Virtual Memory (virtuaalimuist)

- Problem: How can I make my (main) memory as big as my disk drive?
- Answer: Virtual memory
  - keep only most probably referenced data in memory, and rest of it in disk
  - disk is much bigger and slower than memory
  - address in machine instruction may be different than memory address
  - need to have efficient address mapping
  - most of references are for data in memory
- joint solution with HW & SW

Other Problems Often Solved with VM

- If you want to have many processes in memory at the same time, how do you keep track of memory usage?
- How do you prevent one process from touching another process’ memory areas?
- What if a process needs more memory than physically available?

Memory Management Problem

- How much memory for each process?
  - Is it fixed amount during the process run time or can it vary during the run time?
- Where should that memory be?
  - In a continuous or discontinuous area?
  - Is the location the same during the run time or can it vary dynamically during the run time?
- How is that memory managed?
- How is that memory referenced?

Partitioning

- How much physical memory for each process?
- Static (fixed) partitioning (kiinteät partitiot, kiinteä ositus)
  - Amount of physical memory determined at process creation time
  - Continuous memory allocation for partition
- Dynamic partitioning (dynaamiset partitiot)
  - Amount of physical memory given to a process varies in time
    - Due to process requirements (of this process)
    - Due to system (i.e., other processes) requirements
**Static Partitioning**

- Equal size - give everybody the same amount
- Fixed size - big enough for everybody
  - too much for most
- Need more? Can not run!
- Unequal size
  - sizes predetermined
  - Can not combine
- Variable size
  - Size determined at process creation time

**Dynamic Partitioning**

- Process must be able to run with varying amounts of main memory
  - all of memory space is not in physical memory
  - need some minimum amount of physical memory
- New process?
  - If necessary reduce amount of memory for some (lower priority) processes
- Not enough memory for some process?
  - reduce amount of memory for some (lower priority) processes
  - kick (swap) out some (lower priority) process

**Fragmentation (pirstoutuminen)**

- Internal fragmentation (sisäinen pirstoutuminen)
  - unused memory inside allocated block
    - e.g., equal size fixed memory partitions
- External fragmentation (ulkoinen pirstoutuminen)
  - enough free memory, but it is splintered as many un-allocatable blocks
    - e.g., unequal size partitions or dynamic fixed size (variable size) memory partitions

**Address Mapping (osoitteen muunnos)**

Pascal, Java:

while (....)

X := Y+Z;

Compiler

Symbolic Assembly Language:

loop: LOAD R1, Y

ADD R1, Z

STORE R1, X

(Textual) machine language:

1312: LOAD R1, 2510

ADD R1, 2514

STORE R1, 2600

(addresses relative to 0)

Execution time:

101312: LOAD R1, 2510

or

101312: LOAD R1, 102510

+100000?

Execution time:

101312: LOAD R1, 2510

or

101312: LOAD R1, 102510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?

Execution time:

101312: LOAD R1, 102510

or

101312: LOAD R1, 2510

+100000?
Address Mapping, address translation

- At program load time
  - Loader (lataaja)
  - Static address binding (staattinen osoitteiden sidonta)
- At program execution time
  - CPU
  - With every instruction
  - Dynamic address binding (dynaaminen osoitteiden sidonta)
  - Swapping (heitotavoitinta)
  - Virtual memory

Virtual Memory Implementation

- Methods
  - Base and limit registers (kanta- ja rajarekisterit)
  - Segmentation (segmentointi)
  - Paging (sivutus)
  - Segmented paging, multilevel paging
- Hardware support
  - MMU - Memory Management Unit
    - Part of processor
    - Varies with different methods
  - Sets limits on what types of virtual memory (methods) can be implemented using this HW

Base and Limit Registers

- Continuous memory partitions
  - One or more (4?) per process
  - May have separate base and limit registers
    - Code, data, shared data, etc
    - By default, or given explicitly in each mem. ref.
- BASE and LIMIT registers in MMU
  - All addresses logical in machine instructions
  - Exec. time address mapping for address (x):
  - Check: $0 \leq x < \text{LIMIT}$
  - Physical address: $\text{BASE} + x$

Virtual memory

- Only needed reserved areas (chunks) in the memory, no need to be contiguous
- Demand paging (tarvenouto)
- Chunk size?
  - Fixed size = Paging
  - Variable size = Segmentation
  - Combined = Paged segments, multilevel paging
- OS bookkeeping (KJ kirjanpito)
  - Page frame table
    - Which physical page frames are free, which are occupied
  - Each process has its own page table
    - Is this page in memory or on disk? -- “presence-bit”
    - In memory, which physical frame contains this logical page?
    - Other control? Bits: Modified, Referenced

Virtual Memory: Paging (sivutus)

- OS loads process A from disk
- OS course content

Address Mapping with Paging VM

(Sta10 Fig 8.15)

(Sta10 Fig 8.16)
Virtual memory: Translation Lookaside Buffer (TLB) (osoitteenmuunnospuskuri)

- Address translation for each memory reference, at least once for each instruction
- Page table elements in memory = extra memory access?
  - Too slow!
- Solution
  - Principle of locality! Page table element referenced soon again
  - Store recently used page table elements (of this process) in TLB
  - TLB, translation lookaside buffer
  - Just like cache
  - Fast set of registers (Pentium: 32 registers)
  - Associative search
  - Hit ratio (Osumatodennäköisyys) 99.9% ? (Almost always?)

Translation Lookaside Buffer (TLB)

- "Hit" on TLB?
  - address translation is in TLB - real fast
- "Miss" on TLB?
  - must read page table entry from memory
  - takes time = not much, just a memory reference
  - Entry might be in cache!
    - cpu waits idle until it is done
  - Just like normal cache, but for address mapping
  - implemented just like cache
  - instead of cache line data have physical address
  - split TLB? 1 or 2 levels?
Virtual Memory Support Ops

- Hardware support: MMU and its special registers
  - PTR (page table register)
    - Physical start address of process page table (copied from PCB – process control block)
  - TLB (translation lookaside buffer)
    - Caches page table entries from earlier address mappings
    - “Page fault” – Interrupt
    - Updating reference and modified bits
  - Process switch (process switch)
    - PTR register
      - Physical start address of process page table
    - Invalidate old TLB content (it is usually process specific)
      - Each location has valid bit
    - Copy dirty cache lines back to memory

---

TLB and Cache

- Page table entry can be found from cache!

TLB Operation

- Virtual Address
  - Page Table
  - TLB
  - Cache
  - Main Memory
  - Unit

<table>
<thead>
<tr>
<th>TLB Misses vs. Page Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TLB Miss</strong></td>
</tr>
<tr>
<td>CPU waits idling</td>
</tr>
<tr>
<td>HW implementation</td>
</tr>
<tr>
<td>Data is copied from memory to TLB (or from cache)</td>
</tr>
<tr>
<td>Delay 1-8 (?) clock cycles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Page Fault</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Process is suspended and cpu executes some other processes</td>
</tr>
<tr>
<td>SW implementation</td>
</tr>
<tr>
<td>Data is copied from disk to memory</td>
</tr>
<tr>
<td>Delay 10-30 ms(?)</td>
</tr>
</tbody>
</table>

---

Inverted page table (käänteinen sivutaulu)

- Just one shared inverted page table
- MMU: PTR (page table reg), PidR (process id register), TLB

---

Memory/Disk Organisation

- CPU
  - instr
  - regs
  - TLB

- Memory
  - page table

- Disk
  - page table

---

TLB vs. Cache

- TLB Miss
  - CPU waits idling
  - HW implementation
  - Invisible to process
  - Data is copied from memory to TLB
    - from page table data
    - from cache?
    - Delay 4 (or 2 or 8?) clock cycles

- Cache Miss
  - CPU waits idling
  - HW implementation
  - Invisible to process
  - Data is copied from memory to cache
    - from page data
  - Delay 4 (or 2 or 8?) clock cycles

---

Inverted page table

- Just one shared inverted page table
- MMU: PTR (page table reg), PidR (process id register), TLB
Hierarchical page table
(monitasoinen sivutaulu)
- Several systems allow large virtual address space
- Page table split to pages, some of it on the disk
- Top level of page table fits to one page, always in memory

Virtual Memory Policies
- Fetch policy (noutopolitiikka)
  - demand paging: fetch page only when needed 1st time
  - working set: keep all needed pages in memory
  - prefetch: guess and start fetch early
- Placement policy (sijoituspolitiikka)
  - any frame for paged VM
- Replacement policy (poistopolitiikka)
  - local, consider pages just for this process for replacement
  - global, consider also pages for all other processes
  - dirty pages must be written to disk (likaset, muutetut sivut)

Virtual Memory Example
Pentium (IA-32)

Pentium Address Translation

Virtual Memory Policies
- Fetch policy (noutopolitiikka)
  - demand paging: fetch page only when needed 1st time
  - working set: keep all needed pages in memory
  - prefetch: guess and start fetch early
- Placement policy (sijoituspolitiikka)
  - any frame for paged VM
- Replacement policy (poistopolitiikka)
  - local, consider pages just for this process for replacement
  - global, consider also pages for all other processes
  - dirty pages must be written to disk (likaset, muutetut sivut)

Virtual Memory Example
Pentium (IA-32)

Pentium Address Translation

Virtual Memory Policies
- Fetch policy (noutopolitiikka)
  - demand paging: fetch page only when needed 1st time
  - working set: keep all needed pages in memory
  - prefetch: guess and start fetch early
- Placement policy (sijoituspolitiikka)
  - any frame for paged VM
- Replacement policy (poistopolitiikka)
  - local, consider pages just for this process for replacement
  - global, consider also pages for all other processes
  - dirty pages must be written to disk (likaset, muutetut sivut)
Pentium: Page Table (sivutaulu)

Virtual Memory Example

ARM Memory System Overview

ARM Virtual Memory Address Translation for Small Pages - Diagram

ARMv6 Memory Management Formats

Hennessy-Patterson/Computer Architecture, Fig 5.47 Alpha AXP
Virtual Memory Summary

- How to partition physical memory for processes?
  - Fixed partitions (various methods)
  - Dynamic partitions: segments, pages
- Paged virtual memory
  - Multilevel page tables
- How to translate addresses?
  - TBL, multi-level TLB
- How does TBL work with cache?
- Examples: Intel & ARM

Review Questions

- What hardware support is needed for virtual memory implementation?
- Differences of paging and segmentation?
- Why to combine paging and segmentation?
- Relationship of TLB and cache?
  - similarities, differences?