# Design Document for MP3 – Frame Manager I

This machine problem focuses on creating a frame manager for the frame pools created in the previous MP.
Here we aim to manage the simple frame pool and continuous frame pool using a 2 level page table set-up.

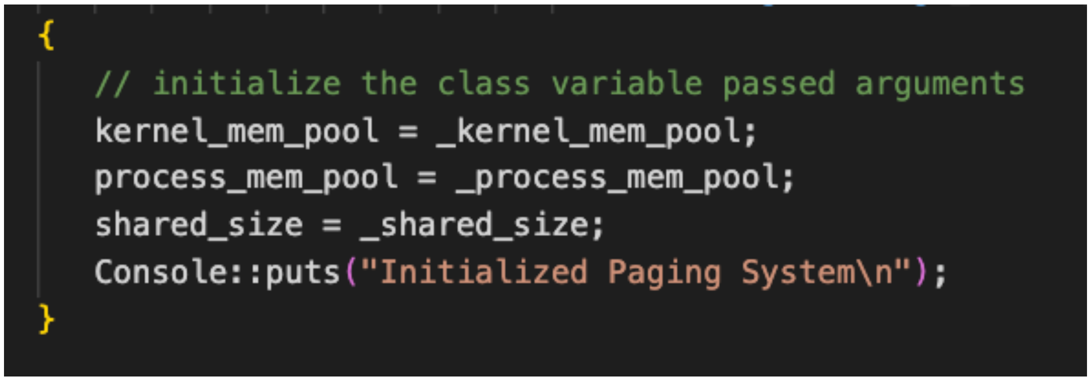
The files modified are:

1. **page_table.C**

I have added descriptive comments in each function to explain the thought process as well as the logic.
Here is a detailed explanation of each of those along with some important code snippets:

1. page_table.C
   a. ***PageTable::init_paging***

      This function primarily sets the variables to their values so that page table can be initiated. I have simply assigned passed parameters to respective page table variables in this function.



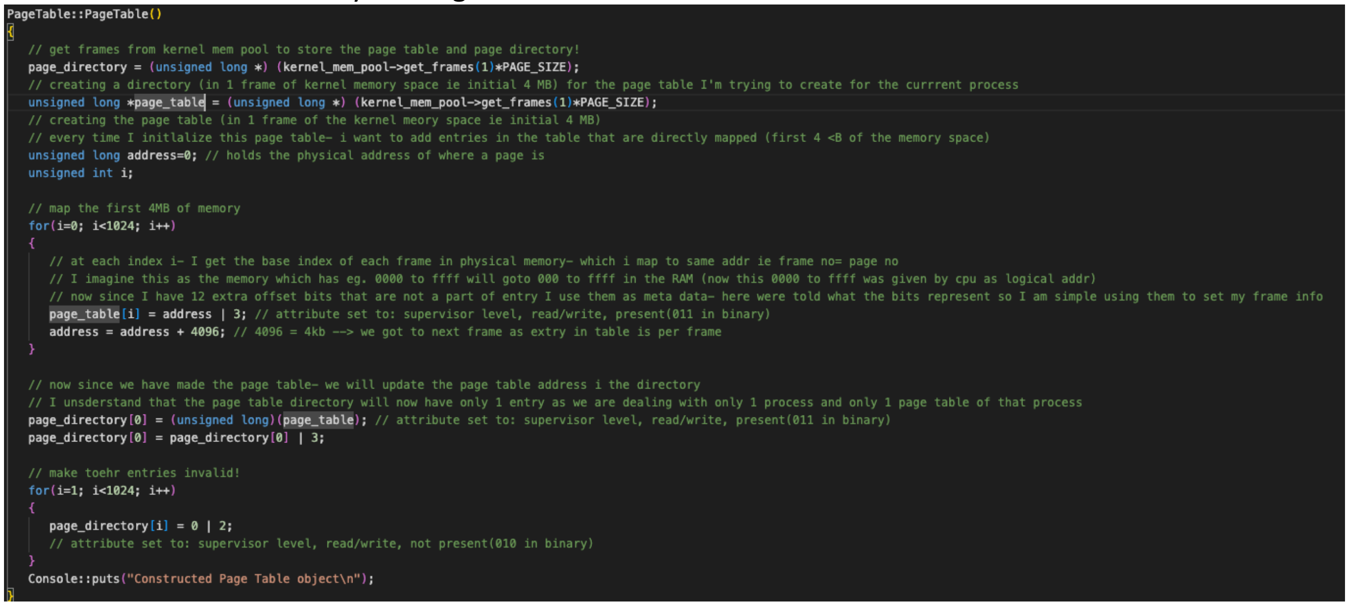
```
{
    // initialize the class variable passed arguments
    kernel_mem_pool = _kernel_mem_pool;
    process_mem_pool = _process_mem_pool;
    shared_size = _shared_size;
    Console::puts("Initialized Paging System\n");
}
```

   b. ***PageTable::PageTable()***

      The constructor plays an extremely important roles in this code and does the following:
      i. I requested a frame from continuous frame pool and assigned it to the page directory pointer which was defined in the page table class. This is the kernel frame pool's frame as instructed in the question.
      ii. Assigned another frame to the page table pointer. Again taken from kernel frame pool as instructed.
      iii. This basically initiates my 2 level page table setup. If more frames are need for Page table or directory it will be all handled if page fault occurs.
      iv. Now it has been instructed to direct map the first 4<B of memory so I start to add entries for first 4MB to my page table. This is clearly a direct

mapping as the actual address is the frame number and hence noted in page table as it is.

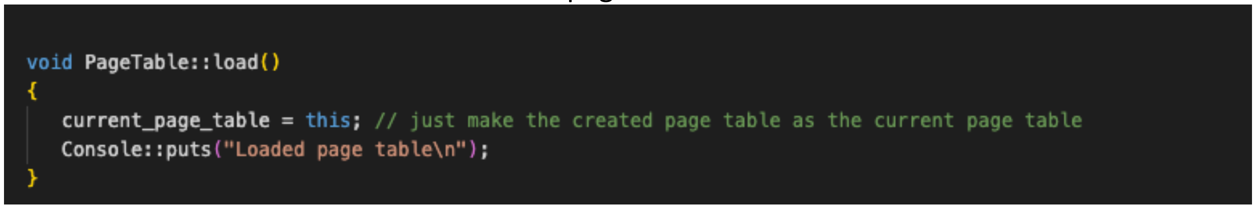
v. I further OR the filled page table entries with 3 (011) to mark my page table entries as supervisor level, read/write, present(011 in binary)
vi. Now since the page has been created in memory – I add its address in page table directory. Again, ORing with3 for same meta data manipulation!
vii. As right not I am inly using 1st entry on page table directory, I mark other entries as invalid so that later my page fault can handle those entries easily looking at the meta data.



```
PageTable::PageTable()
{
  // get frames from kernel mem pool to store the page table and page directory!
  page_directory = (unsigned long *) (kernel_mem_pool->get_frames(1)*PAGE_SIZE);
  // creating a directory (in 1 frame of kernel memory space ie initial 4 MB) for the page table I'm trying to create for the currrent process
  unsigned long *page_table = (unsigned long *) (kernel_mem_pool->get_frames(1)*PAGE_SIZE);
  // creating the page table (in 1 frame of the kernel meory space ie initial 4 MB)
  // every time I initlalize this page table- i want to add entries in the table that are directly mapped (first 4 <B of the memory space)
  unsigned long address=0; // holds the physical address of where a page is
  unsigned int i;

  // map the first 4MB of memory
  for(i=0; i<1024; i++)
  {
    // at each index i- I get the base index of each frame in physical memory- which i map to same addr ie frame no= page no
    // I imagine this as the memory which has eg. 0000 to ffff will goto 000 to ffff in the RAM (now this 0000 to ffff was given by cpu as logical addr)
    // now since I have 12 extra offset bits that are not a part of entry I use them as meta data- here were told what the bits represent so I am simple using them to set my frame info
    page_table[i] = address | 3; // attribute set to: supervisor level, read/write, present(011 in binary)
    address = address + 4096; // 4096 = 4kb --> we got to next frame as extry in table is per frame
  }

  // now since we have made the page table- we will update the page table address i the directory
  // I unsderstand that the page table directory will now have only 1 entry as we are dealing with only 1 process and only 1 page table of that process
  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;

  // make toehr entries invalid!
  for(i=1; i<1024; 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");
}
```

c. ***PageTable::load()***

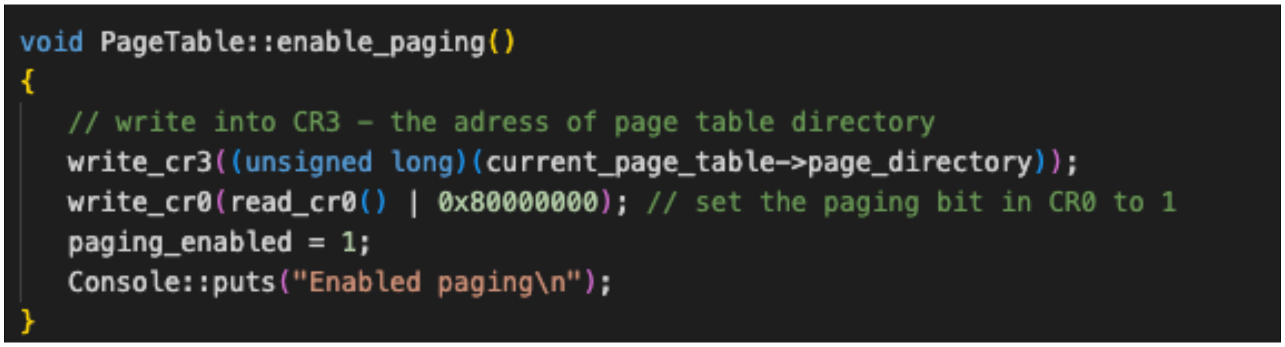
i. Here we simply make the constructed page table as the current page table declared in the page table class.



```
void PageTable::load()
{
  current_page_table = this; // just make the created page table as the current page table
  Console::puts("Loaded page table\n");
}
```

d. ***PageTable::enable_paging()***

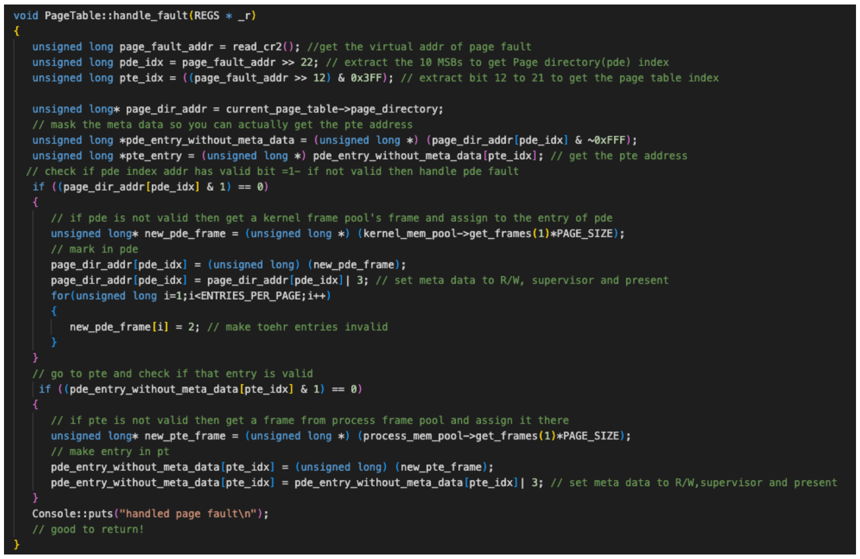
i. This function essentially initiates paging in the system and hence now the use of logical address will be done by the CPU.
ii. To enable paging we make the destined bit of CR0 as 1 and make paging_enabled variable as 1.



```
void PageTable::enable_paging()
{
    // write into CR3 - the adress of page table directory
    write_cr3((unsigned long)(current_page_table->page_directory));
    write_cr0(read_cr0() | 0x80000000); // set the paging bit in CR0 to 1
    paging_enabled = 1;
    Console::puts("Enabled paging\n");
}
```

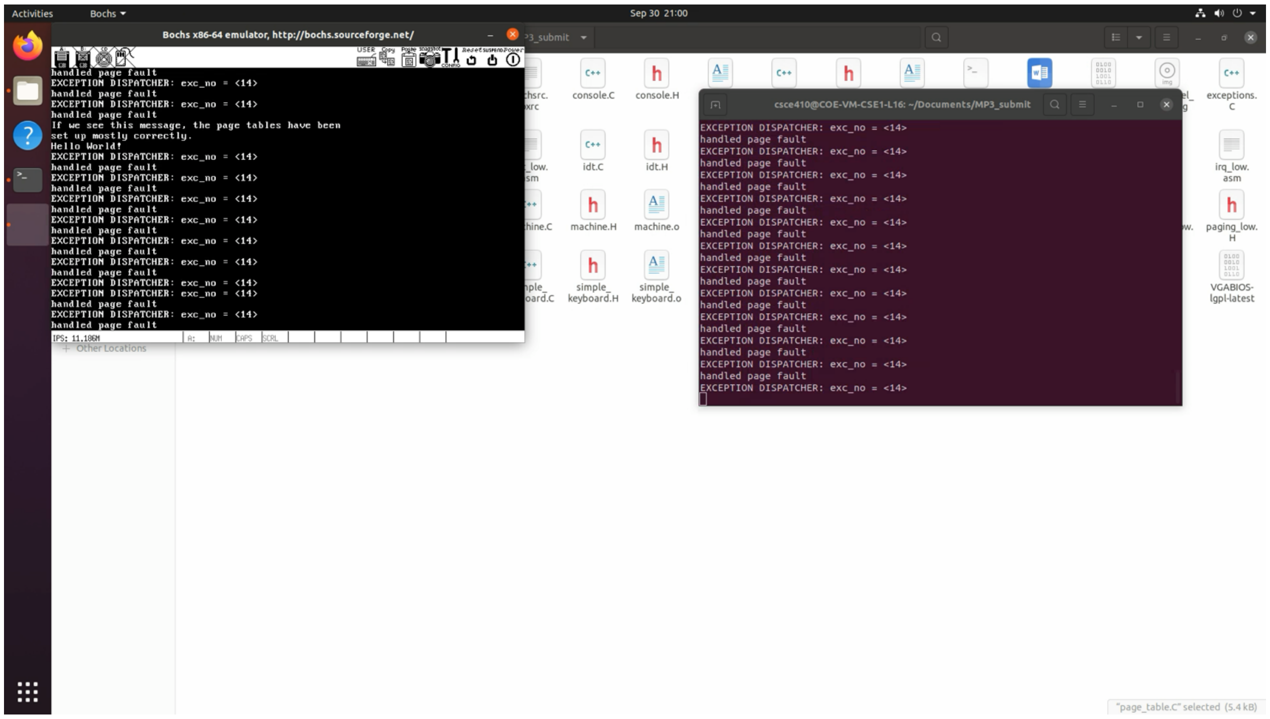
e. ***PageTable::handle_fault(REGS * _r)***

i. As described in the question document, whenever a page fault occurs, the fault address will be stored in CR(). Hence as soon as the fault happens I first read CR2 and take it in variable.
ii. Since we are using 2 level paging, we need to decode 2 addresses (or rather indices!) from CR2.
iii. The 10 MSBs of CR2 will give us the page directory index. This means from the page directory base address(if it is valid) how many indices I need to travel to get the page table address.
iv. The next 10 bits (bit 12 till bit 21) will give us the page table index. This means in the page table where is my entry for the actual page.
v. Also its extremely crucial to note that the entries in page directory as well as page table have last 12 bits of meta data! So to decode the actual address, I am masking the last 12 bits with 0s. however lets first use the meta data to check the valid(here present) bit.
vi. Once it get the directory address, I get the directory entry and first check if the LSB is 1 or 0. If LSB is 0 that means that the entry is not valid and a PDE (page directory) fault has happened.
vii. If a PDE fault has occurred, I request a frame from continuous frame pool and assign it. This is a kernel frame as this will be referring to a page.
viii. I also make sure to make an entry with valid bit as 1 in PDE so that next time this fault does not happen.
ix. Apart from this I also make sure that for the newly assigned frame other entries except  the one made just now are invalid as we don't have any valid entries there.
x. Finally I check the last bit of the PTE entry that I had got from the bit 12 to 21 that if the entry is valid.
xi. If that entry is not valid, I request a process frame pool frame from the continuous frame pool and assign it to the entry where fault has happened. Now here since it was a page, I tool frame from process frame pool.
xii. I make sure to make the entry of this address in page table and mark it valid so that next time the fault will not happen.



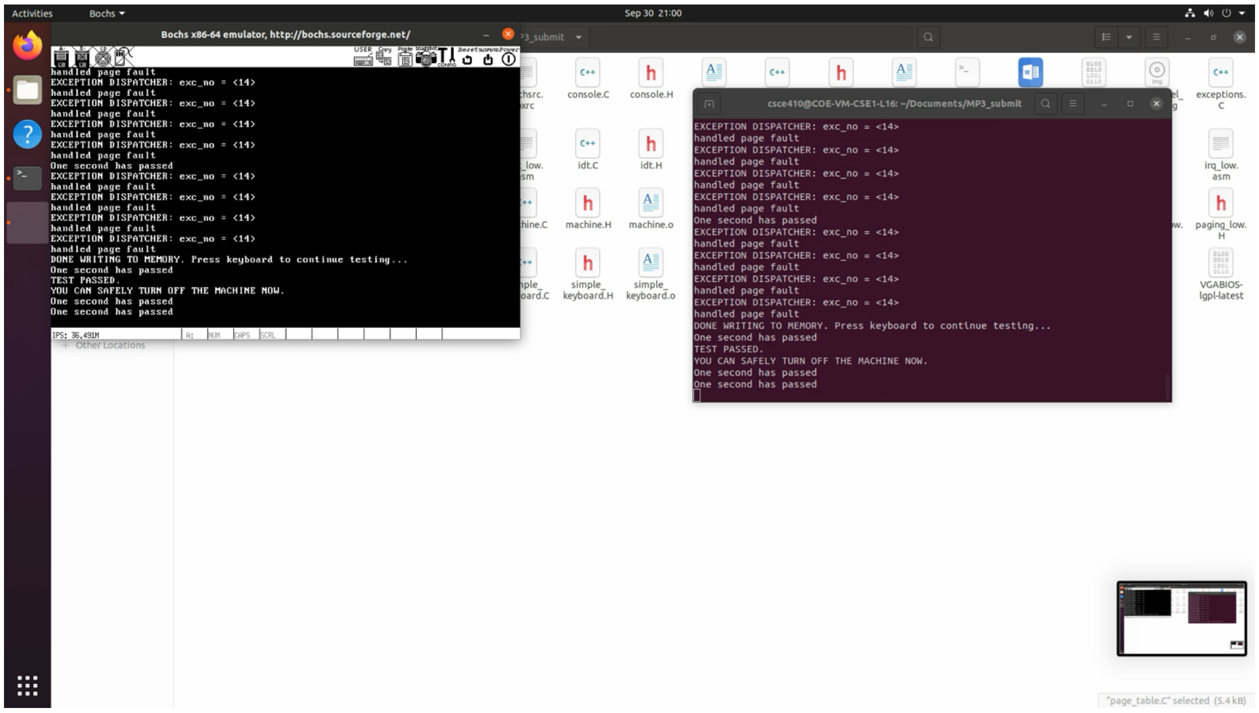
```
void PageTable::handle_fault(REGS * _r)
{
   unsigned long page_fault_addr = read_cr2(); //get the virtual addr of page fault
   unsigned long pde_idx = page_fault_addr >> 22; // extract the 10 MSBs to get Page directory(pde) index
   unsigned long pte_idx = ((page_fault_addr >> 12) & 0x3FF); // extract bit 12 to 21 to get the page table index

   unsigned long* page_dir_addr = current_page_table->page_directory;
   // mask the meta data so you can actually get the pte address
   unsigned long *pde_entry_without_meta_data = (unsigned long *) (page_dir_addr[pde_idx] & ~0xFFF);
   unsigned long *pte_entry = (unsigned long *) pde_entry_without_meta_data[pte_idx]; // get the pte address
  // check if pde index addr has valid bit =1- if not valid then handle pde fault
   if ((page_dir_addr[pde_idx] & 1) == 0)
   {
      // if pde is not valid then get a kernel frame pool's frame and assign to the entry of pde
      unsigned long* new_pde_frame = (unsigned long *) (kernel_mem_pool->get_frames(1)*PAGE_SIZE);
      // mark in pde
      page_dir_addr[pde_idx] = (unsigned long) (new_pde_frame);
      page_dir_addr[pde_idx] = page_dir_addr[pde_idx]| 3; // set meta data to R/W, supervisor and present
      for(unsigned long i=1;i<ENTRIES_PER_PAGE;i++)
      {
         new_pde_frame[i] = 2; // make toehr entries invalid
      }
   }
   // go to pte and check if that entry is valid
    if ((pde_entry_without_meta_data[pte_idx] & 1) == 0)
   {
      // if pte is not valid then get a frame from process frame pool and assign it there
      unsigned long* new_pte_frame = (unsigned long *) (process_mem_pool->get_frames(1)*PAGE_SIZE);
      // make entry in pt
      pde_entry_without_meta_data[pte_idx] = (unsigned long) (new_pte_frame);
      pde_entry_without_meta_data[pte_idx] = pde_entry_without_meta_data[pte_idx]| 3; // set meta data to R/W,supervisor and present
   }
   Console::puts("handled page fault\n");
   // good to return!
}
```

**Following is the screenshot when I run the code: ( Steps in 3 images—pls scroll down)**



```
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
If we see this message, the page tables have been
set up mostly correctly.
Hello World!
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
```

```
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
```

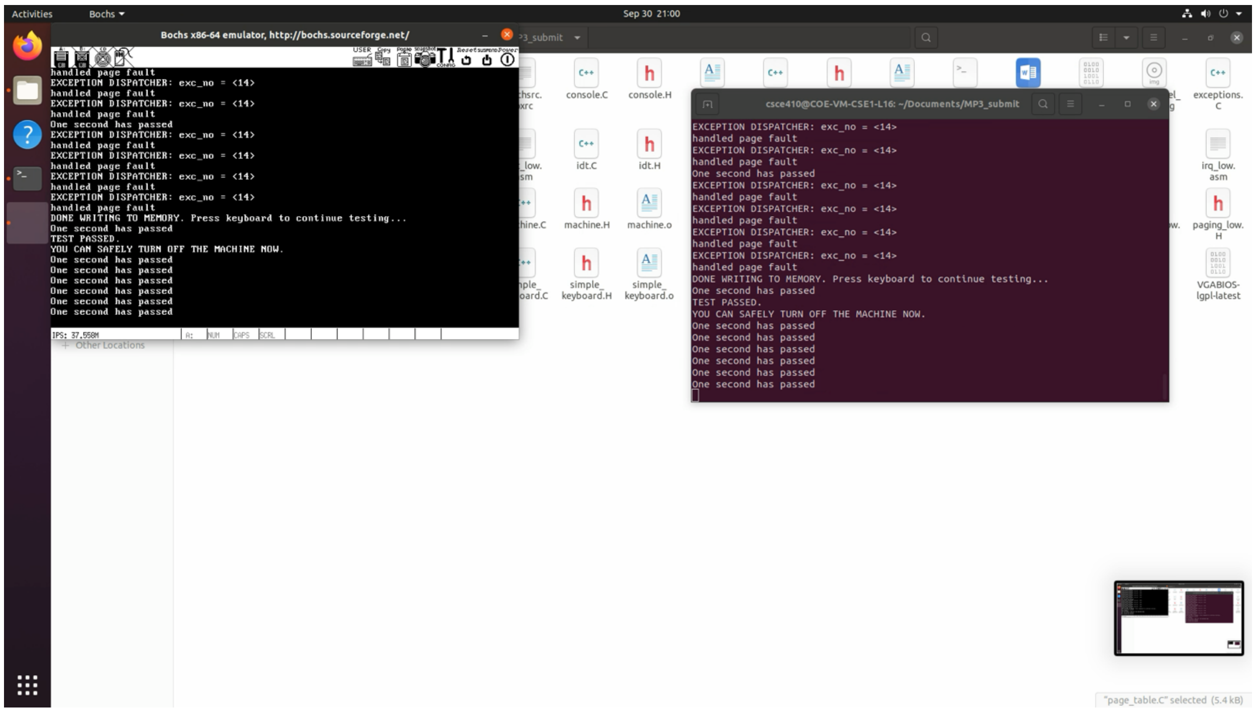


Bochs x86-64 emulator, http://bochs.sourceforge.net/

```
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
One second has passed
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
DONE WRITING TO MEMORY. Press keyboard to continue testing...
One second has passed
TEST PASSED.
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed
```

csce410@COE-VM-CSE1-L16: ~/Documents/MP3_submit

```
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
One second has passed
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
DONE WRITING TO MEMORY. Press keyboard to continue testing...
One second has passed
TEST PASSED.
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed
```



Bochs x86-64 emulator, http://bochs.sourceforge.net/

```
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
One second has passed
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
DONE WRITING TO MEMORY. Press keyboard to continue testing...
One second has passed
TEST PASSED.
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
```

csce410@COE-VM-CSE1-L16: ~/Documents/MP3_submit

```
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
One second has passed
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
EXCEPTION DISPATCHER: exc_no = <14>
handled page fault
DONE WRITING TO MEMORY. Press keyboard to continue testing...
One second has passed
TEST PASSED.
YOU CAN SAFELY TURN OFF THE MACHINE NOW.
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
One second has passed
```