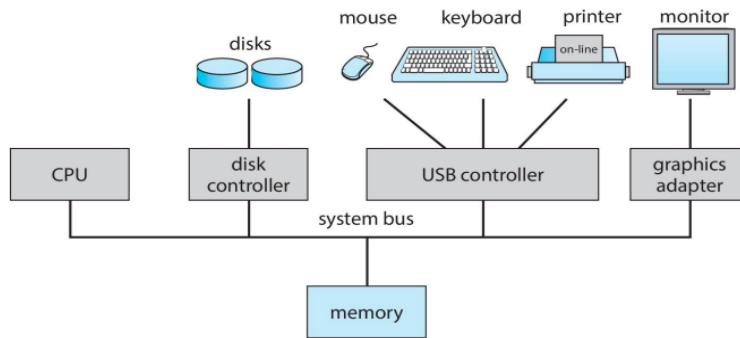


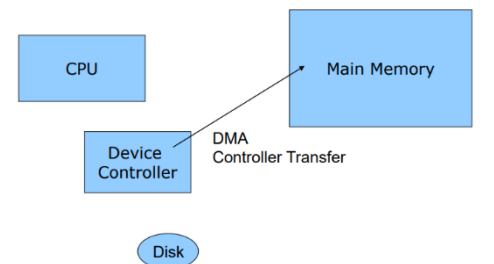
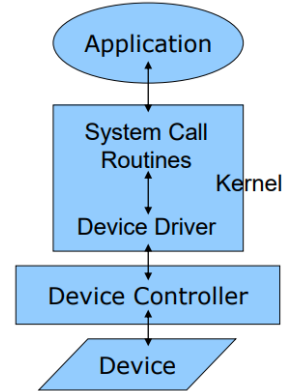
# İŞLETİM SİSTEMLERİ

- **İşletim Sistemi Nedir?:** Bilgisayar donanımı ile kullanıcı arasında bir aracı olarak görev yapan programdır. Ekmek kızartma makinelerinden gemilere, uzay araçlarından oyun makinelerine, TV' lere ve endüstriyel kontrol sistemlerine kadar birçok farklı cihazda bulunabilir. Askeriye için kullanılan sabit kullanımlı bilgisayarların daha genel amaçlı kullanıma ve kaynak yönetimi ve program kontrolüne ihtiyaç duyulduğu zaman ortaya çıkmıştır.
- **İşletim Sistemi Tanımı**
  - İşletim sistemini, mevcut kaynakların ayırıcısı (resource allocator) olarak düşünebiliriz. Bir bilgisayar sisteminde problemin çözülmesi gereken çok fazla kaynak vardır. Bunlar mikroişlemci zamanı, bellek, giriş/çıkış aygıtlarıdır. İşletim sistemi bu kaynakların yöneticisi olarak belli programlar ve kullanıcı ihtiyacına göre kaynakları ayırma işlemini yapmaktadır.
  - İşletim sistemi kontrol programıdır diyebiliriz. İşletim sistemi giriş/çıkış aygıtlarını ve kullanıcı programlarını kontrol ederek hatalı ve yanlış kullanımı önlemeye çalışan bir kontrol programıdır.
  - Daha genel bir tanımla, işletim sistemi bilgisayarda her an çalışan tek programa denir ve bu çekirdek (kernel) olarak adlandırılır.
- **İşletim Sistemi Amaçları**
  - Kullanıcı programlarını çalıştırır ve kullanıcı sorunlarını daha kolay çözüme ulaştırır.
  - Bilgisayar sisteminin kullanımını kolaylaştırır.
  - Bilgisayar donanımını verimli bir şekilde kullanmayı sağlar.
- **Bilgisayar Sistem Yapısı (Structure):** Bilgisayar sistemi 4 bileşene ayrılabilir.
  - **Donanım:** Temel bilgisayar(hesaplama) kaynaklarını sağlar. (CPU, Bellek, Yazıcı, Klavye, Monitör gibi I/O cihazları)
  - **İşletim sistemi:** Çeşitli uygulamalar ve kullanıcılar arasında donanım kullanımını kontrol ve koordine eder.
  - **Uygulama programları:** Sistem kaynaklarını, kullanıcıların bilgisayar (hesaplama) problemlerini çözmek için kullanırlar. (Kelime işlemciler, muhasebe programları, derleyiciler, web tarayıcıları, veritabanı sistemleri, video oyunları gibi.)
  - **Kullanıcılar:** İnsan, makina, diğer bilgisayarlar.
- **İşletim Sistemi Ne Yapar?:**
  - Kullanımı kolay, kaynakların verimli kullanımını sağlayan bir ortam sağlar. Çünkü kullanıcılar; rahatlık, kullanım kolaylığı ve iyi performans isterler. Kaynak kullanımını önemsemezler.
  - Mainframe (sunucu) gibi paylaşılan ya da minicomputer gibi bilgisayarların kullanıcılarının mutlu olmasını sağlar.
  - Workstation (İş İstasyonu) ve server (sunucu) gibi bilgisayarlarda kaynak ayırımını yapıp verimli kullanılmasını sağlar.
  - El (handheld) bilgisayarı, Akıllı telefonlar gibi kaynakları kısıtlı olan sistemlerde kullanılabilirliği ve pil ömrünü optimize eder.
  - Küçük bir arayüzü olan ya da arayüzü olmayan gömülü sistemlerin kullanımını sağlar.
- **Bilgisayarın Başlatılması**
  - Bootstrap (ön yükleme) programı açılışa veya yeniden başlatıldığında yüklenir.
  - ROM ya da EEPROM içinde bulunur, firmware (üretici yazılımları) olarak adlandırılır.
  - Sistem tüm yönleriyle başlatılır.
  - İşletim sistemi çekirdeğini yükler ve yürütme başlar. Çekirdek sürekli çalışır.
- **Bilgisayar Sistem Organizasyonu**

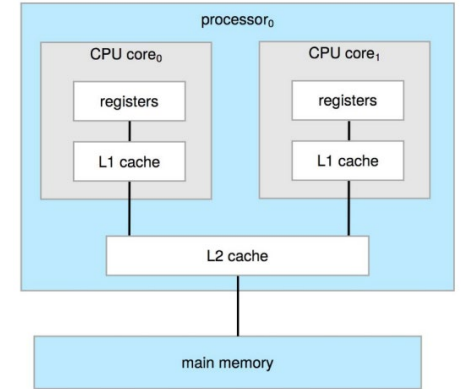
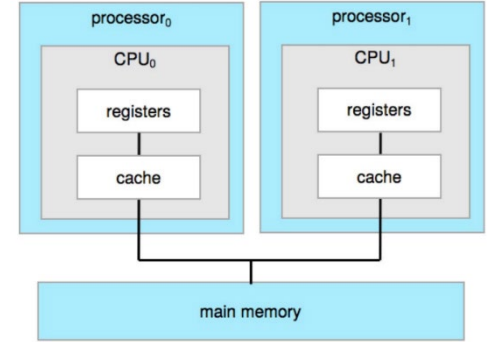


- Bir veya daha fazla CPU ve cihaz kontrolörleri (denetleyici, controller), ortak veriyolu üzerinden paylaşılan hafızaya erişim sağlar.
- Bellek döngüleri (cycle) için yarışan CPU ve I / O cihazları aynı anda (eş zamanlı) yürütülebilir.
- Her aygıt denetleyicisi, belirli bir cihaz türünden sorumludur ve her aygıt denetleyicisinde yerel bir tampon (buffer) bulunur.
- CPU / yerel arabellek (local buffer) ile ana bellek (main memory) arasında veri taşır.
- I/O işlemleri cihazdan, yerel buffer'a ve yerel buffer'dan da ana belleğe gelir.
- Aygıt denetleyici bir kesme (interrupt) ile işlemin bittiğini CPU'ya bildirir.
- **Kesme (Interrupt)**
  - Bilgisayar ilk açıldığında / yeniden başlatma işlemi gerçekleştirdiğinde çalışmasını sürdürebilmek için bir programa ihtiyaç duyar.
  - İlk sistem başlatma programı, sistemin bütün bileşenlerini başlangıç için hazır duruma getirir. Bu program, işletim sisteminin belleğe nasıl yükleneceğini ve nasıl çalışmaya başlatılacağını bilmelidir.
  - İşletim sistemi çalışmaya başladıktan sonra olayların oluşması için beklemeye başlar.

- Bir işin veya olayın oluşu sisteme “kesme” aracılığıyla bildirilir. Donanım sistemi, istediği zaman **sistem veriyolu ile** mikroişlemciye kesme gönderebilir. Yazılım ise **sistem çağrısı** ile kesme gönderebilir.
- Her kesmeden sorumlu ve kesmenin ayrıntılarını içeren bir **servis rutini** vardır. **Kesme servis rutini**, o kesme oluştuğunda yapılması gereken işlemleri içeren komutların bütünüdür.
- **Mikroişlemciye kesme sinyali geldiğinde**, mikroişlemci o anda yaptığı işi bırakır ve kesme tarafından belirlenen yere yönelir. Belirlenmiş yer genellikle kesmenin servis rutininin başlangıç adresini içerir. Mikroişlemci kesme servis rutini gerçekleştirildikten sonra kesmeden önce yapmakta olduğu yarım kalmış görevine döner ve onu yapmaya devam eder.
- Kesmeler, uygun kesme servis rutinine transferi kontrol etmelidir. **Kesmelerin çabuk ele alınması amacıyla**, olabilecek kesmeler için önceden belirlenmiş rakamlar veya kesmeyi işaret eden ve sistemi yönlüten bir **işaret tablosu** kullanılabilir.
- **İşaret tablosu** düşük bellek alanında bulundurulur. Burası birçok aygıtın kesme servis rutininin adresini tutar. Bu kısım **kesme vektörü** olarak adlandırılır.
- **Kesmeler, kesilmiş işlemin de adresini kurtarmak zorundadır.** Yeni sistemler kesilmiş işlemin adresini belleğe (stack) gönderirler. Kesme işlemi icra edildikten sonra, kurtarılan adres program sayacına yüklenir ve yarım kalmış işleme kaldığı yerden devam edilerek tamamlanması sağlanır.
- İşletim sistemi bir kesmeyi yürütürken, diğer kesmeler devre dışı bırakılır ve yürütülen kesme bittiğinde diğerleri aktif duruma geçebilir. Fakat **kesmelerin önemliliklerine göre aralarında öncelik durumları vardır.** Aynı anda birden çok kesme gelirse; kesmelerin öncelik durumlarına bakılır, yüksek öncelikli kesme önce işleme alınır, diğerleri maskelenir.
- **İşletim sistemi kesme denetleyicisidir.** Trap (ayrıcılık) ise hatalar (yanlış bellek erişimi, sıfıra bölme hatası vb) nedeniyle veya kullanıcı programlarının isteği üzerine üretilir.
- **Kesmelerin (Interrupts) Ortak İşlevleri**
  - Tüm servis rutinlerinin adreslerini içeren **Interrupt (Kesme) Vektörü** aracılığı ile kontrol kesme (interrupt) servisine aktarılır.
  - Kesme mimarisi, **kesintiye uğrayan servisin adresini kaydetmelidir.**
  - Bir **tuzak (trap)** ya da **istisna (exception)**, bir hata ya da kullanıcı isteğinin (request) neden olduğu yazılım tarafından oluşturulan bir kesmedir.
  - İşletim sistemi, **interrupt driven** (Kesmelere Dayalı) bir yapıdadır.
- **Kesme İşleme (Interrupt Handling)**
  - İşletim sistemi, **register'ları ve program counter'ı tutarak (saklayarak) CPU'nun durumunu korur.**
  - **Hangi tür kesmenin meydana geldiğini** belirler.
  - **Ayrı kod segmentleri, her bir kesme türü için hangi işlemin yapılması gerektiğini** belirler.
- **I/O Yapısı**
  - **I/O işlemeye yönelik iki yöntem:**
    - 1) I/O başladıktan sonra, **kontrol yalnızca I/O tamamlandığında** kullanıcı programına döner.
      - Wait komutu, bir sonraki kesmeye kadar CPU'yu boşta (idle) bırakır.
      - Wait loop (bellek erişimi için çekişme)
      - Birim zamanda en fazla bir I/O talebi karşılanabilir, eş zamanlı I/O işleme durumu yoktur.
    - 2) I/O başladıktan sonra, **kontrol I/O'nun tamamlanmasını beklemeden** kullanıcı programına döner.
      - **System Call (Sistem Çağrısı)** kullanıcının I/O'un tamamlanmasını beklemesine izin vermek için işletim sistemine yapılan istektir.
      - **Device-status table (Cihaz Durum Tablosu)**, tipini, adresini ve durumunu gösteren her I/O cihazı için giriş içerir.
      - İşletim sistemi, cihaz durumunu belirlemek ve tablo girişini kesinti içerecek şekilde değiştirmek için I/O cihaz tablosuna izin oluşturur.
  - **Uygulama programları, I/O işlemlerini işletim sistemi üzerinden gerçekleştirirler.**
    - **İstek (Request), bir Sistem Çağrısı (System Call) (işletim sistemi rutini) çağrılarak yapılır.**
    - İşletim sistemindeki sistem çağrısı rutini, işletim sistemindeki cihaz sürücüsü (device drive) rutinleri yardımıyla I/O işlemini gerçekleştirir.
    - Bir sistem çağrısı yaptıktan sonra, bir uygulama çağrının bitmesini bekleyebilir (engellenen çağrı) veya başka bir şey yapmaya (engellenmeyen çağrı) devam edebilir.
- **Doğrudan Bellek Erişimi Yapısı (Direct Memory Access Structure)**
  - DMA ile cihaz denetleyicisi, **veri bloklarını CPU müdahalesi olmadan cihaz arabelleğinden doğrudan ana belleğe aktarır.**
  - Byte başına bir kesme yerine, blok başına yalnızca bir kesme üretilir.
  - **Bellek ile I/O birimleri arasında doğrudan veri değişimlerinin yapılmasıdır.**
  - İşleyiş tarzı:
    - İşlemci I/O modülünü hafızaya yazma veya hafızadan okuma ile **yetkilendirir.**
    - Değişim esnasında işlemci **sorumluluğu devreder.**
    - Bu değişim esnasında, **işlemci diğer işleri yapabilir.**
- **Depolama Yapısı (Storage Structure)**
  - **Main memory (Ana Bellek, RAM): CPU'nun doğrudan erişim sağlayabildiği depolama birimidir.**

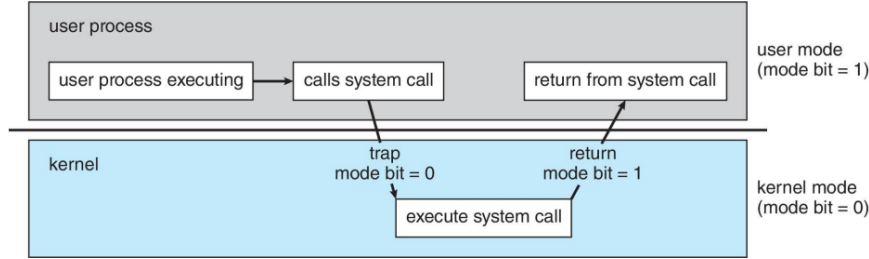


- **Random Access (Doğrudan Erişim):** İstenilen veriye **doğrudan erişim** sağlanabilir. Manyetik teyplerdeki gibi, bir veriye ulaşmak için ondan önceki verileri geçmek gerekmez. Dynamic Random-access Memory (DRAM)'in bir formudur.
- **Volatile:** **Uçucu veriler barındırır. Yani güç kesildiğinde veriler kaybolur.**
- **İkincil depolama:** Büyük, kalıcı (non-volatile) depolama kapasitesi sağlayan ana bellek uzantısıdır.
- **Hard Disk Drives (HDD) / Sabit Disk Sürücüler (HDD):** Manyetik malzeme ile kaplı sert metal veya cam plakalardır. Disk yüzeyi, mantıksal olarak sektörlere bölünmüş track'lere (izlere) bölünmüştür. Disk controller / Disk denetleyicisi, aygıt ve bilgisayar arasındaki mantıksal etkileşimi sağlar.
- **Depolama Hiyerarşisi:** Hız, Maliyet ve Uçuculuk açısından hiyerarşik olarak düzenlenir.
  - **Caching (Ön bellekleme):** Bilgiyi daha hızlı depolama sistemine kopyalamak, boyut ve hız arasında var olan değiş tokuştan kaynaklanır.
  - **Ana bellek bir ön bellek olarak kullanılır.** İkincil depolama birimlerindeki veriler kullanılabilirlik için önce ana belleğe kopyalanmalı, ikincil depolama birimlerine kopyalanacak veriler ise mutlaka önce ana bellekte bulundurulmalıdır.
  - **Küçükten Büyüğe ve Hızlıdan Yavaş Sıralanma:** Maviler uçucu (volatile), kırmızılar kalıcı belleklerdir.  
**Registerlar < Donanım Ön bellekleri L1, L2... < Ana Bellek < İkincil Bellek (Hard Disk) < Manyetik Teypler**
- **Bilgisayar Sistem Mimarisi**
  - **Multiprocessors (Çok İşlemcili):** Çok işlemcili bir bilgisayar sistemi, **iki veya daha fazla işlemcinin (CPU), ortak bir RAM'e tam erişim sağlayarak, tüm işleri paylaştığı** bir sistemdir. Bir işlemcili sistemden farkı, çok işlemciyle yapılacak işlerin paylaşarak daha hızlı bitirilmesi esasına dayanır.
    - **Çok işlemcili sistemin avantajları**
      - **Artan İş Hacmi / Verimlilik:** Daha çok işi daha kısa zamanda gerçekleştirebilirler.
      - **Ölçek Ekonomisi:** Çoklu PC kullanımından daha ucuzdur.
      - **Artan Güvenilirlik:** Hata toleransı.
    - **Çok işlemcili sistemin çeşitleri**
      - 1) **Asymmetric Multiprocessing:** Her işlemciye özel bir görev atanır.
      - 2) **Symmetric Multiprocessing:** Her işlemci tüm görevleri yerine getirir.
  - **Paralel Sistemler:** Bir sistemin paralel sistem olarak nitelendirilebilmesi için, birden çok işlemcisi olması ve her işlemcinin kendisine tahsis edilen, işin belli kısımlarını "eş zamanlı" olarak çalıştırması gerekir.
  - **Çift Çekirdekli Tasarım (Dual-Core Design):** Çok çipli (multichip) ve çok çekirdekli (multicore) dir. Sistem, tüm çipleri içerir. Birden çok ayrı sistem içeren ana gövdeye sahiptir.
  - **Kümelenmiş Sistemler (Clustered Systems):** Çok işlemcili sistemler gibidir, ancak **birlikte çalışan birden çok sistem** vardır. Bilgisayarlar **depolama birimlerini ortaklaşa kullanır**. Genellikle Depolama Alanı Ağı (storage-area network (SAN)) aracılığıyla depolama paylaşımı vardır. Hatalardan kurtulan yüksek kullanılabilirlik hizmeti sağlar. Bir bilgisayarda hata oluşması durumunda, o bilgisayarın görevlerini **sistemdeki diğer bilgisayar yerine getirir**. Bazı kümeler, **yüksek performanslı bilgi işlem / high-performance computing (HPC)** içindir. Bazılarında çalışan işlemleri önlemek için, **distributed lock manager (DLM) dağıtık kilit yöneticisi** bulunur.
    - **Asymmetric clustering:** Çalışırken bekleme modunda bir makineye sahiptir.
    - **Symmetric clustering:** Uygulamaları çalıştıran ve birbirini izleyen **birden çok düğüme sahiptir**.
  - **Çoklu Program İşletimi (Multiprogramming):** Birden çok program başlatılabilir ve yüklenebilir. Sistemdeki toplam işlerin bir alt kümesi bellekte tutulur. Kullanışlı ve verimlidir. **Tek kullanıcı CPU ve I/O aygıtlarını her zaman meşgul tutamaz. Herhangi bir kullanıcıya ait herhangi bir iş, bir sebeple bekleme geçtiğinde, işlemci boş durmaz ve farklı bir kullanıcının iş talebine yanıt verir.** İşlemci bekleme yapmak yerine, tıpkı multitasking örneğinde olduğu gibi farklı bir işe geçer. Hangi işin yapılacağını işletim sistemi seçer. (İş planlaması, Job Scheduling)
  - **Çoklu Görev (Zaman Paylaşımı) Multitasking (Timesharing):** Bir işletim sisteminde, bir kullanıcının **birden fazla process'i aynı anda işleme alabilmesidir**. Multiprogramming sistemlerinin mantıksal bir uzantısıdır. CPU, işleri o kadar sık değiştirir ki kullanıcılar çalışırken her işle etkileşime girerek etkileşimli hesaplama oluşturur. (Yapay eş zamanlılık)
    - Yanıt Süresi (Response time) < 1 saniye olmalı.
    - Her kullanıcının bellekte yürütülen en az bir programı vardır. (process)
    - Birden fazla iş aynı anda çalışmaya hazırsa **CPU Scheduling** yapılır.
    - İşlemler belleğe sığmazsa, **swapping** onları çalıştırmak için içeri ve dışarı taşır.
    - **Virtual memory** tamamen bellekte olmayan işlemlerin yürütülmesine izin verir.
- **İşletim Sistemi İşlemleri (Operations):** Bootstrap programı, sistemi başlatmak için **yürütülen ilk koddur**. Çekirdeğin (kernel) yüklenmesinden sorumludur. Çekirdek (kernel) yüklenir. İşletim sisteminde **geri planda çalışan programlar (system daemons)** başlatılır. Kernel, **kesmelerle yönetilir. (interrupt driven)**



- **Çift Modlu Çalışma (Dual-Mode Operation)**

- İşletim sistemi **kendisi ve diğer sistem bileşenlerini korumak için dual-mode işlemi sağlar.** (User ve Kernel)
- **Mod biti kullanılır:** Sistemin kullanıcı kodunu veya çekirdek kodunu ne zaman çalıştırdığını ayırt etme yeteneği sağlar. Kullanıcı kodu yürütülürken mod biti **"user"**, çekirdek kodu yürütülürken mod biti **"kernel"** olur.
- **Kullanıcının mod bitini açıkça "kernel" olarak ayarlamayacağını nasıl garanti ederiz?:** Sistem çağrısı, modu çekirdeğe değiştirir, çağrıdan geri dönüş onu kullanıcıya sıfırlar. **Ayrıcılık olarak belirlenmiş bazı talimatlar, yalnızca çekirdek modunda çalıştırılabilir.**



- **Zamanlayıcı (Timer)**

- Kaynakları tüketen (Sonsuz Döngü gibi) bir süreci önler.
- Bir süre sonra bilgisayarı kesintiye uğratacak şekilde ayarlanır.

- **İşletim Sistemlerinin Temel Konseptleri/Parçaları/Fonksiyonları**

- **1) İşlem (Process) Yönetimi**
  - Process, **yürütülmekte olan program**dır. Sistem içindeki bir çalışma birimidir. **Program pasif bir varlıktır; process ise aktif bir varlıktır. Program diskte, process RAM'de bulunur.**
  - İşlem (Process) görevini yerine getirmek için **kaynaklara ihtiyaç duyar.** CPU, bellek (memory), I/O, dosyalar (files), Başlatma Verileri (Initialization data) gibi.
  - İşlemin (Process) sonlandırılması, **yeniden kullanılabilir kaynakların geri alınmasını** gerektirir.
  - Bir thread'e sahip bir process, yürütülecek **sonraki komutun yerini belirten bir program sayacına sahiptir.** İşlem, tamamlanana kadar talimatları (instructions) sırayla, birer birer yürütür. **Çok thread'li processin, thread başına bir program sayacı vardır.**
  - Genel olarak sistemde birçok process vardır, bazıları **kullanıcı**, bazıları da bir veya daha fazla CPU üzerinde aynı anda çalışan **işletim sistemi processidir.**
  - **Eş zamanlılık (Concurrency)**, işlemler/iş parçacıklarının arasında CPU'nun çoklaması ile gerçekleştirilir.
  - **İşlem Yönetimi Aktiviteleri (İşletim Sisteminin İşlem Yönetimi ile bağlantılı olarak sorumlu olduğu faaliyetler):**
    - Hem kullanıcı hem de sistem **processlerinin oluşturulması ve silinmesi**
    - **Processleri askıya alma ve devam ettirme**
    - **Process senkronizasyonu** için mekanizmalar sağlamak
    - **Process iletişimi** için mekanizmalar sağlamak
    - **Kilitlenme ile başa çıkmak** için mekanizmalar sağlamak
- **2) Bellek (Memory) Yönetimi**
  - Bir programı çalıştırmak için komutların (instructions) tamamı (veya bir kısmı) bellekte olmalıdır. (Program bellekte olmalı)
  - Programın ihtiyaç duyduğu verilerin tamamı (veya bir kısmı) bellekte olmalıdır. (Örneğin programda kullanılacak db verileri)
  - **Bellek yönetimi, bellekte neyin ne zaman olacağını belirler. (CPU kullanımının optimize edilmesi)**
  - **Bellek Yönetim aktiviteleri:**
    - Şu anda **hafızanın** hangi bölümlerinin **kimler tarafından kullanıldığını takip etme**
    - Hangi işlemlerin (veya bunların bölümlerinin) ve **verilerin belleğe girip çıkacağına karar verme**
    - İhtiyaca göre **bellek ayırma ve belleği serbest bırakma**
- **3) Dosya Sistem (File System) Yönetimi**
  - Dosya ve dizin **oluşturma ve silme**
  - Dosyaları ve dizinleri **değiştirmek için temel öğeler**
  - Dosyaları **ikincil depolamaya eşleme** (mapping)
  - Dosyaları **kararlı (uçucu olmayan) depolama ortamına yedeklenmesi**
- **4) Depolama Yönetimi (Storage Management)**
  - Bilgisayar ile veriler değişik aygıtlara saklanabilmektedir. Veri saklama ortamları; manyetik disk, manyetik teyp ve optik disklerdir. Her bir aygıtın kendi karakteristiği ve fiziksel organizasyonu vardır.
  - Bu depolama birimlerinde **bilgiler mantıksal olarak ünitelendirilmiştir**, buna da **dosya** denir.
  - **Dosya, birbiriyle ilişkili bilgilerin bir araya getirilmesiyle oluşturulur.** Dosyalar genelde **program ve veriyi temsil ederler.**
  - Dosyalar, anlamları oluşturucusu tarafından belirlenen **bit, bayt ve satırların sıralanmasından oluşur.**
  - **Kütle-Depolama Yönetimi (Mass-Storage Management):**
    - Ana bellek (birincil depolama birimi) kalıcı değildir ve tüm programların çalıştırılması için çok küçük kalmaktadır. Bilgisayar sistemi ana belleği yedekleyebilmek için ikincil depolama birimlerini sağlamıştır.

- İşletim Sistem **ikincil belleğe bağlı** şu girişimlerden sorumludur:  
1) Boş disk alanı yönetimi, 2) Diskin paylaşımı, 3) Diskin planlanması.
- **Caching (Ön bellekleme)**: Bilgisayarda birçok seviyede gerçekleştirilen önemli bir prensiptir (donanım, işletim sistemi, yazılım). Kullanımdaki bilgiler geçici olarak daha yavaş depolamadan daha hızlı depolamaya kopyalanır. Bilginin orada olup olmadığını belirlemek için önce daha hızlı depolama (önbellek) kontrol edilir. Öyleyse, bilgiler doğrudan önbellekten kullanılır (Hızlı). Değilse, veriler önbelleğe kopyalanır ve orada kullanılır. Önbellek, önbelleğe alınan depolamadan daha küçüktür. Önbellek yönetimi önemli bir tasarım sorunudur. Önbellek boyutu ve değiştirme politikaları da önemlidir.

#### 5) I/O Kontrol ve Yönetimi

- İşletim sisteminin bir amacı, donanım aygıtlarının özelliklerini kullanıcıdan gizlemektir.
- **I/O alt sisteminin sorumlu olduğu durumlar** şunlardır:
  - I/O bellek yönetimi şu durumları içerir;
    - **Buffering/Ara belleğe alma** (aktarılrken verilerin geçici olarak depolanması),
    - **Caching/Ön belleğe alma** (performans için verilerin bölümlerinin daha hızlı bellek alanlarında tutulması),
    - **Spooling/Kuyruklama** (bir işin çıktısının diğer işlerin girdileriyle örtüşmesi)
  - Genel aygıt sürücüsü arayüzü
  - Belirli donanım cihazları için sürücüler

#### 6) Koruma (Protection)

- **Protection / Koruma**: İşlemlerin veya kullanıcıların işletim sistemi tarafından tanımlanan kaynaklara erişimini kontrol etmek için kullanılan herhangi bir mekanizma olarak tanımlanabilir.
- **Security / Güvenlik**: Sistemin iç ve dış saldırılara karşı savunulmasıdır.
- Sistemler genellikle, kimin neyi yapabileceğini belirlemek için kullanıcılar arasında ayırım yapar.
  - **Kullanıcı kimlikleri (güvenlik kimlikleri)**, kullanıcı başına bir tane olmak üzere ad ve ilgili numarayı içerir. Kullanıcı Kimliği daha sonra erişim kontrolünü belirlemek için o kullanıcının tüm dosyalarıyla, işlemleriyle ilişkilendirilir.
  - **Grup tanımlayıcısı (grup kimliği)**, kullanıcı setinin tanımlanmasına ve kontrollerin yönetilmesine, ardından her işlemle, dosyayla ilişkilendirilmesini sağlar.
  - **Ayrıcalık yükseltme**, kullanıcının daha fazla hakla etkin kimliğe geçmesine olanak tanır.

#### İşletim Sistemi Servisleri

##### 1) İşletim sistemi, programlar ve kullanıcılar için programların ve servislerin yürütülebildiği bir ortam sağlar.

- **Kullanıcı arayüzü**: Command-Line (CLI), Grafik Kullanıcı Arayüzü (GUI), TouchScreen (Dokunmatik Ekran)
- **Program yürütülmesi**: Sistem bir programın belleğe yüklenmesini, bu programın yürütülmesini ve sonlandırılabilmesini sağlamalıdır.
- **Giriş/Çıkış işlemleri**: Yürütülen program bazı giriş/çıkış işlemlerini gerektirebilir. Kullanıcı direkt olarak giriş/çıkış aygıtlarını kontrol edemediği için işletim sistemi bu kontrolü sağlamalıdır.
- **Dosya sistemi idaresi**: Programlar; dosya okumak, yazmak, oluşturmak ve silmek istemektedirler. İşletim sistemi programların bu isteklerini sağlayacak yeterlikte olmalıdır.
- **İletişimler**: Tek bir bilgisayar kendi içinde veya ağda bağlı olan bir bilgisayar, başka bir bilgisayar ile veri alışverişi yapmak isteyebilir. Bu iletişim, ya işletim sistemi ile paylaşılan bellek üzerinden veya mesaj geçişi tekniği ile sağlanabilir.
- **Hata kontrolü**: İşletim sistemi, kendi sistemi üzerindeki her türlü hatayı (hata mikroişlemci, bellek, donanım içinde olabilir) kontrol etmek ve hata olduğunda yapılacak işlemin belirlenmesini ve yapılmasını sağlamak zorundadır.

##### 2) İşletim sistemi, sistemin uygun ve verimli bir şekilde yönetilmesini sağlar.

- **Kaynak paylaşımı**: Çok kullanıcı veya aynı anda birden fazla iş yapılan bir sistemde kaynaklar her bir kullanıcı ve iş için ayrılmalıdır.
- **Hesaplama**: Kullanıcıların sistem kaynaklarından hangilerini ne şekilde kullandıklarını takip etmek, kullanım istatistiklerinin çıkarılması için gerekli olmaktadır. Bu özellik sistemin geliştirilmesi veya yeniden kurmak isteyenler için gerekli olabilir.
- **Koruma**: Çok kullanıcı sistemlerde birden çok iş yürütülürken bu işlemlerin birbirine karışmaması sağlanmalıdır. Koruma, tüm sistem kaynaklarının kontrol altında olmasıdır. Ayrıca sistemdeki herkesin şifrelerle kaynaklara ulaşımı sağlanarak korumalı bir erişim sağlanabilir.

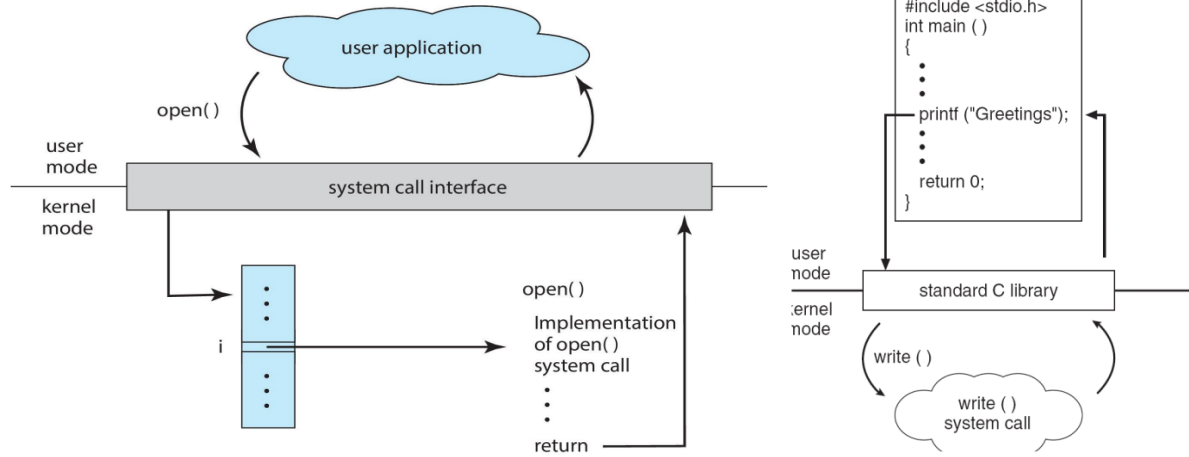
- **Komut Satırı Yorumlayıcısı (Command Line Interpreter (CLI))**: CLI doğrudan komut girişine izin verir. Bazen çekirdekte, bazen sistem programı tarafından uygulanır. Bazen birden fazla yapı uygulanır (kabuklar). Öncelikle kullanıcıdan bir komut alır ve onu çalıştırır. Bazen yerleşik komutlar, bazen sadece programların adları kullanılır.

#### Sistem Çağrıları / System Calls

- **İşletim sistemi tarafından sağlanan servisler için** programlama arayüzüdür.
- Genellikle yüksek seviyeli bir dille (C veya C++) yazılır.
- **Doğrudan sistem çağrısı kullanımı yerine** çoğunlukla yüksek seviyeli Uygulama Programlama Arayüzü (API) aracılığıyla, programlar tarafından erişilir. En yaygın üç API, Windows için Win32 API, POSIX tabanlı sistemler için POSIX API (neredeyse tüm UNIX, Linux ve Mac OS X sürümleri dahil) ve Java sanal makinesi (JVM) için Java API'dir.
- **Her sistem çağrısı bir sayı ile ilişkilendirilir**. Sistem çağrısı arayüzü, bu numaralara göre indekslenmiş bir tablo tutar.
- Sistem çağrısı arabirimi, işletim sistemi çekirdeğinde amaçlanan sistem çağrısını çağırır ve sistem çağrısının durumunu ve herhangi bir dönüş değerini döndürür.

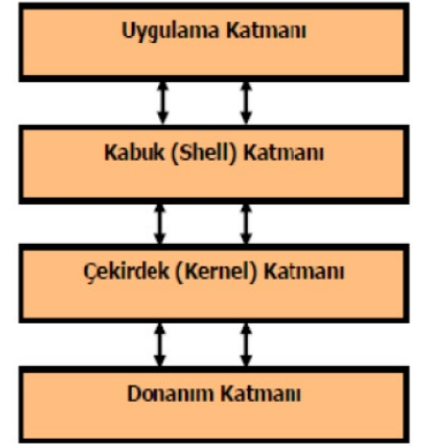


- Sistem çağrısını çağıran uygulama, sistem çağrısını nasıl gerçekleştirebileceğini bilmelidir. API'de durum farklıdır. API'de sistem çağrısının nasıl uygulandığının bilinmesine gerek yoktur. Sadece API'ye uyması ve sonuç çağrısı olarak işletim sisteminin ne yapacağını anlaması gerekir. İşletim sistemi arayüzünün çoğu detayı programcıdan API aracılığı ile gizlenmiştir.
- **Sistem Çağrısı Parametresi Geçişi:** System call süreci içerisinde çağrı numarasından fazla bilgi göndermek de mümkündür. Gönderilecek bilginin tipi ve miktarı gibi bilgiler de gönderilebilir. İşletim sistemine parametre göndermek için 3 yöntem mevcuttur.
  - 1) En basit yöntem **Registers içerisine göndermektir.**
  - 2) Diğer bir yöntem ise **hafızada bir blokta ya da tabloda tutma** yöntemidir. Bu durumda hafızanın adresi register ile gönderilmelidir. Bu sistem Linux ve Solaris işletim sistemleri tarafından uygulanmaktadır.
  - 3) En son yöntem ise **Stack yöntemidir.** Yığın içerisine atılan parametreler işletim sistemi tarafından çekilir. Blok ve yığın yöntemlerinde herhangi bir sınır bulunmaz.



- **Sistem Çağrısı Türleri:** İşlem Kontrolü, Dosya Yönetimi, Cihaz Yönetimi, Bilgi Sağlama, İletişim, Koruma.
- **Sistem Programları:** Program geliştirme ve yürütme için uygun bir ortam sağlayan programlardır. Çoğu kullanıcının işletim sistemi görüşü, gerçek sistem çağrıları tarafından değil, sistem programları tarafından tanımlanır. Bazıları basitçe sistem çağrılarına yönelik kullanıcı arayüzleridir, diğerleri çok daha karmaşıktır. Aşağıda belirtilen başlıklarla sınıflandırılabilirler:
  - **Dosya işlemleri:** Dosyaları ve dizinleri silmek kopyalamak, tekrar isimlendirmek, yazdırmak ve listelemek amacıyla kullanılır.
  - **Durum bilgileri:** Bazı programlar sistemden tarih, zaman, boş bellek, boş disk alanı, kullanıcı sayısı gibi durum bilgilerini isteyebilir. Bu durumda bu bilgiler terminallere veya çıkış olarak aygıtlara gönderilir.
  - **Dosya Değişim İşlemleri:** Birçok text editörü diskteki dosya içeriklerini oluşturmak ve değişiklik yapmak için kullanılmaktadır.
  - **Programlama dili desteği:** Birçok programlama dili, (PASCAL, JAVA, VISUAL BASIC) derleyicisi, assemblyleri ve yorumlayıcısı işletim sistemi ile birlikte sağlanmaktadır.
  - **Program yükleme ve yürütme:** Bir program derlendiğinde, bu programın çalıştırılabilmesi için önce belleğe yüklenmesi gerekir. Sistem; bellek yükleyicileri ve bağlantı editörlerini temin etmelidir.
  - **İletişim:** Bu program, farklı işlemler, kullanıcılar ve bilgisayarlar arasında sanal bir bağlantı sağlamaktadır. Bu şekilde kullanıcılar birbirlerine mesaj gönderip dosya transferi yapabilmektedirler. FTP, browser gibi.
  - **Arka plan hizmetleri.**
  - **Uygulama programları.**
- **İşletim Sistemi Yapısı (Structure)**
  - **Simple Structure (MSDOS) (Basit Yapı):** MS-DOS – Küçük bellek alanında pek çok işlevin sağlanabilmesi için yazılmıştır. Modüllere bölünmez. Arayüzler ve işlev seviyeleri kesin ayrılmamıştır.
  - **Layered Approach (Katmanlı Yaklaşım):** İşletim sistemi çeşitli katmanlara ayrılmıştır. En alt kısımda (katman 0) donanım yer alır, en üst kısımda ise (katman N) kullanıcı arabirimleri yer alır. Her katman, sadece alt düzey katmanlar tarafından sağlanan işlemleri yürütür. Bir katmanın bu işlemleri nasıl yapıldığını bilmesi gerekmez. Dolayısıyla, her bir katman bazı veri yapıları, işlemleri, donanım ve varlığını üst düzey katmanlardan gizler.
  - **Microkernel Approach (Mikro Kernel Yaklaşım):** Kernel'e temel fonksiyonların görevlerini atar. Sadece en önemli işletim sistemi fonksiyonları vardır. Çekirdekten "kullanıcı" alanına doğru kayma sağlanır. İletişim, kullanıcı modülleri arasında mesaj göndermekle gerçekleştirilir. Küçük boyutludur.
  - **Modules Approach (Modül Yaklaşımı):** Bir çok modern işletim sistemi çekirdek modüllerini uygular. Loadable Kernel Modules (LKMs). Nesne yönelimli yaklaşımı kullanırlar. Her temel bileşen ayrıdır. Arayüzler üzerinden haberleşme sağlanır. Modüller Kernel içerisinde gerektiğinde yüklenebilirler. Katman yapısına benzer, ancak daha esnektir. Linux, Solaris örnek verilebilir.
  - **Monolithic Structure: Original UNIX – Donanım işlevselliği ile sınırlı bir yapıya sahiptir.** UNIX OS, iki ayrılabilir parçadan oluşur. 1) Sistem programları, 2) Çekirdek.

- **Hibrit Sistemler:** Çoğu modern işletim sistemi tek bir model üzerine kurulu değildir. Hibrit yapıda, **performans, güvenlik ve kullanılabilirlik ihtiyaçlarını karşılamak için birden çok yaklaşım birleştirilir**. Linux ve Solaris çekirdekleri monolitik yapıdadır, işlevselliğin dinamik olarak yüklenmesi için ise modüler yapıdadırlar. Windows çoğunlukla monolitik, artı farklı alt sistem yapıları için mikro çekirdek yapısındadır. Apple Mac OS X hibrit, katmanlı yapıdadır.



- **Temel İşletim Sistemi Katmanlı Yapı**
  - **Uygulama Katmanı:** Kullanılan programlar bu katmanda yer alır. Word, Excel vb.
  - **Kabuk (Shell) Katmanı:** Genellikle kullanıcı arayüzü de denilen, kullanıcı ile bilgisayar iletişimini sağlayan arabirimdir. MS-DOS komut istemi arayüzü, Linux'da root olarak girildiğinde #, kullanıcı olarak girildiğinde \$ olarak görülen arabirim.
  - **Çekirdek Katmanı:** Kabuk katmanından gelen komutlar doğrultusunda, donanım katmanı ile iletişime geçerek, gerekli işlemleri yürüten kısımdır.
  - **Donanım Katmanı:** Donanım elemanlarının bulunduğu kısımdır.

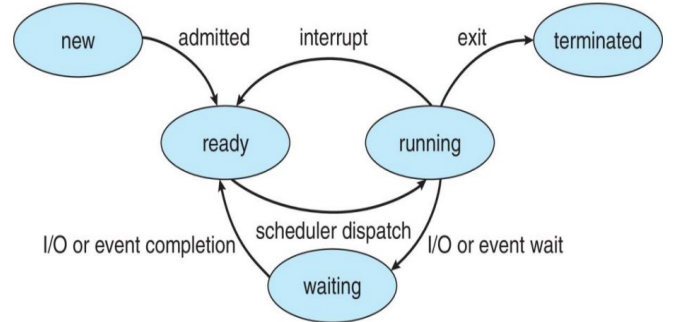
- **Android:** Google ve Open Handset Alliance tarafından kodlanmış, **Linux İşletim Sistemi** tabanlı (ancak değiştirilmiş), mobil cihazlar için geliştirilmiş açık kaynak kodlu bir işletim sistemidir. Çalışma zamanı ortamı, temel kitaplık seti ve Dalvik sanal makinesi içerir. Java artı Android API'de geliştirilen uygulamalar, Java bayt koduna derlenen ve daha sonra Dalvik VM'de çalıştırılardan yürütülebilir dosyaya çevrilen Java sınıfı dosyaları içerirler.

#### İşlem (Process) Kavramı

Program, diskte depolanan **pasif varlıktır** (yürütülebilir dosya/executable file); **process ise aktiftir**. Çalıştırılabilir bir dosya belleğe yüklendiğinde **program processe dönüşür**. Bir **program birden fazla process** olarak yürütülebilir. Aynı programı çalıştıran birden fazla kullanıcıyı düşünebiliriz. Örneğin, Windows işletim sisteminde Word programı, web tarayıcı, e-posta kontrolüne ait programlar aynı anda çalışabilmektedir. İş (job) veya işlem (process) aynı anlamda kullanılmaktadır. Process aşağıdakileri içerir.

- **Program Kodu (Text Section)**
- **Program Counter:** İşlemcide bir sonraki işletilecek komutun ana bellekteki adresini tutan 'register' dir.
- **Stack (Yığın):** Alt program değişkenleri, dönüş adresleri vb. değerleri içerir.
- **Veri:** Değişkenleri, Diziler, Listeleri içerir.
- **Heap:** Nesneleri içerir.

- **İşlem (Process) Durumu:** Processin hangi durumda olduğunu gösterir.



- **Yeni (New):** İşlemin yeni oluşturulduğunu gösterir.
- **Çalışıyor (Running):** İşlemin komutları yürütülmektedir.
- **Bekliyor (Waiting):** İşlem bir olayın gerçekleşmesini beklemektedir. Örneğin bir I/O işlemi.
- **Hazır (Ready):** İşlem bir işlemciye atanmak için beklemektedir.
- **Bitti (Terminated):** İşlem çalışmasını bitirdi.

- **İşlem Kontrol Bloğu (Process Control Block, PCB):** İşletim sisteminde

her işlem, işlem kontrol bloğu ile temsil edilmektedir. Kontrol bloğu içerisinde bulunan bilgiler aşağıdaki gibidir.

- **Process Durumu:** Yukarıda değinmiştik.
- **Process Numarası:** Her process'in bir ID'si bulunmaktadır.
- **Program Sayacı:** Sayaç, process için yürütülmesi gereken bir sonraki komutun adresini gösterir.
- **CPU Registers (Mikroişlemci Kayıt Edicileri):** Kayıt ediciler bilgisayar mimarisine göre sayı ve çeşit olarak değişebilmektedir.
- **Mikroişlemci Programlama Bilgileri:** Bu bilgiler, işlem önceliğini, programın sıra göstergesini ve diğer programlama değişkenlerini içermektedir.
- **Bellek Yönetim Bilgileri:** Bu bilgiler taban ve limit kayıtçılarının değerlerini, sayfa tablolarını veya işletim sistemi tarafından kullanılan bellek sistemine göre değişen segment tablosunu içerir.
- **Hesaplama Bilgileri:** Bu bilgiler, mikroişlemcini kullandığı gerçek zaman sınırlamaları, hesap numaraları, iş ve işlem numaraları gibi bilgileri içerir. Giriş/Çıkış durum bilgileri; işlem için ayrılmış giriş/çıkış aygıtlarının ve açık dosyaların listesi gibi bilgileri kapsar.

- **İşlemden İşleme CPU Geçişi (Context Switch):** CPU **bir processten diğerine** geçtiğinde gerçekleşir. CPU başka bir processe geçtiğinde, sistem **eski işlemin durumunu kaydetmeli ve yeni işlem için kaydedilmiş durumu bir bağlam anahtarı (context switch) aracılığıyla yüklemelidir**. Bağlam değiştirme süresi tamamen ek yüküdür; **sistem kullanımı geçiş yaparken CPU boş (CPU Idle) durumdadır**. İşletim sistemi ve PCB ne kadar karmaşık, bağlam anahtarı da o kadar uzun olur.

- **İş Parçacığı (Thread):** Process'in eş zamanlı olarak işlenen her bir bölümüdür.

- **Thread ve Process arasındaki fark:** **Oluşturuluşu ve kaynakların paylaşılması** açısından farklılıkları vardır. Çoğu durumda **threadler**, **processlerin içinde yer alır ve onları oluştururlar**. Çoklu thread'ler paralel olarak pek çok bilgisayar sisteminde uygulanabilir. **Processler aynı kaynakları paylaşmazlar** (bellek gibi). Fakat aynı process içindeki **farklı thread'ler aynı kaynakları paylaşabilirler**.

- **İşlem Programlama (Scheduling)**

- **Multiprogramming'in amacı CPU kullanımını maksimize etmek** için, **her zaman çalışan işlemler olmasını sağlamaktır**. Ayrıca **ready kuyruğunda bekleyen processlerin bekleme sürelerini azaltabilme**yi sağlar.

- **Process Scheduler**, CPU çekirdeğinde **bir sonraki yürütme için mevcut processler arasından seçim yapar**. Süreçlerin zamanlama sıralarını (scheduling queues) korur.
- **Process Scheduling Kuyrukları** (Process Scheduling Queues)
  - **İş (Job) kuyruğu**: Sistemdeki tüm işlemlerin kümesidir.
  - **Hazır (Ready) kuyrukları**: Ana bellekte bulunan hazır ve yürütülmek için bekleyenler. (CPU'yu bekliyorlar).
  - **Aygıt (Device) kuyrukları**: Bir I/O aygıt için bekleyen işlemlerin kümesidir.
- **Process'ler ile ilgili işlemler**
  - **İşlem Oluşturma (Creation)**: Ana (Parent) process, çocuk (children) processleri oluşturur, bu çocuk processler de sırayla diğerlerini oluşturarak bir işlemler ağacı oluştururlar. Genel olarak processler, "**process id**" (pid) aracılığıyla tanımlanır ve yönetilir.
    - **Kaynak Paylaşım seçenekleri**;
      - Parent ve children işlemler tüm kaynakları paylaşırlar.
      - Children işlemler, parent işlemlerin kaynaklarının alt kümesini paylaşırlar.
      - Parent ve children işlemler hiçbir kaynağı paylaşmazlar.
    - **Yürütme (Execution) Seçenekleri**;
      - Parent ve children işlemler eş zamanlı (concurrently) olarak yürütülürler.
      - Parent işlemler, children işlemler sonlanıncaya (terminate) kadar beklerler.
  - **İşlem Sonlandırma (Termination)**: Process, yapacağı görevleri tamamladığında durarak, işletim sistemine "**çıkış sistem çağrısı**"nı kullanarak durdurulması gerektiğini bildirir. **exit()**. [Child işlemiden parent işleme verilerin durumu döndürülerek (**wait()**) komutu ile]. Child işlemin kullandığı kaynaklar işletim sistemi tarafından geri alınarak.] Parent, **abort()** komutunu kullanarak child işlemin yürütülmesini (Process id ile) sonlandırabilir.
    - **Basamaklı Sonlandırma (cascading termination)**: Tüm children, grandchildren, vb. işlemler sonlandırılmalıdır. Sonlandırma işlemi işletim sistemi tarafından başlatılır. Parent işlem, **wait()** sistem çağrısını kullanarak, bir alt işlemin sonlandırılmasını bekleyebilir. Çağrı, durum bilgisini ve sonlandırılan işlemin **pid**'sini döndürür. **pid = wait(&status);**
- **Neden CPU Scheduling algoritmalarına ihtiyacımız var?**
  - CPU'nun hiç boş bırakılmaması ve **CPU kullanım zamanını maksimize etmek**.
  - Bir process sonlandıktan sonra, **ready kuyruğunda bekleyen process'lerden birinin seçilmesi gerektiği için**, CPU Scheduling algoritmaları **bu kararların doğru ve verimli bir şekilde verilmesini sağlamaktadır**.
- **CPU – I/O Burst Cycle**: **Processin yürütülmesi**, **CPU'da koşturulma** ve **I/O beklemesi**nden oluşur. Processler bu iki durum arasında değişmektedir.
- **CPU Scheduler (İşlemci Zamanlayıcısı)**: Bellekte (RAM) **çalışmaya hazır halde (ready kuyruğunda) bekleyen processlerden birini seçerek** işlemciyi ona ayırır. Seçme kararını CPU Scheduling algoritmalarına göre verir.
- **CPU Scheduling ne zaman devreye girer?** **CPU boşta olduğunda, yani "CPU Idle" denilen zamanda** devreye girer. Bu da **bir processin** sonlanması, bloklanması, I/O işlemi veya bir sistem çağrısı geldiğinde olur. Bu olayları, process durumuna göre ifade edersek, **CPU Scheduling kararlarını ne zaman verir?** Process;
  - 1. Çalışma (running) durumundan **bekleme (waiting)** durumuna geçerken,
  - 2. Çalışma (running) durumundan **hazır (ready)** durumuna geçişte,
  - 3. Bekleme (waiting) durumundan **hazır (ready)** durumuna geçişte,
  - 4. Sonlandığında (terminates) verir.
- Yukarıda 1. ve 4. maddeler için zamanlama açısından seçim yoktur. Ready kuyruğunda process varsa, sıradaki seçilir. (Non-Preemptive) Ancak 2. ve 3. maddeler için bir seçim vardır. Kısaca **Ready durumlarına geçişte bir seçim vardır**. (Preemptive)
- **Kesintili Algoritmalar (Preemptive)**: Yürütülen processin bitirilmeden önce, mikroişlemciden kaldırılması ve istenilen başka bir processin mikroişlemcide yürütülmesidir.
- **Kesmeyen Algoritmalar (Non-preemptive)**: Process, mikroişlemciye yerleştikten sonra; tamamlanıncaya veya durana kadar mikroişlemciyi kullanır.
- **Dispatcher (Görev Dağıtıcısı)**: CPU Scheduler (Kısaca "Scheduler" da diyebiliriz) tarafından seçilen process CPU'nun kontrolünün verilmesini sağlar ve **Context Switch** yapar.
- **Dispatch Latency (Gönderim Gecikmesi)**: Bir process durduktan sonra, diğer process başlayana kadar, o arada önceki processin, kontrol bloğunun kaydedilmesi ve sonraki processin kontrol bloğunun yüklenmesi ve processin çalışmaya başlaması arasında geçen zamandır.
- **Planlama Kriterleri (Scheduling Criteria)**
  - **1. CPU Utilization (İşlemci Kullanımı)**: İşlemciyi olabildiğince **meşgul tutmak**.
  - **2. Throughput (Üretilen iş)**: Birim zamanda bitirilen process sayısı.
  - **3. Turnaround time (Devir zamanı)**: Bir processin CPU'da bitene kadar geçirdiği toplam zaman. (Beklemeler ve Koşturulma dahil)
  - **4. Waiting time (Bekleme zamanı)**: Bir processin **ready kuyruğunda geçirdiği toplam süre**. Bekleme süresi.
  - **5. Response time (Yanıt süresi)**: Bir processin ready kuyruğuna ilk kez girmesi ile, **process'e verilen ilk yanıt** arasında geçen süre.
- Yukarıdaki maddelerde **1 ve 2'nin maksimum, diğerlerinin minimum** olması istenir / amaçlanır.
- **Scheduling Algoritmaları**: Bu algoritmalar ile hesap yaparken, her bir processin ne kadar süreceğini bildiğimizi (tahmin) varsayıyoruz.
  - **FCFS (First Come First Served)**: Non-Preemptive'dir. **Kuyruğa ilk gelen processin işlenmesi** prensibine dayanır. Processler **kuyruğa hangi sırayla dizilirse, o sırayla işlenir**.

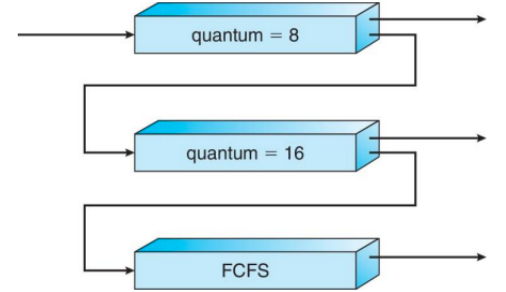


- **SJF (Shortest Job First):** Non-Preemptive'dir. Preemptive versiyonu, **SRTF (Shortest Remaining Time First)** olarak adlandırılır. **En kısa CPU burst time değerine sahip processi seçerek çalışır.** Her processle bir sonraki minimum burst time değerine sahip processi karşılaştırır ve hangisi daha küçükse onu çalıştırır. **SJF optimaldir, belirli bir işlem kümesi için minimum ortalama bekleme süresini verir.** Preemptive versiyonda, bir process çalışmaya başladıktan sonra, ready kuyruğuna burst time'ı daha kısa bir process gelirse, çalışan process durdurulur ve o kısa olan process devreye alınır.
- **Sonraki CPU Burst Uzunluğunu Belirleme:** Üstel ortalama (exponential averaging) kullanılarak önceki CPU burst sürelerinin uzunluğu tahmin edilebilir. 
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$
  $\tau_{n+1}$  yeni CPU burst için tahmin edilen değer,  $t_n$  ise n'inci CPU burst zamanının gerçek değeridir.  $\alpha$  değeri 0 ile 1 aralığındadır ve genellikle 0.5'tir.
- **Priority Scheduling (Öncelikli Planlama):** Her bir process'e öncelik sayısı (tamsayı) atanır. CPU en öncelikli olan process'e tahsis edilir. **En yüksek öncelik, en küçük tam sayıya aittir.** Preemptive ve Non-Preemptive versiyonları vardır. Bu algoritmanın kullanımında, **Starvation durumu ortaya çıkabilir. Starvation, sürekli yüksek öncelikli processlerin yürütülüp, düşük öncelikli processlerin hiç yürütülmemesi problemidir.** Çözüm olarak kullanılan **Aging, zaman geçtikçe processin önceliğinin artırılmasıdır.**
- **Round Robin (RR):** Özellikle **zaman paylaşımli sistemler için** tasarlanmıştır. İnteraktif işlemleri olan sistemler için kullanışlıdır. Round Robin için bir **zaman aralığı (time quantum) tanımlanmıştır**, bu zaman aralığı 10 ile 100 msn arasında değişmektedir. **q değerinin büyük olması durumunda algoritma FCFS algoritmasına benzer. q değerinin küçük olması durumunda context switch işlemi çok fazla yapılacaktır.** Bu yüzden, CPU'nun context switch için harcadığı vakit gereğinden fazla olacaktır. Bu istenen bir durum değildir.
- **Multilevel Queue (Çok Düzeyli Kuyruk):** **Ready kuyruğu bölünüp, başka kuyruklar oluşturulur.** Her bir kuyrukta, farklı tipte processler bir araya getirilmiş olur. Örneğin, **Foreground-İnteraktif processler için Round Robin algoritmasının kullanılacağı ayrı bir kuyruk, Background processler için de FCFS algoritmasının kullanılacağı ayrı bir kuyruk tasarlanabilir.** Multilevel kuyrukta, kuyruklar arası planlamanın da yapılması ve gerekirse, **kuyruklar arası geçişlerin de gerçekleştirilebilmesi gerekir.**
  - **Sabit Öncelikli Planlama (Fixed priority scheduling):** Bu tür bir planlama, örneğin önce foreground daha sonra background processlerin çalıştırılması şeklinde olabilir. Bu tür bir planlamada starvation problemi ortaya çıkabilmektedir.
  - **Time Slice:** Belli bir CPU zamanı, tüm kuyruklara paylaştırılabilir. Örneğin, %80 foreground ve %20 background processler çalışacak şekilde bir planlama yapılabilir.
- **Çok Düzeyli Geri Bildirim (Multilevel Feedback Queue):** **Bir process birçok kuyruk arasında geçişler yapabilir, kuyruklar arasında hareket edebilir. Process bir kuyrukta işini bitiremezse, diğer kuyruğa taşınır.**

Yapıyı oluşturmak için dikkat edilmesi gerekenler:

- Kaç kuyruk oluşturulacak?
- Her kuyruk için hangi algoritmalar kullanılacak?
- Bir processin öncelik değeri hangi yöntemle yükseltilecek? (Bekleme süresine göre mi? Burst time'dan kalan süreye göre mi?)
- Bir processin öncelik değeri hangi yöntemle indirgenecek?
- Process ilk geldiğinde hangi kuyruğa girecek?

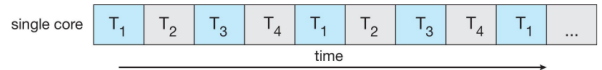
Sağdaki örnekte, 3 kuyruk bulunmaktadır. Round Robin algoritmasının uygulandığı q değeri 8ms olan  $Q_0$  kuyruğu, Yine q değeri 16ms olan  $Q_1$  ve FCFS algoritmasının uygulandığı  $Q_2$  kuyruğu bulunmaktadır. 30ms'lik bir process önce  $Q_0$ 'a gelir, 8ms koşutulduktan sonra bitmediği için  $Q_1$ 'e geçer, orada da 16ms koşutulur ve bitmediği için  $Q_2$ 'e taşınır. En son  $Q_2$ 'te bitirilene kadar devam eder.



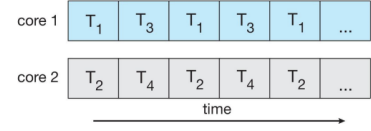
- **Process ve Thread:** Thread, komut komut işlemlerin yürütülmesidir. Programlar ana bir thread ile çalışır. Bu thread'e single-thread, kontrol thread veya execution flow denir. Process içindeki bir thread, inputu beklerken, diğer bir thread başka bir işi yapabilir. Processlerin her biri ayrı bir adres uzayına sahiptir. Process içindeki threadler ise aynı adres uzayını paylaşırlar. Bu açıdan process içinde global bir değişken tanımlanırsa, bu değişken her threadde kullanılabilir.
- **Thread'in Faydaları:**
  - **Responsiveness (Cevap verebilirlik):** Özellikle **kullanıcı arayüzü için önemlidir.** Processin bir parçası engellenmişse, işlemin diğer bölümlerinin yürütülmesine izin verir.
  - **Resource Sharing (Kaynak Paylaşımı):** **Threadler aynı adres uzayını paylaşırlar.** Bu durum, paylaşılan bellekten veya mesaj geçişinden daha kolay uygulanabilir.
  - **Ekonomi:** **Thread oluşturmak, process oluşturmaktan daha ucuzdur.** Threadde context switch'in maliyeti daha azdır. Process değişiminde, önceki processin Process Control Block'unda son kaldığı yerin kaydedilmesi ve diğer processde kaldığı yerin okunması ve çalışmaya hazırlanması maliyetli bir iştir.
  - **Scalability (Ölçeklenebilirlik):** **Paralel çalışma için yeni bir işlemci almak yerine, çok çekirdekli yapılar kullanılabilir.** Her bir thread ile farklı bir çekirdek birimi eşleştirilir. Multiprocessor yapılara göre daha ölçeklenebilir bir tasarım sağlar.
- **Bir sistemde Thread yapabilmek için neler gereklidir?**
  - **Sistem Mimarisi,** thread oluşturmaya uygun yapıda olmalıdır. **Multicore veya multiprocessor işlemci** olması gibi.
  - **İşletim Sistemi,** thread kullanımını desteklemelidir. Örneğin, **MS-DOS olursa thread oluşturulamaz.**

- **Kullanılan program/yazılım**, thread işlemini destekleyen bir program/yazılım olmalıdır.
- **Paralellik**: Bir sistemin **aynı anda, birden fazla görevi yerine getirebilmesidir**.
- **Eşzamanlılık**: İlerleme sağlayan **birden fazla görevi destekler**. Tek işlemcili ve tek çekirdekli yapılarda, eş zamanlılığı sağlayan şey "zamanlayıcı"dır.
- **MultiThread Server Mimarisinde**, Client, Server'a bir istek yaptığında, istekleri bekleyen thread'in yönlendirmesi ile serverda yeni bir thread oluşturulur ve o client'in istekleri ile ilgilenir. Serverdaki ana thread ise yeni client isteklerini beklemeye (listening) devam eder.
- **Multicore / Multiprocessor sistemlerde dikkat edilmesi gerekenler**:
  - **1. Faaliyetleri Bölme**: Programın parçalara ayrılabilmesi gerekir.
  - **2. Denge / Balance**: Bazı işler erken bitip, bitmeyen işleri bekliyorlarsa, dengesizlik var demektir. Hepsinin en kısa ve aynı zamanda bitmesinin ayarlanması (yani dengelenmesi) gerekir. Aksi halde bazı işlerin erken bitirilmesinin hiçbir anlamı olmaz.
  - **3. Veri Bölme**: Threadler arasında verileri bölmek önemlidir. Farklı bölme çeşitleri, sonucu farklı performanslar ortaya koyabilir.
  - **4. Veri Bağımlılığı**: Bir processin çalışması, başka bir verinin varlığına veya hesaplanmasına bağlı ise, onun beklenmesi gerekir.
  - **5. Testing ve Debugging**: Thread sistemlerinin test edilmesi daha zordur.
- **Parallelleştirme Türleri**
  - **Veri Parallelleştirme**: Veriler bölünüp, birden çok çekirdeğe dağıtılır. Her bir çekirdekte aynı işlem yapılır.
  - **Görev Parallelleştirme**: Tüm threadler tüm veriyi kullanır. Her bir thread farklı bir görevi gerçekleştirir.
- **Amdahl's Law**: Seri ve paralel bileşenlere sahip **bir sisteme, ek çekirdekler eklemekle elde edilen performans kazanımı** tanımlar. Bir uygulamanın seri kısmının, ek çekirdekler eklenerek elde edilen performans üzerinde "orantısız" etkisi vardır. Örneğin, **uygulama %75 paralel ve %25 seri ise, 1 çekirdekten 2 çekirdeğe geçiş 1.6 kat hızlanma ile sonuçlanır**. N sonsuza yaklaşırken hızlanma 1/S'ye yaklaşır.
- **Parallelleştirme artıkça; belli bir işlemci veya çekirdek sayısından sonra, hız kazanımı sabitlenir. Parallelleştirmenin sürekli artması, hızın sürekli artacağı anlamına gelmez!**
- **User Threads**: Yönetim ve kontrolü, kullanıcı seviyesindeki thread kütüphaneleri tarafından yapılır. Üç temel thread kütüphanesi vardır. POSIX, Windows ve Java Threads.
- **Kernel Threads**: Kernel seviyesinde desteklenen threadlerdir.
- **Multithread Modeller**
  - **Many to One**: Birçok user thread'in, tek bir kernel thread'e map edilmesi ile oluşturulur. Kernel thread'in bloklanması, tüm user threadlerin de bloklanmasına yol açar. Çok çekirdekli bir yapı olsa bile, çoklu user threadlerin yalnızca biri çekirdekte çalıştığı için, paralel olarak çalıştırılmayabilirler. Örnek olarak; **Solaris Green Threads** ve **GNU Portable Threads** verilebilir.
  - **One to One**: Her bir user thread, ayrı bir kernel thread'e map edilir. Her user level thread oluştuğunda, kernel thread oluşturulur. Many to One modeline göre, daha iyi eşzamanlılık sağlar. Örnek olarak **Windows** ve **Linux** verilebilir.
  - **Many to Many**: Birçok user thread, birçok kernel thread'e map edilir. Pek yaygın değildir. Örnek olarak **ThreadFiber** (Windows).
- **Pthreads**: Thread ve senkronizasyon oluşturmak için kullanılan **POSIX standartı bir API**'dir. User ve Kernel düzeyinde sağlanabilir. UNIX işletim sistemlerinde yaygındır. (Linux & Mac OS X)

• **Tek çekirdekli sistemde eşzamanlı yürütme:**



• **Çok çekirdekli bir sistemde paralellik:**



$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

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

#include <stdlib.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
    pthread_t tid; /* the thread identifier */
    pthread_attr_t attr; /* set of thread attributes */

    /* set the default attributes of the thread */
    pthread_attr_init(&attr);
    /* create the thread */
    pthread_create(&tid, &attr, runner, argv[1]);
    /* wait for the thread to exit */
    pthread_join(tid, NULL);

    printf("sum = %d\n", sum);
}
```

```
/* The thread will execute in this function */
void *runner(void *param)
{
    int i, upper = atoi(param);
    sum = 0;

    for (i = 1; i <= upper; i++)
        sum += i;

    pthread_exit(0);
}
```

```
#define NUM_THREADS 10

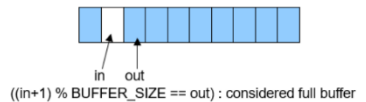
/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
    pthread_join(workers[i], NULL);
```

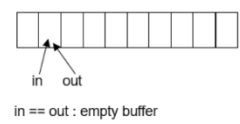
- **pthread\_t tid**: Oluşturulacak olan thread'i tanımlamak için kullanılır.
- **pthread\_attr\_t attr**: Oluşturulacak thread, oluşturulma anında bir takım ilk değerler/ayarlar ile oluşturulacaksa kullanılır.

- **pthread\_create(&tid, &attr, runner, argv[1]);** Thread oluşturma komutudur. İlk parametre tanımladığımız thread id'sinin adresidir. İkinci parametre, oluşturulma anındaki girilmek istenen ilk değerlerin adresidir. NULL olabilir. Üçüncü parametre, thread içerisinde çalıştırmak istediğimiz fonksiyonun adıdır. Dördüncü parametre, komut satırından girilen birinci parametredir.
- **pthread\_join(tid, NULL);** Thread'in bekleneyeceği belirtilir. İlk parametre, hangi thread'in bekleneyeceğini belirtir. İkinci parametre thread için verilen fonksiyonda, **pthread\_exit(return\_val);** komutu ile döndürülen return\_value değerini almak için kullanılır. Dönüş değeri istenmiyorsa, NULL girilir. Eğer kullanılmazsa, threadlerin başlamasına fırsat kalmadan main thread sonlandırılır.
- **pthread\_exit(0);** Thread'i sonlandırma komutudur. Bu komut kullanıldıktan sonra, pthread\_join() komutundan sonraki kodlar çalışmaya başlar. (pthread\_join()'de zaman parametresi NULL verildiyse)
- **Processler Bağımsız (Independent) ve işbirliği halinde (cooperating)** olmak üzere iki şekilde çalışırlar.
  - Bağımsız processler** başka processlerden etkilenmez ve aralarında bilgi alışverişi yoktur.
- **Processler arası iletişim (IPC – Interprocess Communication):** Aynı sistem içindeki farklı **processlerin haberleşebilmeleri** (bilgi alışverişi yapabilmeleri) için kullanılan işletim sisteminin sağladığı bir yapıdır.
- **İşbirliği halinde çalışan processlerin kullanılma nedenleri**
  - **Bilgi Paylaşımı:** Birçok kullanıcı aynı veriye ulaşmak isteyebilir. Bu **veriye çoklu ulaşımı** sağlayacak ortamı oluşturmamız gerekir.
  - **Hesaplama Hızı:** Processleri farklı görevlerde çalıştırarak daha **hızlı sonuçlar** elde edilebilir.
  - **Modülerlik:** Birçok açıdan kolaylık sağlar. Örneğin **belli bir modüle meydana gelen hatalardan diğer modüller etkilenmez.**
  - **Rahatlık:** Kullanıcı birçok işlemin aynı anda yapılmasını isteyebilir. Bu açıdan **kullanıcıya rahatlık** sağlar.
- **IPC için iki farklı model vardır.**
  - **Paylaşılan Bellek:** Processlerin ortak bir bellek alanını kullandığı yaklaşımdır. Ortak alan, her processin adreslerinden farklı da olabilir. Herhangi bir processin bellek alanını ortak olarak da kullanabilirler. Bir sistem çağrısı ile birlikte, **kernel ortak alanı (shared memory)** oluşturduktan sonra devreden çıkar. Yani **her seferinde bir kernel ihtiyacı yoktur.** Bu tip bir iletişim, işletim sisteminin değil, **processlerin kontrolü altındadır.** Bu açıdan hızlı ve avantajlıdır. Konuyu daha iyi anlamak için Üretici-Tüketici (Producer-Consumer) problemini incelemek gerekir.
    - **Üretici-Tüketici Problemi:** Bu problemde bir adet üretici ve bir adet tüketici processi bulunur. Üretici burada veriyi üreten ve tüketici ise veriyi tüketendir. Burada webserver ve client örneği verilebilir. Server bu örnekte üretici, client ise tüketicidir.
    - **Bounded (Sınırlı) ve Unbounded (Sınırsız) Buffer:** Paylaşılan bellek “buffer” olarak da adlandırılır. Üretici veriyi buffer'a koyar ve tüketici veriyi bufferdan okur. Bounded, sabit bir buffer boyutu olduğunu varsayar. Unbounded buffer'da ise buffer boyutunda herhangi bir sınır olmadığı varsayılır. Unbounded buffer'da üretici asla beklemes. Bounded buffer'da ise üretici tüm buffer alanları doluysa bekler. Her ikisinde de tüketici, tüketilecek bir buffer yoksa bekler. Yandaki şekilde **in**, bir sonraki boş buffer alanını; **out** ise ilk dolu buffer alanını temsil eder.
 

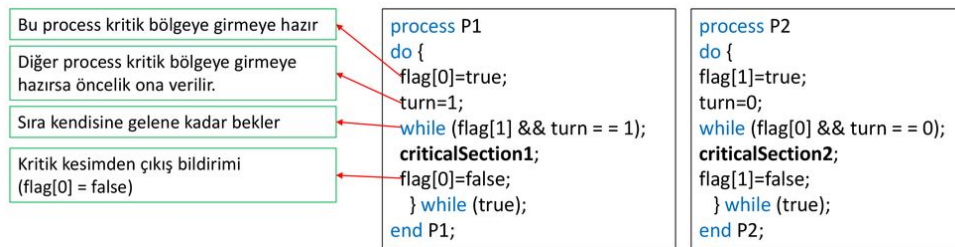
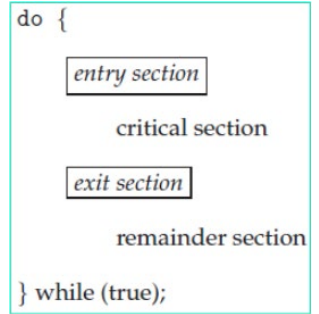
**Buffer Full**



**Buffer Empty**


    - **Race Condition:** İki veya daha fazla prosesin, paylaşımlı kullandıkları bir veri üzerinde yapmış oldukları okuma ve yazma işlemleri, **hangi processin ne zaman çalıştığına bağlı olarak, farklı bir son değere** sahip oluyorsa bu duruma yarış durumu denir. Örneğin, üretici-tüketici problemindeki yarış durumundan kurtulmak için **bir anda sadece bir prosesin counter değişkenini değiştireceği garanti edilmelidir.** Prosesler bir şekilde **senkronize olmalıdır.**
  - **Mesaj Geçişi:** Processler, paylaşılan değişkenlere ihtiyaç duymadan birbirleri ile haberleşebilirler. **Send** ve **Receive** sistem çağrısı ile çalışır. Her bir mesaj, sistem çağrısı ile gerçekleştirilir. Dolayısıyla **kernel sürekli devrededir.** Buna **rağmen avantajı, Race Condition durumu ile karşılaşılması ve Senkronizasyon problemi yaşanmamasıdır.**
    - **Mesaj Geçişinde Uygulama sorunları**
      - Bağlantılar (Links) nasıl kurulur?
      - Bir bağlantı ikiden fazla işlemle ilişkilendirilebilir mi?
      - Her bir iletişim işlemi çifti arasında kaç bağlantı olabilir?
      - Bir bağlantının kapasitesi nedir?
      - Bağlantının barındırabileceği mesajın boyutu sabit mi yoksa değişken midir?
      - Bir bağlantı tek yönlü mü yoksa çift yönlü müdür?
- **Mesaj Geçiş sistemi dizayn ederken dikkat edilmesi gerekenler**
  - **1) Naming:** Doğrudan veya Dolaylı.
    - **Doğrudan Haberleşme:** Processler açıkça birbirini isimlendirmelidir. **iki process arasında gerçekleşir.** Bu komutlarla mesajın hangi processse gönderileceği ve hangi processin mesajı alacağı açıkça belirtilmiştir. **send(P, message)** ve **receive(Q, message)**
    - **Dolaylı Haberleşme:** Mesaj gönderirken, alıcı direkt olarak belirtilmez. Bir **mail kutusu üzerinden iletişim** kurulur. Mail kutusunun bir ismi vardır. İletişim, yalnızca **processler aynı mail kutusunu paylaşıyorsa** kurulabilir. Gönderici mesajı mail kutusuna gönderir. Alıcı da mesajı mail kutusundan okur. **Üsttekinden farklı olarak, ikiden fazla process arasında da iletişim sağlanabilir ve bir process birden fazla mail kutusu ile iletişim kurabilir.** Bu komutlarla mesajın iletimi ve okunmasının A isimli mail kutusu üzerinden gerçekleştirileceği belirtilmiş olur. **send(A, message)** ve **receive(A, message)**
    - **Dolaylı haberleşme için yapılması gereken operasyonlar:**

- Kernel'in yeni bir posta kutusu (mesaj kuyruğu) oluşturması.
  - Posta kutusu aracılığıyla mesaj gönderiminin ve alımının gerçekleştirilmesi.
  - Posta kutusunun silinmesi.
- 2) Senkronizasyon:** Bloklamalı veya Bloklamasız.
  - Engellemeli (Blocking) Haberleşme:** Gönderen, mesaj alınana kadar; Alıcı da, bir mesaj mevcut olana kadar engellenir.
  - Engellemesiz (Non-Blocking) Haberleşme:** Gönderen, mesajı gönderir ve devam eder. Alıcı geçerli veya null bir mesaj alır.
- 3) Kapasite:** Bounded veya Unbounded.
  - Sıfır Kapasite:** İletişimde hiçbir ileti sıraya alınmaz. Gönderenin alıcıyı beklemesi gerekir (randevu-rendezvous).
  - Bounded capacity – Sınırlı kapasite:** Gönderici, bağlantı doluyorsa beklemelidir.
  - Unbounded capacity – Sınırsız kapasite:** Sonsuz uzunluk olduğu için, Gönderici asla beklemez.
- POSIX standardında Shared Memory oluşturmak için gerekli komutlar**
  - shm\_open:** Paylaşılabilecek bellek alanı segmentlerini açmak veya oluşturmak için kullanılır.  
`shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);`
  - shm\_unlink:** Paylaşılan bellek alanının kaldırılması için kullanılır.
  - ftruncate():** Paylaşılan bellek alanının (nesnenin) boyutunun belirlenmesi için kullanılır.  
`ftruncate(shm_fd, 4096);`
  - mmap():** Bir dosya işaretçisini paylaşılan bellek nesnesine eşlemek için kullanılır. Paylaşılan belleğe okuma ve yazma, mmap() tarafından döndürülen pointer (işaretçi) kullanılarak yapılır.
- Client-Server Sistemlerinde Haberleşme**
  - Soketler (Sockets):** İletişim için bir uç nokta olarak tanımlanır ve processlerin haberleşebilmesi için kullanılır. 161.25.19.8:1625 numaralı soket, 161.25.19.8 numaralı adrese sahip ana bilgisayar üzerindeki 1625 numaralı bağlantı noktasına başvurur. İletişim bir çift soket arasında gerçekleşir.
  - Uzaktan İşlem Çağırısı (Remote Procedure Calls):** Ağa bağlı sistemlerdeki işlemler arasında prosedür çağrılarını özetler. Prosedürler burada processlerin bir parçasıdır. Bir programın fonksiyonunun çağırılması gibidir.
- Kritik Bölüm:** Proseslerin ortaklaşa kullandıkları veri alanlarına erişip, veri okuma ve yazma işlemlerini yaptıkları program parçalarına kritik bölge/bölüm/kısım denir.
- Kritik Bölge Problemi:** Paylaşımlı veriye aynı anda ulaşmak her zaman probleme açıktır ve tutarsızlığa neden olabilir. Tutarlılığı sağlamanın yolu, birlikte çalışan proseslerin veya threadlerin kesin bir sırayla işlerini yapabilecekleri bir mekanizma oluşturmaktır. Kritik bölüm problemi (Critical section problem), bunu çözmek için protokol tasarlamaktır. Yandaki şekilde entry section giriş izni için kullanılır. Kritik bölümdeki iş bitirilince exit section ile haber verilir.
- Kritik bölge probleminin çözümü** üç şartı sağlamak zorundadır:
  - Mutual exclusion (Karşılıklı dışlama):** Bir process kritik bölümünü çalıştırıyorsa diğer process'lerin hiç birisi kritik bölümlerini çalıştıramazlar.
  - Progress (İlerleme):** Hiçbir proses kritik bölümde çalışmıyorsa ve bu bölüme girmek isteyen birtakım prosesler varsa, kritik bölüme girme isteği uzun bir süre ertelenemez. Yani prosesler birbirlerinin sürekli iş yapmalarını engellememelidir. Prosesler sıraya girmeli, hangisinin önce çalışacağı belirlenmelidir.
  - Bounded waiting (Sınırlı bekleme):** Bir proses beklerken diğer proseslerin kritik bölüme girme sayısının bir sınırı vardır. Starvation'a yol açılmaması gerekir. Producer çalıştıktan hemen sonra consumer çalışmak zorunda değildir.
- Kritik Bölge Problemi için Çözümler**
  - Peterson Çözümü:** Yazılım tabanlı bir çözümdür. 2 proses için tasarlanmıştır. Bu prosesler 2 veri nesnesini paylaşır ve bu şekilde senkronize olurlar. Eğer **turn = i** ise, kritik bölüme i process'i girecektir. Eğer **flag[i] = true** ise, i.process kritik bölüme girmeye hazırdır. i.process **flag[i] = true** ve **turn = i** olunca kritik bölüme girer.
    - int turn:** Kritik kısma giriş sırasının kimde olduğunu belirtir.
    - boolean flag[2]:** Bir prosesin kritik kısma girişe hazır olup olmadığını belirtir.



#### Peterson Çözümünün Doğruluğu

- İki process, **turn** değişkenini aynı anda değiştirse bile, son değer alınır ve o process kritik bölüme girer. (**Mutual Exclusion**)
- Kritik bölümü tamamlayan process kritik bölüme giriş isteğini iptal eder ve diğer process kritik bölüme girer. (**Progress**)
- Bir process kritik bölüme bir kez girdikten sonra sırayı diğerine aktarır. (**Bounded Waiting**)
- Senkronizasyon Donanımı:** Donanım tabanlı bir çözümdür. Bu çözümde temel mantık kritik bölgeyi kilitlemekten (lock) geçer.



- **Kesmeleri Devre Dışı Bırakmak:** Prosesler kritik bölgeye girdiklerinde tüm kesmeleri devre dışı bırakabilirler. Kritik bölge dışına çıktıklarında ise kesmeler tekrar devreye girer. Çok işlemcili sistemler için uygun bir çözüm değildir. Çok işlemcili sistemlerde, interrupt'ların disable/enable yapılması için tüm işlemcilere mesaj göndermek için zaman gereklidir (time consuming) ve sistem performansı düşer.
- **Kilit Değişkenleri Kullanmak:** Bir kilit değişkeni, paylaşılan ve 0/1 değerlerine sahip olabilen bir tam sayı değişkenidir. Bir kelime içeriğini test edip değiştirme veya iki kelime içeriğini yer değiştirme (swap) işlemlerini atomik olarak (yani herhangi bir kesintiye -interrupt- uğramadan) yapan özel donanım komutları vardır.
  - **Test And Set:** Aktarılan parametrenin orijinal değeri döndürülür ve yeni değeri true olarak ayarlanır.
  - **Compare and Swap:** Aktarılan parametrenin orijinal değeri döndürülür ve yeni değeri true olarak ayarlanır.
- **Mutex Locks:** Donanım tabanlı kritik bölüm çözümleri karmaşıktır ve yazılımcılar tarafından erişilemez. İşletim sistemi tasarımcıları, kritik bölüm problemi için çeşitli yazılım araçları geliştirmişlerdir. En basit yazılım aracı mutex lock aracıdır. Bir proses, kritik kısmına girmeden önce bir lock talep etmelidir. Kritik kısımdan ayrılınca lock değişkenini serbest bırakmalıdır.
  - **acquire() fonksiyonu:** Her proses kritik bölüme girmek ve kilitlemek için izin ister. Lock değişkeni elde edilir ve başka bir prosesin kritik bölüme girmesi engellenir. Bu bir sistem çağrısıdır.
  - **release() fonksiyonu:** Kritik bölümden çıktıktan sonra lock değişkeni serbest bırakılır.
  - Lock durumunun uygun olup olmadığına karar vermek için boolean değişken kullanılır.
  - acquire ve release çağırımları atomik gerçekleşir ve bu fonksiyonun arka planında donanım tabanlı çözümler kullanılır.
  - mutex lock mekanizması **spinlock** olarak da adlandırılır.
  - **Mutex kilitlerindeki temel sorun, Busy waiting (meşgul bekleme) döngüsüdür.** Bir proses kritik kısmında iken kritik kısımlarına girmek isteyen diğer prosesler acquire() fonksiyonu içinde döngüde sürekli işlem görürler. Başka bir iş yapmadan beklerler. Meşgul bekleme olduğundan bekleyen proses de işlemci zamanı harcar.
  - Bir proses kritik bölgeden çıktığında bekleyen birden fazla proses varsa açlık durumu oluşabilir.
- **Semaforlar:** Prosesleri senkronize etmenin diğer bir yoludur. Bir prosesin kaynağı kullanıp kullanmadığını sürekli olarak kontrol etmek, CPU zamanının harcanmasına ve performans düşüşüne neden olur. (Busy waiting) Bunu önlemek için Semafor adı verilen değişken tanımlanır. Semafor S değişkeni bir tam sayıdır ve wait() ve signal() gibi iki temel fonksiyona sahiptir. Bu fonksiyonlar mutex kilitlerindeki acquire ve release fonksiyonları gibi atomik olarak çalışırlar. **wait()** ile S'nin değeri azaltılır, **signal()** ile S'nin değeri artırılır. Kaynağı kullanmak isteyen her proses, semafor üzerinde wait() işlemi gerçekleştirir (sayaç azaltılır). Bir process kaynağı serbest bıraktığında ise signal() işlemi gerçekleştirir (sayaç artırılır). Semafor değeri sıfır olduğunda, tüm kaynaklar kullanılabilir durumdadır. Semaforlar yapı olarak iki çeşittir.
  - **Counting semaphore:** Tamsayı değeri sınırsızdır. Belirli bir sayıda kaynağa erişimin denetlenmesi için kullanılır.
  - **Binary semaphore:** Tamsayı değeri 0 veya 1 olur. Bu tip semaforlar, Mutex kilitleri gibi davranır. bu haliyle semafor kullanımı da busy waiting (meşgul bekleme) durumu oluşmasına ve CPU'nun boşa kullanılmasına neden olacaktır.

**Semafor ile busy waiting probleminin çözümü:** Prosesi CPU üzerinde meşgul bekletmek yerine, bloke edip, bekleme kuyruğuna almak gerekir. Bloke edilen proses, semaforla ilişkilendirilir, durumu "bekleme" konumuna getirilir ve semafor ile ilişkili bir bekleme kuyruğuna yazılır. Kernel tarafında bu tip işlemler için iki fonksiyon kullanılır.

-> **block():** Prosesin durumunu running den waiting'e çevirir. Prosesi semaforun bekleme (waiting) kuyruğuna yerleştirir.

-> **wakeup():** Bekleme kuyruğundan semaforla ilişkili olan 1 prosesi çıkarır ve hazır kuyruğuna koyar. Proses **wakeup()** fonksiyonu ile restart edilir. Bu prosesin durumunu (waiting -> ready) beklemekten hazırına geçirir.

Her semafor, bekleyen proses listesine sahiptir. Bir proses, bir semafor üzerinde beklemeli ise semaforun proses listesine eklenir. Signal operasyonu prosesi semafor listesinden çıkarır.

```
typedef struct {
    int value;
    struct process *list;
} semaphore;

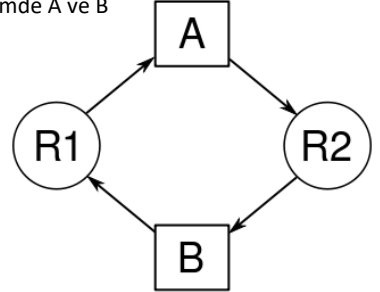
wait()
wait(semaphore *S) {
    S->value--;
    if (S->value < 0) {
        add this process to S->list;
        block();
    }
}

signal()
signal(semaphore *S) {
    S->value++;
    if (S->value <= 0) {
        remove a process P from S->list;
        wakeup(P);
    }
}
```



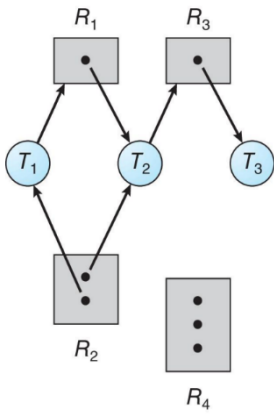
- **Monitörler:** İzleyici Programlardır. Monitörde aynı anda yalnızca bir proses etkin olabilir. Ancak bazı senkronizasyon şemalarını modelleyecek kadar güçlü değildir. Monitör programları paylaşılan nesneye ulaşmada meydana gelebilecek problemleri ortadan kaldırmaya yönelik geliştirilmiştir. **Semafor ile senkronizasyon çözümleri, zamana bağlı hatalar nedeni ile yetersiz olabilir.** Bir prosesin wait() ve signal() sırasını değiştirmesinden dolayı birden fazla proses aynı anda kritik kısımlarına girerler. Bir proses signal() ve wait() yer değiştirir ise deadlock oluşur. Bu hataların önüne geçmek için yüksek seviyeli programlama yapıları geliştirilmiştir. Bu hataların önüne geçmek için yüksek seviyeli programlama yapıları geliştirilmiştir.

- **Deadlock (Kilitlenme):** İki (veya daha fazla) **prosesin (veya thread'in)** karşılıklı olarak **aynı anda kaynak tutma ve kaynak talep etme** durumunda **her ikisinin de birbirini beklemesi** ve çalışmaya devam edememesi durumudur. Yandaki resimde A ve B prosesleri, sırasıyla R1 ve R2 kaynaklarına atanmıştır. Yani A prosesi R1'i kullanırken, B prosesi de R2'yi kullanmaktadır. Aynı zamanda, A prosesi R2 kaynağının serbest bırakılmasını beklerken, B prosesi de R1 kaynağının serbest bırakılmasını beklemektedir. Bu durumda kilitlenme meydana gelir. Kaynaklar CPU zamanı, bellek, I/O vb. olabilmektedir. Her proses **request -> use -> release** sırasıyla kaynakları kullanır. Yani önce kaynağı talep eder, ardından onu kullanır ve en sonunda serbest bırakır. Serbest bırakılan kaynak böylece diğer prosesler tarafından kullanılabilir.



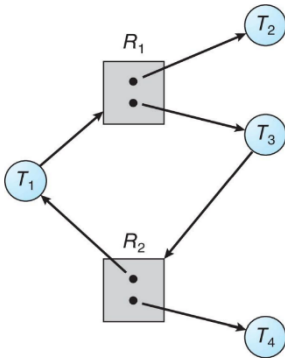
- **Kilitlenme oluşumu için dört şart** vardır. Deadlock oluşumu için, bu şartların **hepsinin sağlanması** gerekir.

- 1. **Mutual Exclusion:** Bir kaynağı aynı anda yalnızca bir proses kullanıyorsa,
- 2. **Hold and Wait:** En az bir kaynağı tutan proses, diğer prosesler tarafından tutulan en az bir kaynağı da elde etmek istiyorsa,
- 3. **No Preemption:** Proses kesintiye uğratılmayan veya sadece prosesin kendisi tarafından serbest bırakılan durumdaysa,
- 4. **Circular Wait:** Dairesel bir şekilde birbirini bekleyen bir yapı varsa deadlock oluşur.



- R1 kaynağının 1, R2'nin 2 ve R3 kaynağının 1 örneği vardır. R4 ise 3 örneğe sahiptir.
- T1, R2 kaynağının bir örneğini tutar. Yani R2 kaynağının bir örneği T1 thread'ine atanmıştır. (Assign)
- T1, aynı anda R1 kaynağının tek örneğini talep etme (Request) durumundadır.
- R1 kaynağının tek örneği T2 thread'ine atanmıştır. T2, R3'ün tek örneğini de talep eder durumdadır.
- R3'ün tek örneği, T3 thread'ine atanmıştır.
- Soldaki graph içerisinde herhangi bir DEADLOCK yoktur.
- Fakat T3, R2 kaynağını talep ederse DEADLOCK oluşur.

- **Graph; cycle (döngü) içermiyorsa**, kesinlikle deadlock yoktur diyebiliriz. Fakat cycle içeriyorsa, diğer deadlock şartlarının olup olmadığına bakmak gerekir. Örneğin, her kaynağın yalnızca 1 örneği varsa, deadlock oluşur.

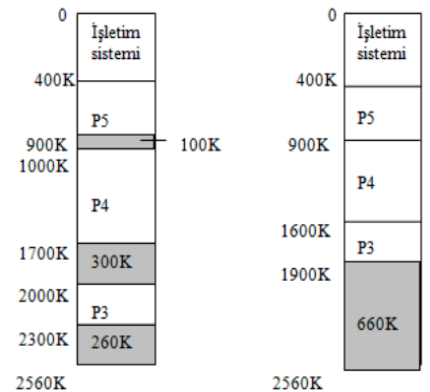
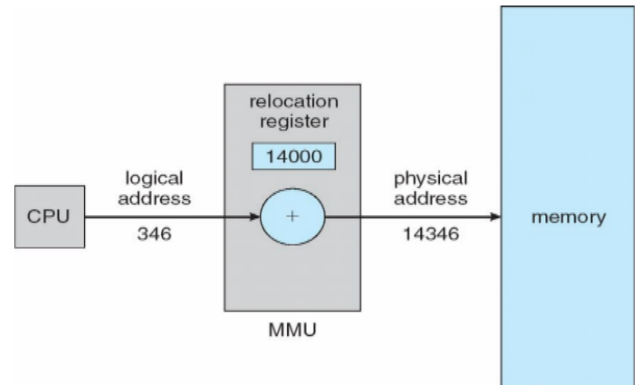
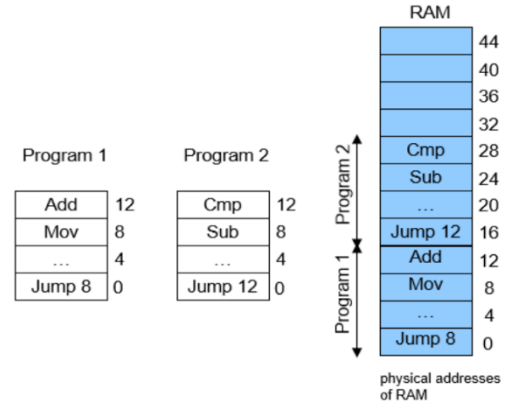


- Soldaki graph için konuşacak olursak;
- T4 sonlanır ve R2'nin ikinci kaynağı boşa çıkar. Yani serbest kalır.
- Böylece T3, R2'nin ikinci kaynağını alır ve sonlanır. R1'in ikinci kaynağı boşa çıkar.
- T2 herhangi bir bağımlılığı olmadığı için serbesttir ve istediği zaman çalışır.
- R1'in kaynakları boşa çıkınca, T1'in talebi karşılanır ve T1 de sonlanmış olur.
- Bu graph için DEADLOCK oluşumu yoktur.
- Fakat R1 ve R2 kaynaklarında, iki yerine bir örnek olsaydı, kesinlikle DEADLOCK oluşurdu.

- **Claim edge (Talep kenarı)  $P_i \rightarrow R_j$ :**  $P_i$  prosesinin  $R_j$  kaynağını talep edebileceğini (talep olasılığını) belirtir; **kesikli çizgi** ile gösterilir.
- Talep kenarı, bir proses bir kaynak istediğinde **uç isteğine (request edge)** dönüşür.
- Kaynak prosese tahsis edildiğinde, request edge bir **atama kenarına (assignment edge)** dönüşür.
- Bir process tarafından bir kaynak serbest bırakıldığında, atama kenarı bir talep kenarına geri döner.
- **Kilitlenmeleri Ele Alma Yöntemleri:** Üç tanedir.

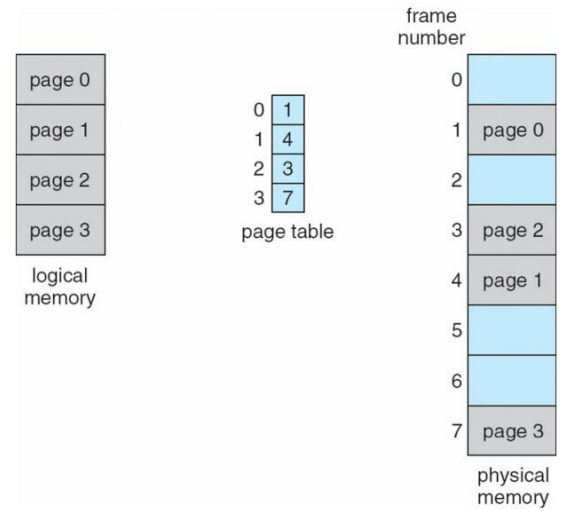
- 1. **Sistemin asla kilitlenme durumuna girmeyeceğinden emin olma:** İki şekilde sağlanır.
  - **Deadlock Prevention (Önleme):** Deadlock için gerekli dört koşuldaki birini geçersiz kılmak. **Döngüsel bekleme koşulunun geçersiz kılınması en yaygın olanıdır.**
  - **Deadlock Avoidance (Kaçınma):** Önceki istatistiklere göre varsayım ile hareket etmek. Bunun için sistemin bazı ek ön bilgilere sahip olmasını gerektirir. İki tür algoritması vardır.
    - **Bir kaynağın tek örneği varsa, Kaynak Tahsis Grafları** kullanılır.

- Bir kaynağın birden çok örneği varsa, **Banker Algoritması** kullanılır.
- 2. **Sistemin bir kilitlenme durumuna girmesine izin vermek ve ardından kurtarmak.**
- 3. **Sorunu görmezden gelmek ve sistemde asla kilitlenmeler oluşmadığını varsaymak.**
- Modern işletim sistemleri **kilitlenmeyi görmezden gelirler**. Çünkü çözmeye çalışmakla harcanan enerji, kazanımdan daha fazladır.
- **Deadlock'ın çözülmesi, farklı problemlerin de ortaya çıkmasına sebep olur**. Örneğin, Deadlock Prevention'da, kilitlenmeyi önlemek için **Mutual Exclusion** geçersiz kılınırsa, bu sefer de **Race Condition** durumu ortaya çıkabilir. Aynı yöntemde **Hold and Wait** geçersiz kılınırsa, bu sefer de **Starvation** durumu ortaya çıkabilir.
- **Banker Algoritması için Veri Yapıları**:  $n$  process,  $m$  kaynak sayısını ifade etmek üzere;
  - **Available**: Her kaynak türünden, kaç adet örneğin kullanılabilir olduğunu gösteren vektördür. (1xm)
  - **Max**: Her bir prosesin, her bir kaynak türünden en fazla ne kadar talep edebileceğini gösteren matristir. (nxm)
  - **Allocation**: Her bir prosesin, her bir kaynak türünden anlık olarak ne kadar kullandığını gösteren matristir. (nxm)
  - **Need**: Her bir prosesin, görevini tamamlamak için, her bir kaynak türünden ne kadar ihtiyacı olduğunu gösteren matristir. (nxm)
 Need matrisi şu formülle oluşturulur:  $Need[i,j] = Max[i,j] - Allocation[i,j]$
- **Bellek Yönetimine geçmeden önce hatırlanması gereken konular**:
  - Program çalıştırıldığı zaman proses olup, **main memory (RAM)**'e getirilir.
  - **CPU sadece Main Memory ve Register'lara erişebilir.**
  - CPU tarafından oluşturulan adresler **Mantıksal (Logical) Adres** olarak adlandırılır. **Mantıksal adres sıfırdan başlar**. Mantıksal adresin var olmasının bir sebebi, **fiziksel adresin yeterli olmama durumudur**. Bir diğer sebep ise, program içerisindeki adresin, fiziksel adresle çakışması ve çeşitli problemler ortaya çıkarmasıdır. Sağdaki şekilde bu durum gösterilmiştir.
  - Bellek ünitelerinin gördüğü adreslere ise **fiziksel adres** denir. Program tarafından üretilen mantıksal adreslerin fiziksel adreslere çevrilmesi gerekir.
  - Program, bellek yönetimine göre sürekli, **disk ile ana bellek arasında** taşınır.
  - **Bellek yönetim ünitesi (memory-management unit, MMU)** mantıksal adresten fiziksel adrese dönüşüm yapan donanım aygıtıdır.
- **Adres Bağlama (Address Binding)**: Üç yerde gerçekleşir.
  - **Compile time**: Bellek konumu önceden biliniyorsa, **absolute code** üretilebilir; başlangıç konumu değişirse kod yeniden derlenmelidir.
  - **Load time**: Derleme sırasında bellek konumu bilinmiyorsa **relocatable code** oluşturmaktır.
  - **Execution time**: Proses, yürütme sırasında bir bellek bölümünden diğerine taşınabiliyorsa, bağlama çalışma zamanına kadar ertelenir. Adresleri eşleştirebilmek için donanım desteğine ihtiyaç vardır. (base ve limit registerlar gibi.)
- **Base ve Limit register'lar PCB (Proses Control Block)'ta tutulur.**
- CPU, **kullanıcı modunda oluşturulan her bellek erişimini kontrol eder**. Base ve Limit arasında olduğundan emin olur.
- **Yer Değiştirme (Swapping)**: Prosesin disk ve bellek arasında sürekli yer değiştirmesinin sebebi, bellek alanının yetersiz olmasıdır.
  - Swapping zamanla devre dışı bırakılabilir.
  - Eğer kullanılan bellek alanı belli bir eşik değerinin üzerinde ise başlatılabilir.
  - Bellek istekleri belli bir eşik değerinin altında kalırsa tekrar devre dışı kalır.
- **Backing store**: Tüm kullanıcılar için **tüm bellek görüntülerinin kopyalarını barındıracak kadar büyük hızlı disk**; bu hafıza görüntülerine doğrudan erişim sağlamalıdır.
- **Dinamik Depolama Tahsis Problemi (Dynamic Storage-Allocation Problem)**
  - **First Fit**: Yeterince büyük olarak **bulunan ilk alana** veriler yerleştirilir.
  - **Best Fit**: Bütün boş alanlara bakar ve eldeki veri **en az boşluk kalacak şekilde** yerleştirilir.
  - **Worst Fit**: Bulunabilen en büyük alana yerleştirme işlemi yapılır.
- **Parçalanma (Fragmentation)**
  - **External Fragmentation**: Bir isteği karşılamak için yeterli bellek alanı vardır. Fakat bu alanlar bitişik değildir. Bellekte yerleşik durumda olan proseslerin arasında boşlukların olması gibi.
  - **Internal Fragmentation**: Ayrılan bellek, istenen bellekten biraz daha büyük olabilir; bu boyut farkı, bir bölümün dahili hafızasıdır, ancak kullanılmamaktadır.
- **Sıkıştırma (Compaction-Defragmentation)**: Sıkıştırma yapılabilecek en kolay yol, bütün **prosesleri bir yönde kaydırıp, boş bellek alanlarını birleştirmek** tir. **Sıkıştırma ile External ve Internal fragmentation problemleri çözülmüş olur**.
- **Sayfalı Bellek Yönetimi (Paging)**: Bir prosesin fiziksel adres alanı bitişik olmayabilir. Proses kullanılabilir olduğunda, fiziksel bellek alanı tahsis edilir.

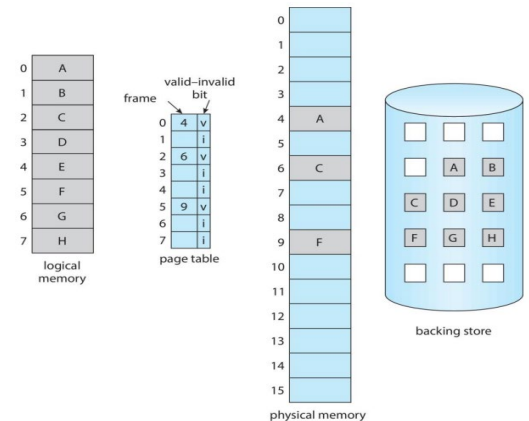


- **Paging ile sağlanan faydalar:** 1) **External fragmentation önlenir.** 2) **Değişken boyutlu bellek yığınları sorunu ortadan kalkar.**
- **Fiziksel bellek** belirli bir büyüklükteki bloklara ayrılmıştır, bu bloklara **çerçeve (frame)** adı verilir. (512B-16MB)
- **Mantıksal bellek** ise eşit büyüklükteki **sayfa (page)** adı verilen bloklara ayrılmıştır.

- Mantıksal adresleri fiziksel adreslere çevirmek için bir **page table (sayfa tablosu)** oluşturulur. **Ana bellekte tutulur.**
- **Internal fragmentation, Paging yapılmasına rağmen devam etmektedir.**
- N page'lik bir programı çalıştırmak için, N adet boş frame bulmanız ve programı yüklemeniz gerekir.



- CPU tarafından üretilen adresler ikiye ayrılır:
  - **Sayfa Numarası (p):** Fiziksel bellekteki her bir frame'in taban adresini tutan sayfa tablosundaki göstergedir.
  - **Sayfa Offseti (d):** Taban adresi ile birleştirilerek, fiziksel bellekte frame'in içerisindeki yerin belirlenmesinde kullanılır.
- **Page-table base register (PTBR)** page tablosunu işaret eder.
- **Page-table length register (PTLR)** page tablosunun boyutunu işaret eder.
- Hem page tablosu, hem de veri/komut için **iki defa RAM erişimi** gerekir. Bu istenmeyen bir problemdir. **Bunu önleyen yapı, TLB (translation look-aside buffer)**'dir. **Page tablosunun ön belleği**dir.
- **Effective Access Time (EAT):** TLB yapısı ile daha az RAM erişimi olduğunda, etkileşim süresinde de azalma meydana gelir. %80'lik bir isabet oranı, istenen sayfa numarasının %80 TLB'de bulunması anlamına gelir. Bir kez belleğe erişimin 10 ns olduğunu kabul edersek, iki bellek erişimi ile (TLB olmadan) **20ns** harcamak gerekir. İsbet oranı **%80** olursa, yani erişimlerin **%80'i 10 ns** ve **%20'si 20ns** zaman alırsa;  $EAT = 0.8 * 10 + 0.2 * 20 = 12ns$  bulunur.
- **Bellek Koruması (Memory Protection):** Yalnızca okuma (read-only) veya okuma-yazma (read-write) erişimine izin verilip verilmediğini belirtmek için koruma bitini her frame ile ilişkilendirerek uygulanan bir yaklaşımdır.
  - **Valid-invalid:** Page tablosundaki her girişe eklenen geçerli-geçersiz biti.
    - **Valid:** İlişkili sayfanın **prosesin mantıksal adres alanında (RAM'de) olduğunu** ve dolayısıyla yasal bir sayfa olduğunu belirtir.
    - **Invalid:** Sayfanın **prosesin mantıksal adres alanında olmadığını** gösterir.
- **Paylaşılan Sayfalar (Shared Pages):** Proseslerin, sayfa tablolarındaki tekrarlanan (aynı olan) değerlerinin, ortak bir veri alanında depolanmasıdır. Böylece **bellek optimizasyonu** sağlanmış olur.
- **Virtual Memory (Sanal Bellek):** Kullanıcı **mantıksal belleğinin fiziksel bellekten ayrılması**dır.
- **Virtual Memory'nin Amacı:** Kodun yürütülebilmesi için bellekte olması gerekir, ancak **programın tamamı nadiren kullanılır**. O yüzden **programın tüm kodu aynı anda gerekli değildir**. Program parçalı bir yapıda yüklenebilir. Yani **programın sadece yürütülecek olan kısmı belleğe getirilir. Diğer kısımlar ihtiyaç duyuldukça getirilir**. Bu şekilde fiziksel belleğin sınırları ile kısıtlanmamış olur. Ayrıca programlar daha az bellek ayırır ve aynı anda daha fazla program çalışabilir. Swap in ve Swap out işlemleri daha az olduğundan daha az I/O ihtiyacı olur ve programlar daha hızlı çalışır. **Swap Out:** Bellekten Diske veri taşıma işlemidir. **Swap In:** Diskten Belleğe veri taşıma işlemidir.
- **Virtual Memory iki şekilde uygulanır:** **Demand Paging (Sayfa Talebi)** ve **Demand Segmentation (Segmentasyon Talebi)**.
- **Demand Paging Faydaları:** 1) I/O ihtiyacı azalır, gereksiz I/O ihtiyacı olmaz. 2) Daha az bellek alanına ihtiyaç duyulur. 3) Daha hızlı cevap verilir. 4) Daha fazla kullanıcıya hizmet verilebilir.
- **Lazy Swapper:** Sayfaya **ihtiyaç oluncaya kadar, o sayfanın belleğe getirilmemesi**dir.
- **Page Fault:** MMU Adres Çevirisi sırasında, eğer sayfa tablosu girişindeki valid-invalid biti **i** ise, sayfa hatası (page fault) meydana gelir. Yani **aranan bir sayfanın fiziksel bellekte olmama durumu**dur. **Fiziksel bellek tamamen boş iken de** meydana gelir.
- **Copy-on-Write (COW):** Prosesler oluşturulurken, **parent ve child proseslerin bellekte aynı sayfayı paylaşmalarına izin verilmesi**dir. Proseslerden biri, paylaşılan sayfayı değiştirirse, ancak o zaman ilgili sayfanın kopyası oluşturulur. Bu şekilde, proseslerin oluşturulması, daha hızlı ve daha verimli bir hale gelmiş olur.
- **Page Replacement (Sayfa Değiştirme):** İhtiyaç halinde boş frame bulunamazsa, **victim frame** (kurban çerçeve) seçmek için bir sayfa değiştirme algoritması (FIFO, Optimal, LRU) kullanılmasıdır.
- **Page Replacement algoritmalarının amacı:** **En az page fault'u** sağlamaktır. Frame sayısı arttıkça, Page Fault hep azalmaz. Belirli bir yere kadar azalır, daha sonra sabitleşir.
- **Belady Anomalisi:** Page Replacement için FIFO algoritması kullanıldığında, yeni frame'ler eklemek page fault'ın bazı anlarda artmasına da sebep olabilir. Buna **Belady Anomalisi** denir.



- **FIFO (First in First Out) algoritması:** Victim frame seçme işlemi ilk gelen frame'e göre yapılır. En önce veya en eski gelen frame silinir.
- **Optimal algoritması:** Referans stringinde ileri bakılır ve hangisi daha uzun süre kullanılmayacaksa, o victim frame olarak seçilir. En az page fault durumu için bu algoritma idealdir. Fakat gerçekte uygulanan bir algoritma değildir. Sadece ölçü alınabilmesi için kullanılır.
- **LRU (Least Recently Used) algoritması:** Geçmişe göre bakılıp, en uzun süre kullanılmayan frame victim frame olarak seçilir. FIFO'dan daha iyi, Optimal'den daha kötü çalışır. Genel olarak iyi bir algoritma sayılır ve sıkça kullanılır. Stack ve Counter implementasyonları şeklinde ifade edilen iki türü vardır.
- **Thrashing:** Bir prosesin sürekli swap in-swap out durumuna düşmesidir. Bu sebeple çok fazla page fault durumu meydana gelir. Kelime anlamı olarak da, boşa çalışmak anlamına gelir. Hemen üstteki şekilden de anlaşılacağı gibi, belirli bir andan sonra, multiprogramming yapsak da, CPU Utilization artmaz. Tam tersine azalır. Bu durumdan kurtulmak için, multiprogramming derecesini düşürmemiz gerekir.
- ...

