13. Eş Zamanlılık
13.1. Giriş

Yazılımların eş zamanlı çalışması 4 farklı düzeyde gerçekleşebilir :

- Komut düzeyinde – iki veya daha fazla makina komutunun eşzamanlı çalışması
- Deyim düzeyinde – İki veya daha fazla yüksek seviyeli dil deyiminin eş zamanlı çalışması
- Birim düzeyinde – İki veya daha fazla alt programın eş zamanlı çalışması
- Program düzeyinde – İki veya daha fazla programın eş zamanlı çalışması

Komut ve program düzeyindeki eş zamanlılığın dil tasatımı ile bir ilgisi olmadığından diğer ikisine odaklanacağız.İlk bakışta eş zamanlılık basit bir konu gibi gözükse de programcı, programlama dili tasarımcısı ve işletim sistemi tasarımcısı (çünkü eş zamanlılık desteğinin büyük bir kısmı işletim sisteminden gelir) için büyük zorluk içerir.

13.1.1. Eş zamanlı kontrol mekanizmaları

Programın esnekliğini artırırlar. Bu mekanizmalar ilk olarak işletim sistemlerinde karşılaşılan problemleri çözmeye yönelik olarak ortaya çıksa da başka birçok uygulamada da kullanılma ihtiyacı oluşmuştur. Bunlardan en geneli web tarayıcılardır. Tarayıcılar aynı anda birçok iş yapmak zorundadırlar. Web servislerine veri gönderme ve web servislerinden veri alma, ekrandaki metin ve görüntüleri işleme ve kullanıcı hareketlerine cevap verme gibi birçok görevi aynı anda yapmak durumundadırlar.

Benzer şekilde, gerçek fiziksel sistemleri simüle eden yazılımlar da birçok alt program biriminin eş zamanlı çalışmasını gerektirir.

Deyim düzeyinde eş zamanlılık ise birim düzeyindekinden oldukça farklıdır. Dil tasarımcısının gözünden deyim düzeyinde eş zamalılık verinin çoklu belleğe nasıl paylaştırılacağı ve hangi deyimlerin eş zamanlı olacağı problemidir.

Eş zamanlı çalışan yazılım geliştirmenin amacı ölçeklenebilir ve taşınabilir eş zamanlı algoritmalar üretmektir. Daha fazla işlemci uygun olduğunda çalışma hızı artıyorsa o eş zamanlı algoritma ölçeklenebilir bir algoritmadır. Bu konu önemlidir çünkü işlemci sayısı her yeni nesil makina ile artmaktadır. Ayrıca algoritmalar taşınabilir olmalı, herhangi bir mimariye bağlı olmalaıdır. Çünkü donanımın yaşam süresi nispeten kısadır.

13.1.2. Eşzamalılık kategorileri

Fiziksel eş zamanlılık – birden fazla işlemcinin bulunduğu durumda aynı programa ait birimlerin gerçekten eş zamanlı olarak çalışabilmesi durumudur.

Mantıksal eş zamanlılık – programların çalışması tek bir işlemci üzerinde gerçekleşir ancak yazılım birden çok işlemci olduğunu varsayar. Programcının ve dil tasarımcısının gözünden mantıksal eş zamanlılık da fiziksel eş zamanlılık gibidir.

Tek bir program akışı içinde birden fazla işlemin gerçekleştirilmesi için iş parçacıkları (thread) kullanılır. Thread kullanımı yoksa program akışı sırasıyla tüm komutların işlenmesi şeklindedir. Thread kullanımı, birden fazla işlemin tek bir akışı paylaşarak eşzamanlı şekilde gerçekleşmesini sağlar.

Fiziksel eş zamanlılıkla çalışan programlar çok tread' li kontrole sahip olabilirler. Her bir işlemci bir threadi çalıştırabilir. Çok threadli(multithreaded) bir program tek bir işlemci üzerinde çalışıyorsa threadleri tek bir thread' e eşlenir.

Deyim düzeyi eşzamanlılık daha kolay bir konudur. Örneğin dizi elemanları üzerinde işlem yapılan döngülerde 500 elemanlı bir dizinin elemanları 10 işlemci üzerine 50 şer dağıtılabilir ve üzerlerinde işlem yapılabilir.

13.1.3. Eş zamanlılık ne sebeplerle kullanılıyor

Eş zamanlı yazılım sistemleri geliştirmek için dört ana neden bulunmaktadır. İlk sebep çok işlemcili makinalarda çalışma hızının yüksek olması ve bu donanımı kullanabilmek için eş zamanlı yazılımlar geliştirilme ihtiyacı duyulmasıdır. İkinci sebep, kullanılabilecek sadece tek işlemci bile olsa, eşzamanlı yazılımların sıralı işleme yapan yazılımlardan daha hızlı çalışmasıdır. Üçüncü sebep, eşzamanlılığın problem çözümüne farklı bir yaklaşım getirmesidir. Nasıl ki bazı problemler doğal olarak özyineleme ile çözülebilmekteyse, benzer şekilde doğası gereği eşzamanlılıkla çözülmesi gereken problemler de vardır. Dördüncü sebep, birçok makina veya internet üzerine dağılmış olan uygulamaları programlamaktır (örneğin internet üzerinden oynanan oyunlar).

Eş zamanlılık şu anda günlük işlerde de kullanılmaktadır. Eş zamanlı doküman isteklerini işleyen web servisleri, web tarayıcıları, işletim sistemleri vb. Örnek olarak verilebilir.

13.2. Alt-program Düzeyi Eş Zamanlılık

Görev (Task) – Alt program benzeri program birimleridir. Aynı programdaki diğer birimlerle eş zamanlı olarak çalışabilir. Bir programdaki her görev, bir iş parçacığı (thread) destekleyebilir. Görevler aynı zamanda süreç (process) olarak da adlandırılabilirler. Java ve C# gibi bazı dillerde belirli metodlar görev olabilirler. Bu tür metodlar Thread nesneleri üzerinde çalışırlar. Görevleri alt programlardan ayıran üç temel özellik aşağıdaki gibidir.

1) Görevler üzeri kapalı bir şekilde başlatılabilirler ancak alt programlar açık şekilde çağrılmalıdır.
2) Bir program birimi bir görev başlattığında kendi çalışmasına devam edebilmek için görevin çalışmasının bitmesi gerekmeyebilir. Ancak alt program çağrısı yapan birim kendi çalışmasına devam edebilmek için alt programın çalışmasını bitirmesi gerekir.
3) Görev çalışmasını tamamladığında kontrol görevi başlatan birime dönmek zorunda değildir.

Görevler iki genel kategoriye ayrılır: Hafif (lightwight) ve ağır (heavyweight). Ağır görevler kendilerine ait adres uzayında çalışırlar. Hafif görevler aynı adres uzayını başka görevlerle paylaşır. Gerçekleştirimleri daha kolaydır. Daha az

zahmetli çalışmasından dolayı daha etkindirler.
Bir görev diğer görevlerle, paylaşımlı yerel olmayan değişkenler, mesaj gönderme veya parametreler aracılığıyla haberleşeblir. Eğer başka hiçbir görevle haberleşmiyorsa veya görevlerin çalışmasını etkilemiyorsa o göreve ayrık (disjoint) görev denir.

Senkronizasyon, görevlerin hangi sıra ile işletileceğini kontrol eden mekanizmadır. İki tip senkronizasyon görevlerin veri paylaşımı sırasında kullanılır: işbirliği (cooperation) ve rekabet (competition).

13.2.1. İşbirliği ve Rekabet Senkronizasyonu

Görev A, çalışmasına başlamak veya çalışmasını devam ettirmek için görev B'nin belirli işlemlerini bitirmesini beklemek zorunda ise orada bir işbirliği senkronizasyonu gereklidir.

İki görevin her ikisi de eş zamanlı kullanılmayan bir kaynağa ihtiyaç duyuyorsa orada rekabetçi senkronizasyon kullanımı gerekir. Örneğin görev A' nın paylaşımlı bellek bölgesi olan X' e erişmesi gerekiyor. Tam o sırada görev B o bölgeye erişim sağlamış. Bu durumda A, B' nin X' i bırakmasını beklemek zorundadır. A' nın çalışmaya başlaması veya devam etmesi B' nin bir sonuç üretmesine veya çalışmasını bitirmesine bağlı değildir.

İşbirliği senkronizasyonu için en basit örnek çok bilinen üretici- tüketici problemidir. Bu problemin kökeni işletim sistemlerinin geliştirilmesine dayanır. Bir program biriminin ürettiği veriyi diğer program birimi kullanmaktadır. Üretilen değer üretici tarafından bir tampon belleğe koyulur ve tüketici de o değeri oradan alıp kullanır. Tampon belleğe veri koyma ve oradan veri alma işlemleri senkronize edilmelidir. Eğer tampon bellek boşsa, tüketici oradan veri alamamalıdır. Benzer şekilde eğer tampon bellek doluysa üreticinin yeni veri koymasına izin verilmemelidir. Bu bir işbirliği problemidir çünkü taraflar bir veri yapısının doğru kullanılabilmesi için işbirliği içinde çalışmalıdır.

Rekabet senkronizasyonu iki görevin aynı veri yapısına aynı anda erişmesini önler. Karşılıklı dışlama (mutual exclusion) ile rekabet senkronizasyonu gerçekleştirilir.

Rekabet senkronizasyonunu daha iyi anlayabilmek için şu senaryoyu düşünelim:

Görev A, TOTAL+=1 ve görev B de TOTAL *= 2 işlemlerini gerçekleştiriyor. Burada TOTAL, paylaşımlı bir integer değişken olsun. Görev A ve görev B aynı anda TOTAL' in değerini değiştirmek isteyebilirler. Makina dili seviyesinde bu değişimi gerçekleştirebilmek için görevlerin tamamlaması gereken adımlar şunlardır:

1) TOTAL' in değerini bellekten al
2) Aritmetik işlemi gerçekleştir
3) TOTAL' in yeni değerini bellekteki yerine yaz

Eğer rekabet senkronizasyonu kullanılmazsa işlem sırasına göre farklı sonuçlar elde edilebilir. Örneğin başlangıçta TOTAL = 3 olsun. Önce A sonra B

görevlerinin çalıştığını düşünelim. Eğer belleğe önce A değeri yazarsa B 8 değerini üretir. Bu durumda bir sorun yoktur. Eğer iki taraf da önce 3 değerini bellekten çekerse o zaman problem oluşabilir. Eğer önce A TOTAL'i yerine yazarsa bellekte 6, eğer B yazarsa bellekte 4 değeri olur. Bu probleme aynı zamanda yarış problemi denmektedir çünkü görevler aynı bellek bölgesi için yarışmaktadır.

Paylaşılacak olan kaynağa karşılıklı dışlamalı erişim sağlamak için en genel yöntem o kaynağa belli bir zamanda sadece tek bir görevin sahip olmasını sağlamaktır. Görev bir kaynağa sahip olabilmek için istek göndermelidir. Eğer o sırada kaynağa sahip olan başka bir görev yoksa kaynak istek gönderen göreve verilebilir. Kaynağa sahip olan bir görev varsa diğer tüm görevlerin bu kaynağa erişimleri engellenir. Karşılıklı dışlama sağlamak için 3 yöntem bulunmaktadır: semaforlar, monitörler ve mesaj geçirme.

Senkronizasyon mekanizmaları görevlerin çalışmalarını erteleyebilmek zorundadır. İster tek ister çok işlemci olasun her zaman işlemci sayısından daha çok görev bulunma ihtimali vardır. İşlemcileri görevler arasında paylaştıran çalışma zamanı sistem programına **scheduler** denilmektedir.

Bir programa ait canlılık kavramı vardır. Bir program çalışmasını sürdürüyorsa ve sonunda tamamlanacaksa o program canlıdır denir.

Paylaşımlı kaynaklrın ve eş zamanlılığın olduğu ortamlarda programın devam edemediği ve hiçbir zaman sonlanamayacağı durumlar oluşabilir. Örneğin, görev A ve B, X ve Y kaynaklarını paylaşıyor olsunlar. Görev A, X'e, görev B de Y' ye sahip olsun. Bir süre sonra A' nın işlerini tamamlaması için Y' ye, B' nin de X'e sahip olması gereksin. Bu durumda ne görev A ne de görev B işlemlerini tamamlayabilir ne de birbirlerindeki kaynaklara sahip olabilirler. Bu durumlarda program canlılığını yitirir. Buna ölümcül kilitlenme (dead lock) diyoruz. Ölümcül kilitlenme, tehlikeli ve önlenmesi gereken bir durumdur. Hem dil, hem de program tasarımında dikkate alınması gerekir.