CIS 314
Introduction to Virtual Memory

Prof. Michel A. Kinsky
Application Compiling Process

- **C Language**

  Human Readable
  
  C program
  
  compiler
  
  assembly code
  
  assembler
  
  object code
  
  library routines
  
  linker
  
  executable
  
  loader
  
  memory

Machine Code
Running a Program

- Operating System (loader) copies a program from permanent storage into RAM
- For PCs and workstations, the OS copies the program (bits) from disk
Running a Program

• Operating System (loader) copies a program from permanent storage into RAM

• For PCs and workstations, the OS copies the program (bits) from disk

• The CPU’s Program Counter is then set to the starting address of the program and the program begins execution
What If Program Does not Fit?

- Some machines will not let you run the program
  - Original DOS
Use of Overlays

- Programmer divides the code into pieces that fit into RAM
- Pieces, called overlays, are loaded and unloaded by the program
- Does not require OS help
Use of Overlays

Load one overlay and run until other overlay is needed.
Use of Overlays

- Programmer divides the code into pieces that fit into RAM
- Pieces, called overlays, are loaded and unloaded by the program
- Does not require OS help
- Problems with overlays
  - Difficult for programmer to manage
Virtual Memory

• Virtual memory
  ‣ Technique that allows execution of a program that may not completely reside in memory (RAM)
  ‣ Allows the computer to “fake” a program into believing that its memory space is larger than physical RAM

• Importance of virtual memory
  ‣ Provides illusion of very large memory
  ‣ Sum of the memory of many jobs greater than physical memory
  ‣ Address space of each job larger than physical memory
Virtual Memory

• Virtual memory
  ‣ Technique that allows execution of a program that may not completely reside in memory (RAM)

• Importance of virtual memory
  ‣ Allows available (fast and expensive) physical memory to be very well utilized
  ‣ Simplifies memory management (main reason today)
  ‣ Removes burden of memory resource management from the programmer
Virtual Memory

• Two memory “spaces”
  ‣ Virtual memory space what the program “sees”
  ‣ Physical memory space what the program runs in (size of RAM)

• On program startup
  ‣ OS copies program into RAM
  ‣ If there is not enough RAM, OS stops copying program and starts it running with only a portion of the program loaded in RAM
Virtual Memory

• On program startup
  ‣ OS copies program into RAM
  ‣ If there is not enough RAM, OS stops copying program and starts it running with only a portion of the program loaded in RAM
  ‣ When the program touches a part of the program not in physical memory (RAM), OS catches the memory abort (called a page fault) and copies that part of the program from disk into RAM
  ‣ In order to copy some of the program from disk to RAM, OS must evict parts of the program already in RAM
  • OS copies the evicted parts of the program back to disk
Virtual Memory

Physical Memory

```
0x00  add r1, r2, r3
0x04  sub r2, r3, r4
0x08  lw r2, 0x04
0x0C  mult r3, r4, r5
0x10  bne 0x00
0x18  add r10, r1, r2
0x1C  sub r3, r4, r1
       sw r5, 0x0c
```

```
0x00  add r1, r2, r3
0x04  sub r2, r3, r4
0x08  lw r2, 0x04
0x0C  mult r3, r4, r5
0x10  add r1, r2, r3
0x14  sub r2, r3, r4
0x18  lw r2, 0x04
0x1C  mult r3, r4, r5
```
Virtual Memory

Translation Table

Virtual Memory

0x00  add r1, r2, r3
0x04  sub r2, r3, r4
0x08  lw r2, 0x04
0x0C  mult r3, r4, r5
0x10  bne 0x00
0x14  add r10, r1, r2
0x18  sub r3, r4, r1
0x1C  sw r5, 0x0c

Physical Memory

0x00  add r1, r2, r3
0x04  sub r2, r3, r4
0x08  lw r2, 0x04
0x0C  mult r3, r4, r5
0x10  Disk
0x14  Disk
0x18  Disk
0x00 0x00
0x04 0x04
0x08 0x08
0x0C 0x0C
0x10 Disk
0x14 Disk
0x18 Disk
0x10
0x14
0x18
Disk

Computer Architecture and Embedded Systems Laboratory (CAES Lab)
Page Fault

Virtual Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>add r1, r2, r3</td>
</tr>
<tr>
<td>0x04</td>
<td>sub r2, r3, r4</td>
</tr>
<tr>
<td>0x08</td>
<td>lw r2, 0x04</td>
</tr>
<tr>
<td>0x0C</td>
<td>mul r3, r4, r5</td>
</tr>
<tr>
<td>0x10</td>
<td>bne 0x00</td>
</tr>
<tr>
<td>0x14</td>
<td>add r10, r1, r2</td>
</tr>
<tr>
<td>0x18</td>
<td>sub r3, r4, r1</td>
</tr>
<tr>
<td>0x1C</td>
<td>sw r5, 0x0c</td>
</tr>
</tbody>
</table>

Physical Memory

Translation Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>bne 0x00</td>
</tr>
<tr>
<td>0x04</td>
<td>add r1, r2, r3</td>
</tr>
<tr>
<td>0x08</td>
<td>lw r2, 0x04</td>
</tr>
<tr>
<td>0x0C</td>
<td>mul r3, r4, r5</td>
</tr>
<tr>
<td>0x10</td>
<td>bne 0x00</td>
</tr>
<tr>
<td>0x14</td>
<td>add r10, r1, r2</td>
</tr>
<tr>
<td>0x18</td>
<td>sub r3, r4, r1</td>
</tr>
<tr>
<td>0x1C</td>
<td>sw r5, 0x0c</td>
</tr>
</tbody>
</table>
Program View

- Program asks for virtual address
- Computer translates virtual address (VA) to physical address (PA)
- Computer reads PA from RAM, returning it to program
Program View

Virtual Memory  Translation Table  Physical Memory

CPU

Virtual address

Instructions (or data)

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Translation Table</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>add r1,r2,r3</td>
<td>0x00</td>
</tr>
<tr>
<td>0x04</td>
<td>sub r2,r3,r4</td>
<td>0x04</td>
</tr>
<tr>
<td>0x08</td>
<td>lw r2, 0x04</td>
<td>0x08</td>
</tr>
<tr>
<td>0x0c</td>
<td>mult r3,r4,r5</td>
<td>0x0c</td>
</tr>
<tr>
<td>0x10</td>
<td>Disk</td>
<td>Disk</td>
</tr>
<tr>
<td>0x14</td>
<td>Disk</td>
<td>Disk</td>
</tr>
<tr>
<td>0x18</td>
<td>Disk</td>
<td>Disk</td>
</tr>
<tr>
<td>0x1c</td>
<td>Disk</td>
<td>Disk</td>
</tr>
</tbody>
</table>

Disk
Next Class

- Memory Mapped I/O and Interrupt Handling