Sistem Programlama

Ders 14

Doç. Dr. Mehmet Dinçer Erbaş
Bolu Abant İzzet Baysal Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

İşlem kontrolü

- Yarış durumu (race conditions)
  - Birden fazla işlem ortak veri üzerinde bir şeyler yapmak istediğinde ve sonucun hangi işlemlerin çalışma sırasına göre değişeceği durumlarda yarış durumu oluşur.
- fork fonksiyonu nedeniyle yarış durumu oluşabilir.
  - Genelde hangi işlemin önce çalışacağını bilemeyiz.
  - Hangi işlemin önce çalışacağını bilsek bile bundan sonra ne olacağı sistemin yük durumuna ve kernel tarafından görevlendirme algoritmasına bağlıdır.

Yarışma durumu

- Bir işlem alt işleminin sonlanmasını beklemek isterse wait fonksiyonlarından birini kullanabilir.
- Bir işlem üst işleminin sonlanmasını beklemek isterse aşağıdakkine benzer bir döngü kullanılabilir.

```
while(getppid() != 1)
    sleep();
```

- Bu tip döngü kullanarak bir durumu kontrol etmeye polling (kuyruklama) denir.
  - Bu yöntem CPU zamanını boşa harcadığı için pek tercih edilmez.

Yarışma durumu

- Yarışma durumunu engellemek için, işlemler arasında bir iletişim yöntemi olmalıdır.
  - Sinyaller kullanılabilir.
  - Farklı işlemler arası haberleşme yöntemleri kullanılabilir.
- Üst ve alt işlemler için genellikle aşağıdakine benzer senaryolar gözlemlenir.
  - Fork işlemi sonrası hem alt hem üst işlemin yapacağı işler olur. Örneğin üst işlem alt işlemin işlem numarasını kullanarak bir kayıt dosyasını değiştirebilir ve alt işlem üst işlem için yeni bir dosya oluşturabilir. Bu örnekte, her iki işlemin birbirine yapacaklarını tamamladıklarını bildirebilmesi gerekir.
  - tellwait1.c
  - tellwait2.c

Yarışma durumu

```
#include "apue.h"
TELL_WAIT();
/* set things up for TELL_xxx & WAIT_xxx */
if ((pid = fork()) < 0) {
        err_sys("fork error");

        } else if (pid == 0) {          * child *

        TELL_PARENT(getppid());

        WAIT_PARENT();

        exit(0);

}

TELL_CHILD(pid);
WAIT_CHILD();
exit(0);
```

- TELL_WAIT, TELL_PARENT, TELL_CHILD ve WAIT_PARENT fonskiyonları apue kütüphanesinde tanımlanmıştır.

exec fonksiyonları

- Daha önce bahsettiğimiz üzere fork fonksiyonu ile yeni bir işlem yaratıp bu işlemde exec fonksiyonları ile bir başka programı çalıştırabiliriz.
- Bir işlem exec fonksiyonunu çağırdığında, çağıran işlemin yerine yeni program geçer ve yeni programın main fonksiyonu çalışmaya başlar.
  - exec fonksiyonu çalışması sonrası işlem numarası değişmez. Çünkü yeni bir işlem yaratılmamıştır.
  - Yapılan işlemin hafızadaki yerine yeni veriler yazılır. Metin kısmı, heap ve yığın bölümlerine yeni programın verileri yazılır.

exec fonksiyonları

- 6 farklı exec fonksiyonu vardır.
- fork fonksiyonu ile yeni işlem yaratırken, exec fonksiyonu ile yeni programı başlatabiliriz.

```
#include <unistd.h>

int execl(const char *pathname, const char *arg0, ... /* (char *)0
*/ );

int execv(const char *pathname, char *const argv []);

int execle(const char *pathname, const char *arg0, .../* (char *)0,

char *const envp[] */ );

int execve(const char *pathname, char *const argv[], char *const
envp []);

int execlp(const char *filename, const char *arg0, ... /* (char *)0
*/ );

int execvp(const char *filename, char *const argv []);
```

Döünş: hata ise -1, OK ise dönüş yok.

exec fonksiyonları

- İlk dört fonksiyon yoladı argüman alırken son iki fonksiyon bir dosya ismi argüman alır.
- Bir dosya adı verildiğinde
  - Dosya adı / ile başlıyor ise yoladı olarak alınır.
  - Aksi halde çalıştırılabilir dosya PATH ismi ile tanımlı çevre değişkeninde belirtilen yerde aranır.
    - PATH=/bin:/usr/bin:/usr/local/bin/:.
- execlp veya execvp yoladı olarak verilen dosyayı bulur ancak bu dosya çalıştırılabilir bir dosya değil ise, fonksiyon bu dosyayı bir kabuk komut dizisi (shell script) olarak kabul eder ve /bin / sh ile bu dosyayı çalıştırır.
- Diğer fark argüman listesinin verilme biçimidir.
  - l ise liste olarak, v ise vektor olarak argümanlar verilir.

exec fonksiyonları

- execl, execlp ve execle fonksiyonları argümanların ayrı ayrı verilmesini bekler. Argümanların bittiği null işaretçi ile belirtilir.
- Diğer üç fonksiyon için ise (execv, execvp ve execve) için ise argümanlar bir diziye yazılmalı ve bu dizinin adresi bu fonksiyonlara verilmelidir.
- Fonksiyonlar arasındaki son fark çevre listesinin yeni programa gönderilme şekli konusundadır.
  - Sonu e harfi ile biten iki fonksiyon (execle ve execve) çevre metinlerine bir işaretçi göndermemize izin verir.
  - Diğer dört fonksiyon ise environ değişkenini kullanır ve işlemin yeni programa iletilmesi için bulunan çevre değişkenlerini kopyalar.

exec fonksiyonları

Function	*pathname*	*filename*	*fd*	Arg list	*argv*[ ]	environ	*envp*[ ]
execl	•			•		•	
execlp		•		•		•	
execle	•			•			•
execv	•				•	•	
execvp		•			•	•	
execve	•				•		•
fexecve			•		•		•
(letter in name)		p	f	l	v		e

exec fonksiyonları

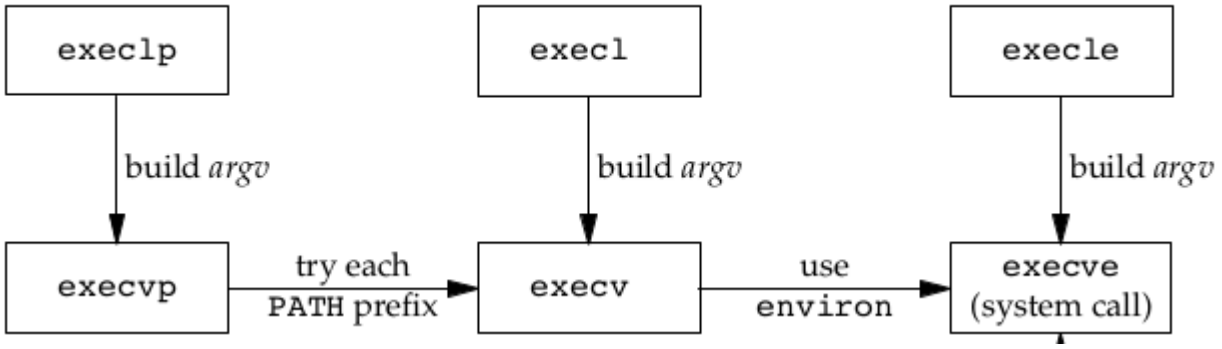
- Yeni çalışacak olan program bu programı başlatan işlemden aşağıdaki bilgileri alır.
  - Işlem numarası ve üst işlem numarası
  - Gerçek kullanıcı numarası ve gerçek grup numarası
  - Destekleyici grup numarası
  - Oturum numarası
  - Kontrol edilen terminal
  - Alarm saatine kalan zaman
  - Kök klasör
  - Dosya modu ve yaratma maskesi
  - Dosya kilitleri
  - Işlem sinyal maskesi
  - Bekleyen sinyaller
  - Kaynak limitleri
  - tms_utime, tms_stime, tms_cutime ve tms_cstime değerleri

exec fonksiyonları

- Açık dosyaların durumu close-on-exec bayrağının değerine göre belirlenir.
  - Bayrak ayarlanmamış ise açık dosyalar açık olarak kalır.
  - Bayrağın ayarlanması için fcntl fonksiyonu ile ayarlamak gerekir.
- POSIX.1 standartlarına göre exec fonksiyonu çalıştırıldığında açık klasör akışlarının kapanması gereklidir.
  - Bu genellikle opendir fonksiyonunun çağırılması ile yapılır.
  - opendir fcntl kullanarak açık olan klasör akışları için close-on-exec bayrağını ayarlar.
- Gerçek kullanıcı numarası ve gerçek grup numarası exec fonksiyonu ile değişmez. Ancak efektif kullanıcı numarası set-user-ID ve set-group-ID bayraklarının değerine göre değişiklik gösterebilir.

exec fonksiyonları

- Unix sistemlerinde genellikle bu altı fonksiyonun sadece biri, execve, kernel içerisinde çalışan bir sistem çağrısı olarak tanımlanır. Diğer fonksiyonlar ise sadece kütüphane fonksiyonlarıdır ve sonuçta execve fonksiyonunu çağırır.



- exec1.c.

Sinyaller

- Sinyaller yazılımsal işkesme (interrupt) metodudur.
  - Sinyaller kullanılarak asenkron olaylar kontrol edilebilir.
    - Örneğin bir kullanıcı belirli bir tuşa basarak işkesme yaptığında.
    - Pipiline kullanıldığında bir sonraki program hatalı şekilde sonlanırsa.
- POSIX.1 sinyal kullanımını standart haline getirmiştir.
- Sinyallerin genel özellikleri ve kullanım şekillerini inceleyeceğiz.

Sinyaller

- Her sinyalin bir ismi vardır.
  - İsimler SIG ile başlar.
    - SIGABRT ==> bir işlem abort fonksiyonunu çağırarak anormal şekilde sonlandığında oluşturulan sinyal.
    - SIGALRM ==> alarm fonksiyonu ile bir zamanlayıcı ayarlandığında ve zaman dolduğunda oluşturulan sinyal.
- FreeBSD 5.2.1, MAC OS X 10.3 ve Linux 2.4.22 31 farklı sinyali destekler. Solaris 38 farklı sinyali destekler.
- Linux ve Solaris sistemlerinde uygulama tarafından yeni sinyaller tanımlanabilir.
- Sinyal isimleri pozitif tamsayı sabitleri ile tanımlanmıştır.
  - Bu sabitler sinyal numarasıdır ve signal.h dosyasında bulunur.
  - Hiçbir sinyal 0 numrasına sahip değildir.
    - kill fonksiyonu bazı durumlarda 0 numaralı sinyal oluşturur.
    - POSIX.1 bu sinyali null sinyal olarak tanımlar.

Sinyaller

- Birçok farklı durum sinyal oluşmasına sebep olabilir.
  - Terminal tarafından oluşturulan sinyaller kullanıcı belli tuşlara bastığında oluşur. Örneğin bir program çalışırken DELETE tuşuna basmak veya CTRL-C tuşlarına basmak işkesme (interrupt) sinyali olan SIGINT oluşmasına neden olur.
    - Bu sayede devam etmekte olan bir program durdurulur.
  - Donanımla ilgili hatalar sinyal oluşturur. Örneğin 0 ile bölünme işlemi yapıldığında, geçersiz hafıza okunması gibi.
    - Bu durumlarda genellikle donanım tarafından algılanır ve kernel uyarılır. Kernel bu durumla ilgili sinyali oluşturur.
    - Örneğin geçersiz hafıza okuması yapılırsa SIGSEGV sinyali oluşturulur.

Sinyaller

- Birçok farklı durum sinyal oluşmasına sebep olabilir (devam)
  - kill (2) fonksiyonu kullanılarak başka bir işleme veya işlem grubuna sinyal gönderilebilir.
    - Bu fonksiyonun kullanılabilmesi için bazı limitler vardır.
      - Sinyal gönderdiğimiz işlemin sahini olmalıyız yada yönetici olmalıyız.
  - kill (1) komutu başka bir sişleme sinyal göndermek için kullanılır.
    - Bu komutu tahmin edeceğiniz üzere kill fonksiyonunu çağırır.
    - Çoğunlukla arka planda çalışmakta olan kaçak işlemler için kullanılır.

Sinyaller

- Birçok farklı durum sinyal oluşmasına sebep olabilir (devam)
  - Yazılımla alakalı olarak işlem ile alakalı bir olay gerçekleştiğinde işleme haber vermek için sinyal oluşturulabilir.
    - Bunlar, 0 tarafından bölünme gibi, donanım kaynaklı olaylar değil yazılımla ilgili olaylardır.
  - Örneğin ağ üzerinde bant dışı veri geldiğinde SIGURG sinyali gönderilir.
  - Örneğin pipe kullanıldığında, pipe okuyan işlem sonlandığında ve pipe için yeni veri yazıldığında SIGPIPE gönderilir.
  - Örneğin bir alarm ayarlandığında ve alarm oluştuğunda SIGALRM gönderilir.

Sinyaller

- Sinyaller asenkron olaylardır.
  - Bir işlem sinyalin ne zaman kendisine gönderileceğini bilemez.
  - Hata yönetiminde gördüğümüz gibi errno değişkeni bir değeri kontrol ederek sinyalin oluşup oluşmadığı kontrol edilemez.
    - Tek yapılacak olan kernel'e sinyal oluşması durumunda ne yapılacağının söylenmesidir.
  - Daha önce bahsettiğiniz üzere bir sinyal alındığında üç farklı işlemden biri yapılmalıdır.
    - Bu sinyal ile alakalı aksiyonlar olarak adlandırılır.

Sinyaller

- Sinyal ile ilgili aksiyonlar
  - Sinyali görmezden gelme
    - Birçok sinyal alındığında görmezden gelmek mümkündür.
    - Ancak iki fonksiyon SIGKILL ve SIGSTOP görmezden gelinemez.
      - Bu sinyaller yönetici tarafından işlemleri durdurmak veya sonlandırmak için kullanılır.
    - Donanım hatası durumunda oluşan bir sinyal görmezden gelinirse, sonuç belirsizdir.

Sinyaller

- Sinyaller ile ilgili aksiyonlar
  - Sinyali yakalama
    - Bunu yapmak için sinyal alındığında çalışacak olan fonksiyonu kernel'e bildirmemiz gerekir.
      - SIGCHLD sinyali yakalanırsa bir alt işlem sonlanmış demektir.
        - Bu durumda yapılması gereken waitpid fonksiyonu ile alt işlemin sonlanma durumunun okunmasıdır.
      - İşleminiz geçici dosyalar ürettiyse ve SIGTERM sinyali yakalanırsa (bu sinyal kill komutu tarafından işlemi sonlandırmak için gönderilir) bu geçici dosyaları temizlemek isteyebilirsiniz.
    - SIGKILL ve SIGSTOP sinyalleri yakalanamaz.

Sinyaller

- Sinyaller ile ilgili aksiyonlar
  - Varsayılan aksiyonun yapılmasına izin verilir.
    - Her işlemin bir varsayılan aksiyonu vardır.
    - Dikkat edilmelidir ki birçok sinyalin varsayılan aksiyonu işlemin sonlandırılmasıdır.
  - Varsayılan işlem “terminate+core” ise işlemin mevcut çalışma klasöründe bir hafıza resmi dosyası oluşturulur.
    - Bu dosya kullanılarak işlemin neden sonlandığı incelenebilir.
    - Bazı durumlarda bu dosya yaratılmaz.
- Kitabınızın 293 numaralı sayfasında Unix sistemlerinde oluşturulan sinyallerin bir listesi bulunmaktadır.

signal fonksiyonu

- Unix sisteminde sinyal arayüzünü kullanmak için signal fonksiyonu kullanılabilir.

```
#include <signal.h>

void (*signal(int signo, void (*func)(int)))(int);
```

Dönüş: OK ise sinyal ile ilgili önceki aksiyon, hata ise SIG_ERR.

  - signo sinyalin ismidir.
  - func değişkenin değeri aşağıdakilerden biri olabilir.
    - SIG_IGN: sistem sinyali görmezden gelir.
    - SIF_DFL: sistem varsayılan aksiyonu yapar.
    - Sinyal yakalandığında çalıştırılacak olan fonksiyonun adresi.
  - shell2.c
  - sigusr.c

signal fonksiyonu

```
$ ./sigusr &
$ kill -USR1 7216
received SIGUSR1
$ kill -USR2 7216
received SIGUSR2
$ kill 7216
[1]+ Terminated
```

Sinyaller

- kill ve raise fonksiyonları
  - kill fonksiyonu bir işleme veya bir işlem grubuna sinyal göndermek için kullanılır.
  - raise fonksiyonu ile bir işlem kendine sinyal gönderebilir.

```
#include <signal.h>

int kill(pid_t pid, int signo);

int raise(int signo);
```

Dönüş: OK ise 0, hata ise -1.

  - raise(signo) fonksiyonu kill(getpid(), signo) ile aynı işe yarar.

kill ve raise fonksiyonları

- pid argümanının alabileceği değere göre dört farklı durum söz konusudur.
  - pid > 0
    - Sinyal işlem numarası pid olan işleme gönderilir.
  - pid == 0
    - Sinyal fonksiyonu çalıştıran işlemle aynı işlem grup numarasına sahip işlemlere gönderilir. Sinyalin gönderilmesi için gerekli iznin olması gerekir.
      - Sistemle ilgili işlemlere sinyal gönderilemez (örneğin init).
  - pid < 0
    - Sinyal işlem grup numarası pid değerinin mutlak değerine eşit olan bütün işlemlere gönderilir. Önceki durumda olduğu gibi sinyalin gönderilme izni olmalıdır ve sistem işlemleri ayrı tutulur.
  - pid == 1
    - Sinyal sistemdeki bütün işlemlere gönderilir. Sinyalin gönderilmesi için gerekli izin olmalıdır ve sistem işlemleri ayrı tutulur.

kill ve raise fonksiyonları

- Bir işlem başka bir işleme sinyal gönderebilmek için gerekli izinleri olmalıdır.
  - Sistem yöneticisi istediği bütün işlemlere sinyal gönderebilir.
  - Öbür kullanıcılar için sinyali gönderenle sinyali alanın efektif kullanıcı numarası aynı olmalıdır.
  - SIGCONT sinyali her türlü işleme gönderilebilir.
  - POSIX.1 standardına göre _POSIX_SAVED_IDS biti ayarlı ise efektif kullanıcı numarası yerine saved-set-used-ID kontrol edilir.
- POSIX.1 standardına göre sinyal numarası 0 null sinyaldir.
  - Bu sinyal gönderildiğinde sinyal gönderilmez. Hata denetimi ise yapılır.
    - Eğer bir işleme null sinyal gönderildi ve bu işlem yok ise, kill fonksiyonu -1 döner ve errno ESRCH olarak değiştirilir.
    - Ancak işlem numaraları tekrar kullanıldığı için sinyali gönderdiğiniz işlemin kontrol ettiğiniz işlem olup olmadığı konusuna dikkat edilmesi gerekir.
    - Ayrıca kill ile test yapma operasyonu atomik değildir.
      - Bu sebeple alınan cevaba fazla güvenmemek gereklidir.

alarm ve pause fonksiyonları

- Alarm fonksiyonu belirtilen süre sonra sonlanacak bir zamanlayıcı ayarlamak için kullanılır.
  - Zaman dolduğunda SIGALRM sinyali oluşur.
  - Bu sinyal görmezden gelinirse veya yakalanmazsa, varsayılan aksiyon işlemin sonlanmasıdır.

```
#include <unistd.h>

unsigned int alarm(unsigned int seconds);
```

Dönüş: 0 veya önceki alarm için kalan süreyi döner.

  - Her işlemin alarm için bir saati bulunur. Bu sebeple daha önce ayarlanmış ve sonlanmamış bir alarm var ise fonksiyon sonlanması için kalan süreyi döner.
  - Eğer ayarlanmış ve sonlanmamış bir alarm var ve sinyal süresi olarak 0 verilirse önceki sinyal iptal edilir. Bu durumda fonksiyon kalan süreyi döner.

alarm ve pause fonksiyonları

- Pause fonksiyonu çağıran işlemi bir sinyal alıncaya kadar durdurur.

```
#include <signal.h>

int pause(void);
```

Dönüş: -1 ve errno EINTR olur.

  - Pause fonksiyonu sadece bir sinyal yakalayıcı çalışır ve dönüş yaparsa döner.
  - read1.c