Memory Management Requirements

- **Relocation**
  - Easily swap process (in part or entirely) in and out of memory
- **Protection**
  - Process should be protected by unauthorized access
- **Sharing**
  - Allow processes to share data and programs in memory
- **Support for Program Modules**
  - Treat program modules as units (e.g., segmentation)
- **Effective Memory Usage**
  - Keep most active parts of a process in main memory
  - Store the rest on secondary storage

Unix Process Memory Layout

- **Low Address**
  - Text
  - Initialized Data
  - Uninitialized Data (BSS)
  - Heap
  - Stack
- **High Address**
  - argv[ ], env[ ]

This “logical address layout” gets mapped (bound) to physical addresses during loading.

Memory Address Binding Time

- **Programming Time**
  - Actual physical address is specified by programmer in the program itself
- **Compile Time**
  - Recompile if starting location changes
- **Load Time**
  - Need to generate *relocatable* code
- **Execution Time**
  - For efficiency, need hardware support if code will be moved during execution
  - Relocation register(s) or Virtual memory system
Program Linking and Loading

- **Module 1**
- **...**
- **Module n**
- **Library**
- **Linker**
- **Load Module**
- **Loader**
- **Memory**
- **Program Linking**
- **Program Image**
- **Object File**
  - Partial Program Image
  - **main( )**
  - `double x = 3.0;` `int y;`
  - `y = (int)pow(x, 4.0);`
  - `printf("y = %d\n", y);`
  - `exit(0);`

- **Linking**
  - Static
    - `ld -static x.o -lm`
    - `OR gcc -static x.c -lm`
  - Dynamic (default)
    - `gcc x.c` // compile and link
    - `pow, printf, exit already in memory`

- **Library**
- **Linker**
- **Loader**
- **Memory**
- **Program Image**

Virtual Memory System

- Memory address mapping occurs for every instruction at run-time
  - Physical (actual) address = f(logical address)
  - Typically, f( ) is implemented as a page table

- Basic Ideas
  - Hide details of real physical memory from user
  - Each user has n contiguous (linear) address spaces
    - Each begins at address 0
    - Paging (n = 1) versus Segmentation (n ≥ 1)

Simple Paging

- Basic Idea
  - Break processes into equal sized pages (e.g., 2^{10} bytes)
  - Break real memory into the same sized page frames
  - Load process pages into frames
    - Not necessarily contiguous frames
    - Some internal fragmentation (last page), but no external fragmentation

- Like fixed partitioning, but:
  - Partitions are small
  - A process can occupy more than 1 partition
  - Partitions need not be contiguous
**Simple Paging Example**

<table>
<thead>
<tr>
<th>Frame</th>
<th>A.0</th>
<th>A.1</th>
<th>B.0</th>
<th>B.1</th>
<th>C.0</th>
<th>D.0</th>
<th>D.1</th>
<th>D.2</th>
<th>D.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>A.0</td>
<td>A.1</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>
</tr>
<tr>
<td>3</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>
</tr>
<tr>
<td>5</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>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Address Translation (Paging)**

```
<table>
<thead>
<tr>
<th>Logical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset</td>
</tr>
<tr>
<td>Page #</td>
</tr>
</tbody>
</table>
```

```
| 0000101110111110 |
```

**Address Translation (Segmentation)**

```
<table>
<thead>
<tr>
<th>Logical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset</td>
</tr>
<tr>
<td>Segment #</td>
</tr>
</tbody>
</table>
```

```
| 0001001011110000 |
```

**Protection in Paging Systems**

```
<table>
<thead>
<tr>
<th>Virtual Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Address</td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>page#</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>frame#</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
</tr>
</tbody>
</table>
```

- read-only
- Text, Constants
- executable
- Text
- writeable
- Heap, Stack

---

- Process Page Table:
- Page Table Address
- Access Privileges (rwx)
- Program
- Paging Hardware
- Memory

- Washington University in St. Louis
Memory Allocation in the Kernel

- Not as easy as user-space memory allocation
- Factors
  - Kernel can not sleep
  - Kernel can not easily deal with memory errors
  - Need lightweight allocation scheme
- Pages
  - Physical page: basic unit of memory management
    - Memory Management Unit (MMU)
      - Hdw that manages memory
      - Maps virtual addresses to physical addresses
    - MMU deals in pages
    - One page descriptor for each physical page
      - 4 GB addr space and 4 KB pages → 1 M descriptors

Memory Management in Linux

- Linux process (32-bit machine)
  - 3 GB virtual address space
  - +1 GB reserved for page tables and other data
    - only visible when in kernel mode
  - Memory is allocated by kernel as needed
- Hardware limitations → memory zones
  - ZONE_DMA pages for DMA operations
    - some devices can DMA only to certain addresses
      - on x86 architectures, < 16 MB
  - ZONE_NORMAL normally mapped pages
  - ZONE_HIGHMEM high-memory addresses, not permanently mapped
    - on x86 architectures, all physical addresses above 896MB
- Actual layout of zones: architecture dependent

Linux Memory Allocation Mechanisms

- Page allocator (Buddy algorithm)
  - Main kernel memory allocation mechanism
  - Allocates physically contiguous pages of memory
  - No external fragmentation
  - But can have internal fragmentation
    - unused space in allocated chunks
- Slab allocator
  - Built on top of page allocator
  - Addresses internal fragmentation
- vm allocator
  - Used when memory need not be physically contiguous
    - allocates virtually contiguous pages
Buddy System (Tree Representation)

- Simple, fast
- Good for allocating contiguous memory blocks
- Subject to internal fragmentation

Free Lists:
- 16MB
- 8MB
- 4MB
- 2MB

Buddy System Example

<table>
<thead>
<tr>
<th>16 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>E</td>
</tr>
</tbody>
</table>