XEN Memory Management (Intel IA-32)

Andrés Krapf
{Andres.Krapf}@sophia.inria.fr

INRIA, Sophia Antipolis - Méditerranée

October 30, 2007

Abstract

This document describes briefly the main characteristics of the memory management carried out by Xen Virtualization Platform over IA-32 architectures.

1 Page Tables - General OS concepts

One of the most important concept to study are page tables. These are the ones that keep all the translations between virtual and physical addresses. For each process there is a page directory (pgd).

In Linux, each process is a pointer `(mm_struct->pgd)` to its own PGD (Page Directory) which is a physical page frame. This frame contains an array of type pgd_t, which is an architecture-specific type defined in `<asm/page.h>`. The page tables are loaded differently depending on the architecture. On the x86, the process page table is loaded by copying `mm_struct->pgd` into the cr3 register, which has the side effect of flushing the TLB. This register is also referred to as the Page Table Base Register (PTBR)[1]

Whenever the cr3 register is written, the TLB is flushed (flushing the TLB is part of the hardware-defined context switch protocol). When there is a *context switch*, the address of the incoming process's page directory is written to the cr3 register.

If there is a TLB miss, the OS is not aware of this (for this architecture, which uses a hardware-managed TLB; the TLB accesses the target Page Table Entry (PTE), using the page table base register, and updates the TLB.

In general, there are different types of TLB flushings. For example[2]

[1]http://www.stanford.edu/~stinson/paper_notes/fundamental/mem_mgmt/ia32_pts.txt
[2]http://tldp.org/LDP/khg/HyperNews/get/memory/flush.html

```
void flush_tlb_all(void)
```

removes all mappings in the TLB. At implementation level this could imply that the cache is also flushed.

```
void flush_tlb_mm(struct mm_struct *mm)
```

removes all the entries corresponding to the address space corresponding to `mm_struct`. (In particular, an mm_struct may map to one or many tasks or none.)

```
flush_tlb_range(struct mm_struct *mm, unsigned long start,
                unsigned long end);
```

removes some pages of the address space, and

```
void flush_tlb_page(struct vm_area_struct *vma,
                    unsigned long address);
```

removes a page.

During address translation, the OS will *page fault* if the page that contains the address is not in memory. The pageFault() function will bring in the page and allocate a frame to it, possibly swapping out a page to obtain a free frame. It will also update the page table and TLB.

When the OS swaps a page out from the currently running process, the OS must invalidate the TLB and page table entry for that page[3].

2 Xen's Management of Page Tables

Xen is responsible for managing the allocation of physical memory to domains, and ensuring safe use of the paging and segmentation hardware[4].

2.1 Pseudo-Physical Memory

Since physical memory is allocated and freed on a page granularity, there is no guarantee that a domain will receive a contiguous stretch of physical memory. However most operating systems do not have good support for operating in a fragmented physical address space. To aid porting such operating systems to run on top of Xen, we make a distinction between machine memory and pseudo-physical memory.

[3]http://www.cs.utexas.edu/users/witchel/372/labs/lab3/index.html
[4]http://www.cl.cam.ac.uk/research/srg/netos/xen/readmes/interface/interface.html

To achieve this, Xen maintains a globally readable machine-to-physical table which records the mapping from machine page frames to pseudo-physical ones. In addition, each domain is supplied with a physical-to-machine table which performs the inverse mapping.

2.2 Memory Management Modes

Page Table Updates. In the default mode of operation, Xen enforces read-only access to page tables and requires guest operating systems to explicitly request any modifications. Xen validates all such requests and only applies updates that it deems safe. This is necessary to prevent domains from adding arbitrary mappings to their page tables.

To aid validation, Xen associates a type and reference count with each memory page. A page has one of the following mutually-exclusive types at any point in time: page directory (PD), page table (PT), local descriptor table (LDT), global descriptor table (GDT), or writable (RW). Note that a guest OS may always create readable mappings of its own memory regardless of its current type.

Writable Page Tables. Xen also provides an alternative mode of operation in which guests have the illusion that their page tables are directly writable. Xen must still validate modifications to ensure secure partitioning. To this end, Xen traps any write attempt to a memory page of type PT (i.e., that is currently part of a page table). If such an access occurs, Xen temporarily allows write access to that page while at the same time disconnecting it from the page table that is currently in use. This allows the guest to safely make updates to the page because the newly-updated entries cannot be used by the MMU until Xen revalidates and reconnects the page. Reconnection occurs automatically in a number of situations: for example, when the guest modifies a different page-table page, when the domain is preempted, or whenever the guest uses Xen's explicit page-table update interfaces.

Shadow Page Tables. Xen also supports a form of shadow page tables in which the guest OSs use a independent copies of page tables which are unknown to the hardware (i.e. which are never pointed to by cr3). Instead Xen propagates changes made to the guest's tables to the real ones, and vice versa.

3 Updating MMU

- Since guest operating systems have read-only access to their page tables, Xen must be involved for making any changes. The following multi-purpose hypercall can be used to

  – modify page-table entries,
  – update the machine-to-physical mapping table,
  – flush the TLB,
  – install a new page-table base pointer,
  – and more.

  ```
  mmu_update(mmu_update_t *req, int count,
             int *success_count)
  ```

  It is used to process a set of requests for efficient in a batch manner. Xen will check that the updates are safe.

- There are some occasions (notably handling a demand page fault) where a guest OS will wish to modify exactly one PTE (page table entry) rather than a batch, and where that PTE is mapped into the current address space.

  ```
  update_va_mapping(unsigned long va, uint64_t val,
                    unsigned long flags)
  ```

  It updates the currently installed PTE that maps virtual address va to new value val. As with mmu_update, Xen checks the modification is safe before applying it. The flags determine which kind of TLB flush, if any, should follow the update.

- Finally, sufficiently privileged domains may occasionally wish to manipulate the pages of others:

  ```
  update_va_mapping(unsigned long va, uint64_t val,
                    unsigned long flags, domid_t domid)
  ```

  It is identical to update_va_mapping except that the pages being mapped must belong to the domain domid.

- An additional MMU hypercall provides an "extended command" interface. This provides additional functionality beyond the basic table updating commands:

  ```
  mmuext_op(struct mmuext_op *op, int count,
            int *success_count, domid_t domid)
  ```

This hypercall is used to perform additional MMU operations. These include updating cr3 (or just re-installing it for a TLB flush), requesting various kinds of TLB flush, *flushing the cache*, *installing a new LDT*, or pinning & unpinning page-table pages (to ensure their reference count doesn't drop to zero which would require a revalidation of all entries). Some of the operations available are restricted to domains with sufficient *system privileges*.

It is also possible for privileged domains to reassign page ownership via an extended MMU operation, although grant tables are used instead of this where possible

4 Changing Memory Management Mode

A hypercall interface is exposed to activate and deactivate various optional facilities provided by Xen for memory management. It toggles various *memory management modes*.

```
vm_assist(unsigned int cmd, unsigned int type)
```

5 Segmentation Support

The Global Descriptor Table or GDT is a data structure used by Intel x86-family processors in order to define the characteristics of the various memory areas used during program execution, for example the base address, the size and access privileges like executability and writability. These memory areas are called segments in Intel terminology.

Xen allows guest OSes to install a custom GDT if they require it; this is context switched transparently whenever a domain is [de]scheduled.

```
set_gdt(unsigned long *frame_list, int entries)
```

installs a global descriptor table for a domain.

Xen also allows guest operating systems to update just an individual segment descriptor in the GDT or LDT:

```
update_descriptor(uint64_t ma, uint64_t desc)
```

updates the GDT/LDT entry at machine address ma; the new 8-byte descriptor is stored in desc. Xen performs a number of checks to ensure the descriptor is valid.

6 Context Switching

When a guest OS wishes to context switch between two processes, it can use the page table and segmentation hypercalls described above to perform the the bulk of the privileged work. In addition, however, it will need to invoke Xen to switch the kernel (ring 1) stack pointer:

```
stack_switch(unsigned long ss, unsigned long esp)
```

requests kernel stack switch from hypervisor; ss is the new stack segment, esp is the new stack pointer.

7 Memory Allocation

The maximum allocation, set at domain creation time, cannot be modified. However a domain can choose to reduce and subsequently grow its current allocation by using the following call:

```
memory_op(unsigned int op, void *arg)
```

8 Trusted OSes

Some platforms, like VLX, consider the existence of trusted OS as well as untrusted OSes. Trusted OSes execute freely without being controlled at all.

Other platforms, like XEN, consider that all guest OSes are untrusted and only one component which "could" be considered as a trusted OS, called Domain0. Domain0 engages in management control and monitoring tasks, and also in device sharing.