Kwangkyu Hwang

MP3 design document

Editted file:

cont_frame_pool.H (same with MP2)
conf_frame_pool.C (same with MP2)
page_table.C (newly implement)

Flow in kernel.C

After operate timer, exception handler, interrupt handler, keyboard etc, initiallize frame pools that we implemented last machine problem.
Kernel frame pool is 2MB size and starts from address 2MB.
Process frame pool is 28MB size and starts from address 4MB.
And then register page fault handler to deal with page fault.
Now, the program initialize the page table by calling
PageTable::init_paging()
PageTable Constructor
PageTable::load()
PageTable::enable_paging()
Next, it access to the memory we didn't allocate at initializing stage(above 4MB address). Thus it causes page fault, and calls page fault handler. A page fault is handled in
PageTable::handle_fault
After allocate some addresses and store values, check the values are stored correctly.

Thus, those methods are what I implemented in this machine problem, and all of them are implemented in page_table.C

PageTable::init_paging()
PageTable Constructor
PageTable::load()
PageTable::enable_paging()
PageTable::handle_fault()

page_table.C

PageTable::init_paging()

```
void PageTable::init_paging(ContFramePool * _kernel_mem_pool,
                            ContFramePool * _process_mem_pool,
                            const unsigned long _shared_size)
{
   kernel_mem_pool = _kernel_mem_pool;
   process_mem_pool = _process_mem_pool;
   shared_size = _shared_size;
   Console::puts("Initialized Paging System\n");
}
```

Init_paging setting the several static variables (kernel_mem_pools, process_mem_pools, shared_size) as what we created in main function of kernel.C

PageTable Constructor

```
PageTable::PageTable()
{
   //setting up page directory
   page_directory = (unsigned long *) (kernel_mem_pool->get_frames(1) * PAGE_SIZE);

   //setting up page table
   unsigned long* page_table = (unsigned long *) (kernel_mem_pool->get_frames(1) * PAGE_SIZE);

   unsigned long address=0;
   unsigned int i;

   // map the first 4MB of memory
   for(i=0; i<ENTRIES_PER_PAGE; i++) {
      page_table[i] = address | 3; // attribute set to: supervisor level, read/write, present(011 in binary)
      address = address + PAGE_SIZE; // 4096 = 4kb
   };

   //filling in the page directory entries
   //fill the first entry of the page directory
   page_directory[0] = (unsigned long) page_table; // attribute set to: supervisor level, read/write, present(011 in binary)
   page_directory[0] = page_directory[0] | 3;

   for(i=1; i<ENTRIES_PER_PAGE; i++){
      page_directory[i] = 0 | 2; // attribute set to: supervisor level, read/write, not present(010 in binary)
   };
   Console::puts("Constructed Page Table object\n");
}
```

Constructor of PageTable Class do several works.

At first, it set up page directory. Since page directory is also a page, we need to get a frame from frame pool. Make it in a stack using kernel_mem_pool. Get one frame from kernel_mem_pool and store it to local variable page_directory.

Now we placed page directory and next step is make one page table. First page table will be located the next of page directory. Get one frame from kernel_mem_pool. And then, allocate 4MB address to the each page table entry. In each PTE store the information of supervisor/user mode, read/write, present. This time we will 011 on each PTE.

Next fill the page_directory. We have first page table, so store the information to page directory entry and also set attribute as present.

The other page directory entries should set as 010 since the other page table's are not created yet. This is for page table constructor

PageTable::load()

```
void PageTable::load()
{
   current_page_table = this;
   write_cr3((unsigned long) page_directory); // put that page directory address into CR3
   Console::puts("Loaded page table\n");
}
```

Set currently made page table at static variable current_page_table and write page directory address into CR3

PageTable::enable_paging()

```
void PageTable::enable_paging()
{
   write_cr0(read_cr0() | 0x80000000); // set the paging bit in CR0 to 1
   PageTable::paging_enabled = 1;
   Console::puts("Enabled paging\n");
}
```

Enable paging on the CPU. Set the paging bit(bit 31)of CR0 to 1, and change our static variable paging_enabled to 1 also.

PageTable::handle_fault()

```
void PageTable::handle_fault(REGS * _r)
{
   unsigned long address = read_cr2();
   unsigned long table_index = address >> 22; //right 10 bit is table index

   unsigned long* pg_directory = (unsigned long*) read_cr3();
   unsigned long* page_table;
   //should check whether need to add a page table or just add a page
   //if address is first address of new page table, need to add page table
   //else, just add page

   //check directory entry to check page table present or not
   if(pg_directory[table_index] & 1 == 1) { //present
      page_table = (unsigned long*) (pg_directory[table_index] & 0xFFFFF000);
   } else { //not present
      //new page table
      page_table = (unsigned long*) (PAGE_SIZE * kernel_mem_pool->get_frames(1));
      pg_directory[table_index] = (unsigned long) page_table;
      pg_directory[table_index] = pg_directory[table_index] | 3;

      unsigned int i;
      for(i=0; i<ENTRIES_PER_PAGE; i++) {
         page_table[i] = 2;
      };
      page_table = (unsigned long*) ((unsigned long)page_table & 0xFFFFF000);
   }
   unsigned long page_index = address & 0x3FF000;
   page_index = page_index >> 12;

   page_table[page_index] = (process_mem_pool->get_frames(1) * PAGE_SIZE);
   page_table[page_index] = page_table[page_index] | 3;

   Console::puts("handled page fault\n");
}
```

Now page fault occurred and we need to handle this. At first we can get address that occurred page fault using CR2 register. And from this address we can calculate table_index since it's left 10 bit of address. Page directory address is written to CR3 already, so we can use it.

Since we know page directory and table index, we can check present bit of the target PDE. If it's present, page table is already created and we only need to refer a frame. But if it's not present, we need to make a page table. Thus, like we did before, get one frame for page table. And set it's attribute as present. Then, each initialize each PTE as 2 since it's not yet present.
Extract page table address from PDE.
Calculate page_index from address mid 10 bit (12-21). And get a frame from process_mem_pool. Lastly set present bit in PTE.