CSCI-GA.2250-001

Operating Systems

Lecture 7: Memory Management III

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
Summary of Paging

• Virtual address space bigger than physical memory
• **Mapping** virtual address to physical address
• Virtual address space divided into fixed-size units called **pages**
• Physical address space divided into fixed-size units called pages **frames**
• Virtual address space of a process can be non-contiguous in physical address space
Paging

- MMU
- OS Involvement
Paging

MMU

OS Involvement

The CPU sends virtual addresses to the MMU

The MMU sends physical addresses to the memory
OS Involvement With Paging

• When a new process is created
• When a process is scheduled for execution
• When process exits
• When page fault occurs
OS Involvement With Paging

• When a new process is created
  – Determine how large the program and data will be (initially)
  – Create page table
  – Allocate space in memory for page table
  – Record info about page table and swap area in process table

• When a process is scheduled for execution

• When process exits

• When page fault occurs
OS Involvement With Paging

• When a new process is created
• When a process is scheduled for execution
• When process exits
• When page fault occurs
OS Involvement With Paging

- When a new process is created
- When a process is scheduled for execution
  - MMU reset for the process
  - TLB flushed
  - Process table made current
- When process exits
- When page fault occurs
OS Involvement With Paging

• When a new process is created
• When a process is scheduled for execution
  – When process exits
    – OS releases the process page table
    – Frees its pages and disk space
• When page fault occurs
OS Involvement With Paging

- When a new process is created
- When a process is scheduled for execution
- When process exits
- When page fault occurs
Page Fault Handling

1. The hardware:
   - Saves program counter
   - Traps to kernel

2. An assembly routine saves general registers and calls OS

3. OS tried to discover which virtual page is needed

4. OS checks address validation and protection and assign a page frame (page replacement may be needed)
Page Fault Handling

5. If page frame selected is dirty
   - Page scheduled to transfer to disk
   - Frame marked as busy
   - OS suspends the current process
   - Context switch takes place

6. Once the page frame is clean
   - OS looks up disk address where needed page is
   - OS schedules a disk operation
   - Faulting process still suspended

7. When disk interrupts indicates page has arrived
   - OS updates page table
Page Fault Handling

8. Faulting instruction is backed up to its original state before page fault and PC is reset to point to it.

9. Process is scheduled for execution and OS returns to the assembly routine.

10. The routine reloads registers and other state information and returns to user space.
Instruction Backup At Page Fault

• In order to restart the instruction, the OS needs to know where the first byte of the instruction is.

```
MOVE.L #6(A1), 2(A0)
```

The value of the PC at the time of trap depends on which operand faulted and the CPU microcode.
Instruction Backup At Page Fault

- In order to restart the instruction, the OS needs to know where the first byte of the instruction is.

```
MOVE.L #6(A1), 2(A0)
```

The value of the PC at the time of trap depends on which operand faulted and the CPU microcode.

The CPU designers usually saves PC in some internal register for the OS.
Interesting Scenario: Virtual Memory & I/O Interaction

- Process issues a syscall to read a file into a buffer
- Process suspended while waiting for I/O
- New process starts executing
- This other process gets a page fault
- If paging algorithm is global there is a change the page containing the buffer be removed from memory.
- The I/O operation of the first process will write some data into the buffer and some other on the just-loaded page!
Interesting Scenario: Virtual Memory & I/O Interaction

- Process issues a syscall to read a file into a buffer
- Process suspended while waiting for I/O
- New process starts executing
- This other process gets a page fault
- If paging algorithm is global there is a chance the page containing the buffer be removed from memory.
- The I/O operation of the first process will write some data into the buffer and some other on the just-loaded page!

One solution: Locking (pinning) pages engaged in I/O so that they will not be removed.
Backings Store

• Swap area: does not a normal file system on it.
• Associated with each process the disk address of its swap area; store in the process table
• Before process starts swap area must be initialized
  – One way: copy all process image into swap area [static swap area]
  – Another way: don’t copy anything and let the process swap out [dynamic]
• Instead of disk partition, one or more preallocated files within the normal file system can be used [Windows uses this approach.]
Backing Store

Static Swap Area

Dynamic
Real Life Program: A Compiler

Virtual address space

- Call stack
- Parse tree
- Constant table
- Source text
- Symbol table

Address space allocated to the parse tree

Free

Space currently being used by the parse tree

Symbol table has bumped into the source text table
Wouldn’t it be much easier to have separate address spaces for each type?
Segmentation

- Provide the machine with many completely independent address spaces
- Each segment consists of linear sequence of addresses.
- Different segments can have different lengths.
- Segment length may change during execution.
- A segment is a logical entity.
- Programmer is aware of segments.
One dimensional address space

Segmented memory
External Fragmentation (Checkerboarding)
<table>
<thead>
<tr>
<th>Consideration</th>
<th>Paging</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need the programmer be aware that this technique is being used?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>How many linear address spaces are there?</td>
<td>1</td>
<td>Many</td>
</tr>
<tr>
<td>Can the total address space exceed the size of physical memory?</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can procedures and data be distinguished and separately protected?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Can tables whose size fluctuates be accommodated easily?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Is sharing of procedures between users facilitated?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Why was this technique invented?</td>
<td>To get a large linear address space without having to buy more physical memory</td>
<td>To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection</td>
</tr>
</tbody>
</table>
Segmentation with Paging

- It may be difficult to keep full segments in memory at once
- Paging segments can help
- Example from real life
  - MULTICS
  - Intel Pentium
MULTICS

• MULtiplexed Information & Computing Service (MULTICS)
• Time Sharing OS (1965 - 2000)
• Provides each program with up to $2^{18}$ segments
• Each segment up to 65,536 words (word size is 36 bits in this system)
• Each program has a segment table with one descriptor per table
MULTICS

• Segment descriptor contains indication whether segment is in memory or not
• If segment is in memory, the descriptor contains 18-bit pointer to its page table
MULTICS

Virtual address
MULTICS

Virtual address
MULTICS

MULTICS virtual address

- Segment number
- Page number
- Offset

Diagram:
- Segment number
- Descriptor segment
- Descriptor
- Page number
- Page frame
- Page table
- Page
- Offset
The Intel Pentium

- 16K independent segments
- Each can hold up to 1 billion 32-bit word
- The heart of the system consists of two tables:
  - LDT (Local Descriptor Table): each program has its own LDT
  - GDT (Global Descriptor Table): for system segments
- To access a segment, a Pentium program first loads a selector for that segment into one of the machine’s six segment registers
The Intel Pentium

- A the time the selector is loaded into a segment register, the corresponding descriptor is fetched from LDT or GDT and stored in microprogram registers.
The Intel Pentium

Address = (Segment, Offset)

Selector

Descriptor

Base address

Limit

Other fields

Offset

+ 32-Bit linear address
The Intel Pentium

Each running program has a page directory.
Conclusions

• We are done with Chapter 3
• Main goal
  – Provide CPU with illusion of large and fast memory
• Constraints
  – Speed
  – Cost
  – Protection
  – Transparency
  – Efficiency
• Memory management
  – Paging
  – Segmentation
  – Paged segments