

Öğrencinin;

ADI : Erdem Çağrı

SOYADI : GÖRÜCÜ

BÖLÜM : Bilgisayar Mühendisliği

Dersin;

ADI : İşletim Sistemleri

EĞİTMEN : Prof.Dr.Ali Yılmaz ÇAMURCU

**Dining Philosophers Problemi**

[](http://www.fatihcelen.com/wp-content/uploads/2014/01/Resim1.png)

Yemek yiyen filozoflar örneği, literatüre **Dijkstra** tarafından kazandırılmıştır ve eş zamanlı işlem yönetimini (concurrent process management) sembolize eder. Filozoflar bu problemde ya pirinç yerler yada düşünürler yani aynı işi birarada yapamazlar.

Masada filozof sayısı kadar tabak ve tabakların arasında birer çatal vardır. Filozoflar makarnayı tek çatalla yemek zor olduğu için 2 çatalla yiyebiliyorlar. Maalesef her filozofa bir çatal düşüyor ve yemeğe hepsi aynı anda başlıyorlar. Bu yüzden hepsinin elinde birer tane çatal olmasına rağmen, hepsi diğer çatalı bekliyor ve hiçbiri yemek yiyemiyor. Böylece sonsuza kadar aç kalıyorlar!

Problem Sorunları

* **Racing** Condition: hepsi iki yanındakini birden almaya kalkarsa, bu durumda, eş zamanlı işlemlerde karşılaşılan yarış durumu (racing condition) ortaya çıkacaktır ve hangisi önce davranırsa o yemeğini yiyebilecektir. Yada bir diğeri yesin diye hiçbiri çatal almayıp aç kalabileceklerdir. Bu tip problemler, genelde kıtlık problemi (starvation) olarak düşünülebilir.
* **Deadlock**: Yanlış bir tasarım sonucunda, tek çatal alan ve çatalı bırakmak için diğer filozofun bırakmasını bekleyen bir filozof sistemi kilitleyebilir. Bu da problemde bulunan ikinci risktir.

Son olarak problemin tanımında, filozoflar birbiri ile konuşamaz kuralı getirilmiştir. Aslında filozoflar birbiriyle çatallar üzerinden iletişim kurmaktadır. Örneğin sağındaki veya solundaki filozofun o anda çatalı alıp almaması, yanındaki filozoflar hakkında bilgi vermekte ve bu da üstü kapalı bir iletişim olarak kabul edilmektedir. Çözümlerin tamamında iletişim sadece çatalların durumuna göre sağlanmaktayken, sadece son çözüm olan chandy misra çözümünde filozoflar doğrudan birbiri ile iletişime geçebilmektedir.

**Problemin çözümü için geliştirilen algoritmalar;**  
1. Rastgele süre çözümü (Random Solution)  
2. Garson çözümü (Conductor Solution)  
3. Monitör Çözümü (Monitor Solution)  
4. Chandy Misra Çözümü (Chandy Misra Solution)

**Rastgele süre çözümü (Random Solution)**

Filozofların problemi çözmek için tamamen rastgele davranması öngörülür. Filozoflar, bir çatal aldıktan sonra ikincisini alabilirse yemeğini yer. Şayet ikinci çatalı alamazsa rastgele bir süre ikinci çatalın boşalmasını bekler. Eğer yanındaki çatalı bırakırsa yemeği yer yada bu süre içerisinde çatalı bırakmazsa yemeği yemeden elindeki çatalı yanındakinin yemesi için bırakır. Sistem rastgelelik üzerine kurulduğu için başarıya ulaşması beklenmez.

**Garson çözümü (Conductor Solution)**

Bu çözümde masanın etrafında bir garsonun dolaşması ile çözüm üretilmeye çalışılmıştır. Garson, sürekli olarak masada boş duran ve filozoflar tarafından yemek için kullanılan çatalların sayılarını takip etmektedir. Bir şekilde her filozof, masadan çatal alabilmek için garsonun iznini istemek zorundadır. Şayet garson izin vermezse filozof masadan çatal alamaz. Eğer ki garson bütün filozoflara yemek imkanı sunarsa deadlock ihtimalinden kurtulmuş olunur ve her filozof aç kalmadığı için kıtlık problemi(starvation) ile karşılaşmaları engellenir.

Örneğin garsonun masayı saat yönünde döndüğünü ve bir filozofu başlangıç kabul ettiğimizi düşünelim.O anda işaretlediği filozof yemek yiyor, sonraki yemiyor , sonraki yiyor ve böylece kaç filozof varsa, sırayla bir yiyor bir yemiyor şeklinde düşünülebilir. Bu durumda her filozof yemek yemek için yeterli çatalı buluyor demektir.7

**Monitör Çözümü (Monitor Solution)**

Bu çözüm, garson çözümüne çok benzemektedir. Amaç sırasıyla her filozofun bir yiyen bir de yemeyen şeklinde sıralanmasıdır. Burada her filozof belirli bir sırayla sıralanmaktadır (örneğin saat yönünde veya saat yönünün tersi istikamette) ardından kendinden önceki filozofun durumunu kontrol ederek yemek yiyorsa yemez, kendinden önceki filozof yemek yemiyorsa bu durumda kendisi yemek yer.

Tek sayıda filozof olması durumunda yemek yeme eyleminin masada bir dalga şeklinde bir noktadan başlayarak sürekli döndüğü görülebilir. Şayet filozof sayısı çift ise bu durumda sürekli aynı filozoflar yemek yerken diğer filozoflar ölecektir. Bir çözüm olarak, şayet toplam filozof sayısı çift ise, sırasıyla tek ve çift filozoflara yemek yedirmek bir çözüm olabilir. Örneğin önce 1,3,5 numaralı filozoflar yemek yerken, sonra 2,4,6 numaralı filozoflar yemek yiyebilir.

**Chandy Misra Çözümü (Chandy Misra Solution)**

Çözümün en önemli özelliği, merkezi bir karar mekanizmasını ortadan kaldırması ancak buna karşılık, filozoflar birbiri ile konuşamaz kuralını çiğnemesidir. Bu çözüm, geliştiren iki kişinin ismi ile anılmaktadır (K. Mani Chandy ve J. Misra). Çözüm 4 adımdan oluşmaktadır.  
Bu Adımlar;  
1. Her filozof ikilisi için bir çatal üretilir ve bu çatal en düşük sayı sahibi olan filozofa verilir. Her çatal kirli veya temiz olarak işaretlenebilir ve başlangıç durumunda bütün çatallar kirlidir.

2. Bir filozof, bir kaynak kümesini kullanmak istediğinde (yani yemek yemek istediğinde), komşusu olan çatalları kullanmak zorundadır. Elinde olmayan (ihtiyacı olan) bütün çatallara bir talep yollar.

3. Bir filozof, elindeki bir çatal için talep aldığında, şayet elindeki çatal temizse kullanmaya devam eder, şayet çatal kirli ise, çatalı masaya koyar. Ayrıca masaya konan çatal temiz olarak işaretlenerek konulur.

4. Bir filozof yemek yedikten sonra çatalı kirli olarak işaretler. Şayet bir filozof, daha önce bir çatalı talep ettiyse, çatalı temizleyerek masaya koyar.  
Çatallara kirli ve temiz şeklinde numaralar verilir. Çatalın talep edilmesi veya edilmemesi durumu kirli veya temiz olma durumuna göre belirlenir. Çatalın üzerinde talep olması çatalın kirli olduğu anlamına gelmektedir. Her çatal sadece talep eden filozof tarafından kullanılacağı için de kullanma işlemi öncesinde tek bir filozofa atama yapılmış oluyor. Ayrıca filozofların talep işleminin gerçekleşebilmesi için iki çatalın birden atamasının yapılması gerekmektedir.

Burda deadlock oluşma ihtimalini ortadan kaldırmak için numaralandırma yapılmıştır. Yani filozofların hepsi aynı önceliğe sahip olduğu durumlarda bir kilitlenme ihtimali bulunmakla birlikte, bu ihtimali bertaraf etmek için her filozofa bir numara verilmiş ve bu numaraya göre en düşük değere sahip filozof öncelikli olmuştur. Bir diğer sebep olan kıtlık sorunu ise çatalların temiz yada kirli olma ihtimali ile engellenmiştir.

**Banker Algoritması**

Aynı kaynaktan birden fazla olan sistemlerde banker algoritması (banker’s algorithm) kullanılabilir.

Banka sisteminde, bir banka parasını tüm müşterilerinin ihtiyacını karşılayamayacak seviyeye hiçbir zaman düşürmez. Bir process sisteme geldiğinde tüm kaynaklar için maksimum ihtiyaçlarını bildirir.

Bu kaynak miktarı sistemin tüm kaynaklarından fazla olamaz! Bir process bir grup kaynak istediğinde, sistem bu atamanın yapılması halinde safe state’in korunup korunmadığına bakar.

Banker algoritması aşağıdaki veri yapılarını kullanır:

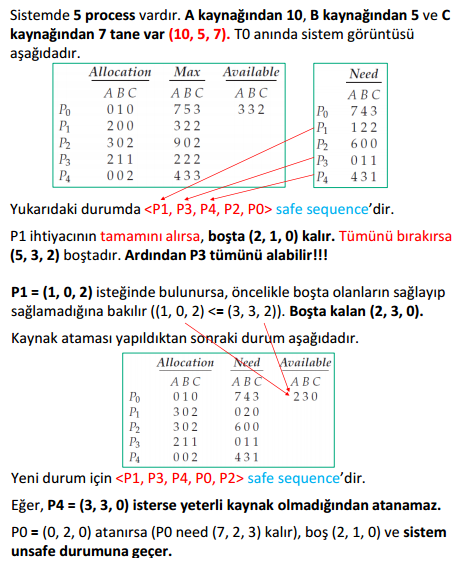
**Available**: Bir vektördür. Sistemde boştaki tüm kaynakların sayısını tutar.  
 Available(j) = k ise, sistemdeki Rj kaynağından k adet boştadır.

**Max**: n \* m matristir. Her process’in maksimum kaynak talebini tutar.   
 Max[i][j] = k ise Pi process’i Rj kaynağından en fazla k tane isteyebilir.

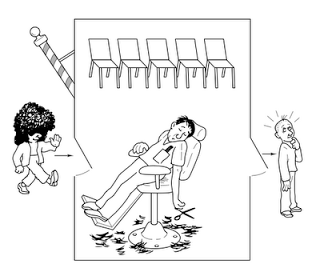
**Allocation**: n \* m matristir. Her process’in kullanmakta olduğu kaynak miktarını tutar.   
 Allocation[i][j] = k ise, Pi process’i Rj kaynağından k adet kullanmaktadır.

**Need**: n \* m matristir. Her process’in kalan ihtiyaç miktarını tutar.   
 Need[i][j] = k ise Pi process’i Rj kaynağından k adet daha kullanabilir.   
 Need[i][j] = Max[i][j] – Allocation[i][j] olur.

Veri yapısındaki veri boyutu (satır) ve değerleri dinamik olarak değişebilir (çalışan process sayısı değişebilir).



**Uyuyan Berber Problemi**

[](http://1.bp.blogspot.com/-Ny6veEdvjxY/ULaA8mHAPJI/AAAAAAAAF5U/A11ks9gL-HY/s1600/sleepingbarber.png)

Bilgisayar bilimlerinde, “Uyuyan berber problemi” klasik bir sorundur ve süreçler arası iletişim (inter-process communication) ile birden fazla bilgisayar arasındaki senkronizasyon problemini çok güzel bir şekilde açıklar. Bu problem, genellikle temel programlama öncülerinden biri olan **Edsger Dijkstra** (1965) ile anılır.  
  
Problemi kısaca açıklarsak, bu problem aynı şuna benzemektedir; bir berber dükkanında çalışan berber vardır. Berber müşterileri olduğu zaman çalışmaktadır, hiç müşteri olmadığı zaman da dinlenmektedir. Bunları sırasıyla yapmaktadır. Berber ve müşterileri yukarıda bahsettiğimiz söz konusu süreçleri temsil etmektedirler. Bu berberin bir tane berber sandalyesi vardır. Bekleme odasında ise birden çok sandalye vardır. Berber, bir müşterinin saçını kesmeyi bitirdiği zaman onunla işi biter. Sonra gidip bekleme odasına bakar başka müşteri var mı diye. Eğer birileri varsa, onlardan bir tanesini alır ve berber sandalyesine getirir. Fakat bekleme odasında kimse yoksa, berber kendi masasına gider ve uyur. Her bir müşteri dükkana geldikleri zaman şöyle bir içeriye bakar. Eğer berberi uyurken yakalarsa, hemen uyandırır ve berber sandalyesine oturur. Fakat berber içerde birinin saçını kesiyorsa, o halde bekleme odasına gider ve bakar. Eğer bekleme odasında boş bir sandalye bulursa ona oturur ve kendi sırasını bekler ama hiç sandalye bulamazsa berberi terk eder. İşte müşteriler böyle davranmaktadırlar. Berber bir müşterinin saçını kesmeyi bitirirse yine gider bekleme odasına bakar, kimse yoksa gider uyur. Sonra yeni bir müşteri gelince, müşteri onu uyandırır. Burada aslında doğabilecek çeşitli sorunlar vardır.

Problemdeki Sorunlar

* Doğabilecek problemlerin başında şu geliyor: yapılan eylemlerin hepsi bilinmeyen sürelerde gerçekleşmektedir ve bu da sorunlara yol açar; mesela eylem olarak bekleme odasını kontrol etmek, dükkana girmek, bekleme odasında bir sandalyeye oturmak... vb. olabilir. Bunu şu şekilde gözümüzde canlandırabiliriz; bir müşteri dükkana gelir ve berberi saç keserken görür, sonra bekleme odasına bakmaya gider. Bu esnada berber saç kesmeyi bitirir ve o da bekleme odasına bakmaya gider. Tabii bekleme odasına müşteri gidene kadar berber çoktan gitmiştir. Berber bakar ki kimse yok. O halde gider uyur. Ama müşteri berber uyurken bekleme odasına girer ve oturur. İşte bu müşteri, başka bir müşteri gelip de berberi uyandırıncaya kadar beklemek zorunda kalır. Başka bir müşteri gelmezse bu ikili sonsuza dek birbirini bekler. Berber müşteriyi bekler, müşteri berberi bekler.
* Diğer bir örnek ise; iki müşteri aynı anda berbere geldi diyelim. Müşteriler berberi kontrol ederler ve onun saç kestiğini görürler. Sonra bu müşteriler bekleme odasına giderler ve orda bir tane sandalye olduğunu görürler. Müşterilerin ikisi de aynı anda bu sandalyeye oturmaya çalışır.

**Çözüm Yolları**  
  
Uyuyan berber problemine çeşitli çözümler önerilmiş. Bu çözümlerin temelinde “aynı anda sadece bir kişi eylem yapabilir” prensibi yatmaktadır. Fakat berber bazı durumlarda bu prensibin dışında tutulur, mesela müşterileri kontrol etmeden önce (uyumaya ya da saç kesmeye karar vermeden önce) gibi. Müşteriler ise berber dükkanına girdikleri anda bu prensibe uygun davranırlar. Mesela müşterilerin bekleme odasında mı bekleyecekler, ya da saç mı kestirecekleri belli olmayan durumda müşteriler tek tek eyleme tabii tutulur. Bu prensip böylece bahsedilen problemi çözer.  
  
Birden fazla semafor kullanarak da sistemi gösterebiliriz, örneğin bekleme odasındaki müşterilerin sayısını ve berberin saçını kestiği kişi sayısını tutmak. Berberlerin sayısı arttıkça problemdeki tuzaklar da artmaktadır ama problemi daha gerçekçi yapmaktadır, çünkü berberlerde genelde birden fazla berber çalışmaktadır. Kullandığımız metafor gerçek hayattaki problemlere ne kadar benzerse karmaşıklığı da o kadar artmaktadır.

**Uyuyan Asistan Problemi ( İşletim Sistemleri Projesi )**

Bilgisayar mühendisliğindeki bir asistan ofis saatleri içerisinde öğrenci bekler. Asistanın odasında bir masa ve bir sandalye vardır. Asistan odası dışında 3 sandalye vardır. Bu sandalyelerde öğrenciler sıra bekler. Ofis saatleri içerisinde eğer öğrenci yok ise asistan masasında uyur. Ofis saatleri içerisinde gelen bir öğrenci asistanı uyurken bulur ve asistanı yardımcı olması için uyandırır. Eğer öğrenci geldiğinde asistanın yanında öğrenci varsa bekler, eğer dışarısı doluysa daha sonra gelmek üzere oradan ayrılır.

* Çözüm Programının Çalışır Haldeki Görüntüsü

