CSCE 611 : MACHINE PROBLEM 3: DESIGN DOCUMENT
AUTHOR: AMALESH ARIVANAN
UIN: 633002190

PROBLEM STATEMENT:

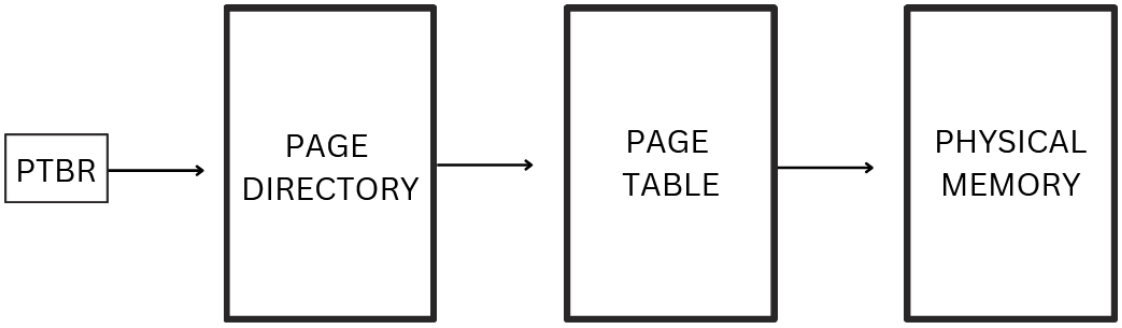
To configure and execute the page table and paging system infrastructure.

CONSTRAINTS:

- The total amount of memory of the machine is 32 MB.
- First 4 MB are reserved for the kernel and shared by all the processes.
- Memory within the first 4 MB will be direct-mapped to physical memory.
- The first 1 MB contains all global data, memory that is mapped to devices, and other stuff.
- The actual kernel code starts at address 0x100000, that is, at 1 MB.

DESIGN OF THE MODEL:

There is a 2-level paging in the proposed design model, where the first level maps to a page directory while the second level maps to a page table.



IMPLEMENTATION

FUNCTIONS USED:

- **PageTable::PageTable()**
  PageTable() is a constructor which sets up entries in the page directory and the page table. The page table entries are directly mapped to the address and its attributes are set to supervisor level, read/write and present.

```
PageTable::PageTable()
{
   //assert(false);
   unsigned long *_page_directory = (unsigned long *) ((kernel_mem_pool-
>get_frames(1))*4096);
   unsigned long *page_table;

   unsigned long address = 0;
   unsigned int i;

   for(i=0; i<1024; i++)
   {
        page_table[i] = address | 3;
        address = address + 4096;
   }
   page_directory[0] = (unsigned long)page_table | 2 | 1;
   page_directory[0] = page_directory[0] | 3;

   for(i=0; i< 1024; i++)
   {
        page_directory[i] = 0 | 2;
   }
   Console::puts("Constructed Page Table object\n");
}
```

- **PageTable::load()**
  Load() function is invoked when the page table is initialized.The page directory index of the present object is extracted and stored in the CR3 register.

```
void PageTable::load()
{
   //assert(false);
   current_page_table = this;
   write_cr3((unsigned long)page_directory);
   Console::puts("Loaded page table\n");
}
```

- **PageTable::enable_paging()**
  The function of enable_paging() is to allocate a particular bit in the CR0 register that is utilized to enable paging.

```
void PageTable::enable_paging()
{
   //assert(false);
   paging_enabled = 1;
   write_cr0(read_cr0() | 0x80000000);
   Console::puts("Enabled paging\n");
}
```

- **PageTable::handle_fault()**
  The function of handle_fault() is to identify faulty addresses, select an available frame to map addresses present in Page Directory and Page Table from the given Frame Pool.

```
void PageTable::handle_fault(REGS * _r)
{
  //assert(false);
  unsigned long * currentpagedirectory = (unsigned long *) read_cr3();
  unsigned long maskaddress = 0;
  unsigned long * pagetable = NULL;
  unsigned long pageaddress = read_cr2();
  unsigned long pdaddress = pageaddress >> 22;
  unsigned long ptaddress = pageaddress >> 12;
  unsigned long er = _r->err_code;

  if((er & 1)== 0)
  {
        if((currentpagedirectory[pdaddress] & 1) == 1)
        {
                pagetable = (unsigned long *) (currentpagedirectory[pdaddress] & 0xFFFFF000);
                pagetable[ptaddress & 0x3FF] = (PageTable::process_mem_pool->get_frames
(1)*PAGE_SIZE) | 2 | 1;
        }
        else
        {
                currentpagedirectory[pdaddress] = (unsigned long) ((kernel_mem_pool ->
get_frames (1)*PAGE_SIZE) | 2 | 1);
                pagetable = (unsigned long *)(currentpagedirectory[pdaddress] & 0x3FF);
                for(int j = 0; j < 1024; j++)
                {
                        pagetable[j] = maskaddress | 4;
                }
                pagetable[ptaddress & 0x3FF] = (PageTable::process_mem_pool-
>get_frames(1)*PAGE_SIZE) | 2 | 1;
        }
  }
  Console::puts("handled page fault\n");
}
```