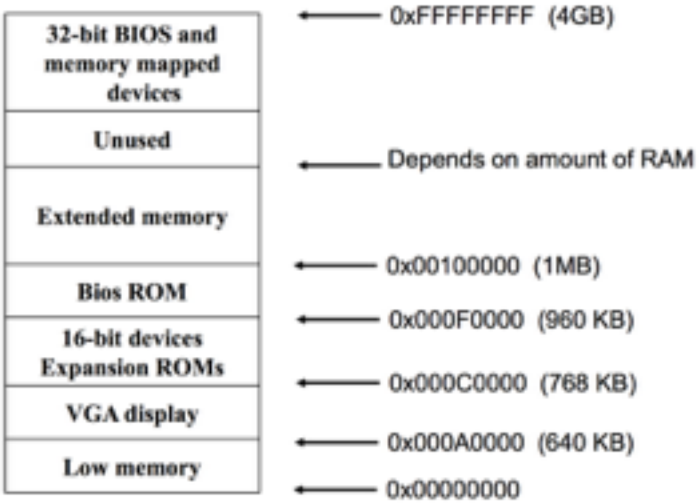
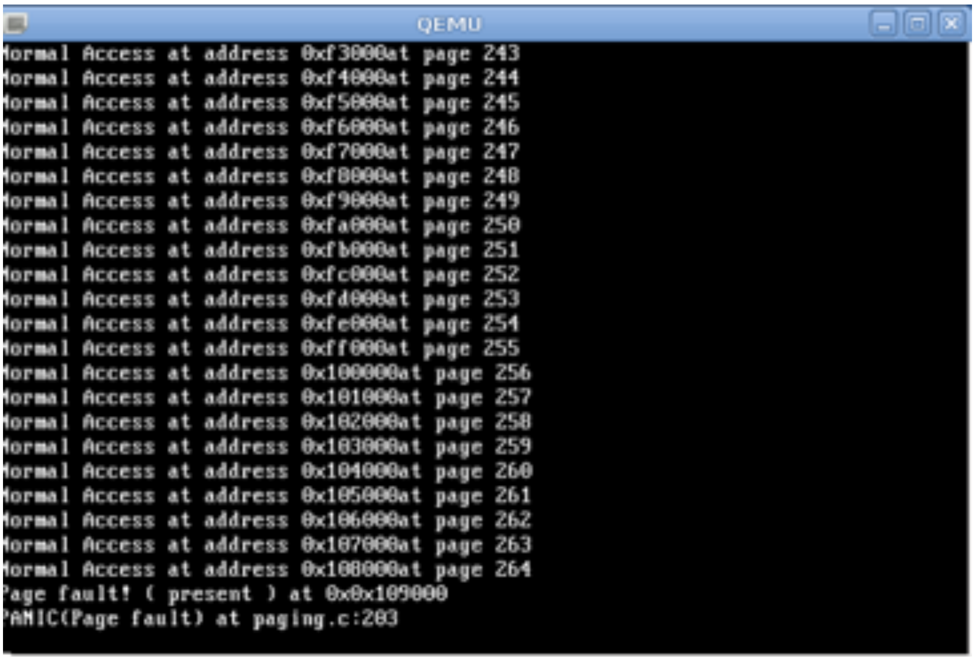
REPORT1:
For the first exercise which asked to find the address for which the program went in page fault, i found that this address number is 0x109000 and the number of the pages were 264.
For that architectures, how the slide explain, the memory address starts from 0x00000000 and arrives until 0x001000000 and after that address there are an extended memory which depends on the RAM , an unused memory and the 32-bit BIOS and memory mapped for devices.

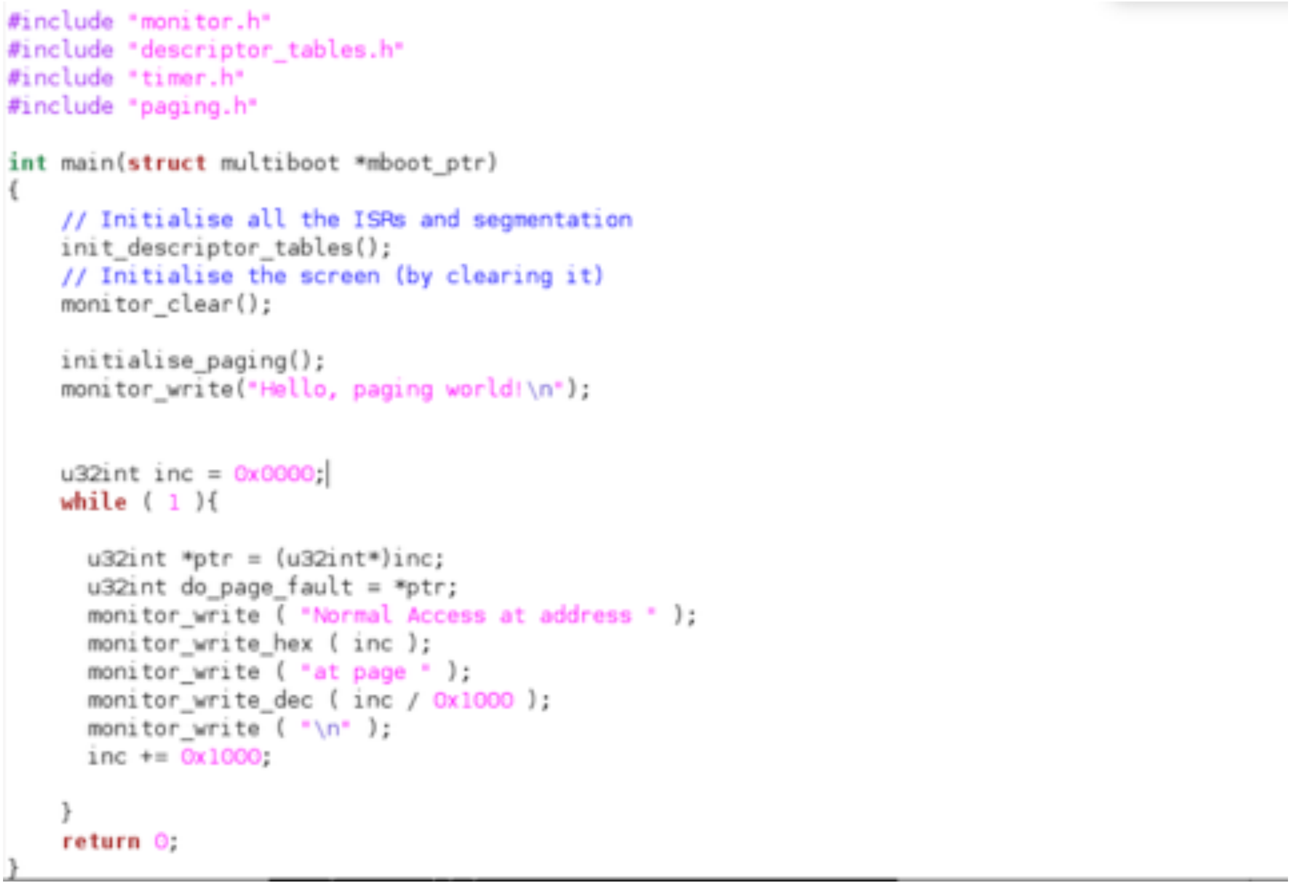


The size of every paging is 4Kb like in almost all processor, for which , how it is possibile to see in the figure above, the user can use only from 0x0000000 address to 0x001000000.
In theory the page for which the page fault occours should be 256, in fact 0x001000000 (1MB) divided by the size of 1 page (0x1000), it results 256 that is the number of page.
But, how we can see in the figure below, the number for which the page faults occurs is not 256 but 264.



```
Normal Access at address 0xf3000at page 243
Normal Access at address 0xf4000at page 244
Normal Access at address 0xf5000at page 245
Normal Access at address 0xf6000at page 246
Normal Access at address 0xf7000at page 247
Normal Access at address 0xf8000at page 248
Normal Access at address 0xf9000at page 249
Normal Access at address 0xfa000at page 250
Normal Access at address 0xfb000at page 251
Normal Access at address 0xfc000at page 252
Normal Access at address 0xfd000at page 253
Normal Access at address 0xfe000at page 254
Normal Access at address 0xff000at page 255
Normal Access at address 0x100000at page 256
Normal Access at address 0x101000at page 257
Normal Access at address 0x102000at page 258
Normal Access at address 0x103000at page 259
Normal Access at address 0x104000at page 260
Normal Access at address 0x105000at page 261
Normal Access at address 0x106000at page 262
Normal Access at address 0x107000at page 263
Normal Access at address 0x108000at page 264
Page fault! ( present ) at 0x0x109000
PANIC(Page fault) at paging.c:203
```

I supposed that 264 comes from the fact that this 8 extra page are used in the extended memory to allocate the paging table.
To find this number, i modified the main given by the professor by inserting an incremental variable by 0x1000 (4Kb) inside an infinite loop and, by assigning at each loop this variable to a pointer i found the number of pages.
I choose to increment by 0x1000 because at each loop i wanted to point to a new page in memory, in fact how i said before the page size is 0x1000 (4KB).



```
#include "monitor.h"
#include "descriptor_tables.h"
#include "timer.h"
#include "paging.h"

int main(struct multiboot *mboot_ptr)
{
    // Initialise all the ISRs and segmentation
    init_descriptor_tables();
    // Initialise the screen (by clearing it)
    monitor_clear();

    initialise_paging();
    monitor_write("Hello, paging world!\n");


    u32int inc = 0x0000;
    while ( 1 ){

      u32int *ptr = (u32int*)inc;
      u32int do_page_fault = *ptr;
      monitor_write ( "Normal Access at address " );
      monitor_write_hex ( inc );
      monitor_write ( "at page " );
      monitor_write_dec ( inc / 0x1000 );
      monitor_write ( "\n" );
      inc += 0x1000;

    }
    return 0;
}
```