CS360 Ders Notları -- Sunucular

Bunu söylemekten nefret ediyorum ama eğer programlarınız bu ders notu rehberliğinde kullanıyorsanız onları Duncan(?) gibi bir SunOS makinasında kullanın. Solaris üzerinde pty(?) çalışmalarım yok.

Bu derste tekrarlanan ve eşzamanlı olan sunculardan ve bunlarla yapabileceğiniz bazı eğlenceli şeylerden bahsedeceğim.

Sunucu alıcılardan bağlantı bekleyen ve sonsuza dek süren bir programdır. Bir alıcı sunucuyla bağlantı kurduğunda , sunucu alıcı için bazı fonksiyonlar gösterir. Eğer bu sunucu tekrarlanan bir sunucuysa bir zamanda sadece bir alıcı işlem yapabilir. Eğer eşzamanlı bir sunucuysa birçok alıcıyla işlem yapabilir.

Alıcı ve sunucu arasında ayrım yapabilmek için aşağıdaki genellikle doğrudur:

Alıcı : lightweight(?)sadece bir kullanıcıyı destekleyebilen, küçük makineleri kullanabilen, ihtiyaç duyulduğunda açılan ve kullanıcı kullandıktan sonra kapanan ,engelleyici çoklu soru sormadan yoksun daha düşük beklemelerde etkili, kaynaklarının çoğunu kullancıların kesişim noktalarına veren

Sunucu : heavyweight(?)Sürekli çalışma kapasitesine sahip, üretilen işte etkili, birden fazla oturumda taşımada çokgörevli tamamen engelleyici.

Suncuların örnekleri yeniden yükleme programları (alıcılara uzaktan makinaya girme imkanı veren),ftp,finger(?) vs. Çoğu zaman alıcı, sadece alıcının stdinini sunucuya gönderen ve sunucunun çıktısını alıcının stdoutunu gönderen telneti kullanır. Daha karmaşık programlar alıcının daha karışık işer yapmasını gerektirir. Örneğin Netscapealıcıların kullandığı grafik kullanıcı arayüzüne sahip ve bu arayüz sunucudan çıktılar açıklar ve hoş bir şekilde gösterir.

Basit bir sunucu örneği olan catmotd.c herhangi birinin program tarafından sunulan mak./porta bağlanarak dosyaya nasıl erişebileceğini belirtir. Ben şu an catmotd sunucu ,port 7001 kullanıyorum. Eğer ‘’telnet plank 7001’’i tuşlarsanız 0600 korumalı olduğu için diğer türlü ulaşamayacağınız motd dosyasının içeriğini göreceksiniz.

Now, suppose you've written a brilliant educational tool like /usr/games/arithmetic. You want to let anyone in the world play it by calling telnet to your machine. Then you could write a program likeiterative_serve1.c.

What this program does is serve a socket, and wait for connections. For each connection, it executes the command

Şimdi usr/ oyunlar/ aritmetik gibi mükemmel bir eğitim aracı yazdığınızı düşünelim. Mak.zi telnet olarak tanımlayarak dünyada herhangi birinin onu oynamasına izin verebilirsiniz. Daha sonra tekrarlanan sunucu1.c gibi bir program yazabilirsiniz. Bu programın yaptığı bit spket görevi görmesi ve bağlantıları beklemesidir. Her bağlantı için komutu gerçekleştirir. Ayrıca **stdin** ve **stdout** 'un ikisi giriş-çıkış olarak kullanılır. Neden **pty** kullanılır? Pekala, **stdin** ve **stdout** 'un gerçekten ne yaptığına bakılmaksızın,bu öyle bir programdır ki,onun argüman programı,**stdin** ve **stdout** ekranda olacak gibi düzeltmeye çalışır.Bu çok iyi çünkü,standart io tamponlar, her satır sonra değil, her 4K karakterden sonra aynı hizada olacak anlamına gelir. Sunucunun doğru çalışması için bu gereklidir. Eğer **pty** hakkında daha fazla bilgi istiyorsanız, kitabın 19. bölümünü okuyun. Kendi bin dizinine **pty** kopyalamak için çekinmeyin. **pty** hakkında başka bir şey bilmeniz gerekmez, ama o çok kullanışlı bir programdır.
Program bittikten sonra ,
Iterative_serve1.c nin **stdin** ve **stdout'**u nasıl kaydettiğini ve geri almayı unutmayın.
Programı deneyin. Kopya iterative_serve1.c derleyin ve (**Duncan** gibi bir SunOS makinede) çalıştırın. Argüman olarak, makinenizin adını ve 4999'dan büyük bazı sayıları kullanın.Şimdi, bir istemci programı **usingtelnet** makine bağlantı noktası çalıştırın.Bunu farklı bir makinede, hatta farklı bir kullanıcı hesabından yapabilirsiniz. Bunu bir deneyin. İstemciniz yapılır ,tipi **CNTL-]** ve ardından **"quit".** Unutmayın , bir anda tek bir istemci ekleyebilirsiniz. Bu bir engel, özellikle bu gibi interaktif bir program ile.Bu problemi çözmek için **doit ()** çağırmak yerine, sunucu, **fork ()** ve **doit ()** concurrent_serve1.c olarak çağırılabilir. Unutmayın **,fork ()** çağırarak**,stdin** ve **stdout** kaydetmek ve geri yüklemek gerekmez.Ayrıca, sistem yerine **execlp** çağırabilirsiniz**.sistem()** fork çağırdığında, forking yürütülmesi kaydedilir.
Concurrent_serve1 çalıştırmayı ve aynı anda birkaç istemci eklemeyi deneyin. Şimdi, istemciler çıkış yapsa da sunucu çalışmasını sürdürecek. Şimdi, sunucu bir makinede "**ps x**" yapın. Şunu göreceksiniz ki işlemler bir çift zombi("ölü" demek) olduğunu. Neden?Çünkü Bu işlemler öldü, fakat onların ana işlemleri (sunucu) **wait ()** çağırmadı. Eğer sonsuza kadar yaşayan bir sunucu

yazmak isterseniz bu bir engeldir. Eğer bu sunucuyu concurrent_serve1 gibi yazmak isterseniz o zaman zombi işlem elde edersiniz.
Bu nasıl düzeltilir?Pekala, iki yolu vardır - birincisi ,işlemlerin bitmesi için **wait** çağırmayı denemektir. Ancak onların ne zaman biteceğini bilmiyorsunuz o zaman bloke olmayan **wait()** versiyonu **wait3 (**man sayfasını okuyun)gibi çağımak zorundasınız.

Diğer yol,prosesi yetim olmaya zorlamaktır. Böylece kolaylıkla istenen gibi olacaktır. Concurrent_serve2.c ' e bak / usr / games / aritmetik işlemini çatalla. İlk proses oluşur (ve proses için **wait()** çağır) ./ usr / games / aritmetik bir yetim olur. Şimdi, oldu ve çalışır.