Sistem Programlama

Ders 16

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

İşlemcik yönetimi

- İşlemcik (ing: thread) işletim sisteminin önemli parçalarındandır.
- Normal bir Unix işlemi sadece bir kontrol akışına sahiptir.
- Birden fazla işlemcik oluşturarak birden fazla kontrol akışına sahip olabilir ve böylece aynı anda farklı görevleri gerçekleştirebiliriz.
- Bu yaklaşımın aşağıda belirtilen avantajları bulunur:
  - Asenkron olayları yöneten kodlamayı basitleştirebiliriz.
    - Her asenkron olayı başka bir işlemcik yönetebilir.
  - Çoklu işlem dosya paylaşımını karmaşık şekilde yapabilir. Aynı işlem içerisindeki çoklu işlemcik aynı adres alanını paylaşır.
  - Çözmeye çalıştığımız problemi daha küçük parçalara ayırabiliriz.
  - Interaktif programlar çoklu işlemcik kullanarak kullanıcı ile etkileşim yapan kısım ile diğer işleri birbirinden ayırabilir. Böylece geri dönüş zamanı düşürülebilir.

İşlemcik tanımlama

- İşlemlerin işlem numaraları olduğu gibi işlemciklerin işlemcik numaraları vardır.
- İki farklı işlemcik numarası aşağıdaki fonksiyon ile karşılaştırılabilir.

```
#include <pthread.h>

int pthread_equal(pthread_t tid1, pthread_t tid2);
```

Dönüş: eşit ise 0 olmayan değer, aksi takdirde 0.

- Bir işlemcik kendi işlemcik numarasını aşağıdaki fonksiyon ile öğrenebilir.

```
#include <pthread.h>

pthread_t pthread_self(void);
```

Dönüş: Çağıran işlemciğin işlemcik numarası

İşlemcik oluşturma

- Yeni bir işlemcik oluşturmak için pthread_create fonksiyonu kullanılabilir.
- **`#include <pthread.h>`**

  `int pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr, void *(*start_rtn)(void *), void *restrict arg);`

  Dönüş: OK ise 0, hata ise hata numarası.
  - tidp yeni oluşturulan işlemciğin işlemcik numarası olur.
  - attr argümanı ile işlemciğin bazı özellikleri ayarlanılabilir.
  - Yeni işlemcik start_rtn fonksiyonunun adresinden başlar.
  - arg argümanı gönderilecek argümanları içerir.
    - Birden fazla argüman gönderilecekse bir yapı oluşturup bunun adresini arg olarak vermek gereklidir.
  - Örnek34.

İşlemcik sonlanma

- Eğer işlem içierisindeki herhangi bir işlemcik exit, _Exit veya _exit fonksiyonlarını çağırırsa tüm işlem sonlanır.
- Benzer şekilde bir işlemciğe varsayılan aksiyonu sonlandırma olan bir sinyal gönderilirse tüm işlem sonlanır.
- Bir işlemcik işleminin tamamını sonlandırmadan aşağıdaki şekillerden biri ile kendini sonlandırabilir.
  - İşlemcik fonksiyonu dönüş yapabilir.
    - Dönüş değeri işlemciğin çıkış kodudur.
  - İşlemcik aynı işlem içerisinde bulunan başka bir işlemcik tarafından iptal edilebilir.
  - İşlemcik pthread_exit fonksiyonunu çağırabilir.

İşlemcik sonlanma

```
#include <pthread.h>

void pthread_exit(void *rval_ptr);
```

- rval tipi olmayan bir işaretçidir ve aşağıdaki fonksiyonda kullanılabilir.

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **rval_ptr);
```

Dönüş: OK ise 0, hata ise hata numarası

- Bu fonksiyonu diğer işlemcikler çağırırsa, belirtilen işlemcik fonksiyonu dönüş yapana veya işlemcik iptal edilene kadar bloklanırlar.
- İşlemcik fonksiyonu dönüş yaparsa, rval_ptr işlemciğin dönüş değerine eşit olur.
- İşlemcik iptal edilirse rval_ptr PTHREAD_CANCELED olur.
- Dönüş değerini öğrenmek istemiyor ve sadece işlemciği beklemek istiyorsak rval_ptr değerini NULL yapabiliriz.
- Örnek35.

İşlemcik sonlanma

- Bir işlemcik aynı işlem içierisindeki başka bir işlemciği iptal etmek isterse aşağıdaki fonksiyonu kullanabilir.

```
#include <pthread.h>

int pthread_cancel(pthread_t tid);
```

Dönüş: OK ise 0, hata ise hata numarası

- Bu isteği alan işlemcik sonlanabilir, veya bu isteği görmezden gelebilir.
- Bir işlem sonlanma sonrası yapılacak işlemleri belirtebilir.
  - atexit fonksiyonun benzeri.

```
#include <pthread.h>

void pthread_cleanup_push(void (*rtn)(void *), void *arg);

void pthread_cleanup_pop(int execute);
```

İşlemcik sonlanma

Process primitive	Thread primitive	Description
`fork`	`pthread_create`	create a new flow of control
`exit`	`pthread_exit`	exit from an existing flow of control
`waitpid`	`pthread_join`	get exit status from flow of control
`atexit`	`pthread_cleanup_push`	register function to be called at exit from flow of control
`getpid`	`pthread_self`	get ID for flow of control
`abort`	`pthread_cancel`	request abnormal termination of flow of control