SiGNALS

Kitaptaki bölüm 10 kısmı sinyaller hakkında tam bi açıklamaya sahiptir.Bu ders notlarını okuduktan sonra daha iyi anlıyacak ve bir şeylerin kazanımına başlıyacaksınız..

Aslında genel yapı itibariyle sinyallerin karmaşık bir akışı yani kontrol dışı çalışması vardır..

Signal i programda bir kesinti parçası gibi düşünebiliriz,Yani programlar çalışırken kesinti yapıp signallerin dahil olması..Tam ve doğru sinyalleri anlamak için biz hatırlamalıyız asenkron oluşan olayları..

Örneğin cntl+z ye bastığımızda biz programımıza SIGINT sinyali göndeririz,

CNTL-\ de ise SIGQUIT signali gönderilir programına..

Segemantation Violation oluştuğu zaman ise SIGSEGV sinyali gönderilir programımıza..

Aslında herhangi bir şeçime bakarsak sinyalin program içinde hangi zamanda oluşacağı program başında belli değildir..

Farklı sistemler farklı signal numarası desteklemektedir..Ve sık sık farklı sinyalleri oluştururlar..

Örneğin Unix Version 7 15 farklı sinyale sahip iken fakat SVR4(System V Release 4) 31 sinyale sahiptir**..**

ANSI C birde tanımlıyor POSIX de çok küçük sinyal oluşumunu..

Aslında bu oluşan farklı sinyaller genel olarak ANSI C standardı olarak kabul edilir ve kullanılır..

Bir kötü haber,C programları içinde oluşturulmuş sinyaller her farklı sistemlerde aynı benzerliği yani farklılıklar oluşturabiliyor..

2.kötü haber ise sinyallerin 2 modeli var.. güvenilir olan ve güvenilir olmayan modelidir..

Version 7 kullanıyor güvenilir olmayan modeli.. SVR4, BSD4.3+ and POSIX ise kullanıyorlar güvenilir modeli..

Yani buradan anlıyacağımız sinyaller farklılık gösterebiliyor..

Programlarımız farklı sinyal yollarına sahiptir..Bütün sinyallerin adları 3 kelime ile başlıyor :**SIG**

Örneğin program çalışırken **CNTL+C** ye bastığımızda çalışan program kapatılır bunun için oluşturulan signal **SIGINT ..**Segmantation violation hatası olduğunda **CNTL-** ye bastığımızda programa satır sonu komutu gönderilir ve program çökmesinden çıkılır ,bunlar için oluşturulan sinyaller **SIGQUIT** ve **SIGSEGV** dir**..**

Program içerisinde bu sinyalleri yeniden tanımlayabilirsiniz..Esnek programlar yazmak için sinyaller kullanışlıdır..

Bir sinyal oluşturulduğunda işletim sistemi çalışan programı devr alır ve onu stack bölgesine kayıt eder..

Sonra sinyal kesme işleyicisi olarak çağrılıyor..Örneğin varsılayan kesici programda **SIGINT** sinyalidir program bittiğinde çıkış için..Kullanıcı program anında çalışırken ise oluşturulan sinyaller **SIGQUIT** ve **SIGSEGV ..**

Aşağıdaki örnek signal in kullanımı verilmektedir..

```
#include < signal.h >

void cntl_c_handler(int dummy)
{
  printf("You just typed cntl-c\n");
  signal(SIGINT, cntl_c_handler);
}

main()
{
  int i, j;

  signal(SIGINT, cntl_c_handler);

  for (j = 0; j < 40; j++) {
    for (i = 0; i < 1000000; i++);
  }
}
```

Bu SIGINT için bir kesme işleyicisi ayarlandı.Şimdi, kullanıcı CNTL-C vurduğu zaman, işletim sistemi programın geçerli yürütme durumu kaydedip ve sonra cntl_c_handler çalıştıracaktır.cntl_c_handler döndüğü zaman, işletim sistemi kaldığı yerden programa devam eder.Böylece, siz sh1 i çalıştırdığınızda,her CNTL-C yazışınızda,sistem "Sadece cntl-c yazdınız." görüntüleyecek ve program çalışmaya devam edecek.Bu yazı 10 saniye ya da biraz daha fazla sürede kendiliğinden kapanacaktır.

Sinyal işleyici cntl_c_handler prototipini takip etmelidir.Diğer bir deyişle, (void)(yani hiç birşey)e geri dönmeli ve eğer argümanı kullanmayacaksa,bir tamsayı integerı kabul etmelidir.Aksi takdirde, gcc size şikayet edecektir.

Ayrıca, sinyal işleyici bir sinyal () çağrısı yapmayı unutmayın.Bazı sistemlerde (örneğin Sürüm 7), eğer bunu yapmazsanız,bu sinyali ele alarak, bir kez CNTL-C için varsayılan sinyal işleyicisini yeniden yükler.Bazı sistemlerde, ekstra sinyal () çağrısı yapmak zorunda değilsiniz.Unix topraklarında bir yaşam gibi.

Bir sinyal çağrısı ile her değişik sinyali işleyebilirsiniz.Örneğin,sh1a.c CNTL-C(SIGINT olan)ve CNTL-\(SIGQUIT olan) için değişik sinyal işlemcileri tanımlar.Sinyal üretildiğinde i ve j değerlerini yazdırırlar.Bunun çalışması için i ve j nin global değişkenler olması gerektiğini unutmayın.Bu global değişkenler kullanmak zorunda olduğunuz bir örnek.

BUnu programı derlerken ve çalıştırırken deneyin, ve CNTL-C ve CNTL-\ ye birkaç kez basın.

```
UNIX> sh1a
^CYou just typed cntl-c.  j is 2 and i is 539943
^CYou just typed cntl-c.  j is 2 and i is 919180
^\You just typed cntl-\.  j is 4 and i is 413031
^CYou just typed cntl-c.  j is 5 and i is 20458
^\You just typed cntl-\.  j is 6 and i is 73316
^\You just typed cntl-\.  j is 6 and i is 683034
^CYou just typed cntl-c.  j is 7 and i is 292244
^CYou just typed cntl-c.  j is 13 and i is 738661
^\You just typed cntl-\.  j is 14 and i is 789583
^\You just typed cntl-\.  j is 16 and i is 42225
^\You just typed cntl-\.  j is 16 and i is 209458
^CYou just typed cntl-c.  j is 17 and i is 260584
^\You just typed cntl-\.  j is 19 and i is 982514
UNIX>
```

```
#include < signal.h >
#include < stdio.h >

void segv_handler(int dummy)
{
  fprintf(stderr, "nfs server not responding, still trying\n");
  while(1) ;
}

main()
...
  signal(SIGSEGV, segv_handler();

  rest of the code
}
```

Bu eğer demo-ing kodu olsaydı ve segmentasyon ihlali oluşsaydı(her zaman demo-ing kodu olduğunuzda gerçekleştiği görünüyor),bu ağ donmuş gibi görünecekti.Çok zekice.(Yani baktığında ve sh1b.c çalıştığında segmentasyon ihlaline neden olur, ama ağa asılı gibi görünür.)SIGHUP hakkında gerçekten harika birşey...

SIGHUP hakkında gerçekten harika birşey

SIGHUP bütün süreçleri çocuğu gibi gören ve onları gönderen kontrol terminali olan bir sinyaldir.Eğer "Is" gibi bir komut yazarsanız, ve ya "Is" den daha gelişmir bir komut yazarsanız,bu süreç çalışırken, kontrol trminalinin kabuk süreci her zaman isletim sistemi tarafından üst süreç olarak kaydedilir.Şimdi, kabuk hakkında onun tasarımı yumurtladığı süreçleri ve dolayısıyla sürecin dizaynından sorumlu olanlar hakkında özel bir bilgi.Özellikle bir şey--kabuk süreci kontrol terminali kapandığında sona erer,kabuk süreci yavru süreçlerine SIGHUP yollar.Peki, eğer kabuk arka planda askıya alınan işler bulursa,örneğin,kabuk sizi uyarmak için uyarı sesi cıkaracaktır.Eğer kabuğu kapatmak için ısrar ederseniz,kabuk kapanır ve bu sinyalleri gönderir.Bu derste öğretilenler ile, bunun varsayılan olarak tüm o yavru süreçleri öldüreceğini söyleyebilirsiniz.

Yani bu sizin bir çoğunuzun deneyimlediği bir durumu getirir.Uzaktan bir Unix makinesine oturum açtığınızda, (çalıştırmak için görsel süreç laboratuvarı ödevinde yaptığınız işi söyleyebilirsiniz)bir işi çalıştırın ve sonra çalışma süresi sırasında,terminaliniz geçici olarak kablosuz bağlatıyı kaybeder.Sonra tekrar oturum açtığınızda işiniz gitmiştir.Bir çok işlem zamanı boşa gitti.Şimdi sistem proglamlama uzmanı olarak, suçlunun kesinlikle SIGHUP olduğunu söyleyebilirsiniz.

Bunun etrafından geçmek için bir yol var mı ? Pekala, biri için,SIGHUP ı yoksayan bir alt sürece sahip olabilirsiniz.Ama tabi bu program sizin yaptığınız değilse.Sonra,alt sürecin kabuk süreçle olan bağlantısını keselim."disown" ya da "nohup" ile yapabilirsiniz bunu.Size örnekler göstereceğim.Daha da iyisi,işin gerçekten doğrudan kabuk ile çalışmadığını nasıl anlayacağız?"at" ya da "batch" standart uygulamalarını deneyin.Bunları kullanarak,iş bittiğinde bir e-posta bile alabilirsiniz.

Sinyal hakkında bazı detaylar

Sinyal çağırılırken, iki giriş argümanı sinyal sayısı ve bir işlev işaretçisi içerir.Biri ikinci argümanın , işlev işaretçisinin SIGIGN ve SIGDFL sabitleri olduğunu bilmelidir.Eğer SIGIGN geçerse, daha sonra süreç sinyali gözardı eder.SIGKILL gözardı edilemez veya yakalanamaz,programcının ayarının anlamı değişik bir sinyal eylemcisi(eger program kontrolden cıkarsa,yönetici hala SIGKILL ile programı yokedebilir.)Eğer SIGDFL geçerse,süreç varsayılan sinyal yakalayıcısına döndürülür.Ne durumda olursa olsun, cagrı sinyali bir sinyal için bir önceki yakalayıcının adresine geri döner.

sinyal üretimi için geçerli nedenler aşagıdaki kategorilere ayrılabilir:

1. terminal Kullanıcı tuş vuruşlarını elde terminali (CNTL-D, örneğin)
2. donanımsal istisna , e.g. divide by 0, bus error.
3. kill fonsiyonu (bir prosesin başka bir prosese sinyal göndermesi yada grupta etkili olan sinyal). Not, bir süreç yükseltmek için bir çağrı tarafından kendisi için bir sinyal gönderebilir.
4. kill komutu (kill işlevi sadece bir komut arayüzü).
5. SIGPIPE ve SIGALARM gibi yazılım koşulları.

Alarm()

Sinyallerin bir başka kullanımı da Unix in sağladığı "çalar saat" dir.alarm() için man sayfasını okuyun.alarm(n) ın geri dönüş değeri yoktur,n saniye sonra SIGALRM sinyali ortaya çıkar.Bunun için sinyal işleyici kullanırsanız istediğiniz sinyali oluşturabilirisniz. Örneğin**, sh2.c** programı 3 saniye çalışıp sonra sadece bir mesaj yazdırır **sh1.c** gibi olur.Bu alarm() yaklaşık değerde –tam olarak 3sn değil,ama biz bu sınıf için buna yakın kabul ettik.

UNIX> sh2

Henüz 3 saniye geçti: j = 26. i = 638663

UNIX>

Son olarak,Unix de her saniye **SIGALARM** ın nasıl gönderildiğini **sh3.c** size gösterir.Bu,

Sadece bir tweak **sh2.c** Unix bir saniye sonra geçerli bir SIGALRM oluşturmak için

UNIX> sh3

1 saniye geçti: j = 8. i = 823534

2 saniye geçti: j = 17. i = 715735

3 saniye geçti: j = 26. i = 610604

4 saniye geçti: j = 35. i = 513675

UNIX>

Bazı sistemlerde, sinyal işleyicisi dönünceye kadar tek bir sinyal için bir sinyal işleyici olduğunda, yine aynı sinyal işleyemiyor. Diğer sistemlerde, aynı sinyal yeniden işleyebilir.Örneğin **sh4.c** ye bakın.

Alarm_handler sonsuz döngüye girdiğinde bir değer döndürmez. Program bir saniye için çalışır ve sonra SIGALRM oluşturulur ve **alarm_handler ()** girilir. Bu sonsuz bir döngüye girer ve bir saniye sonra SIGALRM yeniden oluşturulur. Unix sürümüne bağlı olarak, farklı şeyler ortaya çıkabilir. Solaris' de, sinyal ele alınacak ve yeniden **alarm_handler ()**

gireceksiniz.SunOS ‘de sinyal **alarm_handler** dan dönene kadar şüphesiz ki göz ardı edilecektir.

Burada Solaris deki (kenner üzerinde dene) çıktısı var:

UNIX> sh4

1 saniye geçti: j = 7. i = 697646

1 saniye geçti: j = 7. i = 697646

1 saniye geçti j = 7. i = 697646

1 saniye geçti: j = 7. i = 697646

...

Ve burada da SunOS (duncan üzerinde dene) daki çıktı var:

UNIX> sh4

1 saniye geçti: j = 7. i = 584436

Diğer sinyalleri güvenilir bir sinyal işleyici olsun veya olmasın oluşturabilir ve işleyebilirsiniz.

Örneğin**, sh4.c** içinde **CNTL-** yapıldığında, bu programın veya **alarm_handler ()** ın çalışır durumda olup olmadığı anlaşılmış olur - bir deneyin.

Son olarak, kill komutu ile bir program için herhangi bir sinyal gönderebilirsiniz.Man sayfasını okuyun.

Sinyal sayısı9**(SIGKILL)** program tarafından yakalanmaz yani bunun için bir sinyal işlemci yazamazsınız.Bu güzel bir şey,çünkü eğer bir sinyal işleyici yazarken karışıklık yaparsanız **kill-9**, programı sonlandırmak için tek yoldur.

Üzerinde düşünmeniz için bir kaç değişik soru :

1.**exec** komutu çağrıldığında yeni proses sinyali bir önceki prosesde olduğu gibi aynı yolla mı kullanır ?

2.Proses sinyal işlemeci tarafından yürütülürken , sinyal tetiklenirse , ne yapmalı?