Overview Basic Concepts

• Process must be loaded into memory before being executed.
• *Input queue* – collection of processes on the disk that are waiting to be brought into memory for execution.
• The OS manages memory by allocating and de-allocating memory to processes.
Memory Management

• Swapping
• Overlaying – not used any more
• Partitioning
• Paging
• Segmentation
• Virtual memory – Only loads a portion of program into memory
Addresses

• The **symbolic addresses** are the addresses used in a source program. The variable names, symbolic constants and instruction labels are the basic elements of the symbolic address space.

• The **compiler** converts a symbolic address into a **relative address**.

• The **physical address** consists of the final address generated when the program is loaded and ready to execute in physical memory; the **loader** generates these addresses.
Process Address Space

- A logical address is a reference to some location in the body of a process.
- The set of logical addresses that a process references in its code.
- When memory is allocated to the process, its set of logical addresses will be bound to physical addresses.
Logical and Physical Addresses
Binding

• The association of instructions and data to memory addresses

• Can occur at any of the following steps
  – Compile time
  – Load time
  – Execution time
Phases of a Program

1. **Compile time**: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.

2. **Linkage time**: Program is combined with other modules.

3. **Load time**: Program is loaded into memory.

4. **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., *base* and *limit registers*).
Program Phases and Addresses

- Editor
  - Symbolic addresses

- Compiler/assembler
  - Relative addresses

- Linker
  - Other compiled modules
  - Relative addresses

- Loader
  - Physical addresses

© 2012 Jones & Bartlett Learning, LLC
www.jblearning.com
Managing the Address Space

• The compiler or assembler generates the program as a **relocatable** object module
• The linker combines several modules into a load module
• During memory allocation, the loader places the load module in the allocated block of memory
• The loader **binds** the logical address to a physical address
Dynamic Loading

• Routines or modules to be used by a program are not loaded until called
• All routines are stored on a disk in relocatable form
• Better memory-space utilization; unused routines are never loaded.
• Useful when large amounts of code are needed to handle infrequently occurring cases.
• No special support from the operating system is required to be implemented through program design.
Dynamic Linking

• Linking postponed until execution time.
• Example: Dynamic Linked Libraries (DLL)
• Small piece of code, stub, used to locate the appropriate memory-resident library routine.
• Stub replaces itself with the address of the routine, and executes the routine.
• Operating system needed to check if the routine is in the processes’ memory address.

© 2012 Jones & Bartlett Learning, LLC
www.jblearning.com
Logical vs. Physical Address Space

- **Logical address** – generated by the compiler/assembler; also referred to as **virtual address**.
- **Physical address** – address seen by the memory unit.
- **Logical address space** is the set of all addresses of a program.
- **Physical address space** is the set of addresses used to store the program into memory.
- The logical **address space** is bound to a separate **physical address space**.
Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address.
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program deals with *logical* addresses; it never directly references the *real* physical addresses.
Static Relocation

• A process executes at a specific address.
• Addresses are assigned at load time.
• Relocation modifies instructions that address locations in memory.
• Static relocation does not solve protection.
Dynamic Relocation

• Addresses can be changed at execution time.
• Facilitates Protection using:
  – Addressing via Base (& Limit) Register.
• Additional benefit, program may be moved during execution.
Memory Protection

Necessary because malicious or incompetent programs may access the program or data space of another program and contaminate it.
Protection (2)

• Solution:
  – Divide memory into Blocks of memory, associate a key with each block.
  – Trap when any program attempts to access memory whose key does not agree with its own.

• Alternative:
  – Load a Base Register w/ address at which process is loaded.
  – Load a Limit Register w/ size of partition.
  – Compare each address generated with the contents of the base and base + limit register.
Contiguous Memory Allocation

- Main memory is divided into several **partitions**
- A partition is a **contiguous block** of memory that can be allocated to an individual process
- The degree of multiprogramming is determined by the number of partitions in memory.
Contiguous Memory Allocation

• As mentioned before, when a process completes and terminates, memory is de-allocated
• This type of memory management was used in the early OS’s with multiprogramming
Multiple Partitions

• **Fixed** partitions (static) – the number and sizes of the partitions do not change
• **Variable** partitions (dynamic) – partitions are created dynamically according to:
  – available memory
  – the memory requirements of processes
Fixed Partitions

• Memory is divided into fixed-sized partitions. These are not normally of the same size.
• The number and the size of the partitions are fixed.
• One partition is allocated to each active process in the multiprogramming set.
• There is one special partition, the system partition, in which the memory-resident portion of the operating system is always stored.
Fixed Partitions

Partition 6
900K
Partition 5
850K
Partition 4
700K
Partition 3
450K
Partition 2
300K
Partition 1
100K
Operating system

Unused memory
Partition unused
Unused memory
Unused memory
Unused memory
Unused memory
Fragmentation in Fixed Partition

Fragmentation problem

• Internal fragmentation -- A partition is only partially used.
• A partition is available, but not large enough for any waiting progress.
Memory Allocation Problem

- An important problem fixed partition is finding a fit between the partition sizes and the actual memory requirements of processes
- The goal is to minimize fragmentation
Dynamic Partition Allocation

• The partitions are created **dynamically** (as needed)

• The OS maintains a **table of partitions allocated** that indicates which parts (location and size) of memory are **available** and which have been **allocated**.

© 2012 Jones & Bartlett Learning, LLC
www.jblearning.com
Holes in Memory

- **Hole** – a contiguous block of available memory; holes of various size are scattered throughout memory.
- When a process requests memory, it is allocated memory from a **hole** large enough to accommodate it.
- Operating system maintains data about:
  - allocated partitions
  - Available memory blocks (holes)
Allocation with Dynamic Partitions

- At any given time, there is a list of available blocks of memory of various sizes (holes) and a queue of processes requesting memory.
- Memory is allocated contiguously to processes until there is no available block of memory large enough.
Dynamic Memory Allocation

The memory manager can:

• Wait until a large enough block of memory is available, or

• Skip down the queue to find a process with smaller requirements for memory.
Holes and Allocation

• When a process is to be loaded, the OS searches for a hole large enough for this process and allocates the necessary space.
• When a process terminates, the OS frees its block of memory.
• In general, there is at any time, a set of holes, of various sizes, scattered throughout memory.
• If a new hole is adjacent to other holes, they will be merged to form one larger hole.
Memory Allocation to P7

(a) 1100K
Partition5
Partition4
Partition3
Partition2
Partition1
Operating system
0
Hole
P4
P3
P2
P5
P6

(b) 1100K
Hole
P7
P4
P3
P2
P5
P6
Operating system
De-allocating Memory to P5
Advantages of Dynamic Partitions

• Memory utilization is generally better for variable-partition schemes.
• There is little or no internal fragmentation.
• There can be external fragmentation.
Compaction: External Fragmentation

• External fragmentation is a serious problem.
• The goal of compaction is to shuffle the memory contents to place all free memory together in one large block.
• This is only possible if relocation is dynamic (binding is done at execution time), using base and limit registers.
• Can be quite expensive (overhead).
Memory Allocation

How to satisfy a process 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.
Memory After Compaction

- Partition 5: P4
- Partition 4: P3
- Partition 3: P2
- Partition 2: P5
- Partition 1: P6

Operating system

Hole

Unallocated memory
Swapping

• A process can be *swapped* temporarily out of memory to secondary *storage*, and then loaded into memory again to resume execution.

• Secondary storage – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
Swapping

• *Roll out, roll in* – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.

• Major part of swap time is transfer time; total transfer time is directly proportional to the *amount* of memory swapped.

• Modified versions of swapping are found on many systems, i.e., UNIX and Microsoft Windows.
Fragmentation - Recap

• **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.
Non-contiguous Memory Allocation

Used in modern operating systems

• Paging
• Segmentation
Pages

- A page is a unit of logical memory of a program
- A frame is a unit of physical memory
- All pages are of the same size
- All frames are of the same size
- A frame is of the same size as a page
Paging

- Physical memory is divided into fixed-sized blocks called **frames** (size is power of 2).
- Logical memory is divided into blocks of same size called **pages**.
- A page of a program is stored on a frame, independently of other pages.
- A logical address on the page is converted to a physical address on the corresponding frame.
Paging(2)

• The OS keeps track of all free (available) frames, and allocated frames in the page table.
• To run a program of size \( n \) pages, the OS needs \( n \) free frames to load program.
• The OS sets up a page table for every process
• The page table is used for converting logical addresses to physical addresses.
• There is a small amount of internal fragmentation.
Memory Allocation with Paging

- The frames allocated to the pages of a process need not be contiguous; in general, the system can allocate any empty frame to a page of a particular process.
- There is no external fragmentation.
- There is potentially a small amount of internal fragmentation that would occur on the last page of a process.
Logical vs Physical Memory

- Logical memory corresponds to the user’s view of memory
- Physical memory is the actual contents and addresses of memory
- The user’s view of memory is mapped onto physical memory
- This mapping is done on every reference to logical memory
Logical Address

• Any address referenced in a process is defined by
  – the page that the address belongs to, and
  – the relative address within that page.

• A logical address of a process consists of a page number and an offset.
Logical Address (2)

Address generated by the compiler/assembler is divided into:

- **Page number** \((p)\) – used as an index into a *page table* (which contains base address of each page in physical memory).
- **Page offset** \((d)\) – the relative address in the page.
- This pair of numbers will be converted to the physical memory address that is sent to the memory unit.
Example of a Logical Address

<table>
<thead>
<tr>
<th>Page number</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000010</td>
<td>0111011110</td>
</tr>
</tbody>
</table>

© 2012 Jones & Bartlett Learning, LLC
www.jblearning.com
Memory Reference

What is a memory reference?
Physical Address

• When the system allocates a frame to a page, it translates this logical address into a physical address that consists of a frame number and the offset.

• For this, the system needs to know the correspondence of a page of a process to a frame in physical memory and it uses a page table.
Example of a Physical Address
Address Translation Architecture

- Logical address
  - Page number
  - Page offset

- Physical memory
  - Frame 1
  - Frame 2
  - Frame 0
  - Frame 3

- Page table
  - Page table entry

Page number is index into page table
Page Table Example

<table>
<thead>
<tr>
<th>Page</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>f5</td>
</tr>
<tr>
<td>1</td>
<td>f10</td>
</tr>
<tr>
<td>2</td>
<td>f8</td>
</tr>
<tr>
<td>3</td>
<td>f16</td>
</tr>
</tbody>
</table>
Implementation of Page Table

- A page table is kept in main memory.
- *Page-table base register (PTBR)* points to the page table.
- *Page-table length register (PRLR)* indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
Improving Memory Access

• The two memory access problem can be solved by the use of a special fast-lookup hardware cache called *associative registers* or *translation look-aside buffers (TLBs)*
Memory Protection

• Every memory reference causes a page table lookup to get the appropriate frame number
• Memory protection implemented by associating protection bit with each frame.
• Valid-invalid bit attached to each entry in the page table:
  – “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page.
  – “invalid” indicates that the page is not in the process’ logical address space.
Valid/Invalid Bit

frame number

valid–invalid bit

00000

page 0
page 1
page 2
page 3
page 4
page 5

0
2 v

1
3 v

2
4 v

3
7 v

4
8 v

5
9 v

6
0 i

7
0 i

page table

0

1

2
page 0

3

4
page 1

5

6
page 2

7

8
page 3

9

page 4

page n

© 2012 Jones & Bartlett Learning, LLC
www.jblearning.com
Shared Pages

• Shared code
  – Code shared among processes (i.e., text editors, compilers, window systems). **Only one copy is loaded**
  – Shared code must appear in same location in the logical address space of all processes.

• Private code and data
  – Each process keeps a separate copy of the code and data.
  – The pages for the private code and data can appear anywhere in the logical address space.