Memory Management
Memory Management

• Subdividing memory to accommodate multiple processes
• Memory needs to be allocated efficiently to pack as many processes into memory as possible
Big Picture

kernel memory

proc struct

kernel stack/u area

Stack

Data

Text (shared)

kernel stack/u area

Stack

Data

Text (shared)

kernel stack/u area

Stack

Data

Text (shared)
Requirements

- Relocation
- Protection
- Sharing
- Logical Organization
- Physical Organization
Virtualizing Resources

- Physical Reality:
  Different Processes/Threads share the same hardware
  - Need to multiplex CPU (Just finished: scheduling)
  - Need to multiplex use of Memory (Today)
  - Need to multiplex disk and devices (later in term)

- Why worry about memory sharing?
  - The complete working state of a process and/or kernel is defined by its data in memory (and registers)
  - Consequently, cannot just let different threads of control use the same memory
    - Physics: two different pieces of data cannot occupy the same locations in memory
  - Probably don’t want different threads to even have access to each other’s memory (protection)
Single and Multithreaded Processes

- Threads encapsulate concurrency
  - “Active” component of a process
- Address spaces encapsulate protection
  - Keeps buggy program from trashing the system
  - “Passive” component of a process
Important Aspects of Memory Multiplexing

• **Controlled overlap**:  
  - Separate state of threads should not collide in physical memory. Obviously, unexpected overlap causes chaos!  
  - Conversely, would like the ability to overlap when desired (for communication)

• **Translation**:  
  - Ability to translate accesses from one address space (virtual) to a different one (physical)  
  - When translation exists, processor uses virtual addresses, physical memory uses physical addresses  
  - Side effects:  
    • Can be used to avoid overlap  
    • Can be used to give uniform view of memory to programs

• **Protection**:  
  - Prevent access to private memory of other processes  
    • Different pages of memory can be given special behavior (Read Only, Invisible to user programs, etc).  
    • Kernel data protected from User programs  
    • Programs protected from themselves
Multi-step Processing of a Program for Execution

- Preparation of a program for execution involves components at:
  - Compile time (i.e. “gcc”)
  - Link/Load time (unix “ld” does link)
  - Execution time (e.g. dynamic libs)
- Addresses can be bound to final values anywhere in this path
  - Depends on hardware support
  - Also depends on operating system
- Dynamic Libraries
  - Linking postponed until execution
  - 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 routine
Recall: Uniprogramming

- Uniprogramming (no Translation or Protection)
  - Application always runs at same place in physical memory since only one application at a time
  - Application can access any physical address

- Application given illusion of dedicated machine by giving it reality of a dedicated machine

- Of course, this doesn’t help us with multithreading
Multiprogramming (First Version)

- **Multiprogramming without Translation or Protection**
  - Must somehow prevent address overlap between threads
  
  - Trick: Use Loader/Linker: Adjust addresses while program loaded into memory (loads, stores, jumps)
    
    - Everything adjusted to memory location of program
    - Translation done by a linker-loader
    - Was pretty common in early days

- **With this solution, no protection: bugs in any program can cause other programs to crash or even the OS**
Multiprogramming (Version with Protection)

- Can we protect programs from each other without translation?
  - Yes: use two special registers base and limit to prevent user from straying outside designated area
    - If user tries to access an illegal address, cause an error
  - During switch, kernel loads new base/limit from TCB
    - User not allowed to change base/limit registers
Memory Management: Requirements

- **Relocation**
  - **Why/What:**
    - programmer does not know where the program will be placed in memory when it is executed
    - while the program is executing, it may be swapped to disk and returned to main memory at a different location
  - **Consequences/Constraints:**
    - memory references must be translated in the code to actual physical memory address
Memory Management: Requirements

• **Protection**
  
  - *Protection and Relocation are interrelated*
  
  - Why/What:
    
    • Protect process from interference by other processes
    
    • processes require **permission** to access memory in another processes address space.

  - Consequences/Constraints:
    
    • impossible to check addresses in programs since the program could be relocated
    
    • must be checked at run time
Memory Management: Requirements

• **Sharing**
  - *Sharing and Relocation are interrelated*
  - allow several processes to access the same data
  - allow multiple programs to share the same program text
Memory Management: Requirements

• *Logical Organization*
  - programs organized into modules (stack, text, uninitialized data, or logical modules such as libraries, objects, etc.)
  - Code modules may be compiled independently
  - different degrees of protection given to modules (read-only, execute-only)
  - share modules
Memory Management: Requirements

- **Physical Organization**
  - Memory organized into two levels: *main and secondary memory*.
  - Memory available for a program plus its data may be insufficient.
  - Main memory relatively fast, expensive and volatile.
  - Secondary memory relatively slow, cheaper, larger capacity, and non-volatile.
Relocation

• **Fixed partitions**: When program loaded *absolute* memory locations assigned

• A process may occupy different partitions thus different absolute memory locations during execution

• Compaction will also cause a program to occupy a different partition which means different absolute memory locations
Registers Used during Execution

- Base register
  - starting address for the process
- Bounds register
  - ending location of the process
- These values are set when the process is loaded and when the process is swapped in
Registers Used during Execution

• The value of the base register is added to a relative address to produce an absolute address.
• The resulting address is compared with the value in the bounds register.
• If the address is not within bounds, an interrupt is generated to the operating system.
Hardware Support for Relocation

- Base Register
- Bounds Register
- Adder
- Comparator

Interrupt to operating system

Relative address

Process image in main memory

- Process Control Block
  - Program
  - Data
  - Stack

Absolute address
Memory Partitioning

- **Virtual Memory**
  - Segmentation and/or Paging

- **Non-Virtual memory approaches**
  - Partitioning - Fixed and Dynamic
  - simple Paging
  - simple Segmentation
Fixed Partitioning

- Partition available memory into regions with fixed boundaries
- Equal-size partitions
  - process size \(\leq\) partition size can be loaded into available partition
  - if all partitions are full, the operating system can swap a process out of a partition
- *If program size > partition size, then programmer must use overlays*
Fixed Partitioning - Equal Size

- Main memory use is inefficient.
  - *Internal Fragmentation* - Part of partition unused

Diagram:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>8 M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>process 1</td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>8 M</td>
</tr>
<tr>
<td></td>
<td>8 M</td>
</tr>
</tbody>
</table>

11/1/2006
Fixed Partitioning - Unequal Sizes

- lessens the problem with equal-size partitions

```
Operating System
8 M

2 M
process 1
4 M

6 M
8 M
8 M
12 M
```
Placement Algorithm with Partitions

• **Equal-size partitions**
  - because all partitions are of equal size, it does not matter which partition is used

• **Unequal-size partitions**
  - can assign each process to the smallest partition within which it will fit
    • queue for each partition
  - processes are assigned in such a way as to minimize wasted memory within a partition
One Process Queue per Partition

New Processes

Operating System

Memory Management
One Process Queue for All

- smallest available partition that will hold the process is selected
Dynamic Partitioning

• **External Fragmentation** - small holes in memory between allocated partitions.

• Partitions are of variable length and number

• Process is allocated exactly as much memory as required

• Must use compaction to shift processes so they are contiguous and all free memory is in one block
Example Dynamic Partitioning

<table>
<thead>
<tr>
<th>Operating System</th>
<th>128 K</th>
<th>Operating System</th>
<th>320 K</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Process 1</td>
<td></td>
<td>Process 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>320 K</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Process 2</td>
<td>224 K</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>352 K</td>
</tr>
</tbody>
</table>
Example Dynamic Partitioning

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Process 1</th>
<th>Process 2</th>
<th>Process 3</th>
<th>Process 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>320 K</td>
<td>224 K</td>
<td>288 K</td>
<td>128 K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>224 K</td>
<td>288 K</td>
<td>96 K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>64 K</td>
<td>28 K</td>
</tr>
</tbody>
</table>

11/1/2006
### Example Dynamic Partitioning

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Process 3</th>
<th>Process 4</th>
<th>Process 2</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>288 K</td>
<td>128 K</td>
<td>224 k</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64 K</td>
<td>96 K</td>
<td>96 K</td>
<td></td>
</tr>
</tbody>
</table>

Memory Management
Dynamic Partition Placement Algorithm

• Operating system must decide which free block to allocate to a process

• *Best-fit algorithm*
  - chooses block that is closest in size to the request
  - *worst performer overall*
  - results in minimally sized fragments requiring compaction
Dynamic Partition Placement Algorithm

• **First-fit algorithm**
  - starts scanning from beginning and choose first available block that is large enough.
  - *fastest*
  - may have many process loaded in the front end of memory that must be scanned
Dynamic Partition Placement Algorithm

- **Next-fit**
  - scan memory from the location of the last allocation and chooses the next available block that is large enough
  - more often allocate a block of memory at the end of memory where the largest block is found
  - compaction is required to obtain a large block at the end of memory
Dynamic Partition Placement Algorithm

Last allocated block (14K)

Before

alloc 16K block

After

11/1/2006

First Fit

Best Fit

Next Fit

Allocated block

Free block
Buddy System

Power of two, free buffer coalescing
buffer split into buddy pairs
requests rounded to power of two

Advantages:
- coalesces adjacent buffers

Disadvantage:
- performance (recursive coalescing is expensive)
- poor api
Buddy System in Action

Free list

128  256  512

Bitmap (32B chunks)

0 128 256 512 1023

using a 1024 Byte buffer
alloc(128); // return C

In-use  Free

0 1 1 1 1 0 ... 0
Addresses

- **Logical**
  - reference to a memory location independent of the current assignment of data to memory
  - translation must be made to the physical address

- **Relative**
  - address expressed as a location relative to some known point

- **Physical**
  - the absolute address or actual location
Paging

• Partition memory into small equal-size chunks
  - Chunks of memory are called frames

• Divide each process into the same size chunks
  - Chunks of a process are called pages

• Operating system maintains a page table for each process
  - contains the frame location for each process page
  - memory address = page number + offset
## Paging

<table>
<thead>
<tr>
<th>Frame Number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11/1/2006
# Paging

<table>
<thead>
<tr>
<th>0</th>
<th>A.0</th>
<th>0</th>
<th>A.0</th>
<th>0</th>
<th>A.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A.1</td>
<td>1</td>
<td>A.1</td>
<td>1</td>
<td>A.1</td>
</tr>
<tr>
<td>2</td>
<td>A.2</td>
<td>2</td>
<td>A.2</td>
<td>2</td>
<td>A.2</td>
</tr>
<tr>
<td>3</td>
<td>A.3</td>
<td>3</td>
<td>A.3</td>
<td>3</td>
<td>A.3</td>
</tr>
<tr>
<td>4</td>
<td>B.0</td>
<td>4</td>
<td></td>
<td>4</td>
<td>D.0</td>
</tr>
<tr>
<td>5</td>
<td>B.1</td>
<td>5</td>
<td></td>
<td>5</td>
<td>D.1</td>
</tr>
<tr>
<td>6</td>
<td>B.2</td>
<td>6</td>
<td></td>
<td>6</td>
<td>D.2</td>
</tr>
<tr>
<td>7</td>
<td>C.0</td>
<td>7</td>
<td>C.0</td>
<td>7</td>
<td>C.0</td>
</tr>
<tr>
<td>8</td>
<td>C.1</td>
<td>8</td>
<td>C.1</td>
<td>8</td>
<td>C.1</td>
</tr>
<tr>
<td>9</td>
<td>C.2</td>
<td>9</td>
<td>C.2</td>
<td>9</td>
<td>C.2</td>
</tr>
<tr>
<td>10</td>
<td>C.3</td>
<td>10</td>
<td>C.3</td>
<td>10</td>
<td>C.3</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>11</td>
<td></td>
<td>11</td>
<td>D.3</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>12</td>
<td></td>
<td>12</td>
<td>D.4</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>13</td>
<td></td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>14</td>
<td></td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

11/1/2006
Page Tables for Example

Process A

0 0
1 1
2 2
3 3

Process B

0 ---
1 ---
2 ---

Process C

0 7
1 8
2 9
3 10

Process D

0 4
1 5
2 6
3 11
4 12

Free Frame List

13
14

11/1/2006
Segmentation

• All segments of all programs do not have to be of the same length
• There is a maximum segment length
• Addressing consist of two parts - a segment number and an offset
• Since segments are not equal, segmentation is similar to dynamic partitioning