.

C.3. Çok görevli sunucu stratejisi

Ebeveyn sunucu stratejisinden farklı olarak çok görevli sunucu (threaded server) stratejisi, sisteme daha az yük getirecektir. Sunucu, istemcilerden gelen talepler için yeni çocuk süreçler oluşturmak yerine, kendi adres uzayında yeni thread'ler oluşturacaktır. Çok görevli sunucu yaklaşımı özellikle küçük ve I/O bağımlı işlerde faydalı olacaktır. adres uzayının paylaşımlı olmasından kaynaklanan birçok talep arasında karışıklık mümkün olabilir. Hesaplama yükü olan servislerde, ek threadler ana sunucu threadini bloklayabilir yada verimliliğini etkileyebilir. Süreç başına düşen açok dosya gösterici sayısı limiti aynı anda hizmet verilebilecek istemci sayısını etkileyebilir. Aşağıdaki yalancı kod ifadesi çok görevli sunucu stratejisini göstermektedir:

```
for ( ; ; ) {
        dinlemedeki dosya gösterici üzerinde istemci taleplerini bekle.
        istemci ile aranda özel iki taraflı bağlantı kur.
        istemcinin talebini yönetecek şekilde bağımsız bir thread oluştur.
}
```

İstemci taleplerinin her biri için bir çocuk sürecin oluşturulması sistem kaynakları açısından pahalıya mal olabilir. Her çocuk süreç oluşturulduğunda, işletim sistemi ebeveyn sürecin belleğini, yığınını, dosya ve soket göstericilerini çocuk sürecin uzayına kopyalar. Threadler aynı süreç uzayını kullanarak böyle bir sistem kaynağı maliyetine engel olurlar. Yeni oluşturulan thread ana süreç ile aynı adres uzayını paylaşır. TCPEchoServer-Thread.c isimli kaynak kod dosyası çok görevli sunucu stratejisinin nasıl gerçekleştirildiğini göstermektedir. Kaynak kodu inceleyecek olursak:

- POSIX threadlerini kullanabilecek şekilde eklenen yeni başlık dosyası.
- Threadler için kurulumun tamamlanması. ThreadMain() fonksiyonu POSIX threadinin icra edecek olduğu fonksiyondur. pthread_crate() fonksiyonu thread'in gerçekleştirecek olduğu fonksiyonun parametre olarak geçirilmesini olanaklı kılar. ThreadArgs veri yapısı gerçek parametre listesini belirtir. Thread'in oluşturacak olan süreç, pthread_create() fonksiyonunu çağırmadan önceden ilgili veri yapısını ayırır. Bu programda, thread fonksiyonu yalnızca clntSock argümanına ihtiyaç duyar.
- İstemci soket tanımlayıcısı yeni thread'e argüman veri yapısı kullanılarak geçirilir.
- Yeni thread çağrılır.
- Thread'in icrası: ThreadMain, pthread_create() fonksiyonu tarafından yeni bir thread oluşturulduğunda çağrlır. Thread'in ana fonksiyonu icrasını tamamladığında, ThreadArgs veri

yapısından parametreler çıkarılır. Thread fonksiyonu HandleTCPClient() fonksiyonunu çağırır. Ana süreç thread ile tekrar haberleşmeye ihtiyaç duymadığı için, thread NULL gösterici döndürür.

Ana süreç ve thread aynı adres uzayını paylaştığı için, ana süreç ve istemciler için oluşturulan threadler dinlemedeki soketleri kapamazlar. Çocuk süreçler yerine threadleri kullanmanın birkaç dezavantajı vardır.

1. Eğer çocuk süreç de bir problem oluşursa, komut satırından çocuk sürecin ID değeri kullanılarak monitör edilebilir yada öldürülebilir.
2. Eğer işletim sistemi çok görevli bir sunucunun çalışmakta olduğundan habersiz ise, sistemde oynanan bir oyun programı ile çok görevli sunucu yazılımı eşit önceliği alır. Altında birden çok süreç çalışan sunucu sürece ayrılan bu süre yeterli olmayabilir.

Deney Hazırlığı

1. Sunucu ve istemci mimarileri hakkında genel bir bilgi edininiz.
2. socket(), connect(), bind(), listen(), accept(), send(), recv() sistem çağrılarının kullanımı hakkında bilgi sahibi olunuz.
3. Linux makine üzerinde birden fazla kaynak dosyanın tek bir çalıştırılabilir dosya oluşturacak şekilde gcc'nin nasıl kullanıldığına dair bilgi sahibi olunuz.
4. fork() sistem çağrısı ve pthread kütüphanesi hakkında bilgi edininiz.
5. Çocuk süreç ve thread'ler hakkında bilgi sahibi olunuz.
6. netstat komutunu ve çıktısını yorumlayabilecek düzeyde olunuz.
7. C programlarına, komut satırından nasıl parametre geçirildiği hakkında bilgi sahibi olunuz.

Deney Tasarımı ve Uygulaması

1. Putty programı ile "ktuce.ktu.edu.tr" isimli sunucu bilgisayarına "lab" kullanıcı ismi ile üç farklı SSH oturumu açın. Oturum açmakta kullanılan putty terminal pencereleri T1, T2 ve T3 ile isimlendirilsin.

2. socket_deney isimli klasörün altında çeşitli kaynak dosyalar bulunmaktadır. Öncelikle T1 terminalinden seri sunucu yazılımını derlemek için gerekli işlemleri gerçekleştirin. TCPEchoServer.c isimli dosya kendi içinde DieWithError.c ve HandleTCPClient.c isimli C dosyalarını çağırmaktadır. Aşağıdaki satırın yardımıyla seri sunucuyu derleyip Serial_Server isimli çalıştırılabilir bir dosya oluşturalım.

% gcc –o Serial_Server TCPEchoServer.c DieWithError.c HandleTCPClient.c

3. T2 terminalinde seri sunucuya bağlanacak olan istemci yazılımını derleyip Client isimli çalıştırılabilir bir dosya oluşturalım.

% gcc –o Client TCPEchoClient.c DieWithError.c

4. T1 terminali üzerinden, ana makine üzerindeki (ktuce makinesi) seri sunucuyu aynı makinedeki 2222 nolu port üzerinden dinlemeye açalım.

% ./Serial_Server 2222

5. T2 terminalinden netstat –an | grep 2222 yazarak ilgili portun "LISTENING" modunda açılıp açılmadığını görelim. Ardından ifconfig komutunu çağırarak ana makinenin IP adresini öğrenelim. Ana makinenin IP adresi "xxx.yyy.zzz.ttt" ile gösterilsin.

6. T2 terminalinden aynı makine üzerinde 2222 nolu port üzerinde dinlemede olan sunucu yazılımına herhangi bir karakter dizisi gönderelim. İstemci yazılımının ilk parametresi bağlanılacak olan

sunucunun IP adresi, ikinci parametresi gönderilecek olan karakter dizisi ve üçüncü parametre ise sunucu yazılımın dinlemede olduğu port değeridir.

% ./Client xxx.yyy.zzz.ttt "KTU CENG" 2222

7. Gönderilen ve geri alınan karakter dizilerini karşılaştırın. T1 ekranında sunucuya bağlanan istemcilerin IP numaraları ve gönderilen karakter dizileri gözükmektedir. T2 ekranındaki istemcinin sunucuyla olan bağlantısı koptuktan sonra, sunucunun aynı port üzerinde yeni talepler için beklemede olduğunu, netstat komutu yardımıyla gözlemleyin.

8. Seri sunucu mimarisinde, bir anda yalnız bir istemciye hizmet verildiğini gözlemlemek amacıyla, T3 terminalinde yeni bir istemci program koşulacaktır. "squeeze" isimli istemci program, kullanıcı tarafından belirlenen sayıda paketi, kullanıcı tarafından belirlenen zaman aralıklarıyla sunucuya göndermektedir. Böylelikle squeeze istemcisi, sunucu zamanını kendinde tutarak yeni istemcilerin sunucuya bağlanmasına engel olmaktadır. Çünkü sunucu yazılımı seri sunucu mekanizmasını kullanmaktadır. Squeeze ismli istemci programını T3'de derleyelim.

% gcc -o squeeze squeeze.c DieWithError.c

9. Seri sunucuya 1 sn aralıklarla 20 paket gönderelim.

% ./squeeze xxx.yyy.zzz.ttt 2222 20 1

10. Aynı anda T2 terminalinde Client isimli istemci programi ile sunucu yazılıma bağlanmaya çalışalım.

% ./Client xxx.yyy.zzz.ttt "KTU CENG" 2222

11. T2 terminalindeki istemcinin, sunucudan hizmeti ancak T3'de çalışan squeeze isimli istemcinin talebi karşılandıktan sonra alabildiğine dikkat edin.

12. T1 terminalinde çalışmakta olan seri sunucu ctrl+c ile sonlandırılır. Bir sonraki aşamada seri sunucu yerine ebeveyn-sunucu stratejisini kullanan sunucu yazılımı aşağıdaki gibi T1 terminali üzerinde derlenecektir.

% gcc -o Fork_Server TCPEchoServer-Fork.c AcceptTCPConnection.c DieWithError.c HandleTCPClient.c CreateTCPServerSocket.c

13. Bağlanılan istemcilerin her biri için çocuk süreç oluşturan, Fork_Server isimli ebeveyn istemci yazılımı 2222 nolu port üzerinde dinleme yapacak şekilde koşturulur. T2 terminalinden netstat -an|grep 2222 ile ilgili portun dinlemeye açılıp açılmadığı tespit edilir.

% ./Fork_Server 2222

14.

- T2 terminali üzerinden ./Client xxx.yyy.zzz.ttt "KTU CENG" 2222 yazdığımızda T1 terminalinde ilgili istemci için oluşturulan çocuk sürecin ID değeri verilmektedir (o anki istemci için oluşturulan ID değeri 3950 olsun).
- T2 terminali üzerinde ps -aux | grep Z+ yazdığımızda, sistemdeki Zombie süreçler görüntülenecektir. Bunlar arasında 3950 ID değerine sahip çocuk süreç de gözükecektir.
- T2 terminalinden tekrar istemci programına ./Client xxx.yyy.zzz.ttt "KTU CENG" 2222 ile bağlanılsın. Bu durumda T1 terminali üzerinde yeni bağlanan istemciye atanan çocuk sürecin ID değeri 3955 olsun.
- T2 terminali üzerinden ps -aux | grep Z+ yazdığımızda artık 3950 ID'sine sahip çocuk süreç zombie olarak gözlemlenmeyecektir. Yeni çocuk sürecin ID'si olan 3955 Zombie süreç olarak gözükecektir. Taki bir sonraki istemci bağlanıncaya kadar.

15. T1 ekranında Fork_Server isimli sunucu yazılımı çalışıyorken, T2 terminalinden squeeze isimli istemci programı koşulur.

% ./squeeze xxx.yyy.zzz.ttt 2222 20 1

16. Aynı anda T3 terminalinden de Client isimli istemci programı koşulsun.

./Client xxx.yyy.zzz.ttt "KTU CENG" 2222

17. T3 de çalışan ve T2'dekine göre daha kısa sunucu zamanı gerektiren istemci, kendisine atanan çocuk süreç aracılığıyla istemciden hizmet almıştır. Yani sunucu zamanı T2 ve T3 arasında çocuk süreçler aracılığıyla paylaşımlı olarak kullanılmıştır. T3'deki istemcinin hizmet alabilmesi için, T2'deki istemcinin sunucu ile bağlantısının kesilmesini beklemesine gerek kalmamıştır.

18. T2 ve T3 terminallerinin her ikisinde, işlemci zamanı gerektiren squeeze istemcisi 15'deki parametrelerle koşulsun. Ebeveyn-sunucu yazılımının her iki sürece de hizmet verdiği gözlemlenebilir.

19. T1 terminalinde çok görevli sunucu stratejisini uygulamak amacıyla TCPEchoServer-Thread.c isimli kaynak dosyanın çalıştırılabilir dosyası oluşturulur.

% gcc -o Thread_Server TCPEchoServer-Thread.c AcceptTCPConnection.c CreateTCPServerSocket.c DieWithError.c HandleTCPClient.c -pthread

20. Oluşturulan çalıştırılabilir sunucu yazılımı dosyası T1 terminalinde koşulur. T2 terminalinden netstat -an|grep 2222 yazarak, 2222 nolu portun dinlemeye açılıp açılmadığı kontrol edilir.

% ./Thread_Server 2222

21. Çok görevli sunucu stratejisini gözlemleyebilmek amacıyla T2 ve T3 terminallerinde squeeze istemcisi 15'deki parametrelerle beraber koşulur. T1 terminalinde her bir istemci için oluşturulan thread'lerin ID leri verilmektedir. Fakat ebeveyn sunucu stratejisinden farklı olarak Thread'lerin koşmaya başlamasından sonra sonlandırılmaları olanaksızdır. Bunu gösterebilmek amacıyla T1 terminalindeki çalışmakta olan sunucu Ctrl+c ile beraber sonlandırılıp, ebeveyn sunucu çalıştırılsın.

%./Fork_Server 2222

22. T2 terminalinden 15'deki parametreleri kullanan istemci sunucuya bağlansın. T1 terminalinde istemci ile ilişkili olarak oluşturulan çocuk sürecin ID değeri gösterilecektir. ID değerinin 9999 olduğu varsayılsın.

23. T3 terminalinden, çalışmakta olan çocuk süreci kill komutu ile sonlandırmak mümkün olacaktır.
% kill -9 9999

Deney Soruları

1. Seri sunucu stratejisini kullanan sunucu yazılımının (TCPEchoServer.c), aynı anda hizmet verebilecek olduğu istemci sayısını, komut satırından parametre olarak alacak şekilde düzenleyiniz. Yeni oluşturduğunuz sunucu dosyasını TCPEchoServer_grupno.c olarak veriniz.

2. Seri sunucu yazılımına aynı anda bağlanacak istemci sayısının 2 olarak verilmesi durumunda, bağlanmaya çalışan üçüncü istemcinin hizmet alamayacağını gösteriniz.

3. TCPEchoClient.c dosyasının client_grupno isimli bir kopyasını oluşturunuz. İstemci programın kaynak kodunda alma(receive) buffer'ının büyüklüğü, komut satırından parametre olarak alınız. Echo

verisinin sunucuya gönderilmesi ardından, istemcinin sunucudan gelen verileri almak için kaç tane recv yapması gerektiğini ekrana yazdırınız.

4. Modifiye edilen istemci programı aşağıdaki parametrelerle çağrılırsa ekran çıktısı ne olur, yorumlayınız.

% ./TCPEchoClient_A1 127.0.0.1 "KTU Computer Engineering" 2

5. Ebevyn sunucu stratejisini uygulayan TCPEchoServer-Fork.c kaynak dosyasının ForkServer_Grupno.c isimli bir kopyasını oluşturunuz. Sunucunun ancak yeni bir istemci bağlanması durumunda, bir önceki istemciden kalan Zombie süreci sonlandırdığını gösteriniz.

6. (5). Maddede verilmiş olan problemi çözecek şekilde kaynak kodu değiştiriniz.

7. İstemciden "exit" mesajı gelmesi durumunda, soket bağlantısını kesecek şekilde sunucu yazılımını değiştiriniz.

8. ForkServer_Grupno.c isimli sunucu kaynak kodunu, bir anda N tane istemciye hizmet verebilecek şekilde değiştiriniz. N sayısı kullanıcı tarafından komut satırı parametresi olarak verilsin.

Deney Raporu

1. İstemci – Sunucu mimarisini kısaca tanımlayınız.
2. İstemci ve sunucu yazılımlarının işlevsel farklılıklarını irdeleyiniz.
3. Sunucu stratejilerini birbirleriyle kıyaslamalı olarak değerlendiriniz.
4. Deney sorularını ve çözümleme yaklaşımlarını kısaca anlatınız.