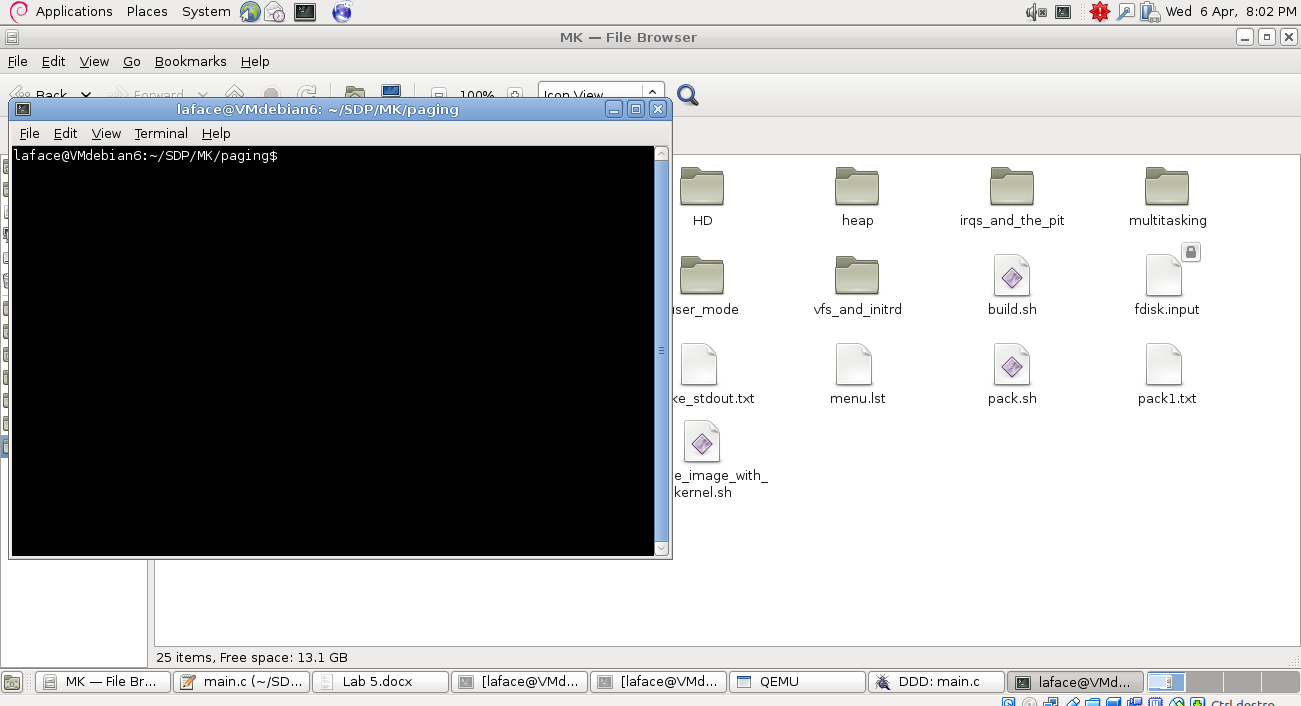
Guide to the lab5 (unofficial)

# Finding the files to use

Once the virtual machine is installed, go to **/home/laface/SDP/MK**. All the files needed are there. For lab 5 the subfolder **paging** is used.

# Build

Open a terminal and go to **/home/laface/SDP/MK/paging**



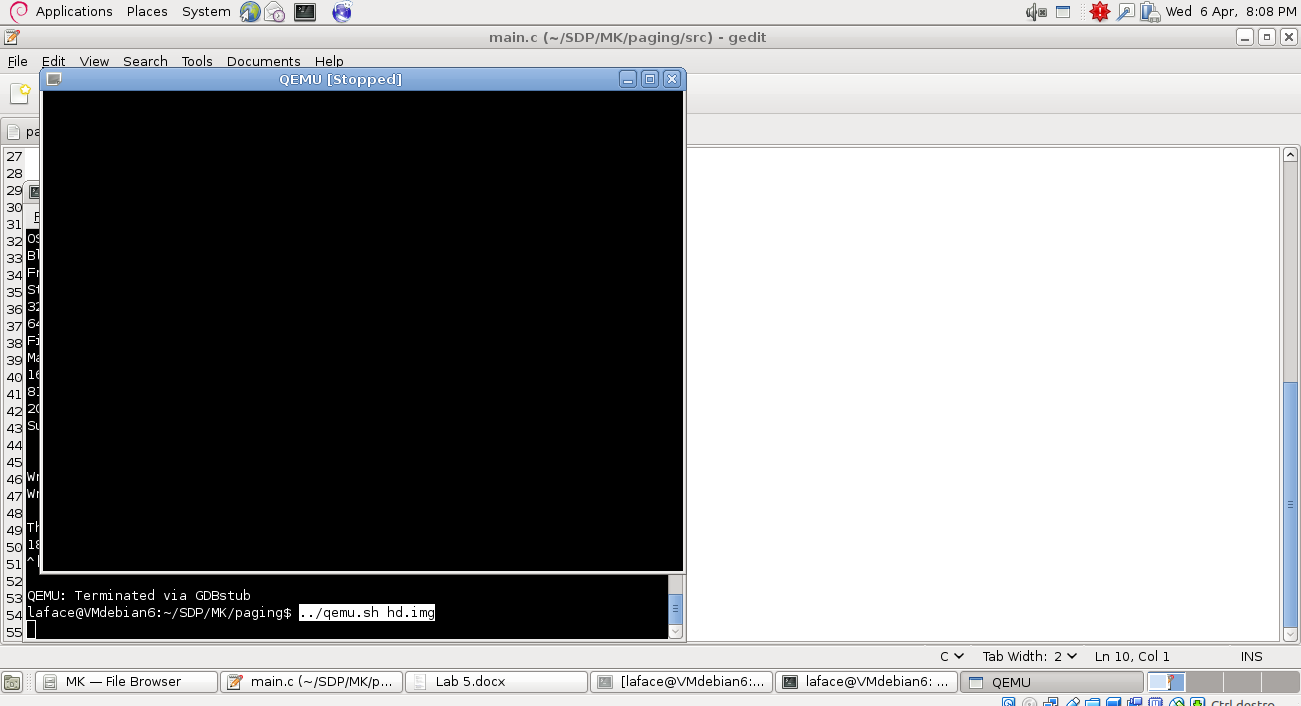
Run: **rm hd.img** to remove the image in order to avoid successive errors in the boot phase.

Run: **./build.sh ; cat src/make\_stderr.txt** to run the build script and to see the output of the compilation. Strange warnings can be discarded

Run: **sudo ../pack.sh 520** to create the **hd.img** file

# Run qemu in debug mode

Run: **../qemu.sh hd.img** and a black window should appear (in a Stopped state)

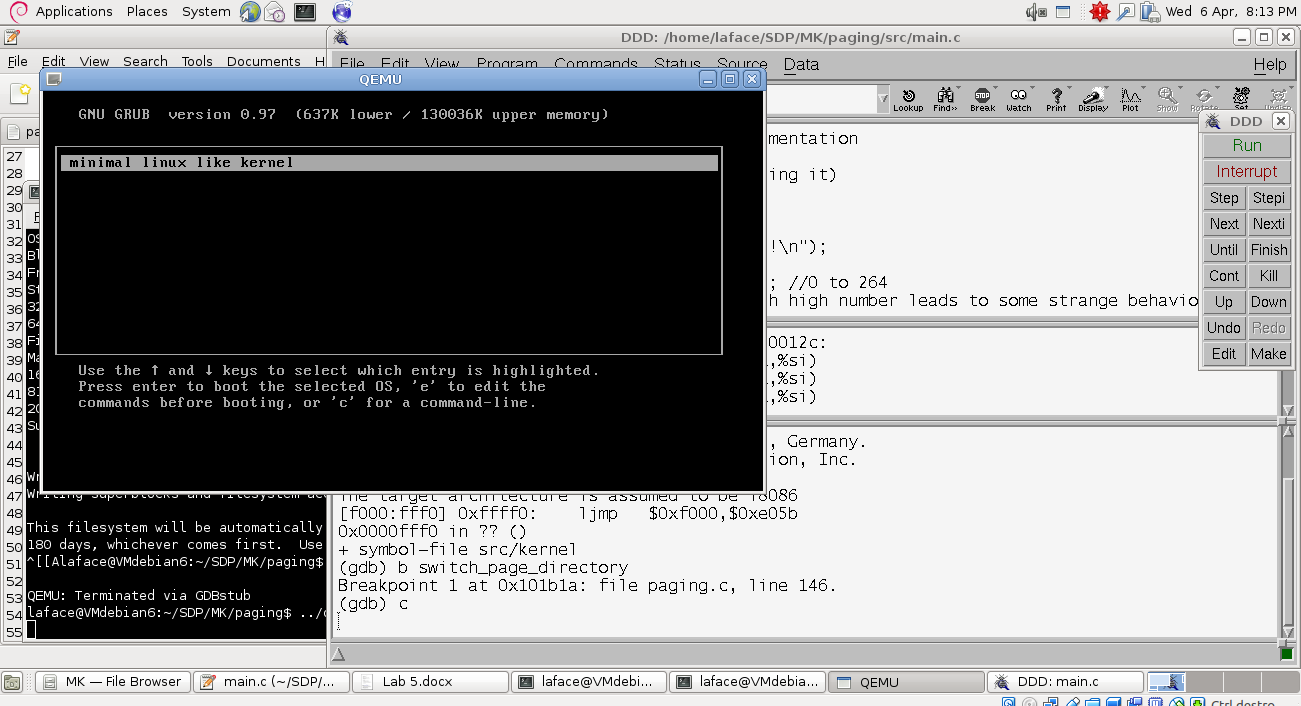


Now open another terminal in the same directory (**/home/laface/SDP/MK/paging**) and run: **ddd**

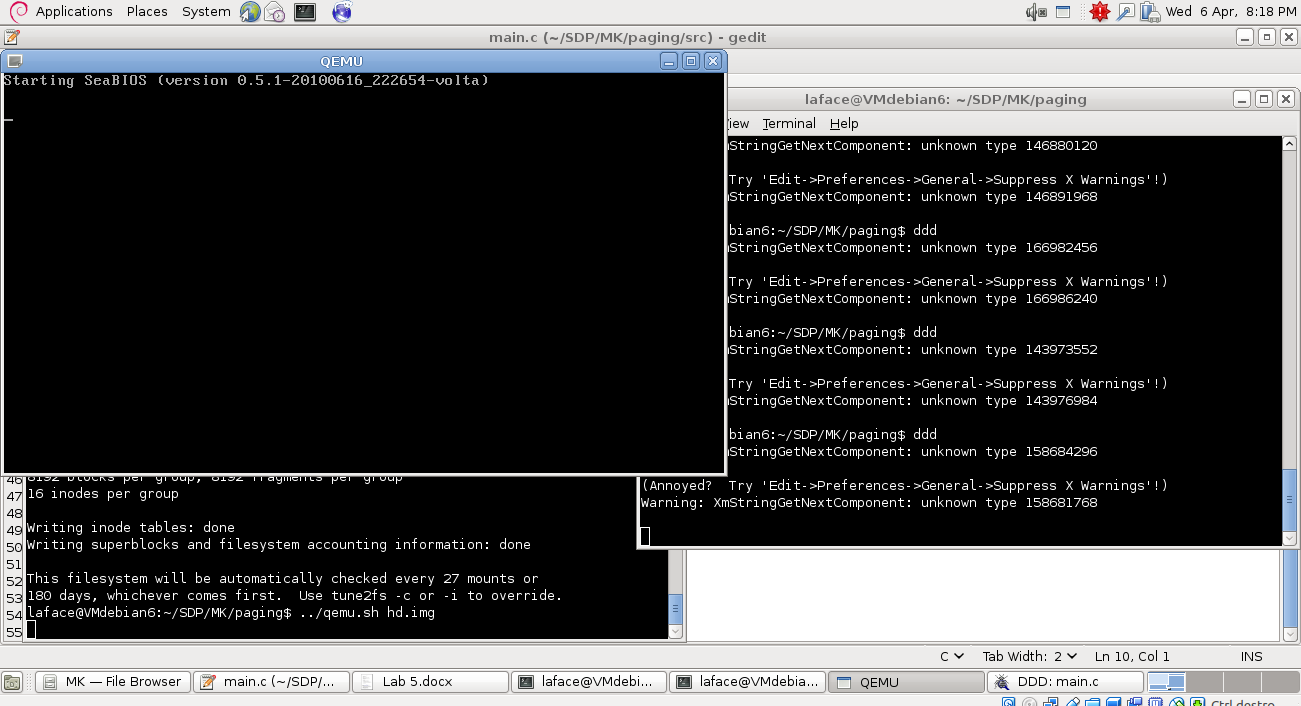
Inside the ddd command window type the following commands:

**b switch\_page\_directory** to create a breakpoint inside the function switch\_page\_directory.

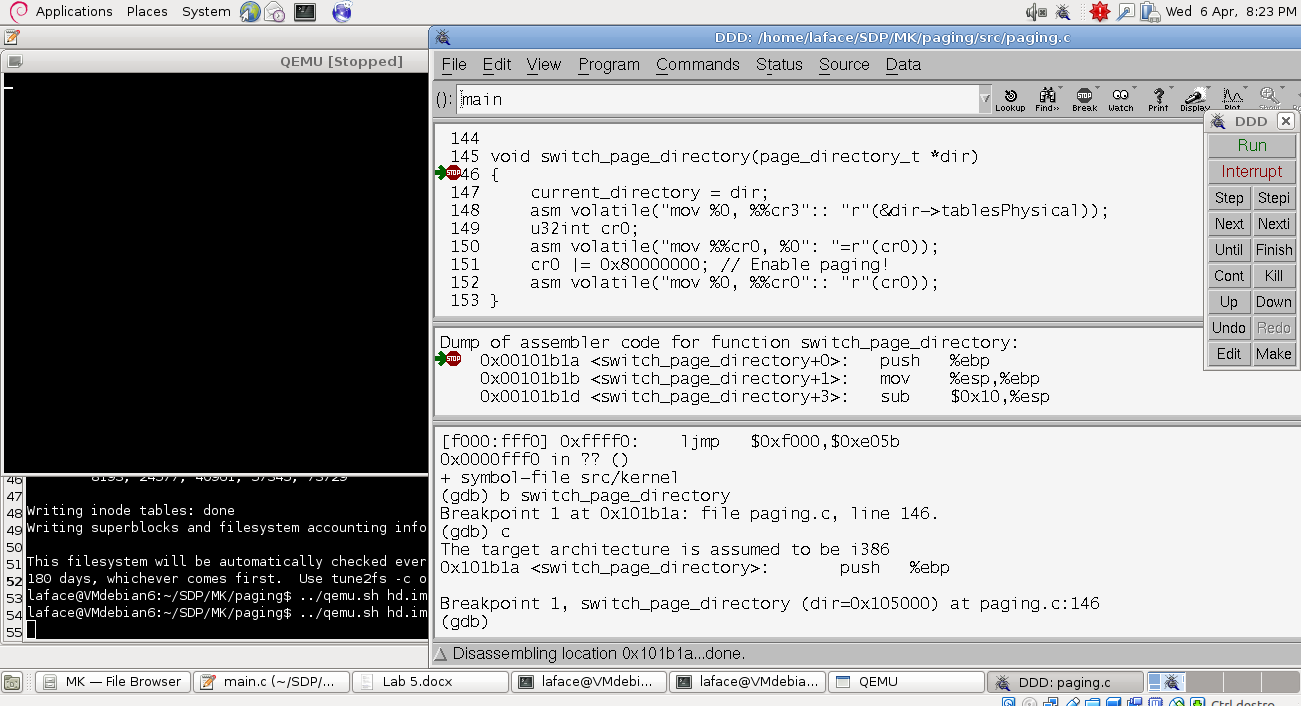
**c** to continue (this should resume the execution of qemu, making it display the grub boot menu (like in the picture).



If instead you see something else instead of grub like in the following image with blinking strings, it means that you need to rebuild (remove, build and pack).



Now press enter into qemu and you should see qemu in state stopped and ddd pointing with a green array to the current istruction

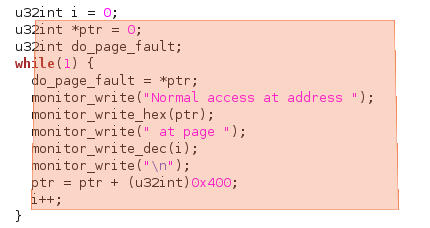


Now you can debug.

# Exercise 1

Only the main.c needs to be modified.

Create a loop that tries to access at increasing addresses (0x0, 0x1000, 0x2000, …) till a page fault occurrs.



# Exercise 2

To swap the mapping between the pages and frames, you need to:

1 – Go back to real mode (without the paging mechanism)

2 – modify the informations on the pages

3 – turn on again the paging mechanism

To be able to perform step 1 and three, be inspired by the switch\_page\_directory function: after setting up sr3, it reads the content of sr0 and places its value inside the variable sr0; then the 31th bit is turned to the value 1 by doing a OR with the appropriate mask (to leave the other bits as they are); at the end the value of the variable is loaded into the register.

Instead to be able to perform the modifications in step 2, you need to modify the information inside the global variable kernel\_directory inside the paging.c file.

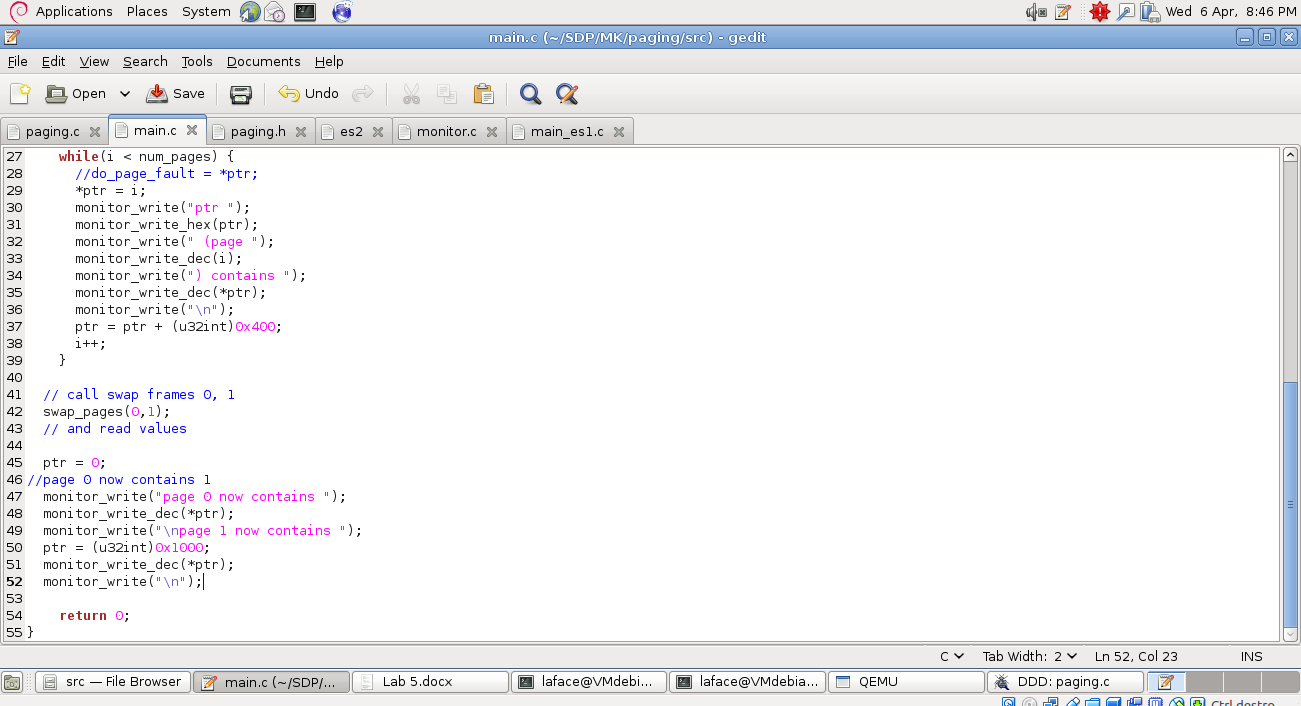
I created inside the file paging.c the following function:

void swap\_pages(u32int pn1, u32int pn2) {  
 u32int fr\_n1, fr\_n2;  
   
 page\_directory\_t \*dir = kernel\_directory;  
 //disable paging  
 u32int cr0;  
 asm volatile("mov %%cr0, %0": "=r"(cr0)); // read register cr0  
 cr0 ^= 0x80000000; // Disable paging by turning off the 31th bit (^ is a XOR)  
 asm volatile("mov %0, %%cr0":: "r"(cr0)); // write register cr0  
   
 // swap pages  
 fr\_n1 = dir->tables[0]->pages[pn1].frame;  
 fr\_n2 = dir->tables[0]->pages[pn2].frame;  
 dir->tables[0]->pages[pn1].frame = fr\_n2;  
 dir->tables[0]->pages[pn2].frame = fr\_n1;  
   
 // now re-enable paging  
 asm volatile("mov %%cr0, %0": "=r"(cr0)); // read register cr0  
 cr0 |= 0x80000000; // Enable paging by turning on the 31th bit  
 asm volatile("mov %0, %%cr0":: "r"(cr0)); // write register cr0  
}

Then in paging.h i make the function visible declaring the prototype:

void swap\_pages(u32int pn1, u32int pn2);

And I use it in the main:



If strange behavior is detected, try to set the value of num\_pages less than the number found in exercise 1. Writing thing to pages >= 160 makes strange things happen. The professor himself in his code limits to 160 pages (0 to 159).