CSCE 613: Project 3 Design

Student: Caio Duarte Diniz Monteiro

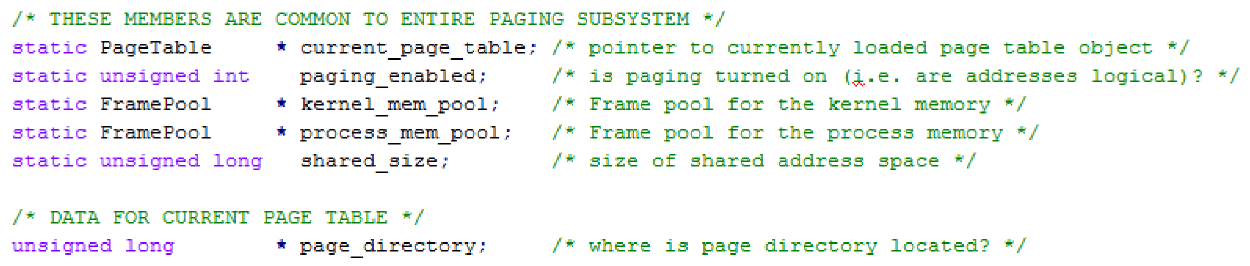
Date: 02/22/16

FramePool class

FramPool class implementation is almost the same from Project 2. The only differences are some error handling for dealing with releasing and marking as inaccessible invalid frames. Those changes were motivated by the Project 2 grading report.

PageTable

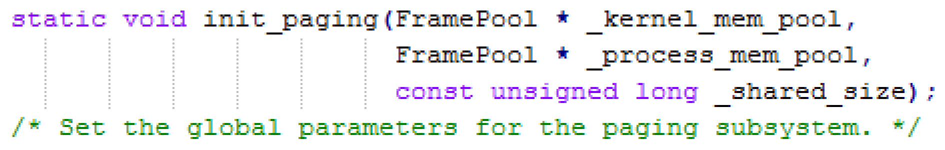
The purpose of the Page Table is to manage the virtual to physical memory address translation. The responsible for filling the entries on the page tables with the appropriated translation is the Page Fault Handler, which triggers whenever a position of the memory not yet mapped needs to be accessed. Some static variable keeps track of the kernel and process memory pools, the current page table in use by the OS (since we will have one PageTable class per process), etc.



```
/* THESE MEMBERS ARE COMMON TO ENTIRE PAGING SUBSYSTEM */
static PageTable     * current_page_table; /* pointer to currently loaded page table object */
static unsigned int    paging_enabled;     /* is paging turned on (i.e. are addresses logical)? */
static FramePool     * kernel_mem_pool;    /* Frame pool for the kernel memory */
static FramePool     * process_mem_pool;   /* Frame pool for the process memory */
static unsigned long   shared_size;        /* size of shared address space */

/* DATA FOR CURRENT PAGE TABLE */
unsigned long        * page_directory;     /* where is page directory located? */
```

PageTable initialization



```
static void init_paging(FramePool * _kernel_mem_pool,
                        FramePool * _process_mem_pool,
                        const unsigned long _shared_size);
/* Set the global parameters for the paging subsystem. */
```

Init paging function serves basically to set the values of the global variables used by the PageTable class, like the memory pools and the size of the shared address space.

PageTable constructor

PageTable constructor has no parameters and is responsible for creating the page directory and the first page table of the directory, which contains the 4 MB of data that are directly mapped. All the other entries of the page directory are marked as not present, supervisor level and read/write.

PageTable load function

```
void load();
```

The load function is responsible to set the global variable of the current page table in use and store its page directory address into the CR3 register.

PageTable enable_paging function

```
static void enable_paging();
```

Enable_paging function simply sets the global variable paging_enable to 1, and set the proper bit of the CR0 register also to 1, making the OS start to use virtual addresses.

PageTable handle_fault function

```
static void handle_fault(REGS * _r);
```

Handle_fault function is the core of the PageTable functioning. This is the function that is evoked whenever an address with no virtual to physical mapping is accessed. The faulty address is read from the register CR2 and the error code is one of the properties of the argument _r.

The error code is used to check if this was really a page not present error and then map the address, or if the fault was due to a protection fault.

Using the fault address on CR2 the page directory index and page table index are found.

```
int directory_entry = fault_addr >> 22; //gets the entry on the page directory
int table_entry = (fault_addr >> 12 & 0x03FF); //gets the entry on the corresponding page table
```

If the specified directory entry is present on the page directory, then a new frame is allocated from the process memory pool and the specified page table entry is updated on the page table represented by the page directory before mentioned. The page table entry is updated with the address allocated by the process memory pool and has the flags present, user level, and read/write set.

If the directory entry is not present, than it is necessary to first allocate a new frame from the kernel memory pool. This frame will hold the newly created page table, where all the entries are not present. Now that the page table was created, a frame from the process memory pool is allocated and its address is set on the specified page table entry, the entry is then marked with the flags present, user level, and read/write. The address of the created page table is then stored on the specified page directory entry, which will have the flags present, supervisor level, read/write.

Now, subsequent memory accesses on addresses belonging to the mapped page will not cause a page fault as long as the page directory and page table entries for it remains valid.

On every step where a frame allocation is requested, a check is performed to make sure that an available frame was found, if this does not happen than an error message is displayed indicating that the memory is full.