1906003052015

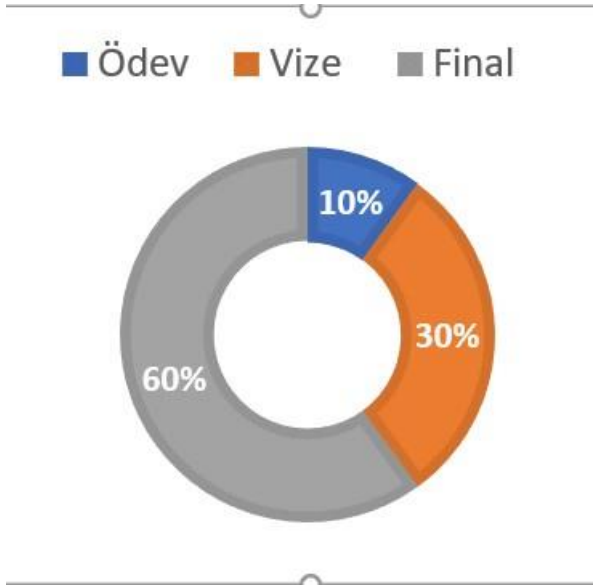
İşletim Sistemleri

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

Giriş

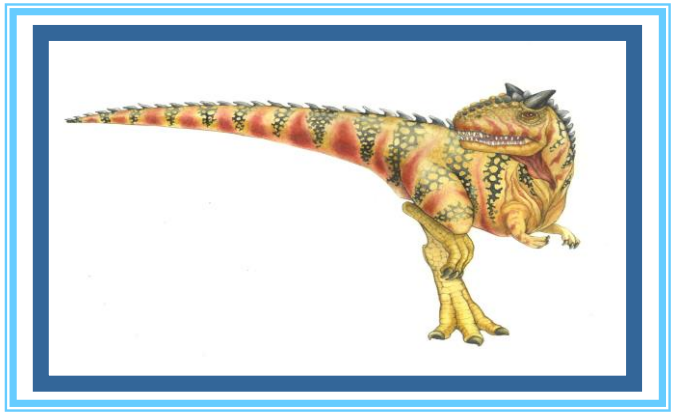
Ders Günü ve Saati:
Çarşamba: 13:00-16:00

- Uygulama Unix (Linux) İşletim sistemi
- Devam zorunluluğu %70
- Uygulamalar C programlama dili üzerinde gerçekleştirilecektir. Öğrencilerden programlama bilgisi beklenmektedir.



HAFTA	KONULAR
Hafta 1	: İşletim sistemlerine giriş, İşletim sistemi stratejileri
Hafta 2	: Sistem çağrıları
Hafta 3	: Görev, görev yönetimi
Hafta 4	: İplikler
Hafta 5	: İş sıralama algoritmaları
Hafta 6	: Görevler arası iletişim ve senkronizasyon
Hafta 7	: Semaforlar, Monitörler ve uygulamaları
Hafta 8	: Vize
Hafta 9	: Kritik Bölge Problemleri
Hafta 10	: Kilitlenme Problemleri
Hafta 11	: Bellek Yönetimi
Hafta 12	: Sayfalama, Segmentasyon
Hafta 13	: Sanal Bellek
Hafta 14	: Dosya sistemi, erişim ve koruma mekanizmaları, Disk planlaması ve Yönetimi
Hafta 15	: Final

Chapter 8: Main Memory



Chapter 8: Memory Management

- Background
- Swapping
- Contiguous Memory Allocation
- Segmentation
- Paging
- Structure of the Page Table
- Example: The Intel 32 and 64-bit Architectures
- Example: ARM Architecture

Bellek Bölümleme (Memory Partitioning)

Ana bellek, birden fazla program arasında paylaştırılmalı ve bölünmelidir.

Bunu yapabilmek için kullanılan yöntemler:

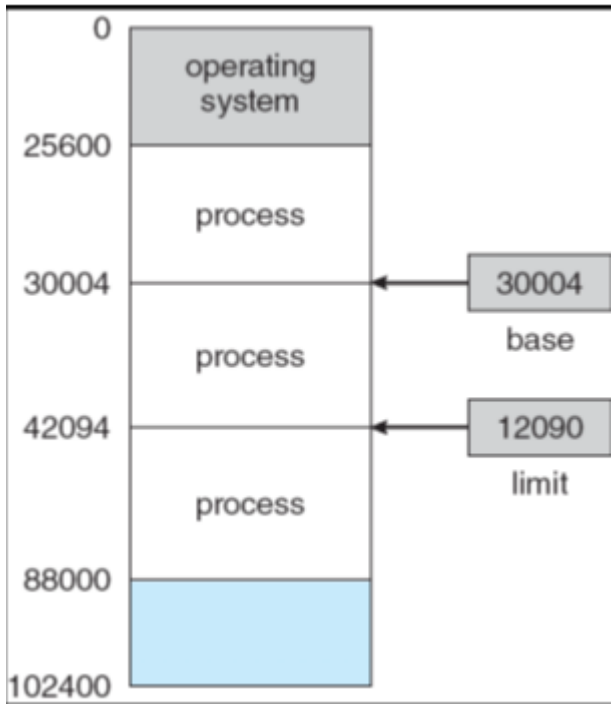
- Bölümleme (Partitioning)
  - Değişmez Bölümlü Belek Yönetimi (**Contiguous Allocation)**
  - Dinamik Bölümlü Bellek Yönetimi (**Dynamic Storage-Allocation Problem)**
- Sayfalı Bellek Yönetimi (**Paging)**
- Kesimli Bellek Yönetimi

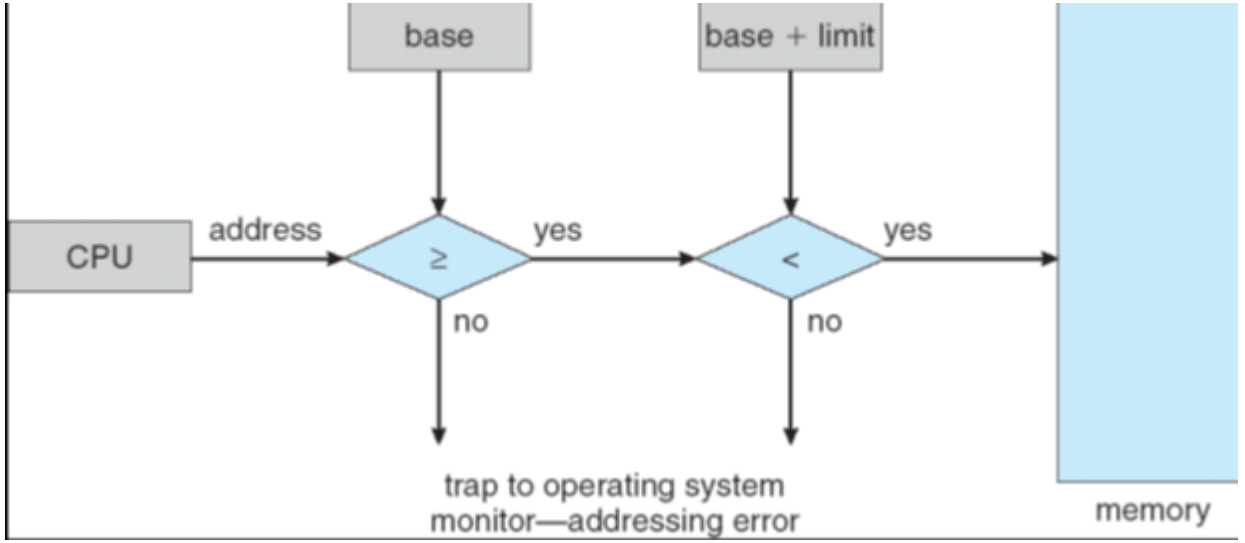
Değişmez Bölümlü Bellek Yönetimi

1. **Tek Bölüm Tahsis (Single Partition Allocation)**

   Koruma için Relocation –Register yaklaşımı kullanılır.

   Re location kaydedici, «base» ve «limit» kaydedici değerlerini tutar.

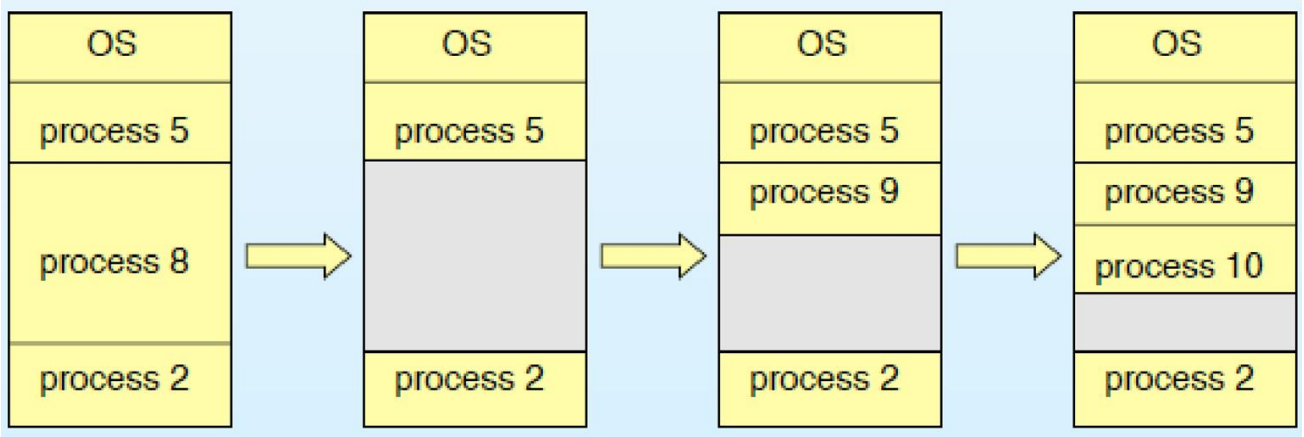




Değişmez Bölümlü Bellek Yönetimi

2 **Çok Bölümlü Tahsis (Multiple Partition Allocation)**

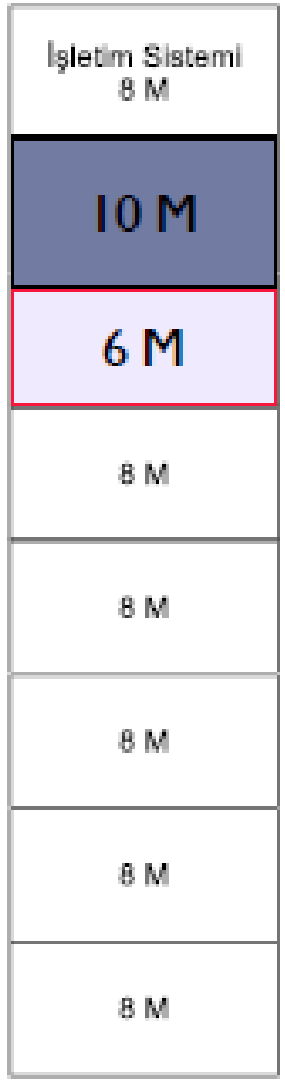
İşletim sistemi, tahsis edilmiş ve boş bellek alanı bilgisini tutar.



- Bu yöntemde ana bellek, işletim sistemi ile birden çok kullanıcı programı arasında paylaşılır.
- Ana bellek, bitişken, irili ufaklı, birden çok bölüm olarak düzenlenir.
- Bir iş, işletime alınmadan önce, ana bellekte kendisine, boyuyla uyumlu bir bölüm atanır.
- İş, kendisine atanan bu bölümü, işletimi tümüyle sonlanana kadar korur.
- Bölümler, bilgisayar sisteminin işletimi başlamadan önce belirlenen boylarını sistem işletime kapanana kadar korurlar

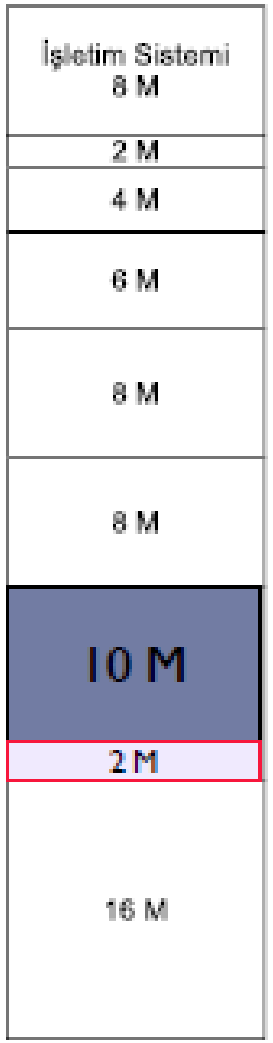
Değişmez Bölümlü Bellek Yönetimi

- Değişmez bölümlü bellek yönetiminde bölümler **sabit** bir büyüklükte olabilir.
- Toplam büyüklüğü 64M, bölüm boyutları da 8M olan bir bellek yan tarafta görünmektedir.
- 10M bir işlem istek yolladığında 2 bölüm birleştirilecek ve bu istek kaydedilince 6M'lık bir **iç parçalanma** (internal fragmentation) oluşacaktır.



Değişmez Bölümlü Bellek Yönetimi

- Değişmez bölümlü bellek yönetiminde bölümler **farklı** büyüklükte olabilir.
- Toplam büyüklüğü 64M, bölüm boyutları da farklı olan bir ana bellek yan tarafta görünmektedir.
- 10M bir işlem istek yolladığında en az iç parçalanmanın oluşacağı bölüm bulunacak istek bu bölüme konulacaktır.



Değişmez Bölümlü Bellek Yönetimi

- Bellek kullanımı etkin değil:
  - her program ne kadar boyu küçük de olsa tam bir bölmeyi elinde tutar ⇒ iç parçalanma (internal fragmentation)
  - eşit boyda olmayan bölmeler kullanılması sorunu birderece çözer
- Maksimum aktif proses sayısı sınırlı
- İşletim sistemi tarafından gerçeklenmesi kolay
- Getirdiği ek yük az.

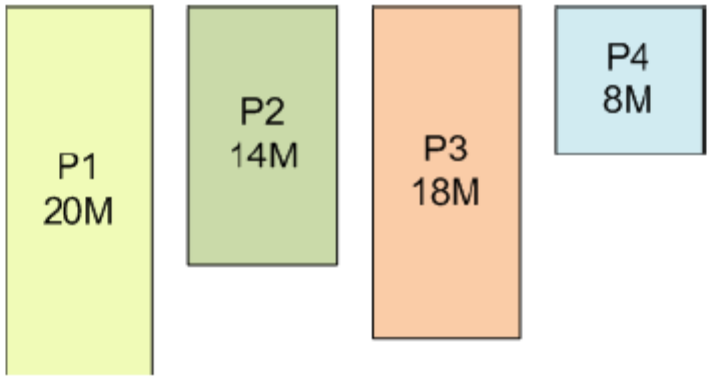
Devingen Bölümlü Bellek Yönetimi

- K5->7. L7->5 C8->23
- Bu yöntemin temel ilkesi, bölümlerin, işlerin görevlere dönüştürülüp sisteme sunuluşları aşamasında, devingen olarak yaratılmasıdır.
- Bu yöntemde, ana bellekte kullanıcı işlemleri boşluklara bloklar halinde yerleştirilmektedir.
- Bu yöntemle bir işlem ana belleğe yükleneceği zaman, işlem için gerekli miktarda yeni bir bölüm oluşturularak, işlem oluşturulan yeni bölüme yüklenir.
- Yeni bir işlem geldiğinde bu belleğe işlemin yeteceği kadar bir alan ayrılarak konumlandırılmakta ve işletim sistemi boş alanlar ve dolu alanlar ile ilgili bilgileri kayıt altına almaktadır.
- Bölümlerle ilgili bölüm başlangıç adresi, boyu gibi bilgiler tutulmaktadır.
- Bu yöntemde görevlere atanan bölümlerin yanı sıra, bu bölümler arasında kalan boş alanların da izlenmesi gereklidir

Devingen Bölümlü Bellek Yönetimi

Örnek:

- 64M ana belleğin aşağıdaki dört program için kullanılacağını varsayınız.



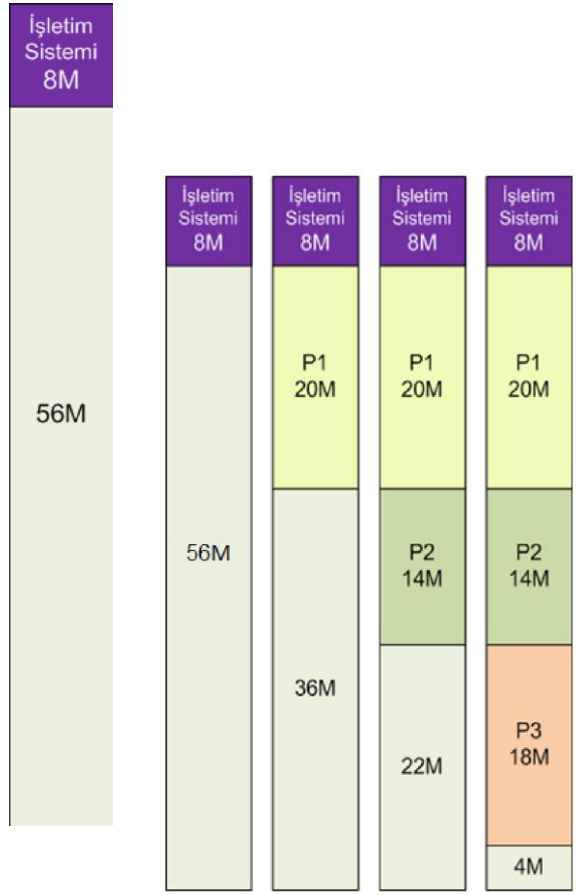
Devingen Bölümlü Bellek Yönetimi

Başlangıçta ana belleğin işletim sistemi dışında boş olduğunu varsayınız.

- Programlar birbirleri ardına gerekli bölümleri oluşturarak ana belleğe yüklenirler.

Bu durumda P4 işlemi için bellekte yeterli yer yoktur.

- Yer değiştirme (swap) işlemiyle bir işlem sanal belleğe taşınır.
- P2'nin yer değiştireceğini varsayalım.

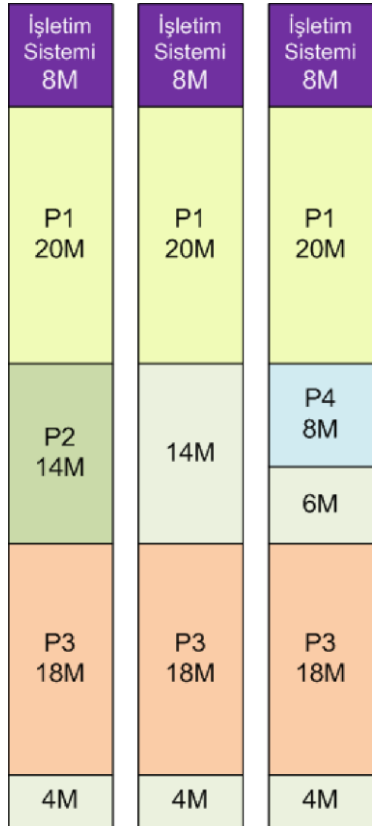


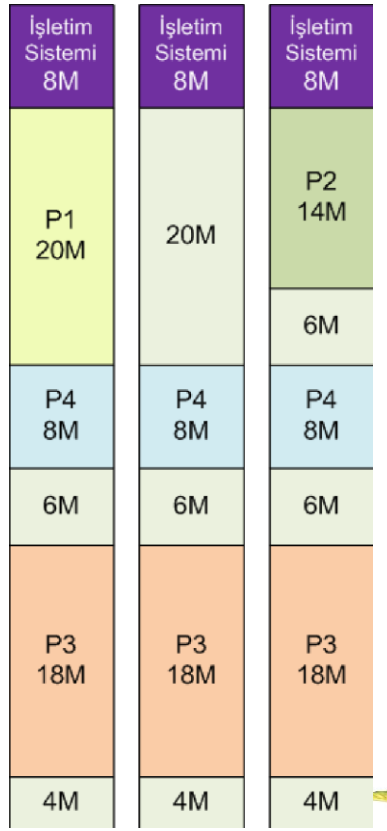
Devingen Bölümlü Bellek Yönetimi

P2'nin yer değiştirmesinden sonra bellekte 14M'lık bir bölüm serbest kalır.
Bu Bölüme P4 yüklenebilir.

P2'nin çalışması için tekrar ana belleğe yüklenmesi gerektiğini varsayalım.

- Bu sefer P1'in bekleme durumunda olduğunu ve yer değiştireceğini varsayalım.
- Yer değiştirme olayından sonra P2 tekrardan ana belleğe yüklenebilir.



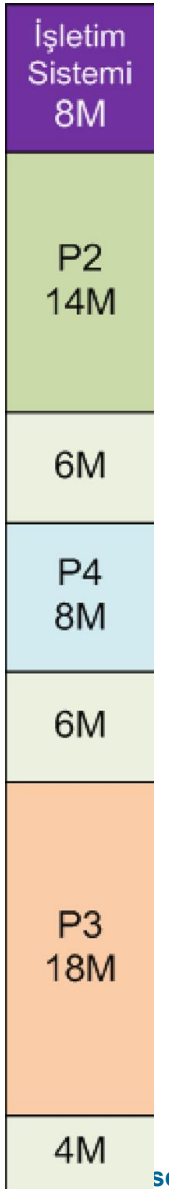


Devingen Bölümlü Bellek Yönetimi

Ana Belleğin Parçalanması Sorunu

- Ana belleğin parçalanması, bitişken alanların görevlere atanan bölümlerle, zaman içinde ufalanması olarak tanımlanır.
- Bu sorun, kullanılan bölümler arasına sıkışmış, işletim için bekleyen görevlerin gereksinimini karşılayamayan boş alanların varlığıyla ortaya çıkar.
- Belleğin parçalanması sonucu, bellekteki boş alanların toplamı, gerekli sığmaları karşılıyor olmasına karşın yeni görevlere yer sağlanamaz durumlarla karşılaşılır.

- Görevlere sağlanan alanların konumlarının işletim sırasında değiştirilememesi parçalanma sorununun temel nedenleridir.
- Bölüm içi yararlanılamayan boş alanlar **iç parçalanma**, bölümler arasında kalan boş alanlar ise **dış parçalanma** olarak adlandırılır.
- Devingen bölümlü bellek yönetiminde bölümler arasında boş alanlar kalmakta, yani dış parçalanma oluşmaktadır.



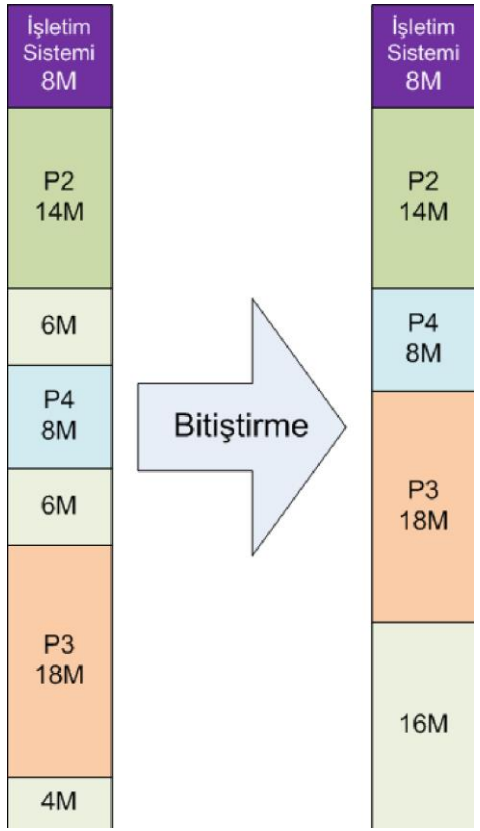
Devingen Bölümlü Bellek Yönetimi

Bitiştirme (Compaction)

Tüm bellek alanına dağılmış durumdaki bölümleri, yerlerini değiştirerek yan yana yerleştirip tek bir bitişken boş alan yaratma işlemine **bitiştirme işlemi (compaction)** denir.

- Verilen örnekte, görevlere bellek ataması yapılması ve bir göreve atanan bellek konumunun işletim sırasında değiştirilememesi sonucu bitişkin alanların parçalandığı görülmektedir.
- Bu sakıncalı durumun yok edilebilmesi için bitiştirme işlemi kullanılabilir.

- Bitiştirme işlemiyle tüm kullanılmayan parçalar bir yerde toplanır.
- Bitiştirme işlemi, işlemciyi meşgul eden, zaman alan ve dolayısıyla bilgisayarı yavaşlatan bir işlemdir.
- Bitiştirme ihtiyacını azaltmak için farklı **yerleştirme algoritmaları** kullanılır.



Devingen Bölümlü Bellek Yönetimi

Yerleştirme Algoritmaları

İşletim sistemi, bellekteki boşlukların bulunduğu listeden işlem için en uygun boşluğu belirlemede 3 başlıca strateji kullanır:

- İlk Uygun Yer Algoritması (First Fit)
- Sonraki Uygun Yer Algoritması (Next Fit)
- En Uygun Yer Algoritması (Best Fit)

Yerleştirme Algoritmaları

İlk Uygun Yer Algoritması

- Bir süreç (işlem) bellek isteğinde bulunduğunda, bellek baştan sona taranır ve süreç için gerekli olan bellek boyutunu sağlayan ilk boş alana yerleştirilir.
- Gerçekleştirmesi kolay ve en hızlı çalışan algoritmadır.

Sonraki Uygun Yer Algoritması

- İlk uygun yer algoritması ile aynı mantıkta çalışır, fakat bu algoritma en son bulduğu uygun yer bilgisini saklar.
- Bir sonraki aramada belleğin en başından değil de, saklamış olduğu noktadan itibaren aramaya başlar.
- İlk uygun yer algoritmasına göre biraz daha yavaş çalışır.

En Uygun Yer Algoritması

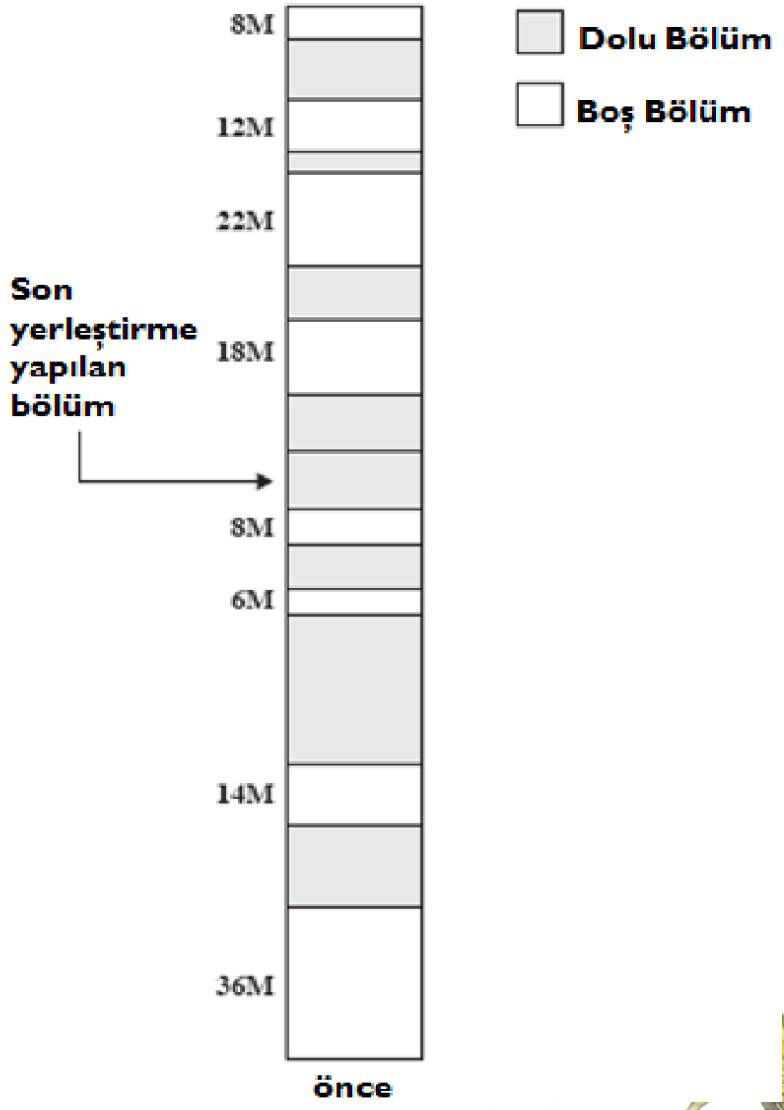
- Bu algoritma belleğin başından sonuna kadar tüm boş alanları tarar ve süreç için gerekli olan bellek boyutuna en uygun olan (en yakın boyuttaki) boş bellek alanına süreci yerleştirir.
- En yavaş algoritma çalışan algoritmadır.
- Çok küçük ve kullanışsız parçalanmalara sebep olur.

Kod:mem_fit.c

Yerleştirme Algoritmaları

Örnek 1

- Ana bellekteki boş ve dolu bölümlerin yan tarafta göründüğü gibi olduğunu varsayınız.
- 16M'lık bir yerleştirme isteğini ilk uygun, sonraki uygun ve en uygun yer algoritmasına göre belleğe yerleştiriniz.

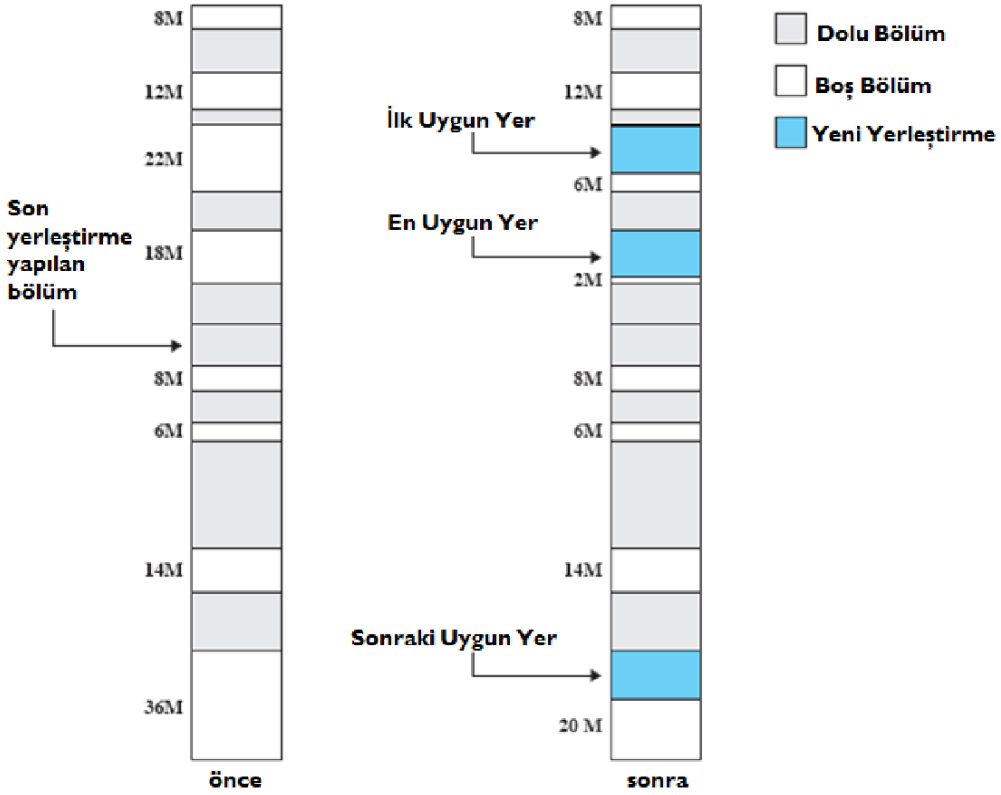


Yerleştirme Algoritmaları

Örnek 1

- Ana bellekteki boş ve dolu bölümlerin yan tarafta göründüğü gibi olduğunu varsayınız.
- 16M'lık bir yerleştirme isteğini ilk uygun, sonraki uygun ve en uygun yer algoritmasına göre belleğe yerleştiriniz.

İlk uygun yer algoritması 22MB'lık bölümü kullanarak 2MB'lık dış parçalanma, **sonraki uygun** yer algoritması son 36MB'lik bölümü kullanarak 20MB'lik dış parçalanma, **en uygun** yer algoritması da tüm yerlere bakarak 18M'lık bölümü kullanarak, 2M'lık dış parçalanma oluşturur.



Yerleştirme Algoritmaları

Örnek 2

Ana bellekte sırayla aşağıdaki boş bölümlerin olduğunu varsayınız

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3	BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	40 K	36 K	14 K	18 K	24 K	30 K

24K, 20K ve 18K'lık bellek kullanım istekleri için sırasıyla hangi boşlukların kullanılacağını ve oluşacak parçalanmaları aşağıdaki algoritmaları kullanarak belirtiniz.

- İlk Uygun Yer Algoritması
- Sonraki Uygun Yer Algoritması
- En Uygun Yer Algoritması

Yerleştirme Algoritmaları

Örnek 2

İlk Uygun Yer Algoritması

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3	BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	40 K	36 K	14 K	18 K	24 K	30 K

24 K **Boşluk 3** (16K parçalanma)

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3		BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	24 K	16 K	36 K	14 K	18 K	24 K	30 K

20 K **Boşluk 1** (0K parçalanma)

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3		BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	24 K	16 K	36 K	14 K	18 K	24 K	30 K

18 K **Boşluk 4** (18K parçalanma)

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3		BOŞLUK 4		BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	24 K	16 K	18 K	18 K	14 K	18 K	24 K	30 K

Yerleştirme Algoritmaları

Örnek 2

Sonraki Uygun Yer Algoritması

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3	BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	40 K	36 K	14 K	18 K	24 K	30 K

24 K **Boşluk 3** (16K parçalanma)

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3	BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	24 K	16 K	36 K	14 K	18 K	24 K	30 K

20 K **Boşluk 4** (16K parçalanma)

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3	BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	24 K	16 K	20 K	16 K	14 K	18 K	24 K	30 K

18 K **Boşluk 6** (0K parçalanma)

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3	BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	24 K	16 K	20 K	16 K	14 K	18 K	24 K	30 K

Yerleştirme Algoritmaları

Örnek 2

En Uygun Yer Algoritması

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3	BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	40 K	36 K	14 K	18 K	24 K	30 K

24 K **Boşluk 7** (0K parçalanma)

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3	BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	40 K	36 K	14 K	18 K	24 K	30 K

20 K **Boşluk 1** (0K parçalanma)

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3	BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	40 K	36 K	14 K	18 K	24 K	30 K

18 K **Boşluk 6** (0K parçalanma)

BOŞLUK 1	BOŞLUK 2	BOŞLUK 3	BOŞLUK 4	BOŞLUK 5	BOŞLUK 6	BOŞLUK 7	BOŞLUK 8
20 K	8 K	40 K	36 K	14 K	18 K	24 K	30 K

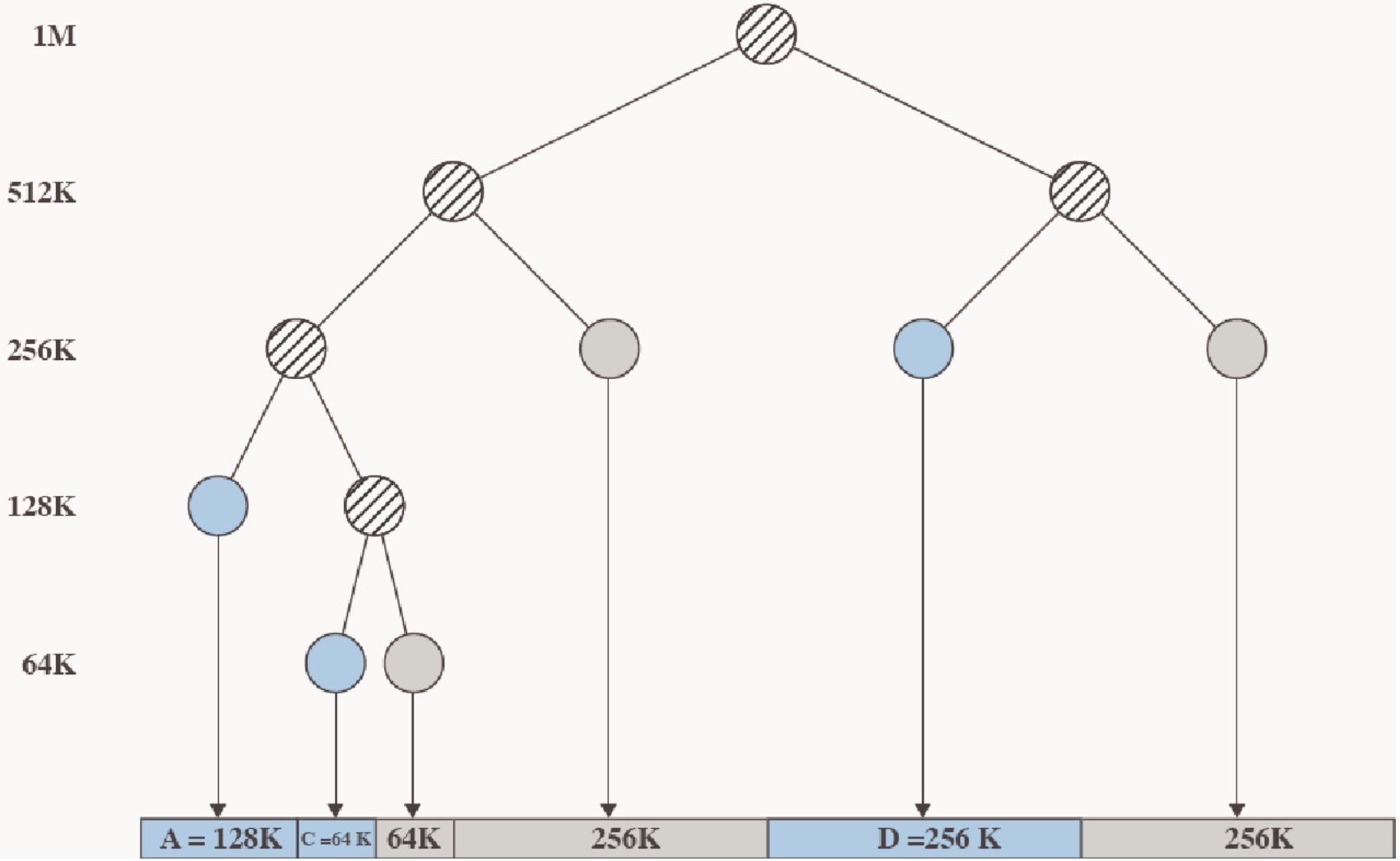
Buddy Yöntemi

- Tüm boş alan 2^U boyutunda tek bir alan olarak ele alınır
- s boyutundaki bir istek eğer $2^{U-1} < s <= 2^U$ ise tüm blok atanır
- Aksi halde blok 2^{U-1} boyutunda iki eş bloğa bölünür (buddy)
- s'den büyük veya eşit en küçük birim blok oluşturulana kadar işlem devam eder

Buddy Yöntemi

1 Mbyte block	1 M				
Request 100 K	A = 128K	128K	256K	512K	
Request 240 K	A = 128K	128K	B = 256K	512K	
Request 64 K	A = 128K	C = 64K	64K	B = 256K	512K
Request 256 K	A = 128K	C = 64K	64K	B = 256K	D = 256K	256K
Release B	A = 128K	C = 64K	64K	256K	D = 256K	256K
Release A	128K	C = 64K	64K	256K	D = 256K	256K
Request 75 K	E = 128K	C = 64K	64K	256K	D = 256K	256K
Release C	E = 128K	128K	256K	D = 256K	256K	
Release E	512K			D = 256K	256K	
Release D	1M					

Buddy Yöntemi



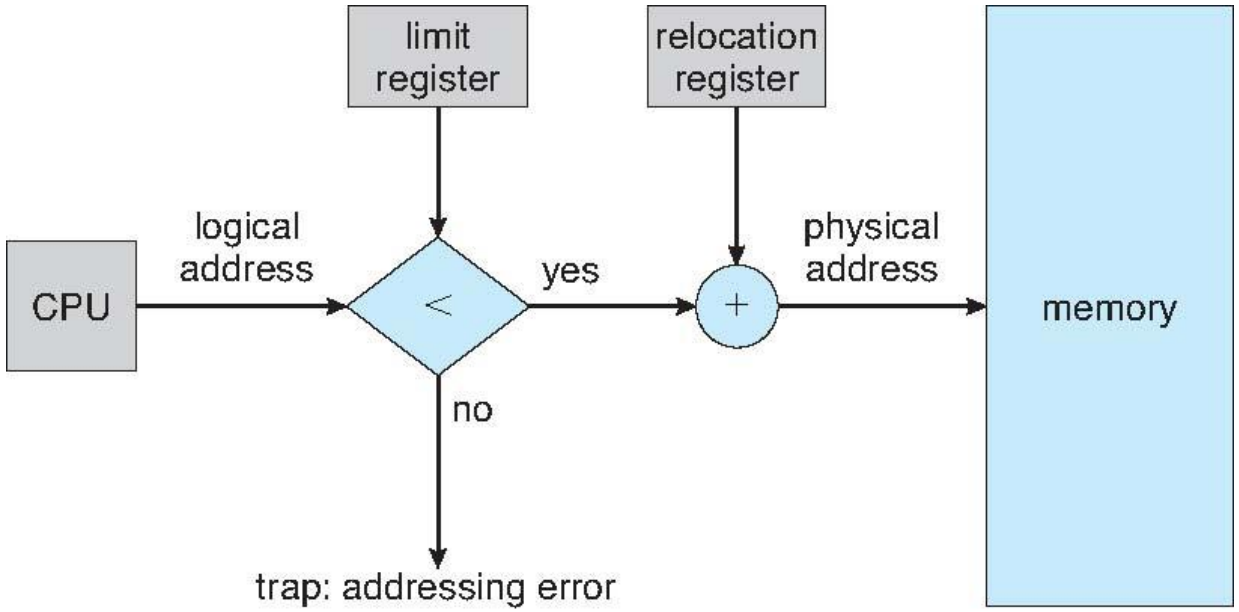
Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
  - Each process contained in single contiguous section of memory

Contiguous Allocation (Cont.)

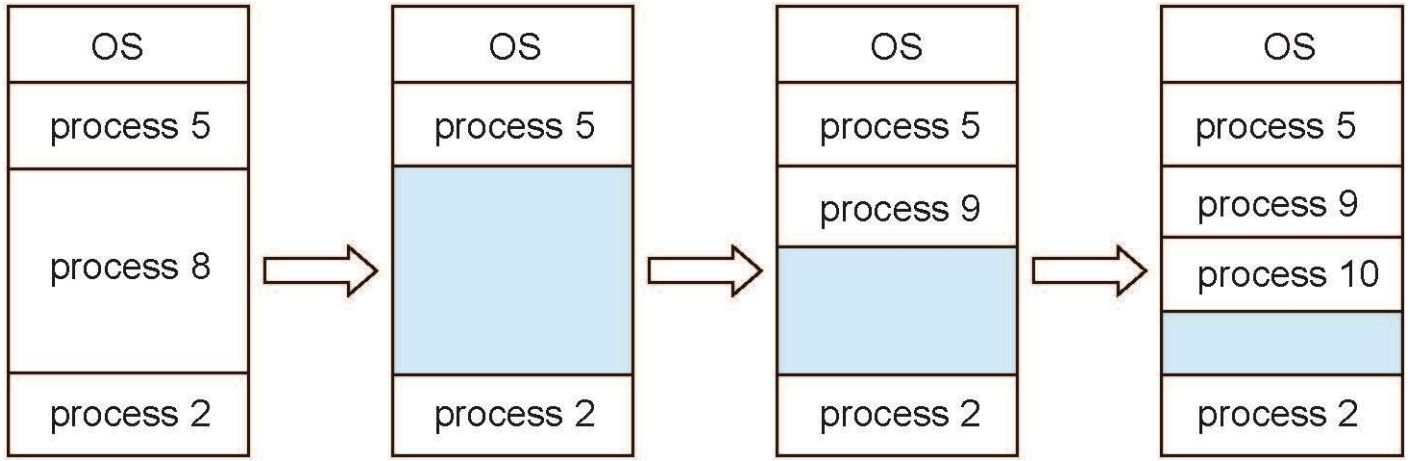
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*
  - Can then allow actions such as kernel code being **transient** and kernel changing size

Hardware Support for Relocation and Limit Registers



Multiple-partition allocation

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **Hole** – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



Dynamic Storage-Allocation Problem

How to satisfy a request of size ***n*** from a list of free holes?

- **First-fit**: Allocate the ***first*** hole that is big enough
- **Best-fit**: Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
  - Produces the largest leftover hole

First-fit and best-fit better than worst-fit in terms of speed and storage utilization

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- First fit analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable -> **50-percent rule**

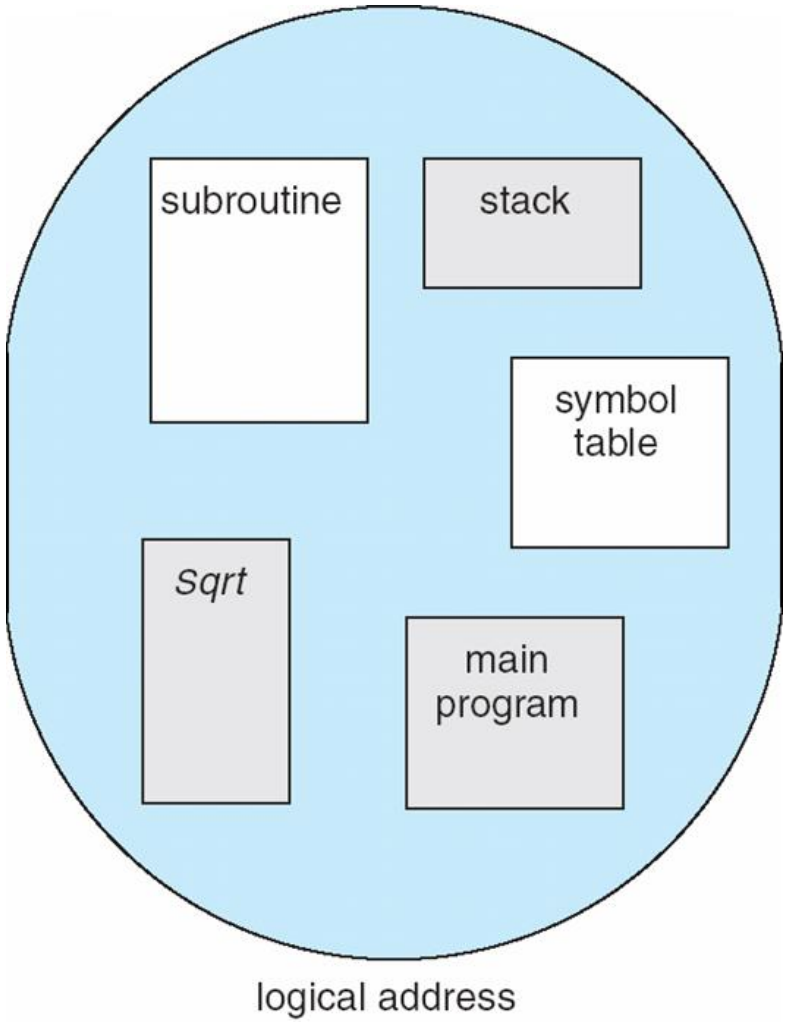
Fragmentation (Cont.)

- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems

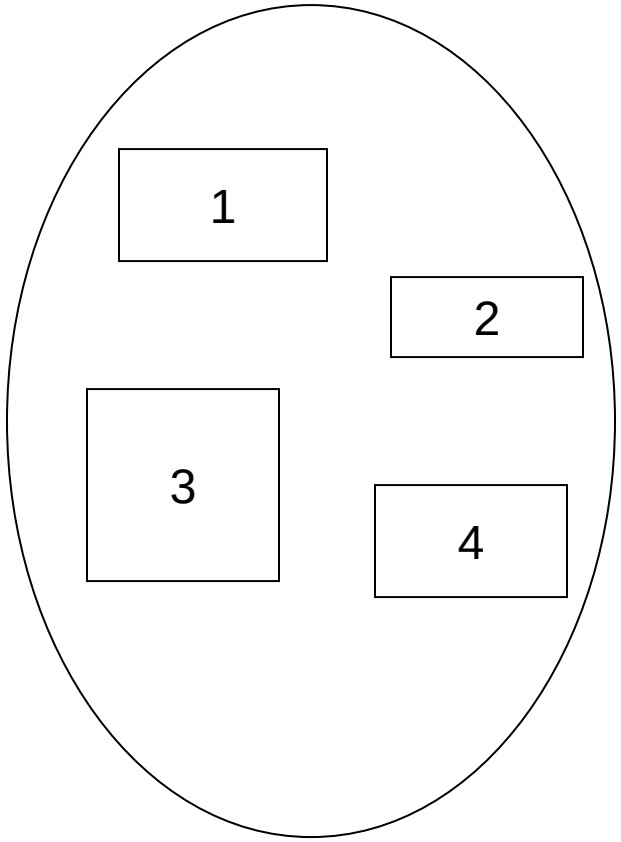
Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
  - A segment is a logical unit such as:
    - main program
    - procedure
    - function
    - method
    - object
    - local variables, global variables
    - common block
    - stack
    - symbol table
    - arrays

User's View of a Program



Logical View of Segmentation



user space

physical memory space

Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < **STLR**

Segmentation Architecture (Cont.)

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 ⇒ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram

Segmentation Hardware