Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Fall Term 2012

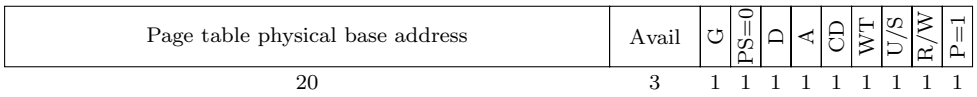
SYSTEMS PROGRAMMING AND COMPUTER ARCHITECTURE
Assignment 10: Page table programming assignment

Assigned on: **6th Dec 2012**
Due by: **13th Dec 2012**

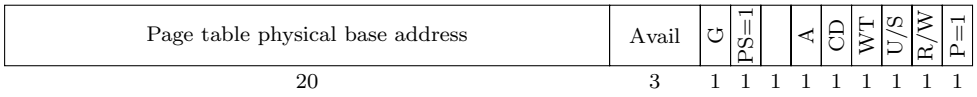
Task 1

In this assignment you will implement basic virtual memory functionality in C based on the structure of a x86_32 page-table. x86_32 has a two-level page table. The first-level page table is often referred to as the page-directory.

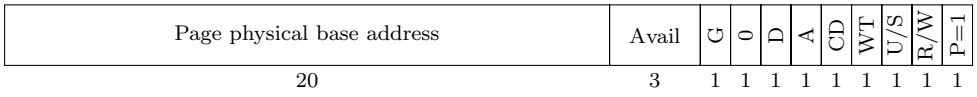
If the present bit is set, the 32 bit page-directory entry looks like this:



or for large pages (4 MiB):

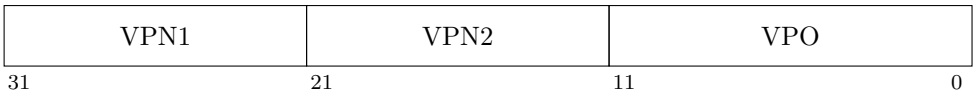


The 32 bit width second-level page-table format for entries having the present bit set is:



See Table 1 for a description of flags used in above figures.

The structure of a 32 bit virtual address is:



VPN 1 and 2 are the virtual page numbers used as indices for the first and second level page table, whereas VPO is the virtual page offset which addresses the byte within the page.

We prepared a skeleton that can be used as a basis for the implementation. You will find it on the course website as a tar archive. Download and extract that file. You will find the following typedefs there:

Avail	Bits available to system programmers
G	global page: not evicted from TLB on task switch
PS	page size: 0 = 4 KiB, 1 = 4 MiB
A	accessed: set by hardware cleared by software
CD	cache disabled
WT	write-through
U/S	user or supervisor mode access
R/W	read-only or read-write access
P	page table or directory is present
D	dirty: set by MMU on writes

Table 1: Page table and page directory bits

Type	Base type	Description
pde_t	uint32_t	Entry in the page-directory (first-level page table)
pte_t	uint32_t	Entry in the page-table (second-level)
vaddr_t	uint32_t	Virtual address
paddr_t	uint32_t	Physical address

As a first step implement functions *parse_virt_addr*, *parse_pde*, *parse_pt*, *set_pde*, *set_pt*. These functions are low-level functions to read and write the page-directory and page-table entries as well as for extracting the page-table indices from a virtual address.

Task 2

Have a look at the skeletons for functions *map_large* and *map*. These functions map physical pages into the virtual address space by inserting entries to the first and second-level page tables. They also set some attributes in these entries. At this point, you only have to handle the r/w flag.

For regular 4 KiB pages, *map* needs to touch both levels of the page-table hierarchy, whereas *map_large* (for 4 MiB mappings) only modifies the content of the page-directory. Mapping a large page works by pointing the page-directory directly to a 4 MiB piece of memory. It needs to be page aligned, as only the 20 uppermost bits of the physical address can be stored in the page-directory entry. The PS bit in the page-directory entry is used to distinguish large and regular pages. Implement these two functions.

In case the page-directory or the appropriate page-table does not yet exist, you will have to allocate some space for it. Have a look at and implement functions *alloc_pt* and *alloc_pd*. Remember: Both the beginning of the page-table and the page-directory data-structures need to be page-aligned. Do not forget to implement *free_pagetable* to free all allocated memory again.

Task 3

Next, you are going to implement the *unmap* operation. It should work for both, 4 KiB and 4 MiB pages. The present bit in the appropriate page-directory or page-table entries should be set to 0 by this operation.

Evaluation

You can use the *./correctness* script in the handout archive to verify your solution. The program will run your page-table implementation and compare it to the master solution. *dropAddresses* is executed on your output first to remove all dynamic addresses from the page-table entries. These are the addresses stored in the page-directory for regular 4 KiB pages. Then, the Unix *diff* command is executed to compare the files.

Hints

Run *make* after extracting the archive we provide on the website to compile your page table manipulation program. gcc will link against a static library *libdump.a* required to dump your page table in a format we can parse with the *./correctness* script. Your program will be compiled in 32 bit mode. As always you will have to install *gcc-multilib* for cross-compiling on 64 bit systems.

Hand In Instructions

Upload your source files to a subfolder **assignment10** of your SVN folder.