The Formal Verification of a Distributed Realtime System

Steffen Knapp Wolfgang Paul
{sknapp, wjp}@wjpserver.cs.uni-sb.de

Institute for Computer Architecture and Parallel Computing,
Saarland University

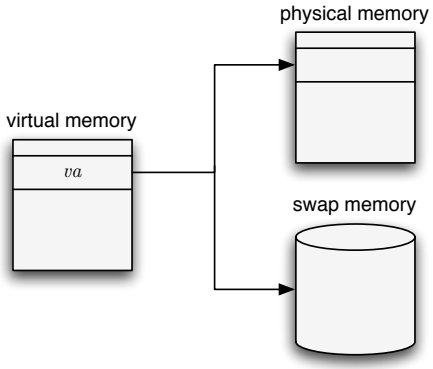
Lecture 6: Memory Management

March 19, 2007

Overview

Virtual memory simulation:

- applications require large address space (up to 2^{32} bytes)
- not enough physical memory (too expensive)
- use cheap but slow disks as swap memory
- enforce process separation via address translation

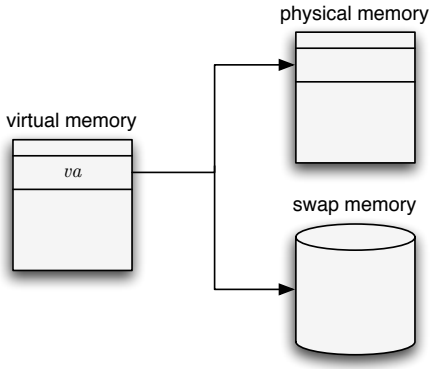


Overview

Virtual memory simulation:

- applications require large address space (up to 2^{32} bytes)
- not enough physical memory (too expensive)
- use cheap but slow disks as swap memory

- enforce process separation via address translation

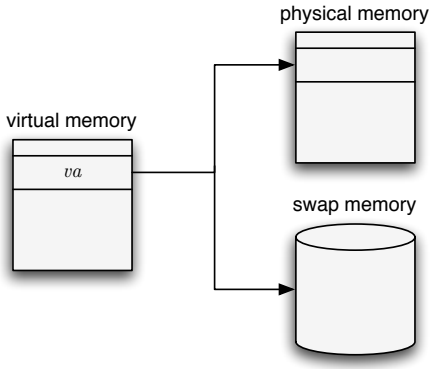


Overview

Virtual memory simulation:

- applications require large address space (up to 2^{32} bytes)
- not enough physical memory (too expensive)
- use cheap but slow disks as swap memory

- enforce process separation via address translation



Guideline

1. Physical Machines
   - Configuration
   - Address Translation
2. Virtual Memory Simulation
   - Correctness Theorem

Guideline

1. Physical Machines
   - Configuration
   - Address Translation
2. Virtual Memory Simulation
   - Correctness Theorem

Physical Machine Configuration

Physical machine configuration:

- processor registers $d.R$:
  - 'old' register (gpr, sr, eca, ...)
  - mode register $mode$
  - page table origin pto
  - page table length ptl
- the physical memory $d.m$
- the swap memory $d.sm$

Operating Modes

Behavior depending on $mode$ register:

- system mode $d.mode = 0$:
  d operates like basic processor model with extra registers
- user mode $d.mode = 1$:
  d emulates basic processor model using address translation

Naming conventions:

- denote the configuration of the physical machine by d
- machines simulated in user mode are called virtual machines
- denote configurations of virtual machines by vm
- addresses used by virtual machines are called virtual addresses va

Operating Modes

Behavior depending on $mode$ register:

- system mode $d.mode = 0$:
  d operates like basic processor model with extra registers
- user mode $d.mode = 1$:
  d emulates basic processor model using address translation

Naming conventions:

- denote the configuration of the physical machine by d
- machines simulated in user mode are called virtual machines
- denote configurations of virtual machines by vm
- addresses used by virtual machines are called virtual addresses va

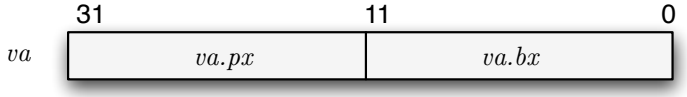
Virtual Addresses

Aim:

- define the translated physical memory address $pma(d, va)$

Virtual addresses va:

- pages size is $4K = 2^{12}$ bytes
- page index $va.px = va[31 : 12]$
- byte index $va.bx = va[11 : 0]$

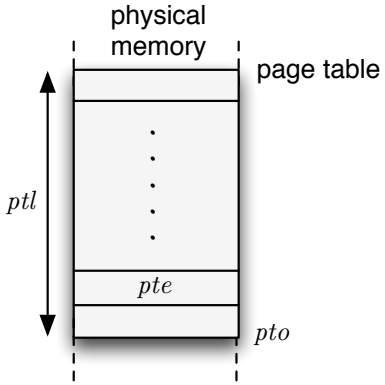


$$va = (va.px \circ va.bx)$$

Page Table Entries

Page table:

- region of the physical memory
- starting at page table origin $d.pto \cdot 4K$
- having $d.ptl$ entries of four bytes



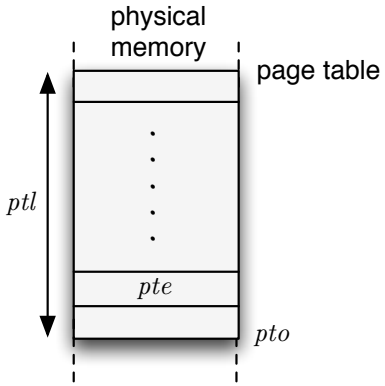
Address translation via page tables:

- page table entry address ($ptea$):
  - $ptea(d, va) = d.pto \cdot 4K + 4 \cdot va.px$
- page table entries (pte):
  - $pte(d, va) = d.m_4(ptea(d, va))$
- physical page index (ppx):
  - $ppx(d, va) = pte(d, va)[31 : 12]$
- valid bit (v):
  - $v(d, va) = pte(d, va)[11]$

Page Table Entries

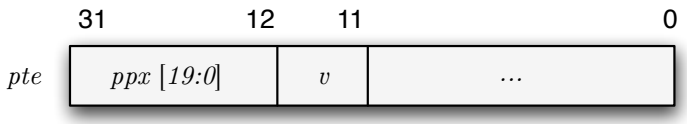
Page table:

- region of the physical memory
- starting at page table origin $d.pto \cdot 4K$
- having $d.ptl$ entries of four bytes



Address translation via page tables:

- page table entry address ($ptea$):
  - $ptea(d, va) = d.pto \cdot 4K + 4 \cdot va.px$
- page table entries (pte):
  - $pte(d, va) = d.m_4(ptea(d, va))$
- physical page index (ppx):
  - $ppx(d, va) = pte(d, va)[31 : 12]$
- valid bit (v):
  - $v(d, va) = pte(d, va)[11]$



Translation

Being in user mode and accessing memory address va:

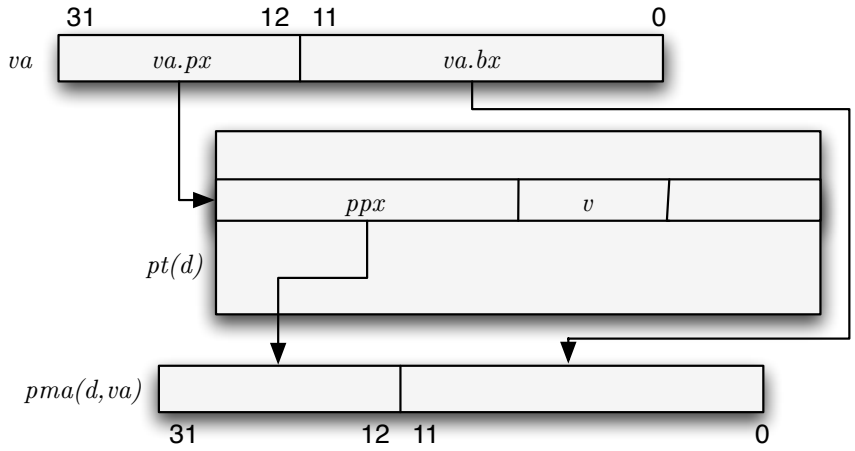
- either generate a page fault:
  - if the page index exceeds the page table length, i.e. $va.px > d.ptl$
  - if the page table entry is not valid, i.e. $v(d, va) = 0$
- or perform the memory access:
  - using the (translated) physical memory address $pma(d, va)$

Translation

Define physical memory address pma:

- concatenation of the physical page index and the byte index

$$pma(d, va) = ppx(d, va) \circ va.bx$$



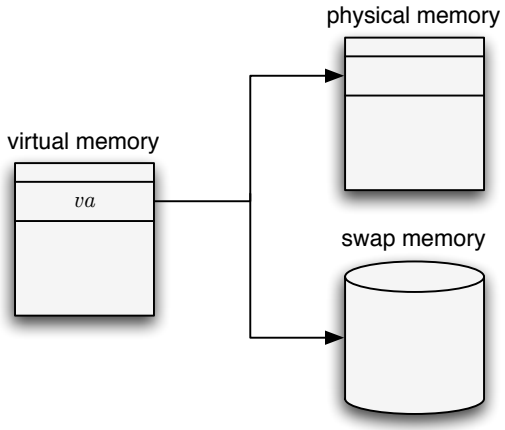
Note

Require page fault handler specification (for completeness):

- based on intermediate model between physical machines and hardware:
  - a processor with a disk as an I/O device used as swap memory

Prove: Physical memory acts like a cache for the swap memory

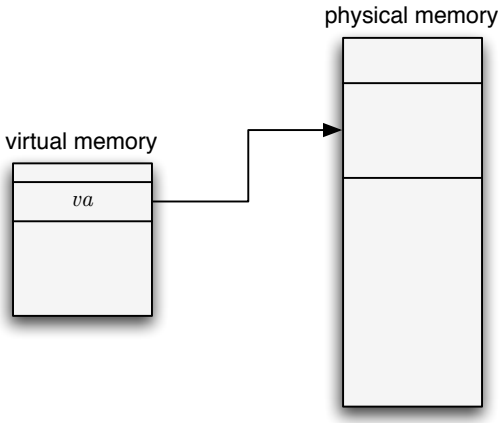
- use page fault handler correctness [HIP05]



Note

Real time systems:

- are programmed such that no page faults occur
- deal with 'small' programs (fitting in the physical memory)
- do not use swap memory
- enforce process separation via address translation



Guideline

1. Physical Machines
   - Configuration
   - Address Translation
2. Virtual Memory Simulation
   - Correctness Theorem

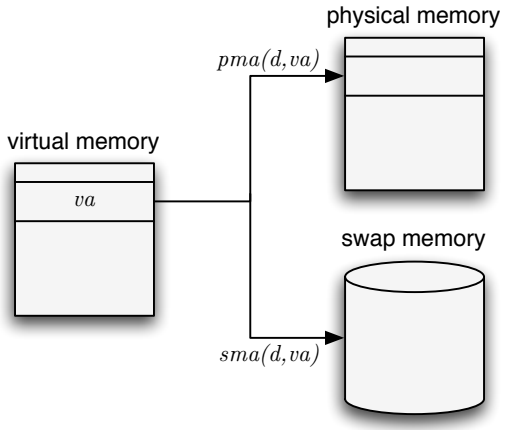
Virtual Memory Simulation

Physical machines with correct page fault handlers simulate virtual machines:

- virtual memory is stored in the swap memory of the physical machine
- the physical memory acts as a write back cache

Page fault handler maintains:

- architecturally defined physical memory address $pma(d, va)$
- swap memory address function $sma(d, va)$



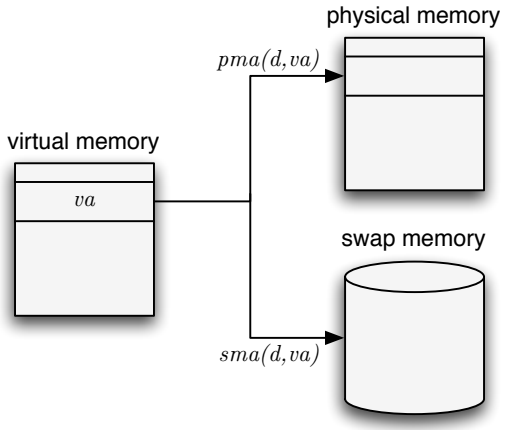
Virtual Memory Simulation

Physical machines with correct page fault handlers simulate virtual machines:

- virtual memory is stored in the swap memory of the physical machine
- the physical memory acts as a write back cache

Page fault handler maintains:

- architecturally defined physical memory address $pma(d, va)$
- swap memory address function $sma(d, va)$



Virtual Memory Simulation

On page faults not violating the page table length:

- the handler selects a physical memory page to evict
- loads the missing page from the swap memory

Simulation Relation

Denote maximal virtual address by $d.b$

Simulation relation $B(vm, d)$:

- (user mode) physical machine d encodes virtual machine vm
- there exits a pte in d for all $d.b/(4K)$ pages of virtual memory, i.e.

$$d.b/(4K) = d.ptl$$

- the content of virtual memory addressed by va is stored in d:
  - if $v(d, va)$ in the physical memory at $pma(d, va)$
  - otherwise in the swap memory at $sma(d, va)$

$$vm.m(va) = \begin{cases} d.m(pma(d, va)) & v(d, va) \\ d.sm(sma(d, va)) & \text{otherwise} \end{cases}$$

Physical memory serves as a write back cache for swap memory.

Simulation Relation

Denote maximal virtual address by $d.b$

Simulation relation $B(vm, d)$:

- (user mode) physical machine d encodes virtual machine vm
- there exits a pte in d for all $d.b/(4K)$ pages of virtual memory, i.e.

$$d.b/(4K) = d.ptl$$

- the content of virtual memory addressed by va is stored in d:
  - if $v(d, va)$ in the physical memory at $pma(d, va)$
  - otherwise in the swap memory at $sma(d, va)$

$$vm.m(va) = \begin{cases} d.m(pma(d, va)) & v(d, va) \\ d.sm(sma(d, va)) & \text{otherwise} \end{cases}$$

Physical memory serves as a write back cache for swap memory.

Simulation Relation

Denote maximal virtual address by $d.b$

Simulation relation $B(vm, d)$:

- (user mode) physical machine d encodes virtual machine vm
- there exits a pte in d for all $d.b/(4K)$ pages of virtual memory, i.e.

$$d.b/(4K) = d.ptl$$

- the content of virtual memory addressed by va is stored in d:
  - if $v(d, va)$ in the physical memory at $pma(d, va)$
  - otherwise in the swap memory at $sma(d, va)$

$$vm.m(va) = \begin{cases} d.m(pma(d, va)) & v(d, va) \\ d.sm(sma(d, va)) & \text{otherwise} \end{cases}$$

Physical memory serves as a write back cache for swap memory.

Theorem

Theorem (Virtual Memory Simulation)

For all computations of the virtual machine $(vm^0, vm^1, \dots)$ *there exits:*

- *a computation of the physical machine* $(d^0, d^1, \dots)$
- *step numbers for the physical machine* $(s(0), s(1), \dots)$

such that for all i*:*

$$B(vm^i, d^{s(i)})$$

Attention: Proof not completely obvious!
One instruction might cause two page faults (fetch and load/store).

Solution:

- page fault handler must not swap out the page swapped in last (see [Hil05])

Theorem

Theorem (Virtual Memory Simulation)

For all computations of the virtual machine $(vm^0, vm^1, \dots)$ *there exits:*

- *a computation of the physical machine* $(d^0, d^1, \dots)$
- *step numbers for the physical machine* $(s(0), s(1), \dots)$

such that for all i*:*

$$B(vm^i, d^{s(i)})$$

Attention: Proof not completely obvious!
One instruction might cause two page faults (fetch and load/store).

Solution:

- page fault handler must not swap out the page swapped in last (see [Hil05])

Theorem

Theorem (Virtual Memory Simulation)

For all computations of the virtual machine $(vm^0, vm^1, \dots)$ *there exits:*

- *a computation of the physical machine* $(d^0, d^1, \dots)$
- *step numbers for the physical machine* $(s(0), s(1), \dots)$

such that for all i*:*

$$B(vm^i, d^{s(i)})$$

Attention: Proof not completely obvious!
One instruction might cause two page faults (fetch and load/store).

Solution:

- page fault handler must not swap out the page swapped in last (see [Hil05])

Synchronization Conditions

Problem:
Read after write (RAW) hazard:

- pipelined or out-of-order architectures
- instruction $I(d^i)$ can modify $I(d^j)$ for $j > i$ after it has been fetched

Instruction $I(d^i)$ may:

- overwrite the instruction itself
- overwrite its page table entry
- change the mode

Synchronization Conditions

Problem:
Read after write (RAW) hazard:

- pipelined or out-of-order architectures
- instruction $I(d^i)$ can modify $I(d^j)$ for $j > i$ after it has been fetched

Instruction $I(d^i)$ may:

- overwrite the instruction itself
- overwrite its page table entry
- change the mode

Synchronization Conditions

On RAW hazards instruction fetch does not work.

Solution:

- detect such dependencies in hardware and roll back the computation
- or introduce software synchronization conditions:
  - let $iaddr(d^j)$ denote the address of instruction $I(d^j)$, possibly translated
  - $I(d^k)$ for $i < k < j$ must drain the pipe if:
    - $I(d^i)$ writes to $iaddr(d^j)$
    - d^j is in user mode and $I(d^i)$ writes to $ptea(d^j, d^j.dpc)$
  - switched to user mode only by an return from exception rfe instruction (which drains the pipe)

Synchronization conditions are hypotheses in the correctness proof [DHP05].

Synchronization Conditions

On RAW hazards instruction fetch does not work.

Solution:

- detect such dependencies in hardware and roll back the computation
- or introduce software synchronization conditions:
  - let $iaddr(d^j)$ denote the address of instruction $I(d^j)$, possibly translated
  - $I(d^k)$ for $i < k < j$ must drain the pipe if:
    - $I(d^i)$ writes to $iaddr(d^j)$
    - d^j is in user mode and $I(d^i)$ writes to $ptea(d^j, d^j.dpc)$
  - switched to user mode only by an return from exception rfe instruction (which drains the pipe)

Synchronization conditions are hypotheses in the correctness proof [DHP05].

Summary

Topics dealt with in this lecture:

- address translation
  - page tables
  - physical memory address
- virtual memory simulation
  - page fault handler
  - synchronization conditions

Iakov Dalinger, Mark Hillebrand, and Wolfgang Paul.
On the Verification of Memory Management Mechanisms.
In *Proceedings of the 13th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 2005)*, volume 3725 of *Lecture Notes in Computer Science (LNCS)*, pages 301–316. Springer, 2005.

Mark Hillebrand.
Address Spaces and Virtual Memory: Specification, Implementation, and Correctness.
PhD thesis, Saarland University, Computer Science Department, June 2005.

Mark Hillebrand, Thomas In der Rieden, and Wolfgang Paul.
Dealing with I/O Devices in the Context of Pervasive System Verification.
In *ICCD '05*, pages 309–316. IEEE Computer Society, 2005.