Multicore/Multithread Programlama

Onur Tolga Şehitoğlu

Bilgisayar Mühendisliği

7 Şubat 2008

Başlıklar

Çok Dallılık Gereksinimi

- Çokluişlem (multitasking) → Çoklu süreç (multiprocess) → Çoklu dal (multithreading)
- Tek işlemcide Çok dallılık gereksinimi
  - Süreçler arası iletişim,
  - Bağlam değişimi (kernel mode/user mode) ↔ işlemci kullanımı
- Kullanıcı düzeyinde dallar (user level threads)

Çok Dallılık Gereksinimi

- Çokluişlem (multitasking) → Çoklu süreç (multiprocess) → Çoklu dal (multithreading)
- Tek işlemcide Çok dallılık gereksinimi
  - Süreçler arası iletişim,
  - Bağlam değişimi (kernel mode/user mode) ↔ işlemci kullanımı
- Kullanıcı düzeyinde dallar (user level threads)

Çok Dallılık Gereksinimi

- Çokluişlem (multitasking) → Çoklu süreç (multiprocess) → Çoklu dal (multithreading)
- Tek işlemcide Çok dallılık gereksinimi
  - Süreçler arası iletişim,
  - Bağlam değişimi (kernel mode/user mode) ↔ işlemci kullanımı
- Kullanıcı düzeyinde dallar (user level threads)

Çok Dallılık Gereksinimi

- Çokluişlem (multitasking) → Çoklu süreç (multiprocess) → Çoklu dal (multithreading)
- Tek işlemcide Çok dallılık gereksinimi
  - Süreçler arası iletişim,
  - Bağlam değişimi (kernel mode/user mode) ↔ işlemci kullanımı
- Kullanıcı düzeyinde dallar (user level threads)

Çok Dallılık Gereksinimi

- Çokluişlem (multitasking) → Çoklu süreç (multiprocess) → Çoklu dal (multithreading)
- Tek işlemcide Çok dallılık gereksinimi
  - Süreçler arası iletişim,
  - Bağlam değişimi (kernel mode/user mode) ↔ işlemci kullanımı
- Kullanıcı düzeyinde dallar (user level threads)

Dal ile Süreç

- Bellek alanı
  aynı alan $\leftrightarrow$ farklı alanlar
- Geçiş
  aynı bağlam $\leftrightarrow$ bağlam değişimi
- Senkronizasyon ve iletişim
  İşletim sistemi/karmaşık $\leftrightarrow$ kullanıcı modu/basit

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Dal Modelleri

- Kullanıcı düzeyi dallar
  - Çok hızlı, basit, kullanıcı kütüphanesi
  - Tek işlemci için ideal
  - Çok işlemciden faydalanamaz
- Kernel düzeyi dallar (Linux 2.2-2.4)
  - Dallar kernel tarafından süreç gibi yönetilir
  - Çoklu sürece benzer sorunlar
  - Yavaş
- Çok Dallı Kernel (Solaris, Linux 2.6)
  - Kullanıcı düzeyi/kernel etkileşimi
  - Etkin çok işlemci/çoklu süreç/çoklu dal
  - Kernel birden fazla işlemciyi kullanır

Neden Çoklu Dal Programlama?

- Tek işlemci: I/O yoğun işlemlerde işlemci verimliliği
- Çok işlemci: Her tür uygulamada paralellik ve verim
- İki işlemcili SMP sistemler artık dizüstü bilgisayarınızda!
- 4,8 hatta 16 çekirdek hazır.
- 64,80,128,160,... Çok yakında!

Neden Çoklu Dal Programlama?

- Tek işlemci: I/O yoğun işlemlerde işlemci verimliliği
- Çok işlemci: Her tür uygulamada paralellik ve verim
- İki işlemcili SMP sistemler artık dizüstü bilgisayarınızda!
- 4,8 hatta 16 çekirdek hazır.
- 64,80,128,160,... Çok yakında!

Neden Çoklu Dal Programlama?

- Tek işlemci: I/O yoğun işlemlerde işlemci verimliliği
- Çok işlemci: Her tür uygulamada paralellik ve verim
- İki işlemcili SMP sistemler artık dizüstü bilgisayarınızda!
- 4,8 hatta 16 çekirdek hazır.
- 64,80,128,160,... Çok yakında!

Neden Çoklu Dal Programlama?

- Tek işlemci: I/O yoğun işlemlerde işlemci verimliliği
- Çok işlemci: Her tür uygulamada paralellik ve verim
- İki işlemcili SMP sistemler artık dizüstü bilgisayarınızda!
- 4,8 hatta 16 çekirdek hazır.
- 64,80,128,160,... Çok yakında!

Neden Çoklu Dal Programlama?

- Tek işlemci: I/O yoğun işlemlerde işlemci verimliliği
- Çok işlemci: Her tür uygulamada paralellik ve verim
- İki işlemcili SMP sistemler artık dizüstü bilgisayarınızda!
- 4,8 hatta 16 çekirdek hazır.
- 64,80,128,160,... Çok yakında!

Çok İşlemci/Çok Çekirdek

- Symmetric Multi Processor systems
- Çok Çekirdekli İşlemciler
- Aynı bellek alanını paylaşan süreç ve dallar
- İletişim masrafı çok düşük
- Ön bellek verimliliği hala önemli

POSIX Thread Kütüphanesi

- Kullanıcı düzeyi kütüphane → kernel/kullanıcı etkileşimi
- Bir süreç birden fazla dal çalıştırabilir
- Her dal farklı bir işlemcide çalışabilir.
- Dallar aynı bellek alanına erişebilir
- Her dalın ayrı bir çalışma zamanı yığıtı vardır.

Dalları yaratma ve Sonlandırma

- `pthread_create(pthread_t * tidp, pthread_attr_t *attr, void *(*start)(void), void *arg)`
  Yeni bir dal yaratarak belirtecini ilk parametreye koyar,
  yeni dal `start` isimli fonksiyonu, `arg` parametresi ile çağırır
- `pthread_exit(void *rval);`
  Bir dal bu çağrıyı yaparak yada başlangıç fonksiyonundan `return(rval)` yaparak sonlanır.
- `pthread_join(pthread_t id, void **rval)`,
  Başka bir dal sonlanan dalın çıkış değerini almak ve beklemek için çağırır.
- `pthread_t pthread_self()`
  Dalın kendi belirtecini almasını sağlar

Örnek: Merhaba Dünya

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>

void *helloworld(void *arg) {
        printf("Hello, I am: pid %u tid %u \n",
                getpid(),(unsigned)pthread_self());
}
int main(int argc,char *argv[])
{
        pthread_t tr[5];
        int i,r;

        for (i=0;i<5;i++)
                pthread_create(&tr[i],NULL,helloworld,NULL);
        for (i=0;i<5;i++)
                pthread_join(tr[i],NULL);

        return 0;
}
```

Örnek: Paralel Sıralama

- Böl, bölümleri sırala, birleştir: $O\left(\frac{N}{P}log(\frac{N}{P}) + N\right)$
- Dağıtık sistemler:
  $O(Nlog(N)) < comm. + O\left(\frac{N}{P}log(\frac{N}{P}) + N\right)$
- Merge Sort ya da Quick Sort

Sonuçlar

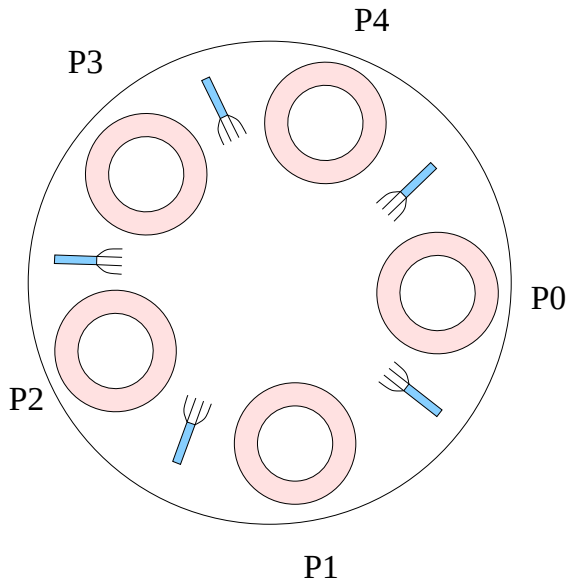
...

Senkronizasyon/Mutex

- Mutex: karşılıklı dışlama
- İkili semafor eşleniği
- Kritik alanlar yaratma
- ```
  int pthread_mutex_init(pthread_mutex_t * mutex,
                         pthread_mutexattr_t *r attr);
  int pthread_mutex_destroy(pthread_mutex_t * mutex);
  ```
- ```
  int pthread_mutex_lock(pthread_mutex_t *mutex);
  int pthread_mutex_trylock(pthread_mutex_t *mutex);
  int pthread_mutex_unlock(pthread_mutex_t *mutex);
  ```

Örnek: Yemek Yiyen Düşünürler

- Düşünürler bir süre düşünüp sonra yiyorlar
- Yemek için iki çatala birden gerek var
- Dolayısıyla komşu düşünürler beraber yiyemez
- Bir süre yedikten sonra iki çalalı da bırakıp düşünmeye başlıyorlar.



```
pthread_mutex_t forks[5];
void philosopher(int *ip) {
        pthread_mutex_t *sol,*sag;
        int i=*ip, z;

        sol=i==0?forks+4:forks+(i-1);        sag=forks+i;
        for (z=0;z<1000;z++) {
                usleep(10000);
                pthread_mutex_lock(sol);
                pthread_mutex_lock(sag);
                printf("Philosopher %d is eating %d\n",i,z);
                usleep(10000);
                printf("Philosopher %d finished eating %d\n",i,z);
                pthread_mutex_unlock(sol);
                pthread_mutex_unlock(sag);
        }
}
int main() {
        pthread_t phils[5];
        int phids[5]={0,1,2,3,4}, i,n;

        for (i=0;i<5;i++) {
                pthread_mutex_init(&(forks[i]),NULL));
        for (i=0;i<5;i++) {
                pthread_create(&(phils[i]), NULL, philosopher, (void *) &(phids[i]));
        }
        for (i=0;i<5;i++) {
                pthread_join(phils[i],NULL);
        }
        return 0;
}
```

İki Fazlı Kilitleme

- Okuma kilitleri paylaşılır, yazma kilitleri diğerlerini dışlar
- Farklı rollerde çok dal
- ```
  int pthread_rwlock_init(pthread_rwlock_t * rwlock, ...
  attr);
  int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
  ```
- ```
  int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
  int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
  int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);

  int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
  int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);
  ```

Bekleme ve Sinyalleşme

- Koşul değişkenleri üzerinde senkronizasyon
- Dal A: koşul gerçekleşene kadar bekle
  Dal B: koşulu sağla ve A'ya haber ver
- Koşulu korumak için mutex kullanılır
- ```
  int pthread_cond_init(pthread_cond_t * cond, ...  * attr);
  int pthread_cond_destroy(pthread_cond_t *cond);

  int pthread_cond_wait(pthread_cond_t * cond,
                        pthread_mutex_t * mutex);
  int pthread_cond_timedwait(pthread_cond_t * cond,
         pthread_mutex_t * mutex, const struct timespec * timeout);

  int pthread_cond_signal(pthread_cond_t *cond);

  int pthread_cond_broadcast(pthread_cond_t *cond);
  ```
  ```

- Üretici:
  - Kuyrukta yer varsa veriyi kuyruğa sok, tüketiciye haber ver
  - Kuyruk doluysa boş yer oluncaya kadar bekle
- Tüketici:
  - Kuyruk boş değilse veriyi kuyruktan al, üreticiye haber ver
  - Kuyruk boşsa veri oluncaya kadar bekle

Özet

- Çok çekirdek/çok işlemci yaygınlaştıkça çok dallı programlama elzem
- YBB dışında da paradigma bu tarafa kayıyor.
- Basit ama denetimi, sorun giderilmesi performans ölçümü zor.
- Önceden tasarlamak önemli.
- Teşekkürler Sorular?