Exercise 1: Size of virtual memory on systems with 64-bit addressing

Consider an address space of 64 bits. Compute:

- the size of the addressable memory in bytes.
  \[ 2^{64} \text{bytes} = 1.8 \cdot 10^{19} \text{bytes} \]

- the number of 100GB disks needed to persistently store the whole address space.
  \[ \frac{2^{64} \text{B}}{100 \text{GB}} = 171'798'692 \text{disks} \sim 10^8 \text{disks} \]

- the time needed to read the whole memory with transfer rate of 1GB/s.
  \[ \frac{2^{64} \text{B}}{1 \text{GB/s}} = 17'179'869'184 \text{s} \sim 544 \text{years} \]

- the size of the page table with the following page sizes: 4KB, 128KB and 4MB

One address is 8 bytes. The size of a page table is equal to the number of pages multiplied with the size of one table line. A minimal table line is composed by (see Figure 1 for an example):

![Figure 1: Page table line (example).](image-url)
– the real page address (64 - offset bits)
– present flag (1 bit)
– protection flags (3 bits)
– modified flag (1 bit)
– referenced flag (1 bit)
– cache disable flag (1 bit)

The size of the page table is:

– for 4KB pages
  * an address is composed by 52 bits for the page address and 12 for the offset
  * we have \(2^{52}\) pages
  * each page table entry is 52 + 7 = 59 bits (7.3 bytes, 8 bytes because of alignment)
  * size = \(2^{52} \cdot 8 = 32768\) TB

– for 128KB pages
  * an address is composed by 47 bits for the page address and 17 for the offset
  * we have \(2^{47}\) pages
  * each page table entry is 47 + 7 = 54 bits (6.75 bytes, 7 bytes because of alignment)
  * size = \(2^{47} \cdot 7 = 896\) TB

– for 4MB pages
  * an address is composed by 42 bits for the page address and 22 for the offset
  * we have \(2^{42}\) pages
  * each page table entry is 42 + 7 = 49 bits (6.125 bytes, 7 bytes because of alignment)
  * size = \(2^{42} \cdot 7 = 28\) TB
Exercise 2: Alternative mapping structures for 64-bit wide virtual memories

Discuss some alternatives to linear page tables considering the case of a sparse occupied memory. Describe how trees and inverted tables work. Which parts of these structures can potentially be swapped out? Which parts not? Compute the worst-case memory access time for a three indirection levels page table.

- Inverted tables
  
  With 32 and 64 bits architectures the size of the page tables is a major problem and system designers decided to build tables organized around physical memory. This inverted page tables contain one entry for each physical page. On a page table line we store the PID of the process owning the pages, some flags and the virtual address.

  The virtual page number an the process id are hashed and the result is used to get a line in the inverted page table (which has one line for every real page frame). There we find a PID and a virtual page number: if they match our page is in memory otherwise we have a page fault. In this rare case we look up in the process page table (which is not inverted).

  In this way we reduce the size of the table to one line for each real memory page but when we look for a certain virtual address we must scan the table (this search can be sped up with an hash table).

- Trees and multilevel directories
  
  Another possibility is to use multi-level page tables. The page frame address is divided into more indexes (addr = index_1 | index_2 | index_3 | offset). Each index is then used to access one level of the tree (see Figure 2). This has the advantage the unused sub-tress can be left out and that the lookup happens in fixed time:

  The worst-case memory access time will be:

  \[ time = 4 \cdot \text{accessTime}_{\text{page}} \]

  ![Figure 2: Multilevel page table.](image-url)
Exercise 3: Virtual memory by IA64

Read carefully the information [1] about the architecture chosen by Intel for the virtual addressing his Itanium Processors and answer to the following questions:

- which is the idea behind region registers?
  The first three bits of an address corresponds to a register which specifies which memory region has to be used. Each 24 bit register can address $2^{24}$ regions and in this way the range of possible addresses is increased to 85 bits (61 bits for each region). Regions also allows an efficient management of processes (heavy-weight) for multi-tasking environments.

- what does *pinning* mean? Why and where is it used?
  Pinning is used to mark a page as *non-swappable*. For example to keep kernel pages in memory.

- which page sizes are supported? Which size would you choose for stack, heap, mapped I/O and video frame buffer?
  Supported sizes: 4KB, 8KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB, 256MB, 4GB
  - stack: as small as possible (4KB)
  - heap: depending on system constraints
  - mapped I/O: device matching size (usually 4KB – 16KB)
  - video-buffer: 4MB or 16MB

- which TLBs are implemented? Which are the allowed operations on TLBs?
  - instruction (ITLB) and data (DTLB)
  - loading, unloading and pinning

- which access rights are supported?
  write disable, read disable, execute disable and domain

- what is a virtual hash page table (VHPT)? How can this be used?
  The VHPT is an hash table to map addresses. It is implemented in software and is used as an extension of the TLB hierarchy to speed up an address lookup.

- what is virtual aliasing? How and where is this harmful?
  We have virtual aliasing when more than one virtual page is mapped to the same physical page. On some Itanium models the performance can be degraded when the distance between virtual aliases is less than 1 MB.

References