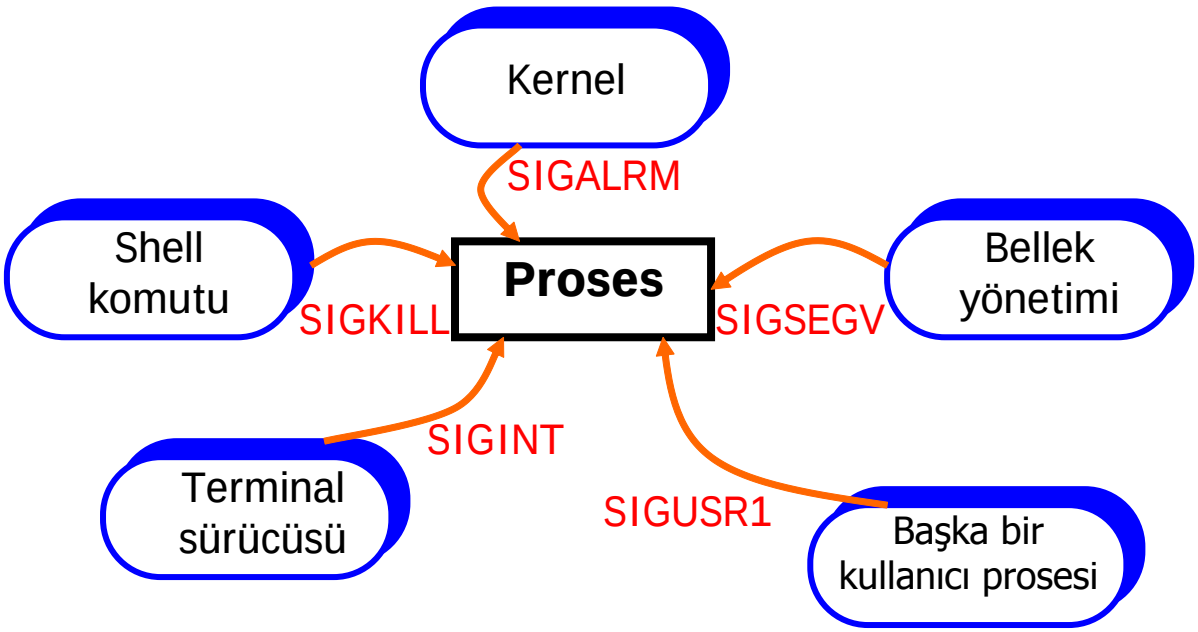
UNIX de Sinyaller(signals)



<sys/signal.h>

Sinyallerin kaynakları (bazıları)

Sinyal ismi	**Varsayılan(olağan) etki**	**Sinyalin sebebi**
SIGINT	Prosesi sonlandır	Durdurma karakterine basılır
SIGKILL	Prosesi sonlandır	Shell komutu “kill” kullanılır
SIGALARM	Prosesi sonlandır	Alarm saatinin zamanı dolar
SIGSEGV	core image yaratır	Geçersiz bellek kullanımı
SIGUSR1	Prosesi sonlandır	Kullanıcı tarafından tanımlanır

core image : ana (çekirdek) görüntü

Belli tipteki bir sinyal alındığı zaman bir proses nasıl bir karşılık vereceğini seçebilir:

1. Sinyali yok sayabilir
2. Belirli bir sinyal-yakalayıcı(signal-handler) fonksiyonu çalıştırabilir
3. Varsayılan etkiyi kabul edebilir

Bu seçim **sinyal disposition** olarak tanımlanır.

Program içerisinde sinyal disposition tanımlama:

signal(<sinyal ismi>, <bu sinyale yanıt>) ;

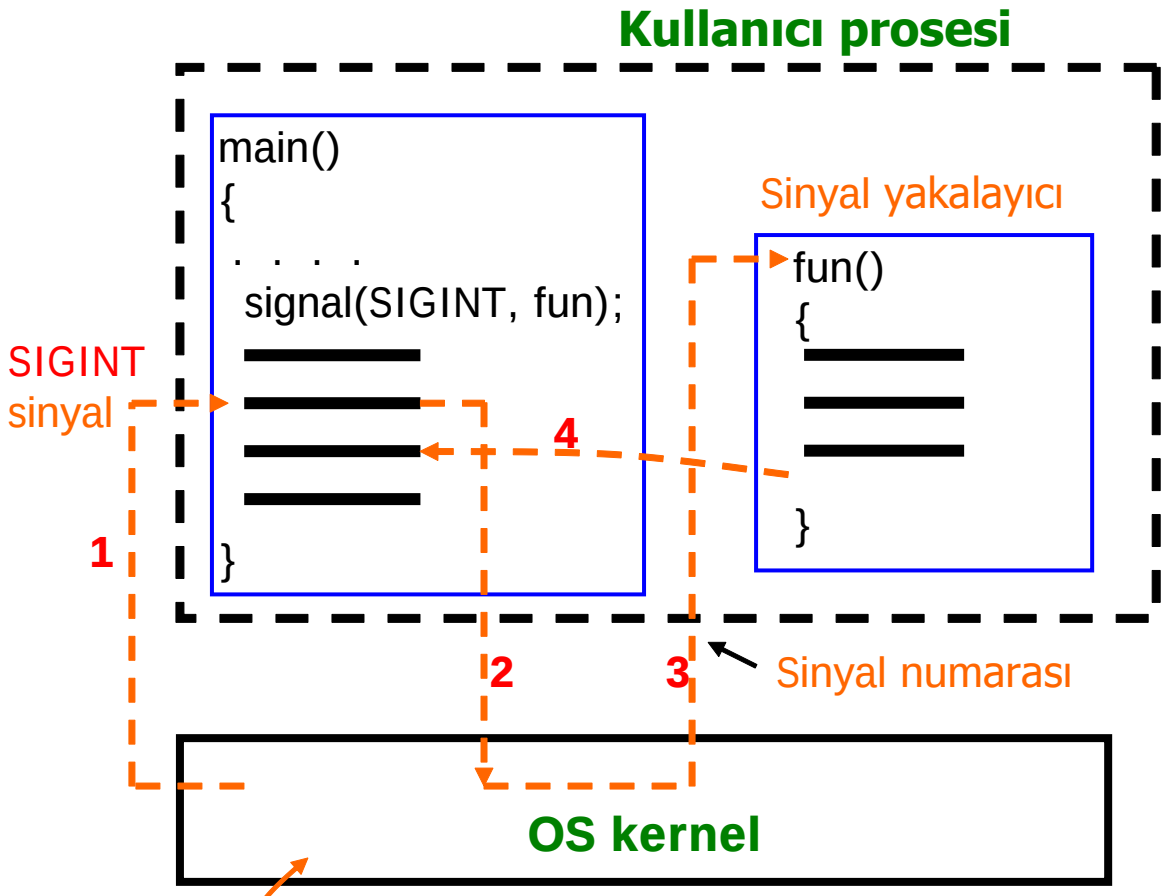
<sinyal ismi> ∈ {SIGINT, SIGKILL, SIGALARM, . . . }

Yaklaşık 35 sinyal

<Bu sinyale yanıt> =

SIG_IGN	sinyali yok say,
SIG_DFL	farzedilen etkiyi kabul et,
fonksiyon ismi	bu fonksiyonu çalıştır (programın içinde olmalı).

Sinyal alındığı zaman yapılan işlemler



Farklı kaynaklardan üretilen sinyaller işletim sisteminin çekirdeği tarafından prosese taşınır

Örnek:

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

void handler(int signum)
{
  printf("Durdurma sinyali aldım\n") ;
  signal(SIGINT, handler) ;

main()
{
  int i ;
  signal(SIGINT, handler) ;
  for ( i=0 ; i<1000 ; i++)
    {  printf("çalışıyor . . .%d\n", i) ;
       sleep(1) ;
     }
}
```

sinyal yakalayıcı

yakalayıcı yeniden kur

Bir kullanıcı prosesinden başka bir kullanıcı prosesine sinyal gönderme

kill(pid, signum) ;

Alıcı prosesin **ID** si veya
0 veya
–1 veya
negatif sayı

Sinyal numarası

Örnek: Alt prosesten ana prosese sinyal gönderme

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

void handler(int signum)
{
   printf("alt prosesten gelen sinyal SIGUSR1 \n");
   sleep(15);
   wait(0);
}

int main()
{
   int i, pid, ppid;
   signal(SIGUSR1, handler);
   pid = fork;
   if (pid == 0) {/*Alt proses*/
                     sleep(30);
                     ppid = geppid(); /* Ana prosesin ID si ?*/
                     kill (ppid, SIGUSR1); /* Ana prosese sinyal */
                     exit(0);
                    }
   for (i=0; i<300; i++)
       {
       printf("ana proses çalışıyor …%d\n", i);
       sleep(1);
       }
    return 0;
}
```

Örnek: Alt proseslere sinyal gönderme

```
#include <stdio.h>
#include <sys/signal.h>
main()
{ int i ;
  setpgrp() ;   /* ana proses, proses gurup numarası set eder */
  for ( i=0 ; i<10 ; i++) /* proses gurup numarası tekrardan set edilir */
   { if (fork() == 0 )
        {  /* alt proses */
          if ( i & 1 ) setpgrp() ; /*grp no tekrardan set edilir */
          printf("pid = %d pgrp = %d\n", getpid(), getpgrp() ) ;
          pause() ;          /* sinyal alana kadar  bekle */
        }
    }
  kill(0, SIGINT) ;          /* ana proses kendi gurubunda olan
                              bütün alt proseslere sinyal gönderir */
}
```

Sinyallerle uğraşma

1. Bir sinyali yok saymak

```
main()
{  signal(SIGINT, SIG_IGN) ;
    signal(SIGQUIT, SIG_IGN) ;
       /*  v.b. */
       /* şimdi bazı ış yap */
}
```

2. Temizlemek ve sonlanmak

```
int child_pid ;                                          /* global */
void clean_up(int signum)
{  unlink("/tmp/my_file") ;                          /* dosyayı sil */
    kill(child_pid, SIGTERM) ;            /* alt prosesi sonlandır */
    wait(0) ;                    /* alt prosesin sonlanmasını bekle */
    printf("Program sonlandı\n") ; exit(1) ;
}

main()
{  signal(SIGINT, clean_up) ;
    open("/tmp/my_file", O_RDWR | O_CREAT, 0644) ;
    child_pid = fork() ;                          /* alt proses yarat */
       /* Şimdi iş yap */
}
```

3. Implementing a timeout

```
#include . . .    /* <stdio.h>, <sys/signal.h>, <setjmp.h> */
jmp_buf tpoint ;                                  signal handler
void handler(int signum)
{
    longjmp(tpoint, 1) ;                          /* return 1 */
}
                    buffer      timeout
int tgets(char *s, int t )
{ char *ret ;
   signal(SIGALRM, handler) ;
   if ( setjmp(tpoint) != 0 )  return(-2) ;        /* timed out */
   alarm(t) ;                                     /* set timeout */
   ret = gets(s) ;
   alarm(0) ;                                  /* cancel the alarm */
   if ( ret == NULL ) return (-1) ;               /* EOF  -  ^D */
   else
      return strlen(s) ;
}

main()
{  int v ; char buf[100] ;
    while( 1 )
       { printf("Enter a string: ") ;                    /* Prompt */
         v = tgets(buf, 5) ;                /* Asking 5 sec timeout */
         switch( v )
            { case –1: exit(1) ;                    /* Probably EOF */
              case –2: printf("Timed out\n") ; break ;
              default: printf("You typed %d chars\n", v ) ;
            }
       }
}
```