Bellek Yönetimi - 1

Bellek Yönetim Birimi

- Ana bellek kritik kaynaklardan biridir
- Yürütülebilmesi için, bir programın disk alanından ana belleğe taşınması ve bir proses ile özdeştirilmesi gerekir
- MİB sadece saklayıcılara ve ana belleğe doğrudan erişebilir
- Bellek Birimi sadece
  - Okuma isteği + ilgili adres
  - Yazma isteği + ilgili adres ve veriyi görür

Bellek Yönetim Biriminin Temel Amaçları

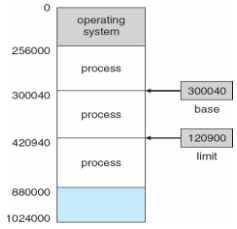
- yeniden yerleştirme (relocation)
  - prosese ayrılan fiziksel bellek her çalışmada farklı olabilir
  - fiziksel ve mutlak adresler kullanılmaz
- koruma (protection)
  - prosesler birbirlerinin bellek bölgelerine erişemez
- paylaşma (sharing)
  - prosesler bazı bellek alanlarını paylaşmak isteyebilir
    - kod / veri paylaşımı

Bellek Yönetim Birimi

- Çekirdekte yer alan Bellek Yönetim Birimi (memory management unit –mmu), belleğin
  - Etkin kullanımı, ve
  - Paylaşımından sorumludur
- Bellek koruması: doğru çalışma için gereklidir
  - İşletim sistemini proseslerden korur
  - Bir prosesi diğer proseslerden korur

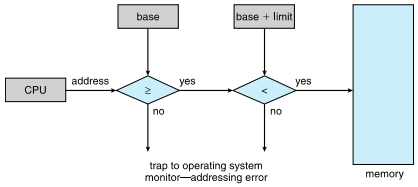
Bellek Koruması

- Bellek koruması nasıl gerçekleştirilir?
- Basit bir çözüm: donanım ile
  - Taban (base) saklayıcısı ve sınır (limit) saklayıcısı prosese ait sanal adres uzayını tanımlar.



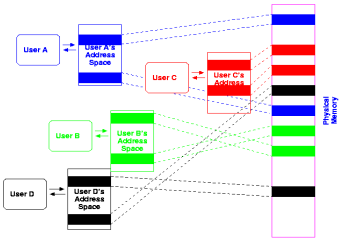
Bellek Koruması

- MİB, kullanıcı düzeyinde oluşan her bellek erişim isteğinin o kullanıcıya ait taban ve sınır adresler arasında olduğunu kontrol eder.



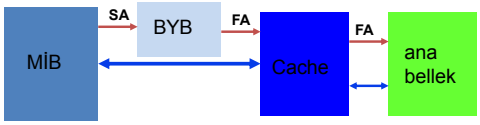
Bellek Yönetim Birimi

- MİB sanal (virtual) adresler üretir
- Bellek Yönetim Birimi sanal adresleri fiziksel adreslere dönüştürür



Bellek Yönetim Birim

- Bellek Yönetim Birim (BYB)
  SANAL ADRES(SA) → FİZİKSEL ADRES(FA)
  adres dönüşümünden sorumludur



Adres Bağlanması (binding)

- Bir program yürütülmeden önce bir çok aşamadan geçer: derlenme, bağlanma, yüklenme aşamaları
- Her aşamada, adresler farklı temsil edilirler
  - Kaynak kodda adresler simgeseldir, değişken adları
  - Derleyici simgesel adresleri yeri değişebilir (relocatable) adreslere dönüştürür
    - Bu modülün başından itibaren 24 sekizli ötesi
  - Bir bağlayıcı/yükleyici yeri değişebilir adresleri fiziksel adreslere dönüştürür
    - 85024 fiziksel adresi

Adres Bağlanması

- Veri ve komut adreslerinin bellek adreslerine bağlanması üç farklı aşamada gerçekleşebilir:
  1. Derlenme aşaması: prosesin bellekte nereye yerleşeceği biliniyor ise, mutlak adresler içeren kod üretilebilir. Yer değişirse yeniden derlenmeli
  2. Yüklenme aşaması: derleme sırasında bellek adresi bilinmiyor ise derleyici yer değiştirebilir kod (relocatable) üretir. Adresler modül başlangıcına göreceli olarak oluşturulur.

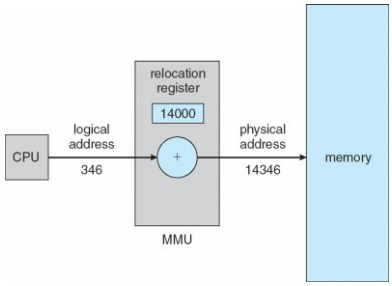
Adres Bağlanması

3. Yürütme aşaması: eğer proses yürütme sırasında bir bellek alanından başkasına taşınabilecekse, bağlanma adımı yürütme zamanına kadar ertelenir.
   - Esnek bellek kullanımı sağlar
   - Bu tür çalışma için özel donanım gereklidir
   - Çoğu genel amaçlı işletim sistemi tarafından uygulanır

Bellek Yönetim Birimi

- Bellek YönetimBirimi yürütme sırasında sanal adreslerin fiziksel adreslere dönüşümünden sorumludur
  - Taban saklayıcısı mevcuttur (relocation register)
  - Mİ Biriminin prosesi yürütürken ürettiği her sanal adres Taban Saklayıcısının içeriği ile toplanarak fiziksel bir adres oluşturulur ve belleğe gönderilir
- Kullanıcı programı sadece lojik adresler üretir, fiziksel adreslerden haberdar değildir
  - Bellekteki bir alana referans verildiğinde, yürütme aşamasında adres bağlanması gerçekleşir ve lojik adres fiziksel adrese bağlanır.

Taban Saklayıcısı ile Adres Bağlanması



Bağlanma (Link) Süreci

- örnek program: kullanıcı ve sistem (derleyici) tarafında tanımlı veri yapıları ve prosedürleri olan bir program
- programın ayrı ayrı derlenen parçaları: modül
  - her biri 0 adresine göre kodlanmış
- modüller birbirlerinin veri ve prosedürlerine erişebilir-dış referans olarak adlandırılır
  - sadece veri ve prosedür adlarını bilir (taban adresi belli değil)
- bağlanma aşaması: program belleğe yüklenirken adres bilgisinin sağlanması gerek : bağlanma aşaması

Bağlanma (Link) Süreci

- bağlanma 3 şekilde:
  - tüm prosedürler derleme öncesi hazırlanır ve derleyiciye birlikte sunulur; bağlanmayı derleyici gerçekleştirir
    - yüksek düzeyli diller destekler
    - derleyicinin ürettiği kod doğrudan belleğe yüklenir veya sistem rutinleri ile bir bağlanma aşaması geçirir
  - önceden derlenmiş olan modüller belleğe yüklenirken bağlanır
    - bağlanma işlemi, derleme ve yüklemeden ayrı yürütülür
    - "linking loader" iki adımı birleştirir
    - 0 adresine göre kodlanmış adresler düzenlenir (relocation)
  - bağlanma yürütme anına kadar ertelenir (late-binding)
    - rutinler çağrıldıkça bağlanır

Bağlanma (Link) Süreci

- bağlanma yöntemleri donanıma bağlıdır
- modüller 0 adresine göre göreceli derlenir (relocatable module)
- modül belleğe yüklenirken mutlak başlangıç adresi taban saklayıcısına yüklenir
  - Modül içindeki diğer referanslar başlangıçtan uzaklık değerinin taban saklayıcısı içeriğiyle toplanarak hesaplanır

Bağlayıcı Yükleyici (Linking Loader)

- tek geçişli -- yükleme sırasında bağlama da gerçekleşir
- her modül şu bilgileri yükleyiciye bildirir:
  - modül içi tanımlı ve diğer modüllerin çağırabildiği isimler
  - dışarıda tanımlı olan ama modül içinde kullanılan isimler
  - bağlanmış modül içinde diğerleri ile bağlandıktan sonra yeniden hesaplanacak adresler

Bağlayıcı Yükleyici (Linking Loader)

- yükleyici dışarıda tanımlı isimler için tablo oluşturur
  1. Modül içinde tanımlanmış olup dışarıdan çağrılabilecek isimler
  2. Dışarıda tanımlı olup modül içinde kullanılacak isimler
  3. Bağlanmış modul içindeki konumuna göre yeniden hesaplanması gereken adresler – kaymalar olabilir
     - isim önceden yüklenmiş bir modülde ise tabloda isme karşılık bağlanmış modülün başına göre göreceli adresi
     - isim yüklenmemiş bir modülde ise adres belli olunca hesaplanacaklar listesine eklenir
- modüller belleğe birbirini izler şekilde yüklenir
  - adres referansları önceki modüle göre ötelenmeli

İki Aşamalı Yöntem

- bağlanma ve yüklenme ayrı adımlar
- önce bağlanma
  - sonucunda bağlı tek bir modül (load module)
  - tüm bağlanma işlemleri yapılmış
- yükleyici ayrı çalışır

Yöntemlerin Karşılaştırılması

- ayrı iki adım olmasının avantajları
  - programın her koşturulmasında bağlanma gerekmez
  - sadece yükleyici bellekte daha az yer kaplar
- problem: bağlanma sonrası modüller dış ortama yazılır; yükleyici okumak zorunda $\Rightarrow$ G/Ç için zaman kaybı
- sık değişen veya az koşturulan programlar için tek aşamalı yöntem daha iyi çözüm

Dinamik Yükleme

- Yükleme (Loading): yeterli bellek alanı ayrılıp, yürütülecek programın ana belleğe taşınması
- Dinamik Yükleme : Yükleme aşaması yürütme anına kadar ertelenir, modüller/prosedürler çağrılana kadar diskten ana belleğe yüklenmezler
- Kullanmayan modüller bellekte yer kaplamadığı için bellek alanı kullanımı çok daha verimli olur
- Tüm modüller bellekte yer değiştirebilir şekilde tutulur

Dinamik Bağlanma (linking)

- Statik Bağlanma – yükleyici, sistem kütüphanelerini ve program kodunu bir ikili program modülü olarak birleştirir
- Dinamik Bağlanma - bağlanma aşaması yürütme zamanına kadar ertelenir
- Genellikle sistem kütüphaneleri veya programlama dillerine ait kütüphaneler için uygulanır
- Bu yöntem uygulanmayacak olursa, her program için, çağırılan kütüphanenin bir kopyasının belleğe yüklenmesi gerekir.
- Bu yöntemle sadece tek kopya yüklenir ve bellek etkin bir şekilde kullanılır.

Dinamik Bağlanma

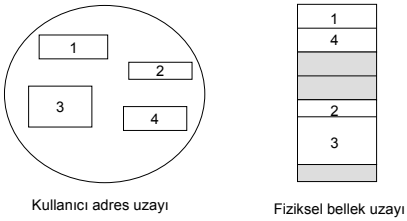
- Dinamik bağlı Kütüphaneler (Dynamically linked libraries:dll)
  Kullanıcı programlarına yürütülme aşamasında bağlanan kütüphanelerdir
- Dinamik bağlanmada , her kütüphane fonksiyonu referansı için koda bir koçan (stub) eklenir
- Koçan küçük bir kod parçasından oluşur:
  - Bellekte yer alan kütüphane fonksiyonuna nasıl erişileceğini, veya
  - Bellekte yer almıyor ise, kütüphanenin belleğe nasıl yükleneceğini gösterir
- Koçan fonksiyonun adresini edinir ve fonksiyonu yürütür
  - Böylece, aynı fonksiyona bir daha referans verilirse, dinamik bağlanma işlemine gerek duyulmadan, fonksiyon doğrudan yürütülür.
  - Bellekte kütüphanenin bir kopyası tutulur ve tüm prosesler aynı kopyayı yürütürler.

Segmanlama

- programlar “segman” adı verilen mantıksal bölümlerden oluşur – data, kod, yığın segmanları gibi.
- segmanlama programın adreslemesinin lojik yapısını yansıtır
- program segmanlara bölünür
  - segmanlar farklı boylarda olabilir
  - örneğin segman veri alanı veya prosedür olabilir

adres = segman başl. adresi + segman içi konum

Segmanlama



Kullanıcı adres uzayı

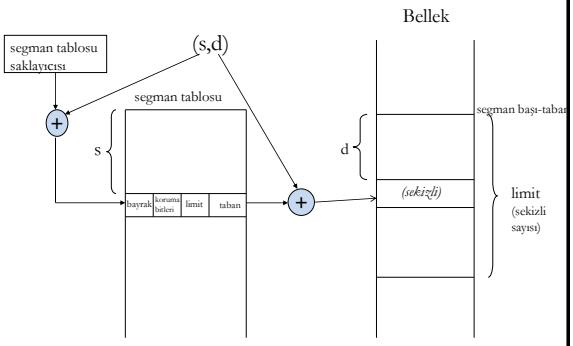
Fiziksel bellek uzayı

- Programın segmanları bellekte ayrı yerlerde bulunabilir, ayrıca diskte de yer alabilir; gerekince yüklenir.
- Adres hesabı özel donanım gerektirir

Segmanlama

- adres: (s,d)
  - s: segman adı
  - d: kayıklık
- her proses için ayrı bir segman tablosu oluşturulur
  - segman bellekte bayrağı
  - segmanın taban adresi
  - segmanın uzunluğu (limit)
  - erişim hakları bitleri
- bir saklayıcı segman tablosuna bir işaretçi tutar

Segmanlama



Segmanlama

- adres hesabı öncesi *bayrak* kontrol edilir
  - bellekte yoksa segman hatası
    - kesme oluşur
  - segman belleğe yüklenir
    - yer yoksa bir segman bellekten atılır
      - segman boyları eşit olmayabilir → bellekte parçalanma
  - segman tablosu saklayıcısı koşan prosesin segman tablosunu gösterir
    - segman tablosunun bellekteki başlangıç adresini saklar

Segmanlama

- segmanlama faydaları
  - bağlanma: modüller segmanlar olarak yazılır
  - yüklenme segman tablosu doldurulmasıyla olur
  - paylaşılabilir kodların paylaşımı daha kolay
    - segman bilgileri biren fazla tabloya konur
  - adresleme donanım ile yapılıyor ama bellek ⇔ disk akışı düzenlemeli

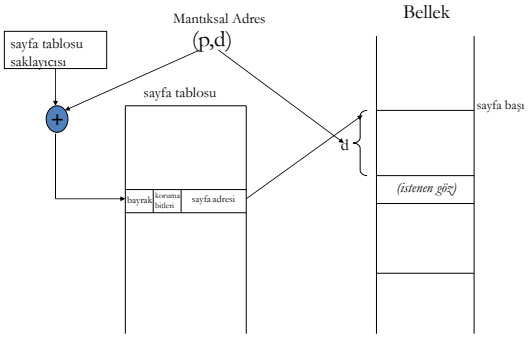
Sayfalama (Paging)

- bellek sabit uzunluklu fiziksel bloklara ayrılır
  - sayfa çerçevesi
- program ve veri alanları da sabit uzunluklu sayfalara ayrılır
  - sayfa
- her çerçeveye bir sayfa yüklenir
- adres: (p,d)
  - p: sayfa adı
  - d: sayfa içindeki kayıklık

Sayfalama

- her prosesin sayfalarına ilişkin bilgiler kendine ait sayfa tablosunda tutulur
- sayfa tablosundaki bilgiler:
  - sayfa bellekte bayrağı
  - sayfanın yeri (bellek/ikincil bellek adresi)
  - erişim hakları için koruma bitleri
- sayfa tablosu saklayıcısı
  - koşan prosesin sayfa tablosuna işaret eder

Sayfalama



Sayfalama

- adres hesabı öncesi *bayrak* kontrol edilir
  - bellekte yoksa: sayfa hatası kesmesi (page fault)
  - sayfa ikincil saklama alanından ana belleğe taşınır
- koruma bitleri kontrol edilir
- boş çerçeveler bilgisini işletim sistemi tutar
- ana bellek ⇔ ikincil bellek sayfa aktarımı = **sayfa trafiği**

Sayfalama

- Sayfalamanın yararları:
  - segmantasyonun sağladığı yararlar aynen var
  - adres hesabı donanım ile (hızlı) ve kullanıcıya saydam
  - bellek ayırma daha basit (sayfa boyu sabit)
- Sorunlar
  - sayfa boyu programın mantıksal biriminin tamamını içerecek büyüklükte olmayabilir
  - parçalanma (fragmentation) : bellekte kullanılmayan alanlar oluşur

Sayfalama

- dış parçalanma (external fragmentation)
  - bloklar arasında boşluklar oluşması
- iç parçalanma (internal fragmentation)
  - blok içinde boşluk oluşması
- sayfalama varsa dış parçalanma oluşamaz

Sayfalama

- sayfa boyu belirlemede dikkat edilecekler:
  - sayfa trafiği
  - iç parçalanma
- büyük sayfa boyu
  - ana bellek ⇔ ikincil bellek aktarımlarında bir büyük sayfa aktarmak daha kolay
  - proses daha az sayfadan oluşur ⇒ sayfa trafiği azalır
  - iç parçalanma artar
- küçük sayfa boyu
  - sayfa trafiği artar
  - iç parçalanma azalır

Sonuç: iç parçalanma ve sayfa trafiği maliyetleri arası denge kurulmalı

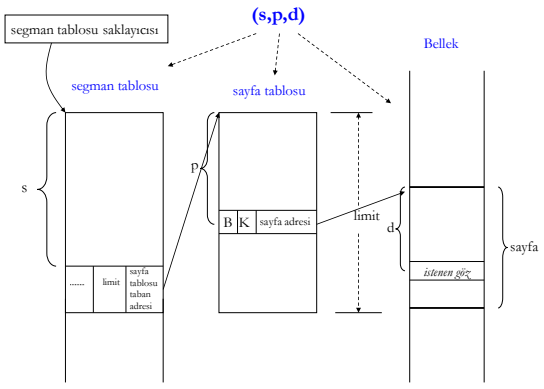
Sayfalama

- Örnek (iç parçalanma): prosesin boyu 1545 sözcük
  - sayfa boyu 1500 sözcük ise:

  - sayfa boyu 500 sözcük ise:

Sayfalamalı Segmanlama

- segmanlar sayfalara bölünür
- her segmanın ayrı bir sayfa tablosu vardır
- adres: (s,p,d)
  - s: segman numarası
  - p: segman içindeki sayfa tablosuna erişim için bilgi
  - d: sayfa içi kayıklık



Sayfalamalı Segmanlama

- adres hesabı üç aşamalı
- donanım ile gerçeklense bile uzun sürer
  - asosyatif saklayıcılar kullanılır ⇒ **TLB**

Sayfalamalı Segmanlama

- hem sayfalamanın hem de segmantasyonun faydaları aynen var
- segmanlama olduğundan paylaşım ve bağlanma daha kolay
- sayfalama olduğundan bellek ayırma daha kolay
- dış parçalanma yok
- TLB kullanımı ile adres hesabı kabul edilebilir sürelere gelir