PA3 - Part 2 – myMemory Manual

Paolo Ratti Tamayo - A20352801

Jiateng Sun - A20368601

MYMEMORY PA3’s User Manual MYMEMORY

NAME

myMemory

SYNOPSIS

int myMemory(void);

DESCRIPTION

The myMemory system call helps system programmer to find out the physical memory space occupied by a user process.

It displays the number of pages that are accessible, and the number of pages that are writable by the current running program.

RETURN VALUE

The myMemory() function returns the number of pages that are allocated by user program.

XV6 Memory Structure

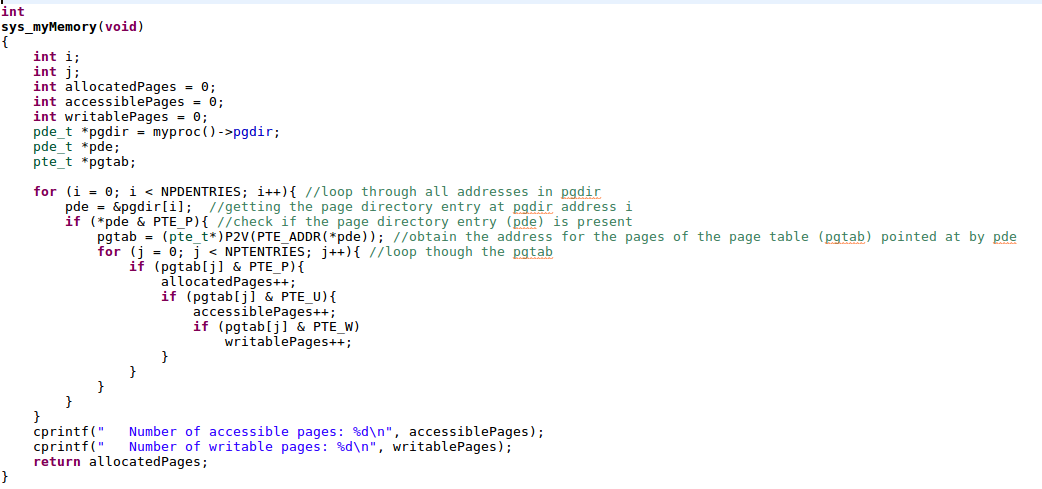
XV6 uses an x86 page table, it is an array of 2^20 page table entries (PTEs). Each PTE contains a 20-bit physical page number (PPN) and flags bits telling the paging hardware how the associated virtual address is allowed to be used.

The paging hardware translate a virtual address by its first 20 bits to index into the page table to find its PTE. It replaces the address’s top 20 bits with PPN in the PTE while the last 12 btis are unchanged.

Flag PTE\_P indicated whether the PTE is present; flag PTE\_W controls whether instructions are allowed to issue write operations to the page; flag PTE\_U controls whether user programs are allowed to use the page.

A page table is stored in a two level tree. The root is a 4096 byte page directory which contains 1024 PTE-like references to page table pages. Each of these page table pages is an array of 1024 32 bits PTEs.

Implementation of System Call myMemory()



We initialized three integers variables to store numbers of allocated pages, user accessible pages and user writable pages.

The current process’s information is stored in a structure proc. The attribute pgdir of proc, with type pde\_t\*, stores the page directory.

We iterate through the page directory, storing each page directory entry to a variable pde. We check if each pde is present by masking it with PTE\_P (0x001), which is defiend in MITxv6/mmu.h. If pde is present we get the first 20 bits by casting it to a unsigned integer and masking it with ~0xFFF using the macro PTE\_ADDR() defined in MITxv6/mmu.h. We then translate the first 20 bits into the corresponding virtual address by adding the first kernel virtual address (0x80000000) using macro P2V() defined in MITxv6/memlayout.h. This is how we obtain the address for the pages of the table (pgtab) pointed .

We iterate through the pages. During each iteration we check:

- If each page is present by masking it with PTE\_P (0x001). If the page is present, we increase the number of allocated pages by one.

- If the page is present we further check if that page is user accessible by masking it with PTE\_U (0x004) defined in mmu.h. If the page is accessible, we increase the number of user accessible pages by one.

- If the page is present and user accessible we further check if that page is user writeable by masking it with PTE\_W (0x002) defined in mmu.h. If the page is writable, we increase the number of user writable pages by one.

Finally, we print out the number of user accessible pages and writable pages, and returns the number of allocated pages.