Virtual Memory

- Virtual Memory (VM) basic concepts
- The Page Table
- The Translation Lookaside Buffer (TLB)
Virtual memory

“program too big to fit in memory”

• what if a program is larger than the memory available?

• what if many processes run at the same time?

IDEA: hard disks are much larger than the main memory. Why not use the hd as a main memory, and the main memory as a cache?
A single user program can be larger than the main memory:

Virtual memory

MAIN MEMORY:

DISK:

V. ADD SPACE:
Virtual memory

Many processes (including the OS) can be run “at the same time”, as if the memory were unlimited:
Virtual memory

Virtual vs. physical addressing

- programs are compiled with their own virtual addressing space
- processes are run from memory — virtual addresses are translated into physical addresses
Virtual memory

The Virtual Memory Manager

- part of the OS
- manages the memory mapping (also called address translation) between virtual and physical addressing space for all processes
Terminology:

**page**: minimum amount of information allocated and/or transferred  
(analagous to a cache *block*)

**page fault**: the page containing the memory location specified by the CPU  
is not present in memory and must be fetched from the disk (analagous to  
a cache miss)

**relocation**: virtual addresses can be mapped to any location in main  
memory

**virtual addressing space**: the addressing space relative to a program —  
only depends on the instruction set architecture

**physical addressing space**: depends on the actual amount of physical  
memory in the system
Virtual memory

Page number and page offset

Both virtual and physical addresses are broken down into a page number and a page offset.

Example (fig 7.21): 4 GB virtual addressing space, 1 GB physical addressing space, 4 KB page size
In virtual memory systems

Typically:

- pages should be large enough to amortize the high access time
- fully associative placement of pages in memory
- page faults are handled in software
- sophisticated LRU replacement policy
- consistency maintained using write-back (here called *copy back*)
Virtual memory

Keeping track of pages: the page table

- all programs use the same virtual addressing space
- each program must have its own memory mapping

Each program has its own page table to map virtual addresses to physical addresses.

The page table resides in memory, and is pointed to by the page table register.

The page table has an entry for every possible page (in principle, not in practice...), no tags are necessary.

A valid bit indicates whether the page is in memory or on disk.
Example (fig 7.22): 4 GB virtual address space, 1 GB physical memory, 4 KB page size.
Virtual memory

- bits for page offset
- bits for virtual page number
- number of virtual pages
- entries in the page table
- bits for physical page number
- number of physical pages
- bits per page table line
- total page table size
Virtual memory

The page table

\[ \begin{array}{c|c|c|c|c|c|c} \hline V & 1 & 1 & 1 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ \hline \end{array} \]
Page faults

What happens on a page fault?

- the CPU must save the state of the process and transfer control to the OS
- the OS must look up the page table to locate the page on the disk
- unless there are unused pages, the OS must choose a page to replace
- if the chosen page is dirty, it must be written back
- read the new page from disk
- update the page table and the TLB
- restart the instruction
Page replacement policy

Exact Least Recently Used (LRU) is expensive.

Approximate LRU:

- a *use bit* (or *reference bit*) is added to every page table line
- the bit is set at every access
- the OS periodically clears all use bits
- the page to replace is chosen among the ones with their *use bit* at zero
The Translation Lookaside Buffer (TLB)

Page tables are stored in the main memory.
Every memory access would take twice as long:

- one access to get the physical address
- one access to get the data

The TLB: a cache for page tables (only holds memory mappings).

Based on the same principle of temporal locality as the cache memory.

The TLB is a cache, so it needs a valid bit, a tag field, and a dirty bit (for write-back).
Virtual memory

Virtual memory with TLB

**TLB**

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>TAG</th>
<th>P. P. Add</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PAGE TABLE**

<table>
<thead>
<tr>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

**PHYSICAL MEMORY**

**DISK**
Virtual memory, TLB, and cache

Example:
4 GB virtual addressing space; 1 GB physical memory; 4 KB pages; 32-entry, direct-mapped write-back TLB; 32 KB, direct mapped write-back cache with 1-word data block. The cache is \textit{physically addressed} and \textit{physically tagged}.

- bits for page offset:
- bits for virtual page number:
- bits for physical page number:
- bits for TLB tag:
- bits for cache index:
- bits for cache tag:
### Virtual Memory

**Virtual address:**

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>TLB TAG</th>
<th>Physical p. number</th>
</tr>
</thead>
</table>

**TLB:**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Physical address:**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Cache:**

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>CACHE TAG</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Virtual memory

Context switching and TLB

Every time the CPU, in a multi-tasking system, changes the running process, the TLB becomes inconsistent, since all processes use the same virtual addresses.

Solution 1: clear ...

Solution 2: add ...
Protection and privacy

A system with VM also provides protection and privacy in a multi-tasking environment, by preventing programs from:

- reading from another program’s space
- writing into another program’s space
- messing with the OS

How?
Virtual memory

- virtual pages of a process never map onto another process’ pages
- processes can NOT modify their own page table
- the CPU must support two operating modes: \textit{user mode} and \textit{supervisor mode}
- only the OS can use the CPU in supervisor mode
  - from user to supervisor: \textit{system call}
  - from supervisor to user: \textit{return from exception}


Exercise

A system has 26-bit physical addresses, 32-bit virtual addresses, 1 KB pages, and a 2-way set-associative TLB with 16 sets.

- diagram a virtual address

- diagram a physical address

- diagram a TLB lookup

- diagram a TLB line, assuming a 6-bit PID field is included in each entry
Recommended exercises

• 7.32, 7.33, 7.34, 7.35, 7.36