HAFTA 1 - HİSTORY OF OPERATİNG SYSTEMS

Multiprograming

- Aynı anda birden fazla programın çalıştırılmasıdır.
- Program çalıştırıldığında RAM'a yüklenir ve işlemciye girmek için beklemeye alınır. RAM'da bekleyen programlar sırayla işlemciye alınır.
- İşlemciye alınan program işi bitinceye kadar(herhangi bir interrupt gelmezse) işlemcide kalır. Yani her program işlemcide eşit sürede kalmaz.

Time sharing

- Multiprograming in bir alt versiyonu.
- Farkı çalışacak programların hepsi için işlemcide kalma süresinin eşir olmasıdır. Her program işlemcide eşit sürede kalır.

NOT: İşlemcide tüm programlar aynı anda çalışmamasına rağmen bilgisayar görüntülemede herhangi bir kopukluk olmaz. Çünkü bilgiler bufferda tamponlanır.

- Diğer bir yenilik olan **Spooling**(Biriktirme), işleri bir arabelleğe veya bellekte bir alana veya bir aygıtın hazır olduğunda erişimine izin veren bir diske koyma.

Monoprograming

- Bir OS'un bir anda tek bir iş yapması durumudur.

* Unix, timesharing mantığıyla daha çok çalışmaktadır.

* MS-DOS multitasking özelliği yok.

İŞLETİM SİSTEMİ TÜRLERİ(OS TYPES)

1. **Mainframe OS**
- yoğun input-output gerektiren çok sayıda görev çalıştırmaya yönelik sistemler için kullanır.
2. **Server(Sunucu) OS**
- Bilgisayar ağı üzerinden çok sayıda kullanıcıya hizmet verirler.
3. **Multiprocessor OS**
- Birden fazla işlemcisi olan tek sistem bilgisayarlarda kullanırlar.
- Bağlanma şekillerine göre : Paralel sistemler, Grid sistemler ve Çok işlemcili sistemler olarak 3'e ayrılırlar.
4. **Personal Computer OS**
- Amacı kullanıcıya etkin ve kolay kullanım ara yüz sunmaktadır
5. **Real-Time OS**
- Zaman kısıtlaması çok önemlidir. (Bir programın başlangıç ve bitiş zamanı bellidir.)
6. **Embedded OS(Gömülü işl. Sist.)**
- PDA, TV Sets, microwave ovens, mobile phones
- Ex: PalmOS, Windows CE, Symbian OS
7. **Smart Card OS**
- Kredi kartı boyutlarında üzerinde işlemci olan kartlarda çalışır.
- Java tabanlıdır. JVM içerir.
- Ex: MULTOS, Windows Embedded CE, Smartec OS

HAFTA 2 - INTRODUCTION TO OPERATING SYSTEMS

- Bilgisayar donanımı çok karmaşıtır. OS bu karmaşıklığı kullanıcılardan gizler ve kullanıcıya basit bir arayüz sunar.
- OS un temel görevi donanımı yönetmektir.
- Bazı sistem programları:
  - En temel sistem programı OS'dir.
  - İnterpreter, compilers
  - Database management systems
  - Editors
- Bunlara SHELDS adı verilir.
- Operating systens kernel modda çalışabilir yada diğer adıyla supervisor modda çalışabilir. Ama diğer sistem programları user modda çalışır.

1. Managing Resources

- Her program istediği her kaynağa istediği an ulaşamaz.
- İşlemci aynı anda tüm programları çalıştıramaz.
- Bunun için OS devreye girer ve verileri tampon bellekte tutar, bekletir sırası geldikçe işlemciye alır. Yani bir düzen oluşturur.
- Her program belleğin her alanına yani her adres bölgesine ulaşırsa başka verilere ait olan adresleri de kullanırsa bu çeşitli sorunlara yol açar. OS bir adres ya da kaynak kullanılmadan onun boşta olup olmadığını kontrol eder ve veri kayıplarının oluşmasını engeller.
- Yani adres ya da kaynak boşta ise oraya atama yapar. Kimin hangi kaynağı kullanacağını izler.
- Kaynak yönetimi 2 temel parametreye göre yapılır
  - 1.İşlemci Paylaşımı **time** parametresine göre yapılır.
  - 2.RAM ya da Harddisk paylaşımı **space** parametresine göre yapılır.

BİLGİSAYARIN İŞLETİM SİSTEMİNİ YÖNETTİĞİ DURUMLAR

1. Processor(İşlemci)

- PCdeki en hızlı cihaz. Bilgisayarın beyni.
- Pipelining bilgisayarı hızlandırmak için kullanılır.
  - Mesela **n.** Kod **excute** edilirken, **n+1.** Kod **decode** edilir, **n+2.** Kod **fetch** edilir
- Öncelikle veri işlemci üzerindeki cache belleğe yüklenir ve veri cache bellekten register'lara atanır ve veri register'lardan çağırılır. Bazı register'lar:
  - PC(Program Counter): bir sonraki işlenecek komutun adresini tutar.
  - SP(Stack Pointer): bellekteki yığının en üst adresi tutar.
- **Kernel Mod** ile çalışan donanım üzerinde tüm işlemleri gerçekleştirebilir. OS Kernel Mod ile çalışır.
- **User Mod** ile çalışan program sadece okuma yapıp, çok kısıtlı olarak bazı alanları değiştirebilir. Donanımsal işlemler yapamasa da gerektiğinde OS'tan yardım ister. Bu duruma system call yani sistem çağrısı denir. Bu çağrı sonucunda OS işlemi gerçekleştirir ve sonucu döndürür.

2. Memory

- Bilgi önbellekte aranır varsa registerlara alınarak kullanılır yoksa bellekte aranır.
- Bellek yönetimindeki problem: bilgisayarda birden fazla program çalışıyorsa RAM de birden fazla program var demektir, her birinin farklı adres alanı vardır ve bu alanlar korunmalıdır. Yani birbirlerinden izole olmalıdırlar. OS bunu çözer. Bu koruma olayına bloked etmek denir. Blocked edilmelidir ki farklı program tarafından kullanılmasın.
- **Physical Addresses,** programın RAM e yerleştiğindeki adrestir.
- **Virtual Addresses,** mesela compiler der ki 10 000. Adrese git, program 10 000.adrese gider ama RAM deki adres aynı değildir. **MMU(memory management unit)** sanal adresi fiziksel adrese çevirir.
  - Bu olay şu şekilde gerçekleşir:
    - **Base register** programın taban adresini tutar, **Limit register** ise programın tavan adresini tutar (ulaşabileceği en üst nokta).
    - Base register ile sanal adres toplanarak fiziksel adres bulunur.

NOT: Harddisteki veriyi okuyup yazılabilecek en küçük birim SECTOR dür.

3. I/O DEVICES (RAM ve CPU HARİÇ DİĞER BİRİMLER)

- I/O devices genellikle 2 parçadan meydana gelir. Aygıtın kendisi ve controller.
- OS komutu controller a gönderir, sonuçları da oradan alır.
- Kontrolcü ile iletişimi sağlayan yazılıma **device driver** denir.
- **Driver larda kernel modda çalışabilir.**

I/O OPERATION TYPES

- 3 Farklı şekilde gerçekleşir.
    - **a.** En basit method
        - **i.** Performans açısından en düşük olandır. Mesela aygın memoryden okuma için sistem çağrısı yaptı. Okuma gelene kadar beklerken veri geldi mi diye sürekli kontrol yapar ve zaman kaybeder bu olaya **busy waiting** denir. I/O işlemi bitene kadar işlemci boşta kalır.
    - **b.** Busy waiting olayından kurtulunuyor . Driver işleme başladığında işlemci kontrol devresine bildirim yapar, kontrol devresinden cevap gelene kadar işlemci işlemi bloklar. Veri bekleyen işlem işlemciden çıkarılıp RAM e gönderilir. Beklediği veri gelince interrupt gelir ve önceden beklenen verinin geldiği anlaşılır ve işlem işlemciye girer ve beklemedeki boşa giden zaman başka işte kullanılmış olur.
    - **c.** DMA(direct memory access), CPU kullanılmadan bellek ile kontrolcü arasındaki akışı yönetir. CPU DMAyı bilgilendirir işler DMA halleder, CPU başka işlerle uğraşır. DMA işi hallettiğinde bir interrupt oluşturur ve CPU u bilgilendirir.

4. BUSES

- **PCI Bridge (Kuzey Köprüsü):** işlemci, RAM ve Ekran Kartı arasındaki veri transferini gerçekletirir.
- **ISA Bridge (Güney Köprüsü):** I/O cihazları ile işlemci arasındaki veri transferini gerçekleştirir.

HAFTA 3 - İŞLETİM SİSTEMİNİN TEMEL GÖREVLERİ

1.Process Yönetimi:

-programın, belleğe yüklenerek çalıştırılmasıyla ortaya çıkan yeni yapıya Process denir.

-Bir processin işi yapabilmesi için, belleğe yüklenmesi gerekir. Her bir process için bellekte bir adres alanı ayrılır.

-Bu adres alanında programın kodları, programın kullanılacağı veriler ve program yığını bulunur.

-Periyodik olarak işletim sistemi, çalışan bir processi durdurup askıya alabilir ve yerine başka bir processi başlatabilir.

-Birçok işletim sistemi, process ile ilgili gerekli verileri process tablosu ya da process kontrol bloğu denen, bir dizi yapısında tutar. Bu tabloda her bir kayıt, bilgisayarda başlatılmış durumda olan bir process için ayrılmıştır.

2.Bellek Yönetimi

- Çalışmakta olan bütün programların verileri ana bellekte tutulur.

-Çalışan processlerin verilerinin bir kısmı ana bellekte, bir kısmı ise disk üzerinde Sanal Bellekte tutulur.

3.Dosya Yönetimi

-Birçok işletim sistemi dosyaları gruplandırmak için dizin(directory) yapısını kullanır.

-Dosyalar hiyerarşik bir yapıda bilgisayarda saklanır.

-Her dosya ana dizinden itibaren, nerde olduğunu gösteren bir adres yoluna sahiptir.

4.Güvenlik

-UNIX'te koruma işlevi 9 bitlik binary bir kod ile gerçekleşir. Bu bitler 3e bölünmüştür. İlk 3 bit admin yetkilendirmesi, ikinci 3 bit kullanıcı grupları, son 3 bit ise sıradan kullanıcılar için ayrılmıştır.

5.Shell

-Kullanıcı modundaki işlemlerden gelen sistem çağrılarına işletim sistemi cevap verir.

-UNIX'in Shell isimli komut yorumlayıcısı, işletim sisteminin bir parçası değildir ama sistem çağrılarının nasıl çalıştığını anlamak için iyi bir örnektir. Aynı zamanda GUI kullanılmadığı zaman, kullanıcı ile işletim sisteminin arasında arayüz görevi görür.

6.Sistem Çağrıları

-Sistem çarpıları makineye bağlıdır. Dolayısı ile Assembly de kodlanması gerekir.

-İşletim sistemleri, C ve diğer dillerden sistem çağrılarını aktif etmek için bir prosedür kütüphanesi sunarlar.

-Eğer kullanıcı modunda çalışan bir kullanıcı, bir servise ihtiyaç duyarsa, bir kütüphane prosedürü ile sistem çağrısı komutu çalıştırılarak kontrolü işletim sistemine vermelidir.

Bir sistem çağrısı 3 tane parametreye sahiptir:

-Birincisi dosyayı bildirir.

– ikincisi Buffer, belleğe dosyanın yerini bildirir.

-Üçüncüsü ise verinin kaç byte uzunluğa sahip olduğunu bildirir.

NOT: İşletim sistemi, sistem çağrısını yerine getirir ve kontrolü sistem çağrısını çağıran programa geri verir.

NOT: Sistem çağrısı yapmak, işletim sistemine sapmak anlamına gelir. İşletim sistemi gerek duyduğunda SVC(supervisor call) ve INT(interrupt) gibi özel makine komutları çalıştırarak ilgili işletim sistemi kesimine sapılır.

THE WINDOWS WIN32 API:

-API, bir kütüphane ya da ortamın(framework) sunduğu yararlı işlemler yapan fonksiyonlara verilen genel addır.

-Sistem çağrıları KERNEL modda çalışan fonksiyonlardır. API ise daha geneldir. Her sistem çağrısı bir API'dir. Fakat her API bir sistem çağrısı değildir.

-Windows'un sistem çağrıları API'ler aracılığıyla dolaylı olarak çağrılmaktadır. Sistem çağrıları, İşletim sistemi tarafından servisler için sağlanan arayüzdür.

-Programcılar, doğrudan sistem çağrılarını kullanmak yerine, API kullanmayı tercih eder.

NEDEN API?

-API'ler programın her sistemde çalışabilmesini sağlar.

-Birden fazla sistem işlemini kısa fonksiyonlarla yapabilmeyi sağlar.

-UNIX sistemlerde genellikle sistem çağrısı ile çağrılacak olan kütüphane fonksiyonunun ismi aynıdır.

-POSIX fonksiyonları UNIX türevi sistemlerde bulunan fonksiyonlardır. "Open" fonksiyonu UNIX türevi sistemlerde kullanılır.

-WINDOWS'ta durum bu şekilde değildir. Microsoft win32 API adını verdiği bir prosedür kümesi tanımlamıştır.

-İşletim sistemleri değişse bile, bir kullanıcı, programı API'ler aracılığıyla bütün Windows İşletim sistemlerinde sorunsuzca çalıştırabilmektedir.

-Win32 API çağrılarının sayısı, binlerle ifade edilecek kadar çoktur.

HAFTA 4 - PROCESSLER

-Processler çalışmakta olan programlara verilen genel addır.

- Multi-Programming bir sistemde CPU, çalışmakta olan programlar arasında anahtarlama yapar ve işlemci her bir process için saniyenin yüzde biri ya da binde biri adar süre ayırır.

Process Model

-Çalışmakta olan bütün processler ardışık şekilde bellekte beklerler.

-CPU, processler arasında anahtarlanır. Bu hızla anahtarlama işlemine Multi-Programming denir. -

Gerçekte fiziksel olarak sadece bir tane program sayacı vardır. O da işlemcinin içerinde bulunur.

-Process işlemciden çıkarken, fiziksel program sayaç değeri, bellek üzerindeki o processe ait mantıksal program, sayaç alanına yazılır.

NOT: İşlemci, bir processten daha öncelikli bir processe anahtarlanabilir.

NOT: Process, bir programa, input/output verilerine ve ayrı bir duruma sahip olan bir aktivitedir.

NOT: Sistemde arkada çalışan processlere **deamon** denir.

Process Oluşturma

1. İşletim sistemi başlatıldığında

2. Çalışmakta olan process, bir sistem çağrısı yaparak yeni processleri oluşturup, kendisine yardımcı processler oluşturabilir.

3. Etkileşimli sistemlerde kullanıcılar bir komut vererek ya da bir ikona tıklayarak yenir bir programı çalıştırabilir. Dolayısı ile yeni bir processi oluşturmuş olur.

4. Özellikle mainframe üzerinde icra edilen Batch işlemlerdir. İşletim sistemi ardışık olan bu processlerden biri bittiğinde diğerinin oluşturulup çalışmasını sağlar.

-Teknik olarak, yeni bir process oluşturmak için sistem çağrısı yapılır. Oluşturulan bu process çağrıda bulunan programla ilişkilendirilir.

-Yeni bir process oluşturmak için UNIX'te "**fork**", WINDOWS'ta ise Win32 API "**Create Process**" kullanılır.

-İki işletim sisteminde de process oluşturulduktan sonra, ana process ile çocuk process için bellek üzerinde birbirinden farklı olanlar ayrılır. İki process de birbirinin yaptığı değişiklikleri göremez.

-Fakat UNIX'te çocuk processin başlangıç adres alanı, ana processin adres alanının bir kopyasıdır.

Sadece çocuk process, ana processin başlangıçtan gelen bazı verilerini okuyabilir.

-WINDOWS için ise ortak paylaşılan bir alan yoktur.

Process Sonlandırma

1. Normal çıkış

2.Program kodlarındaki bir hatada bir hatadan dolayı process sonlandırılır. Örneğin sıfıra bölünme hatası, overflow hatası gibi.

3. Kullanıcının olmayan dosyayı açmak için komut vermesi durumunda, derleyici hemen processin sonlandırılması için çağrı yapar.

4.Bir process işletim sistemine diğer bir processi sonlandırmak için çağrıda bulunur. UNIX'te "**kill**", WINDOWS'ta ise "**Terminate Process**" ile sağlanır.

Process Hiyerarşisi

- UNIX'te, kullanıcı klavyeden bir sinyal gönderdiğinde, bu sinyal gruba bağlı olan bütün processlere iletilir. Gruptaki ilgili process ya da processler bu sinyalleri dikkate alırken, diğer processler ise bu sinyali yok sayar.

-WINDOWS'ta ise hiyerarşi yoktur. Bütün processler eşit durumdadır.

Process Durumları

- Eğer bir process işlemcide çalışırken, dış kaynaklı bir giriş verisine ihtiyaç duyarsa, ihtiyaç duyduğu veri gelene kadar bloklanır.

1. **RUNNING:** Process şu anda işlemci içinde çalıştırılıyor.
2. **READY**: Process çalışabilecek durumda, fakat diğer processlerin de işlemciden faydalanması için geçici olarak durduruldu.
3. **BLOCKED**: Process dış kaynaklı bir olay gerçekleşinceye kadar çalışamaz.

Processlerin İcrası

-İşletim sistemleri, process tablosu adı verilen bir tabloyu kullanır. Bu tablo bir dizi yapısındadır ve bu tabloda her bir process için ayrı bir kayıt tutulur.

- Her bir kayıt, ilgili processin durumu, program sayacı, yığın göstergeci, bellek alanı açık dosyaları, zamanlama parametreleri gibi verileri tutar. Böylece bu process işlemciye bir daha girdiğinde, sanki hiç çalışmamışçasına ara vermemiş gibi kaldığı yerden devam edebilir.

HAFTA 5 – THREADS

-Her bir process ana bellek üzerinde kendine ait bir adres alanına sahiptir. Bu adres alanında program metni, veriler ve processin çalışması için gerekli olan kaynak bilgileri yer alır.

-Process, çalışması için gerekli olan bütün kaynakları kendi adres bölgesinde gruplandırarak, daha hızlı çalışan ve daha kolay yönetilebilen bir forma sokulmuş olur.

-Processin icra edilen bu kısmına **Thread** adı verilir.

-Thread, bir sonraki komutun adresini tutan bir program sayacına, yığın göstericisine ve register değişkenine sahiptir.

- Processler kaynakları gruplandırmak için kullanılırken, threadler işlemci içerisine giren ve processin işlem kısmını icra eden kod parçalarıdır.

-Threadler, bir process içinde birden fazla işlem biriminin çalışmasına olanak sağlayan yapılardır.

-Threadlerin multi-threading(paralel) çalışması, bir işletim sisteminde birden fazla processin aynı anda çalışmasına benzerdir.

-Bir process içerisindeki tüm threadler aynı adres uzayını, açık dosyaları, genel değişkenler gibi kaynakları paylaşırlar.

-Bir processe ait bütün threadler aynı adres alanını kullanırlar. Bir thread, ait olduğu processin adres alanını paylaştığı gibi, açık olan dosyalarını, çocuk processlerini, verilerini de ortak kullanabilirler.

-Multi-threading, multi-programming'e benzer bir çalışma yapısına sahiptir. CPU threadler arasında hızlı bir şekilde anahtarlama yaparak, sanki aynı anda birden fazla thread'i çalıştırıyormuş izlenimi verir.

-Threadlerin içinde bulunabileceği durumlar: Running, blocked, ready, terminated.

-Her bir thread kendine ait bir yığın alanına sahiptir. Her bir kendi bünyesindeki prosedürleri çağırabilir. Dolayısı ile her bir threadin kendi yığınına sahip olması gerekir.

-Yeni bir thread "thread_create" komutu ile oluşturulur."thread_exit" komutu ile sonlanır.

-"thread_yield" komutu ise, thread'in gönüllü olarak, işlemci zamanını kardeş thread'e vermesini sağlar.

Thread Kullanımı

-Thread kullanımındaki temel avantaj, bir processe ait birçok işlevin eş zamanlı olarak yapılabilmesidir. Processler bloklandığı zaman, processin threadlere ayrılması ile, diğer thread devreye girip, işleme devam edilir. Böylece tam olmasa da yarı-paralel bir çalışma özelliği oluşur.

-Threadlerin performans artışı sağladığı sistemler, çok işlemcili sistemlerdir. Çünkü threadler bu sistemlerde, gerçek anlamda paralel çalışırlar.

Thread Kullanım Nedenleri

1. Bazı paralel çalışan işlevler, ortak bir adres bölgesi kullanmak isterler. Bu ortak kullanım ancak threadler ile mümkün olur.

2. Threadlerin kendine ait verileri çok fazla olmadığı için, processlere oranla daha kolay oluşturulup yok edilebilirler. Birçok sistemde thread oluşturma işlemi, process oluşturmaktan 100 kat daha hızlı gerçekleştirilir.

3. Threadlerin hepsi CPU bağımlı ise, performans kazanmak çok mümkün değildir. Fakat threadlerin bir kısmı CPU bağımlı, Bir kısmı I/O bağımlı ise, bu threadlerin eş zamanlı çalışması, dolayısı ile performans kazancı mümkün olur.

Kullanıcı Uzayında Çalışan Threadler

- Bu threadleri işletim sistemi tanımaz. Kernel, bir process içindeki threadlerden habersizdir ve bu processe tek bir thread'den oluşuyormuş gibi davranır.

  ** bu yöntemin en önemli avantajı, işletim sisteminin kendisi, threadleri desteklemese dahi gerçekleştirilebilir olmasıdır.

- Threadler bir Run-Time sisteme bağlı olarak çalışırlar. Bu Run-time sistemler, birçok fonksiyona sahip kütüphanelerdir. Bunlardan bazıları: thread**_create, thread_exit, thread_wait,thread_yield**.

  ** Her bir process kendi thread tablosunu tutar. Her bir threade ait program sayacı, yığın gösterici, durum ve yazmaç değerleri tutulur.

- Bir thread kendini bloklayacak bir işlem gerçekleştirdiğinde Run-Time sistem prosedürlerini çağırır. Eski threade ait yazmaç değerlerini thread tablosuna yazar. Yeni threade ait değerleri thread tablosundan, işlemcideki yazmaçlara yükler.

Kernelda Çalışan Threadler

-Bu tip sistemlerde kernel, threadleri tanır ve yönetir. Run-Time sistemlere ihtiyaç duymaz. Thread tablosu bizzat işletim sistemi tarafından tutulur. Bir thread oluşturmak ya da yok etmek istediğinde, sistem çağrısı yaparak kerneldaki thread tablosu üzerinde işlem yapar.

-Bir thread bloklandığında kernel, thread tablosundan başka bir thread anahtarlar.

** Bu sistemin en büyük dezavantajı, thread işlemleri sistem çağrıları ile yapılır ve bu da sisteme yük bindirir.

Hybrid Thread Uygulamaları

- Run-Time sistemler ile kernelda çalışan threadlerin avantajları birleştirilerek hybrid bir kullanım sağlanmıştır.

-Bu tasarımda kernel, sadece kernel threadlerini tanır ve sadece onları anahtarlar.

-Kullanıcı uzayındaki threadlerle yapılan işlemler, yine işletim sisteminde habersiz bir şekilde Run-Time sistem üzerinde gerçekleştirilir.

HAFTA 6 – PROCESSLER ARASI İLETİŞİM

Processler arası iletişimde üç konu önemlidir.

1. Bir process diğerine nasıl veri gönderir?
2. Processlerin ortak alanları kullanılırken dikkatli olmaları
3. İletişimdeki uygun sıra nasıl olmalıdır?

İletişim neden yapılır?

1. Kaynak paylaşımı
2. Karşılıklı haberleşme
3. Senkronizasyon

Race Condition(Yarış Durumu)

- Bir process bir dosyayı yazdırmak istediğinde, spooler adı verilen bir dosyaya, yazdırmak istediği bilgisini ekler.

- Ortak dosyaların kullanımı birden fazla processe açılırsa, Race Condition oluşur.

Kritik Bölgeler

-bir programın ortak alanlara ulaşmaya çalıştığı kod kısımlarına **kritik bölge** denir.

-Race Conditiondan sakınmak için kullanılan yöntemlere **Mutual Exclusion** yöntemleri denir.

- İki süreç aynı anda kritik bölgeye girmemelidir.

- Sistemdeki işlemci sayısı ve hızı ile ilgili kabuller yapılmamalıdır. Bu değerlerden bağımsız olmalıdır.

-Kritik bölgede çalışmayan bir process, diğer processleri bloklamamalıdır.

- Bir süreç, kritik bölge içinde sonsuza kadar beklememelidir.

NOT: Bir processin kritik bölgeye girmesi demek, işlemciye girdiği anlamına gelmiyor.

Yoğun Beklemeli Mutual Exclusion Yöntemleri

1. Disable Interrupts

- Süreçler kritik bölgeye girdiklerinde tüm kesmeleri devre dışı bırakabilirler.

- Sistemde birden fazla işlemci varsa, bu işlemci sadece tek bir işlemciyi etkiler.

- Bu görevi işletim sistemi yapar.

2. Lock Değişkeni

- Bu yöntem yazılımsal bir çözümdür. Başlangıç değeri 0 olan, ortak kullanılan bir lock değişkeni olsun. Bir process, kritik bölgesine girmek istediğinde lock değişkenini kontrol eder.

-Lock değişkeni 0 olduğunda hiçbir process kendi kritik bölgesinde değildir. 1 olduğunda ise kritik bölgesinde olan bir process vardır.

-Bu yöntemde tam manası ile çözüm sağlamaz. Çünkü lock değerinin kendi koruması olmadığı için, değeri anahtarlama sırasında değişir. Bu da sistemde soruna yol açar.

TSL Komutu

- Çok işlemcili sistemler şöyle bir komuta sahiptir: TSL RX, LOCK

- TSL komutu kullanıldığında bellekteki lock değişkenine giden veri yollarını kilitleyerek, diğer işlemcilerin lock değişkenine ulaşımı engellenir.

- Lock 0 olduğunda herhangi bir process TSL komutunu kullanarak, lock'ı 1 yapar ve ortak bellek bölgesinden veri okur ya da bu alana veri yazar. İşlem bittiğinde de lock değeri tekrar 0 yapılır.

Sleep And Wake Up Çağrıları

- Processleri sonsuz döngülerden kurtaran çağrılardır. Bir process, diğer bir processi beklerken kritik bölgeye girip işlemciyi terk edemez. Sonsuza kadar bu durumun devam etmesini engellemek amacıyla bu çağrılar kullanılır.

-Sleep, çağıran processi bloklayan bir sistem çağrısıdır. Bu processi diğer bir process onu uyandırıncaya kadar bloklanmış bir şekilde bekletilir. Wake up çağrısına, uyandıracağı process parametre olarak gönderilir.

Producer – Consumer Problemi

- İki process belirli bir buffer alanını ortak kullanıyor olsun. Üretici process(Producer) bu buffera veri koyarken, tüketici process(Consumer) bu bufferdaki veriyi alıp kullanır.

- Buffer tamamen dolu olduğunda, producer veri yazamaz. Bu durumda sleep durumuna geçmesi gerekir. Uyuyan bu process, consumer tarafından bufferdan bir ya da daha fazla kayıt çektiğinde tekrar uyandırılır.

- Aynı şekilde buffer tamamen boş olduğunda consumer, bufferdan bir şey alamaz ve sleep moduna geçer. Producer buffera kayıt eklediğinde ise consumer uyandırılır.

- Buradaki sıkıntı ise, buffer sayacı olan count değişkeni için kısıtlama olmadığından, Race Condition olması muhtemeldir. Bunun önüne geçmek için wake up bekleme biti kullanılır.

-Uyanık durumdaki bir processe wake up çağrısı gönderildiğinde, bu bit off durumundan on durumuna geçer. Bu process uyku moduna geçemeye çalıştığında ise wake up bitine bakılır. Eğer bu bit on durumunda ise uyku moduna geçmez, uyanık kalır ve bu biti off yapar.

Semafor(Semaphore)

- Semaforlar uyanık durumdaki processlerin sayısını tutan bir integer değişkendir.

- Bu değer 0 ise, uyanık durumda olan process yoktur. Bu değer pozitif bir sayı ise, bir ya da daha fazla uyanık durumda process vardır.

- Semaforların iki metodu vardır : **Down(wait)** ve **Up(signal). Down->sleep** çağrısına karşılık gelirken, **Up->wake up** çağrısına karşılık gelir.

- Down metodu, semafor değerinin 0'dan büyük olup olmadığını kontrol eder. Eğer büyükse, semafor değerini 1 azaltıp işleme devam eder. Eğer semafor değeri 0 ise, process semafor değerini azaltmadan uyku moduna geçer.

-Up metodu ise, semafor değerini 1 arttırır. Eğer bir ya da daha fazla process semafor üzerinde uyku modunda ise, bekleyen bu proseslerden biri rastgele seçilir.

Producer-Consumer Probleminin Semaforlar İle Çözümü

- Semafor işlemleri çağrıldığında, işletim sistemi bütün kesmeleri iptal eder. Böylece semaforun değerinin kontrol edilmesi, değiştirilmesi ve processin uyku moduna geçirilmesi, kesintisiz ve tek bir işlem olarak gerçekleştirilmiş olur.

MUTEXLER

- Mutexler belirli bir ortak alanın mutual exclusion olayını sağlamak için kullanılır ve uygulaması kolay bir yöntemdir.

-Mutex değerinin 0 olması, mutexin açık olduğu, 0 haricindeki bütün değerler ise mutexin kapalı olduğu anlamına gelir.

-Bir thread ya da process kritik bölgesine girmek istediğinde, **mutex_lock** metodunu çağırır. Eğer mutex açık ise, bu çağrıyı yapan thread kritik bölgeye girebilir.

- Mutex kapalı durumda ise, mutex_lock metodunu çağıran thread bloklanır, kritik bölgede çalışan thread ise mutex_lock komutunu çağırarak kritik bölgeden çıkar. Kritik bölgeye girmeye çalışan birden fazla thread var ise, bunlardan bir tanesi rastgele seçilir.

BARİYERLER

- Bütün processler ancak birlikte bir sonraki aşamaya geçebilir. Bu şekilde bir çalışma için bariyer kullanımına ihtiyaç duyulur.

- Her bir aşamanın sonuna bariyer konulur. Process grubundaki processlerden biri, bariyere ulaşınca bloklanarak diğer processleri beklemeye başlar.

HAFTA 7 – ZAMANLAMA (SCHEDULING)

Zamanlama (scheduling) Nedir? *

- Kernel modda çalışan zamanlayıcı(scheduler), bilgisayardaki en çok çalışan proseslerden birisdir.
- İşletim sisteminde birden fazla hazır durumda bulunan prosesten hangisinin işlemciyi kullanacağına scheduler karar verir.
- Temel görevi proses switching dir.
- Scheduling işlemi çok sık yapılmamalıdır çünkü bu işlem performans açısından pahalıdır.
- Proses anahtarlaması yapıldığında user moddan kernel moda geçiş olur ve yeni prosesin bellek haritası MMU tarafından yüklenir.

Proses Davranışları(Proses Behaviour)

1. İşlemcide çok fazla kalan işlemlere **compute-bound proses** denir.
2. Daha çok veriye ihtiyaç duyduklarından çok fazla işlemci dışında bekleyen, işlemcide işi olunca kısa sürede çıkıp giren işlemlere **I/O-bound proses** denir. İşlemcide çok kısa süre kaldıklarından öncelik durumları vardır.

Ne Zaman Scheduling Yapılmalıdır? *

- Yeni bir proses oluşturulduğunda.
- Bir proses sonlandığında.
- Bir proses bloklandığında.
- Bir I/O kesmesi geldiğinde

Bilgisayarlarda bir donanım saati(clock) bulunur. Bu clock periyodik aralıklarla işlemciye kesmeler gönderir. Scheduler işlemi her clock kesmesinde ve ya belli sayıda clock kesmesinde gerçekleştirir. Scheduling algoritmaları 2 ye ayrılır.

1. **None-preemptive(Kesintisiz)**: Proses, bloklanıncaya kadar ya da kendi isteğiyle işlemciden çıkıncaya kadar işlemciden çıkmaz. Yani clock kesmelerine karşı tepkisizdir. Daha hızlıdır.
2. **Preemptive(Kesintili)**: Proses belli süre çalıştırılır. Clock kesmeleri dikkate alınır.

Scheduling Algoritma Kategorileri *alt başlıkları bil

1. Batch Sistemlerde Zamanlama

- Kullanıcının acil cevap beklemediği yani kullanıcı ile direk iletişimde olmayan bir sistemdir. Prosesler seri halinde işletilirler. Kesintili ve kesintisiz olarak çalışabilirler. Ama genellikle kesintisiz çalışma tercih edilir.

a. First Come First Served

- Kesintisiz, en kolay zamanlama algoritmasındır. İlk gelen proses ilk hizmeti alır. Prosesler istek sırasına göre çalışırlar.

CPU ← **ABCD** →(A işini bitirdi)**BCD**→(B beklerken bloklanmış)**CD**→(B uyandı)**DB**

b. Shortest Job First

- Kesintili ve kesintisiz olarak çalışabilir fakat kesintisiz çalışmayı tercih eder. Çünkü proseslerin çalışma sürelerinin belirli olduğunu düşünerek işlem yapar. Yani en kısa süreli işi ilk yapmayı hedefler.

A	B	C	D
8	4	4	4

A: 8
B: 12
C: 16
D: 20

TOP/4=14

Turnaroundtime=14

B	C	D	A
4	4	4	8

B: 4
C: 8
D: 12
A: 20

TOP/4=11

Turnaroundtime=11

Turn around time kullanıcıyı bekletme süresidir. Yukarıya bakarsak 11 olan daha avantajlıdır.

C. Shortest Remaining Time Next

- Kesintili çalışır. Yeni bir iş geldiğinde mevcut proses ile yeni prosesin kalan sürelerini karşılaştırır ve kısa olanı tercih eder.

2. Etkileşimli (Interactive) Sistemlerde Zamanlama

- Kullanıcı ile sürekli etkileşim halinde olan bir sistemdir. Bu yüzden kesintili zamanlama kullanımı zorunludur.

a. Round-Robin Scheduling (Dönüşümlü)

- Etkileşimli sistemlerde en geniş kullanım kitlesine sahip algoritmadır. Her prosese eşit süre ayrılır bu süreye **quantum** süresi denir. Her quantum süresi sonunda proses switching gerçekleşir.
- Proses quantum süresi bitmeden bloklanır ya da sonlanırsa sürenin bitmesi beklenmez hemen proses switching gerçekleştirilir.
- Proses quantum süresi dolduğunda proses hala çalışıyorsa tekrar sıraya alınır ve yeni prosese yer verilir.

Örnek:

Quantum süresi : 4ms
Proses switching : 1ms
} 5ms → %20si proses switching işleminden dolayı kayıp

Quantum süresi : 99ms
Proses switching : 1ms
} 100ms → %1i proses switching işleminden dolayı kayıp

- Örnekte de görüldüğü gibi quantum süresi arttıkça kayıp azalmaktadır. Fakat quantum süresi arttıkça kullanıcının bekleme süreside artacaktır. Bu yüzden ideal quantum süresi 20-50 ms arasındadır.

b. Priority(Öncelik) Zamanlama

- Her prosese bir priority değeri atanır ve yüksek öncelikli prosesin çalışmasına öncelik verilir.
- Yüksek öncelikli proses için quantum süresi sonlandığında priority değeri ile karşılaştırılır, sonsuza kadar aynı prosesin çalışmasını önlemek için. Yani prosesin değerleri dinamik olarak verilmelidir.

Not: I/O bound prosesler zaten veri girişi çıkışı için çok fazla bekletildikleri için birde CPU da bekletilmezler. CPU da çalışmak istediklerinde bu yüzden öncelikleri vardır. Zaten CPU da çok kısa süre çalışırlar.

Örnek:

P=q/f → priority değeri

q : quantum süresi

f : prosesin son kullandığı quantum süresi

p=q/f = 50/1=50 (en yüksek öncelikli proses)
p=q/f = 50/2=25
p=q/f = 50/25=2
p=q/f = 50/50=1 (en düşük değerlikli proses) →verilen sürenin tamamı kullanılmış.

NOT: Prosesler öncelik durumlarına göre gruplandırılırlar. Öncelik durumu eşit olan birden fazla proses varsa Round-Robin uygulanır.

c. Multiple Queues

- Öncelik durumu azaldıkça ve ya giriş seferi arttıkça verilen quantum süresi artar.

Örnek: Bir proses düşünün işlemciye 100 kez girmesi gerekiyor.

$2^{n-1} \leq 100$ (n: giriş seferi)

Giriş Seferi	→	1	2	3	4	5	6	7
Verilen qua. Süresi	→	1	2	4	8	16	32	64

↓

Buraya kadar 63 quantum süresi harcandı. 37 quantum süresine
Daha ihtiyaç var. 7 girişinde verilen 64 ün 37'si kullanılır 23 kullanılmaz

d. Shortest Proses Next

- Tahmin(estimated) edilen süreden en kısa süreye sahip olan proses seçilip çalıştırılır.

****Örnek:** a= ½ için;

T2 = a.T0 + (1-a).T1 T3 = a.T1 + (1-a)T2

T2= ½.T0+ ½.T1 T4= a.T2 + (1-a)T3

- T0 aralığı gittikçe düşüyor. Yani prosesin çalışma sayısı arttıkça önceliği azalır.

e. Quaranteed(Garantili) Zamanlama

- Her prosese eşit süre vererek adil olmayı garantiler. N kullanıcı varsa her proses işlemcinin 1/n 'lik kısmını kullanır. Bu sistemde her prosesin çalışma süreleri proses tablosundan ayrıntılı olarak izlenir.

$$r = \frac{\text{prosesin kullandığı zaman}}{\text{prosesin hak ettiği zaman}}$$

r1=5/10= 0.5 →verilenden az kullanılmış artık öncelikli yani kendisine en yakın olan orana kadar çalıştırılır.

R2=15/10=1.5 →asılnda amaç oranı 1 yapabilmektir

R3=20/10=2.0

f. Lottery(Piyango) Zamanlama

- Her bir prosese öncelik durumlarına göre bir bilet ve ya biletler verilir. Bir zamanlama kararı verilince bir bilet seçilir ve bu bilete sahip olan proses çalıştırılır. Örneğin proses 100 biletten 20 sine sahipse %20 çalışma şansı vardır. Prosesler arasında bilet paylaşımı olabilir.

g. Fair-Share(Adil Paylaşım) Zamanlama

- Eğer sistemde 2 tane kullanıcı varsa her biri işlemcinin %50'sini kullanır. Yani işlemci paylaştırılır.

Örnek:

User1→A,B,C,D
User2→E

İşlemci: AEBCEDEAEBECEDE… %50-%50

Örnek:

User1→A,B,C,D
User2→E

İşlemci: AEBEDEAEBEDE… %50-%50

User1 kullanıcısı C prosesini hiç çalıştırmamış bunun kararı Round-Robin çalışma sistemi ile verilir.

Örnek:

User1→ A,B,C,D
User2→E

İşlemci: ABECDEABECDE…
%66,6… →User1'e
%33,3… -->User2'ye hizmet edilmiştir.

HAFTA 8 – ÖLÜMCÜL KİLİTLENMELER (DEADLOCKS)

- Bir kaynak bir anda sadece tek bir proses tarafından kullanılabilir.

Örnek : A ve B prosesleri tarayıcıdan aldıkları veriyi CD'ye yazdırmak istesin. OS, tarayıcı kullanım hakkını Aya, CD Writer kullanımını B'ye vermiş olsun. A prosesi CD Writer kullanmak istediğinde ulaşımı sağlayamaz çünkü hak B'nin, B'de tarayıcıya ulaşımı sağlayamaz ve iki proseste sonsuza kadar bloklanır. Bu duruma **deadlock** denir.

- Bir bilgisayarda kaynaklar 2 şekilde kategorileştirilir.

1. Preemptable Resources (yarıda kesilebilir kaynaklar):

- Bir proses tarafından kullanılırken, kaynağın elinden alınıp başka prosese tahsis edilmesi durumunda olumsuzluk, hata çıkarmayan kaynak tipleridir.

Örnek: Bellek.

3. Non- Preemptable (yarıda kesilemeyen kaynaklar)

- Bir proses bir kaynağı kullanırken başka proses bu kaynağı elinden alamaz. Bu kaynaklarda deadlock oluşur.

Örnek: CD-Writer

- Bir kaynağın kullanımı için aşağıdaki adımlar gerçekleşir.

a. **Kaynağı iste.** → Bir proses kaynak istediğinde kaynak müsait değil ise bekletilir. OS prosesi bloklar ve kaynak müsait olunca uyandırır. Bazı prosesler hata kodu üretir ve proses tekrar tekrar kaynak müsait mi diye kontrol eder.
b. **Kaynağı kullan.**
c. **Kaynağı serbest bırak.**

Kaynakların Kullanımı

- Sistemde kaynakların düzgün bir şekilde kullanılması için her kaynağa bir semafor atanır. Proses kaynağı kullanmadan önce semaforu ait **down()** metodunu çalıştırarak kaynağı elde eder, işi bitince **up()** metodunu çalıştırarak kaynağı serbest bırakır.

NOT: Deadlock oluşması için en az 2 kaynak ve 2 proses olmalıdır ve kaynakların non-preemptable olması gereklidir.

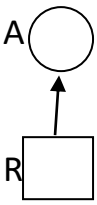
Deadlock Oluşumuna Neden Olan Durumlar

- Deadlock oluşması için aşağıdaki durumların hepsinin sağlanması gereklidir aksi halde deadlock oluşmaz.

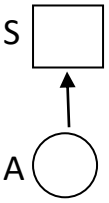
1. Mutual Exclusion (Karşılıklı Dışlama) : 1 anda 1 kaynağı 1 proses kullanabilir.
2. Hold and Wait Condition : Proses yeni bir kaynak ister alana kadar elindeki kaynağı bırakmadan bekler.
3. NO-Preemptable Condition: Bir prosese verilen kaynak elinden alınamaz.
4. Circular Wait Condition: İki yada daha fazla proses zincir şeklinde bir sonraki sürece ait kaynağı bekler

Deadlock Modelleme

- Deadlock'lar graf yapısı ile modellenebilir.



- A prosesi, R kaynağına sahip yani a prosesi R kaynağını kullanıyor.



- S kaynağı başkası tarafından kullanılıyor ve A prosesi S kaynağı tarafından Bloklanmış halde bekliyor.

- Cycle oluştuğunda deadlock durumu ortaya çıkar.

- Deadlock durumları ile başa çıkmak için 4 farklı yaklaşım vardır;

1. **Deadlock durumlarını ihmal etme(ignore) (Ostrivh Algoritması)**
2. **Deadlock Detection(Tespit) and Recovery(geri yükleme):**

Deadlock oluşu engellenmiyor. Oluştuğu zaman geri dönüş için bazı işlemler yapılır.

Deadlock Tespiti

a. Her kaynak tipinden sistemde 1 adet olduğu durumda deadlock tespiti için graf yapısı kullanılır. Graf'ta bir ya da daha çok cycle varsa deadlock oluşmuştur.

- Dışa çıkış yoksa ölü nokta. Graf sonlanır.

b. Her kaynak tipinden birden fazla olduğunda deadlock tespiti

- Bu tip durumlar için matris temelli bir çözüm üretilmiştir. Şöyle ki; n tane proses, m tane farklı kaynak olsun.

E: aynı kaynaktan kaç tane
A: kullanılabilir kaynak sayısı
C: kullanılmakta olan kaynak sayısı (Cij) i:proses j: kaynak
R: istenilen kaynakları gösteren matris.

Deadlock Onarılması

a. Keserek Onarma (Recovery Through Preemption)

- Kaynağa sahip olan proses bloklanır ve 2.prosese tahsis edilir. 2.proses işlemini bitirince ilk proses uyandırılır ve kaldığı yerden devam ettirilir. Çok zor bir yoldur, ekstra performans gerektirir.

b. Geri Yükleyerek Onarma (Recovery Through Rollback)

- Periyodik olarak kontrol noktaları (checkpoint) oluşturulur. Deadlock durumunda check point e dönüş yapılır. O arada yapılanlar kaybolur.

c. Sonladırarak Onarma

- Deadlock oluşmasına sebep olan prosesler sonlandırılır. Sonlandırılan proses cycle oluşmasına sebep olan proses ise deadlock oluşumu önlenir. Cycle oluşturan proses bulunana kadar proses sonlandırma işlemi devam eder.

4. Dikkatli Kaynak Dağıtımı ile Deadlock Oluşumunu Önleme

- Prosesin ihtiyaç duyduğu tüm kaynakların en başta bilindiğini varsayalım. (Gerçekte mümkün değildir. Bu yüzden uygulanması zor.) OS istenilen kaynağın verilmesi durumun da - deadlock oluşur mu? oluşmaz mı? – güven kontrolü yapar ve ona göre kaynak tahsisini yapar.

Güvenli (Safe) ve Güvensiz (Unsafe) Durumları

- Sistemde deadlock yoksa ya da sistemdeki tüm prosesler ihtiyaç duyduğu kaynakları tek seferde sıralı şekilde çalışıp işlemlerini bitirebiliyorsa bu durum **güvenli durum**dur.

A	3	9 → 9 - 3 = 6	İhtiyaç duyulan kaynak sayısı
B	2	4 → 4 - 2 = 2	
C	2	7 → 7 - 2 = 5	

Sistemdeki 10 kaynaktan 7si kullanılıyor

Tüm işlemler için gereken kaynak sayısı.

NOT: Tüm proseslerin işlemlerini bitirmesinin garanti edildiği durumdur.

HAFTA 9 – BELLEK YÖNETİMİ

- Bellek yöneticisi (Memory manager), scheduler kadar çok çalışan bir prosestir.
- Görevleri;
  - Belleğin hangi parçaları kullanımda hangileri değil, izler.
  - Proseslere bellek tahsisi yapar ve geri alır.(allocate, deallocate)
  - Belleğin yetersiz olduğu zamanlarda disk ve bellek arasında takas yapar.
- Bellek yönetim sistemleri 2 temel sınıfta incelenebilir.
  - Bellek ve disk arasında sürekli yer değiştirenler(swapping, paging)[(2),(3)]
  - Bellek ve disk arasında yer değiştirme işlemi yapmayanlar.[(1.2),(1.2),(1.3)]

1.1. Sayfalama ve Takaslama Yapmayan Monoprogramming

- ***** Monoprogramming, bilgisayarda sadece tek bir prosesin çalışmasına izin veren ve belleğin OS ile çalışan proses arasında paylaşıldığı kullanım şeklidir.*****

Not: Monoprogramming sadece gömülü(embedded) sistemlerde bulunur.

1.2. Sabit Bölümlü (Fixed Partitions) Multiprogramming

- Multiprogramming ile ilk 3.nesil OS/360 bilgisayarında kullanılmıştır. Bellek n parçaya ayrılır ve parça boyutları sabittir. Bir proses çalıştığında kendisinden büyük en küçük parçaya yerleşir. BATCH sistemlerde kullanımı mümkündür.

1.3. Yer Değiştirme (Relocation) ve Koruma (Protection)

- Multiprogramming işleminde relocation ve protection birer problemdir. Bu problemlere çözüm olarak bilgisayarlarda base ve limit adında iki register eklenmiştir.

Base: Prosesin bulunduğu partition'un başlangıç adresi
Limit: ilgili partition'un uzunluğu(son adresi)

- **100** numaralı adrese gidilecekse yapılan **çağrı 100 değil 100+base** olacaktır.

2. Swapping

- Bir proses belleğe girer ve görevini tamamlayana kadar bellekte kalır. **Bellekte yer bulamayan prosesler ise disk üzerinde tutulur** ve **zamanı gelince belleğe dinamik olarak yüklenir. Partitionlar kesinlikle sabit değildir**, değişkendir.
- Bölümlerin boyutları sabit olmadığı için bellek bazen büyük bazen küçük parçalara ayrılır ve daha verimli bir kullanım sağlanır. Ama bellek yönetimi daha zor olur.
- Bellek dinamik olarak tahsis ediliyorsa 2 yapı kullanılır.
  a. Bitmap b. Listeler

a. Bitmap Bellek Yönetimi

- Bellek boyutları birbirine eşit küçük birimlere ayrılır. Her bellek birimi bir bit ile temsil edilir. **Bu bit 1 ise dolu alan, 0 ise boş alan demektir.**

b. Bağlı Listeler ile Bellek Yönetimi

- Liste, her biri boşluğu yada prosesi temsil eden düğümlerden oluşur.

P: Dolu
H: Boş
** Bu yöntemle çalışan proses sayısı belirlenebilir.

- Yeni proses oluşturulduğunda ya da mevcut proses diskten RAM'e yükleneceği zaman bellekte yer tahsisinde kullanılan algoritmalar vardır;

a. First Fit

- İlk yer algoritmasıdır.
- Her seferinde **en baştan** tarama yaparak **ilk bulduğu yeterli alana** prosesi tahsis eder. Eğer boşluk proses boyutundan fazla ise bölüm parçalanır ve gerektiği kadar alanı kaplar

ÖRNEK:

P,0,5 → H,5,3 → P,8,6

Bu kısma yerleşecekse 1 birim gerekliyse

P,0,5 → P,5,1 → H,6,2 → P,8,6 olur.

b. Next Fit

- Sonraki uygun yer algoritmasıdır.
- First fit gibi yerleştirme yapar ama farkı hep **en baştan tarama yapmaz**. İlk en baştan yapar sonra kaldığı yerden devam eder.

c. Best Fit

- En uygun yer algoritmasıdır.
- Her seferinde **en baştan** tarama yapar. Bellekte boşlukları çok parçalamadan en uygun en verimli alana ulaşmayı hedefler.

NOT: Hız sıralaması;

First Fit > Next Fit > Best Fit

3. Sanal Bellek (Virtual Memory)

- Bilgisayarda kullanılan programların bellek boyutlarını aşması ile sanal bellek kullanılmaya başlanmıştır.
- Örneğin; 16MB'lık bir program **page** adı verilen parçalara bölünür ve 4MB'lık kısmı RAM'de(bellekte) 12MB'lık kısmı diskte tutulur. Program aradığı parçayı bulamadığında **system call** yapar ve diskten getirilip RAM'de kullanmadığı parça ile değişimi gerçekleştirilir.

Not: Swapping RAM'de ve disk arasında proses değişimi yaparken paging aynı prosesin parça değişimi demektir.

3.1. Paging

- Sanal bellek işlemleri genellikle paging tekniğini kullanır. Bilgisayarda her bellek bölgesinin bir sanal adresi vardır, işlemcide çalışan proses tarafından üretilen. Bu sanal adres direkt bellek yolunda kullanılmaz ve sanal adres **MMU(verilen adres ile Base Register toplanır)** tarafından fiziksel adrese dönüştürülür.

Sayfa Tabloları (Page Tables)

- 16 bitlik bir adres ve sayfa boyutu 4KB olan bir sistemde; ilk 4 bit hangi sayfa üzerinde yerleştirme yapılacak onu belirler yani 2^4=16 tane sanal bellek sayfası adresler, son 12 bit ise ofset adresini adresler yani seçilen sayfa içinde kaçıncı satıra yerleştirme yapılacağını belirler.

Sayfa boyutu 4kb ise ofset 12 bit olmalı. ($4 * 2^{10}=2^{12}$)
Sayfa boyutu 8kb ise ofset 13 bit olmalı. ($8 * 2^{10}=2^{13}$)

Çok Katmanlı (Multi Level) Page Tables

- Büyük boyutlu sayfa tablolarının bellekte tutulduğu sistemlerce kullanılır.

Örnek:
32 bitlik sanal adresi olan bilg.
4KB'lık sayfa boyutu (12 bit ofset)
} 32-10=20
2^{20} adet sanal sayfa

Sayfa Tablolarındaki Kayıt Yapısı

Protection: 1: yalnız okunabilir RAM sayfası
0: hem okulabilir hem yazılabilir RAM sayfası
Modified: Sayfaya bir şey yazıldığında bu bit 1 yapılır ve sayfa **dirty** olur, disk üzerine yeniden kayıt yapılması gerekir.
Referenced: Bir sayfa proses tarafından kullanılmaya başlandığı an bu bit 1 yapılır.

TLBs – Translation Lookside Buffers

- TLB, sanal adresten fiziksel adrese dönüşümü sayfa tablolarını kullanmadan yapan bir donanımdır. Associative memory de denilebilir.
- **TLB de sık kullanılan sayfalar bulunur.**

Sayfa Değiştirme Algoritmaları (Page Replacement Algorithms)

- Sistemde sayfa hatası (page fault) alındığında OS bellekten en az kullanılan sayfayı seçer ve diske gönderir ve RAM'de boşluk oluşur. Prosesin aradığı sayfa da bu boşluğa yerleştirilir. Bu işlemi yapan algoritmalar disk-RAM ve cache-RAM arasında kullanılır. Veri ilk cache de aranır.

a. Optimal Sayfa Değiştirme Algoritması

- En iyi algoritmadır fakat kullanmak imkânsızdır.
- Şöyle çalışır; Bir sayfa kaç komut sonra çalışacak, o sayı ile etiketlenir. Etiket numarası en büyük olan diske gönderilmek üzere seçilir.

b. The First-In, First-Out (FIFO)

- Bir sayfa hatası oluştuğunda liste başındaki sayfa çıkarılır ve yeni gelen sayfa liste sonuna eklenir.

c. Son Zamanlarda Kullanılmayan(The Not Recently Used =NRU)

- R: Proses tarafından kullanımda olan sayfa.
- M: Üzerinde değişiklik yapılmış sayfa.
- Bir sayfa harası oluştuğunda OS R ve M bitlerine göre sayfaları ayırır.
- Bellekten çıkarılacak sayfa boş olmayan, En düşük numaralı sınıftan seçilir.

d. İkinci Şans (The Second Chance)

- R=0 ise sayfa çıkarılır.
- R=1 ise R=0 yapılır ve liste sonuna eklenir.

e. Saat(The Clock)

- Random selection olarak sürekli seçim yaparak R=0 olan sayfa aranır ve bulunduğunda bellekten çıkarılır.
- Saat kolu genelde en eski sayfaları işaret eder.

f. En Yakın Zamanda Kullanılan (The Least Recently Used = LRU)

- Uzun zamandır kullanılmayan sayfa diske gönderilir. Bu methodun donanımsal çözümü için **counter** kullanılır. Her komuttan sonra counter 1 arttırılır. En küçük sayaca sahip sayfa bellekten çıkarılır.

g. Çalışma Kümesi (Working Set)

- Proses çalışmadan önce anlık olarak(currently) ihtiyaç duyduğu sayfaları RAM'e bildirir. Buna **prepaging** denir ve bu sayede page fault minimuma indirilir.
- R=0 olanları al ve en son page fault zamanından son kullanılan ms leri çıkar T den büyük olanı RAM'dan kaldır.

HAFTA 1o – GİRDİ – ÇIKTI AYGITLARI (I/O DEVİCES)

- I/O aygıtları 2 kategoriye ayrılır.

a. Blok Aygıtları

- Bir blok aygıtı, verileri sabit boyutlu **bloklar** halinde saklar. Bu adreslenebilir ve birbirinden bağımsız olarak okunup yazılabilir.
- En temel blok aygıt: **Diskler**

b. Karakter Aygıtları

- Karakter aygıtları, verileri **stream** halinde alıp, gönderirler. Bu veriler adreslenemez ve bu aygıtlarda arama işlemi yapılamaz.
- En temel karakter aygıtları: Yazıcılar, Ağ aygıtları, Mouse gibi cihazlardır.

Aygıt Kontrolcüleri (Device Controller)

- Kontrolcü yerine adapter da denir.
- Temel görevleri: veri okumak, veri yazmak, hata kontrolü yapmak, belleğe veri kopyalamak gibidir.

Bellek Eşlemeli I/O (Memory Mapped I/O)

- Her kontrolcü içinde yazmaç(register) ve veri tamponlar (buffers) vardır.
- Yazmaçlar, işlemci ile iletişimin sağlanmasında kullanılır. İşlemci yazmaçlara komut ve bilgi yazarak aygıt üzerinde kontrol sağlar.
- Veri tamponları ise OS tarafından okunur ve ya veri tamponuna OS tarafından veri yazılabilir.
- Örneğin; ekran kartlarındaki veri tamponuna OS veri yazar, ekran kartı da bunu gösterir.

- İşlemcinin, kontrolcünün yazmaçlarını okuyup veri yazabilmesi için 2 seçeneği vardır.

1. Her kontrol yazmacının bir port numarası vardır. Bu port numarasıyla register'a ulaşım sağlanır.

IN REG, PORT ; gerekli kontrol yazmacının port numarası, işlemci içindeki register'a kopyalanır.

OUT PORT,REG ; işlemci kendi register'ındaki değeri gerekli kontrol yazmacının port'una yazar.

2. Her kontrol yazmacı bellekteki bir alan ile eşleştirilirler. Bu bellek alanları sanal bellek uzayına koyulmaz ve kullanıcının direk ulaşımı engellenir. Bu sistemlere **memory-mapped I/O** denir.

Doğrudan Bellek Erişimi (DMA – Direct Memory Access)

CPU'nun boşa giden zamanını engellemek için kullanılır.

DMA olmadan;
- Aygıttan veri okunurken tampona koyulma, hata kontrolü vb. işlemler yapılır ve DMA yoksa bu işlemleri OS ve CPU üstlenir, bu yüzden işlemci zamanı boşa harcanır.

DMA kullanıldığında;
İşlemci DMA'yı bilgilendirir. DMA gerekli bilgileri alır. İşlemleri yapar ve işlemciye interrupt gönderilir. Bu sayede CPU kullanılmadan bellek ile kontrolcü arasında akış sağlanmış olur.

INTERRUPTS(Kesmeler)

- Bir I/O aygıtı kendisine verilen görevi bitirince yola(bus) bir sinyal koyarak interrupt oluşturur. Kesme kontrolcüsünün işi yoksa hemen işleme alınır işi varsa interrupt kuyruğuna koyulur (yüksek öncelikli önce işleme alınır). Bu kesme işlendiğinde işlemciye gönderilir ve işlemciden ACK mesajı gelene kadar kesme sinyali işlemciye devamlı gönderilir. İşlemciye gelen kesme numarasını işlemci, **interrupt vector** adı verilen tablodan kontrol eder ve bu kesmede yapılacak kodların bellek bölgesinin başlangıç adresini elde eder. Bu adresi PC sayacına koyar ve prosedür işlemeye başlar ve ilk iş kesme kontrolcüsü bilgilendirilir.
Not: İşlemci ACK mesajını biraz geç göndererek ölümcül kilitlenmeleri engeller.

Aygıt Sürücüleri (Device Drivers)

- Aygıta özel çalıştırılan ve aygıt üzerinde istediğimiz işlemleri gerçekleştiren koda device driver denir. Device Drivers direk çekirdek(kernel) içinde olabileceği gibi kullanıcı uzayında da olabilir. Kullanıcı uzayında bulunan sürücülere system call ile erişilir.
- Aygıt sürücü temel görevleri:
  - okuma
  - yazma
  - günlükleme
  - aygıt açılıp, kapatılması
  - hata kontrolü
  - aygıt kullanımda mı?

HAFTA 11 – DİSK

- Manyetik diskler silindir şeklinde organize edilirler.
- Cylinder, track, sector.
- Sector, veriyi diskte okuyabildiğimiz, yazabildiğimiz ve adresleyebildiğimiz en küçük birimdir. Boyutu büyüdükçe veriye ulaşım hızlanır. Fakat sectorden çok küçük bir veri sectorun küçük bir bölümünü kapalar ama sectorün diğer boş alanları kullanılamaz.

RAID (Redunant Array Of Independent Discs)

- Disk güvenliğini ve performanını arttırmak için çıkarılmış disk organizasyon biçimidir.
- Temel fikir bilgisayara birden fazla disk ekleyip bu disklerin tek disk gibi çalışmasını sağlamaktır.
- Veriler RAID disklerde paylaştırıldığından **paralel** işlem gerçekleştirmek mümkündür.

RAID Level 0:
- Sanal diskin her birinde k adat sector bulunur.
- Strip0→(0, k-1) Strip1→(k,2k-1)
- Performans mükemmel ve uygulama(implemantation) basittir. Çünkü veri aynı anda farklı sürücülere yazılır. Bu yüzden okuma yazma hızı yüksektir.
- Disklerden biri arızalanırsa, veriler kaybolur.

RAID Level 1:
- Her diskin kopya(duplucates) diski bulunur. Yani 4 birincil disk, 4 yedekleme diski var.
- Yazma işlemi iki diskede yapıldığından yazma hızı düşüktür. Okuma performansı ise iyidir. Çünkü bir disk arızalandığında diğer yedek diske geçilir. Yanı hata toleransı mükemmeldir.
- Onarma(Recovery) işlemi de basittir. Arızalı disk yerine yenisi konulur ve sağlam olan diskten birebir kopyalama yapılır.

RAID Level 3:
- Bu seviyeden sonraki RAID yöntemlerde parity check ilave diski verinin saklanması için kullanılır.
- Parity biti hata denetiminde kullanılır, hata düzeltmede değil.

Disk Kolu Zamanlama Algoritmaları (Disk Arm Scheduling Algorithms)
- Bir disk bloğunun okuma yazma hızını etkileyen faktörler;
  a. **Arama Zamanı (Seek Time)**
- Kolun uygun silindir üzerine konulması
  b. **Rotational Delay (Dönmeden Kaynaklı Gecikme)**
- Sectörün okuyucu kafanın altına getirilmesi olayı.

** En önemlisi arama zamanıdır. Bu yüzden arama zamanının azaltılması(reducing) performansı arttırır,geliştirir.

FCFS Disk Scheduling Algorithm

- FCFS: First-Come, First-Served
- İlk gelen ilk işlem görür. Yani istek(request) sırasına göre işlem yapılır.

SSF Disk Scheduling Algorithm

- SSF: :Shortest Seek First
- Mevcut olan noktadan en yakın(closest) silindire gidilir.

Elevator Disk Scheduling Algorithm

- Gidilmeye başlanan yönde cevap verilmeyen istek kalmayana kadar devam eder. Daha sonra geri döner.