**오퍼레이팅시스템\_1주차**

**12180626 성시열**

1) Install Gentoo Linux on virtual machine.

1.0) Download Virtualbox from Internet and install.

1.1) Download Gentoo virtualbox files (gentoo.zip) from the I-class and un-compress it.

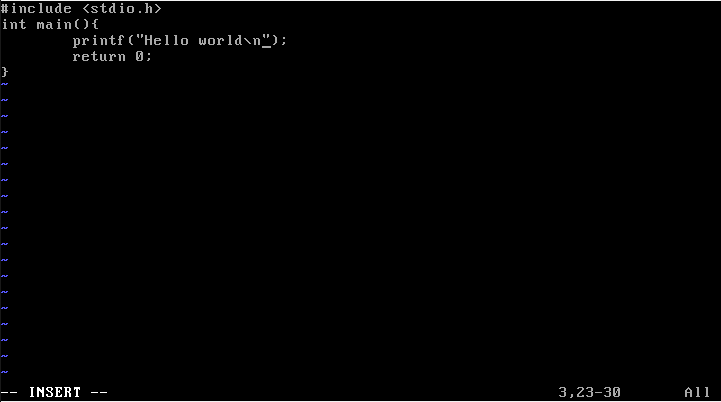
1.2) Run VirtualBox, and click File>Import and go to the gentoo directory. Select Gentoo2.ovf. Uncheck USB controller. Select Import. This will install Gentoo Linux on your virtual box.

1.3) Run Gentoo. If you have an error, run VirutalBox as administrator and try again. For USB error, simply disable usb controller in "Setting" tab. For Hyper-V error (Raw-mode is unavailable), turn off Hyper-V feature in control panel>program and feature>window feature. Select My Linux. Login as root and hit Enter for the password prompt. If VirtualBox still cannot open the session, you need to look at the log file (right click on Gentoo VM and select “log file”) and see what is the error and fix it.

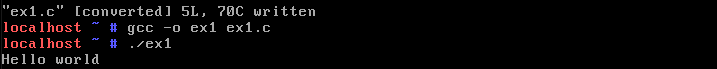
(In some cases, you may need to download and install virtualbox extension package.)

1.4) Make a simple program, ex1.c, with vi which displays "hello world". Compile and run it.

가상환경을 세팅한 뒤 vi 편집기를 활용하여 ex1.c의 코드를 작성한다. vi ex1.c



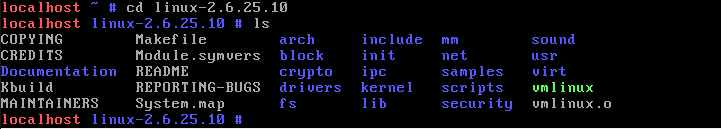
:wq를 통해 ex1.c에 코드를 저장하고 vi 편집기를 탈출한다.



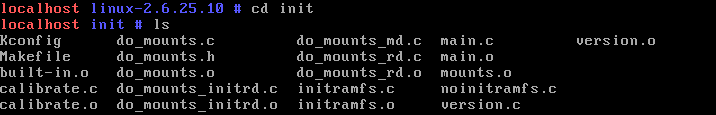
이후 gcc -o ex1 ex1.c를 통해 컴파일을 하고 ./ex1 을 통해 실행하면

printf를 통한 Hello world를 출력한다.

2) Go to linux-2.6.25.10 directory and find all the files referred in Section 1 such as main.c, fork.c, entry\_32.S, etc.



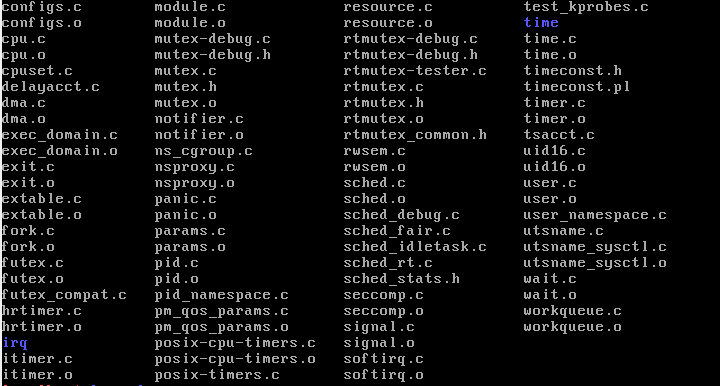
cd to linux-2.6.25.10 를 통해 해당 디렉토리로 이동한 뒤 ls를 통해 해당 디렉토리에 존재하는 파일들을 모두 확인하였다.



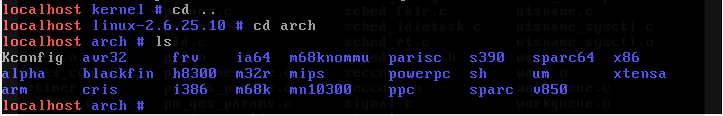
처음으로는 cd init으로 이동한 뒤 ls를 통해 파일을 확인해본 결과 main.c가 init에 존재하는 것을 알 수 있다.



다음은 kernel로 이동하여 파일을 확인하였다.



fork.c가 존재함을 확인하였다.



다음은 arch로 이동하여 파일을 확인한 결과 위 처럼 다양한 디렉토리가 내부에 존재함을 확인할 수 있다.



entry\_32.S를 찾기 위해 디렉토리를 이동시켜본다. ls를 통해 파일을 확인한다.

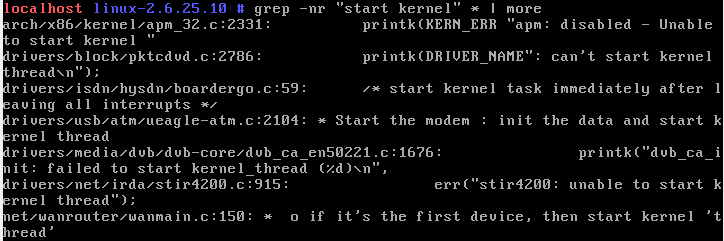
3) Find the location of start\_kernel().

To find a string "start\_kernel", go to the linux top directory (linux-2.6.25.10) and do

$ grep -nr "start\_kernel" \* | more

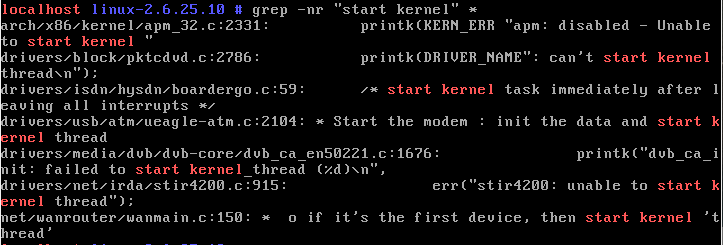
Use "space" to move down the screen, "q" to exit

cd를 통해 디렉토리 위치를 변경한 뒤



grep -nr "start\_kernel" \* | more를 통해 start\_kernel이 들어간 문자열을 찾아보았다. -nr 옵션g 중 -n은 문자열이 나온 번호를 알려주는 옵션이고, -r은 하위 디렉토리를 내려가면서 모두 찾는 옵션이다. \*은 디렉토리의 모든 파일에 대해 명령어를 실행한다는 뜻이다.

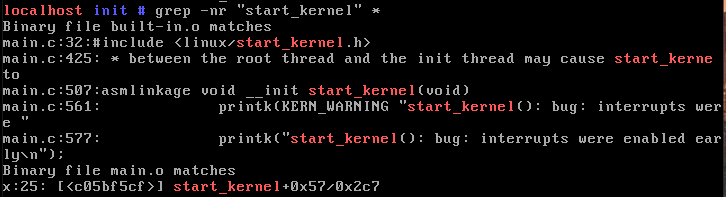
| more은 페이지 단위로 값을 출력한다.



전체 출력 내용이 한페이지가 되지 않아 | more을 제외하고 출력해본 결과

start kernel이 붉은 색 글씨로 표시됨을 확인하였다.

init 디렉토리에서 grep -nr "start\_kernel"를 해본 결과 main.c 507번 라인에서 호출됨을 알 수 있다.



Once you found the file that has "start\_kernel", use "vi" to read the file.

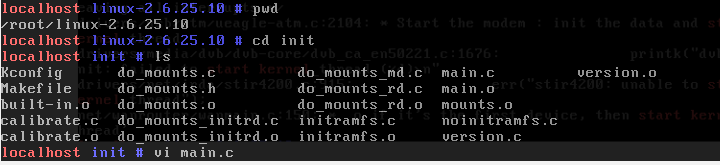
In vi, type "/start\_kernel" to search for the first instance of "start\_kernel".

For the next string, simply type "/". Repeat "/" until you find the start\_kernel() function.

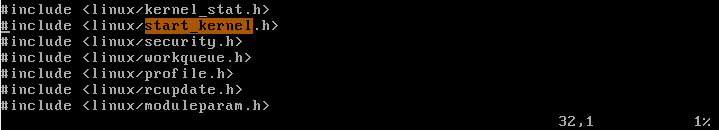
Use "j" to mode down the cursor, "k" to move up, "^f" to move one screen down, "^b" to move

up one screen up.

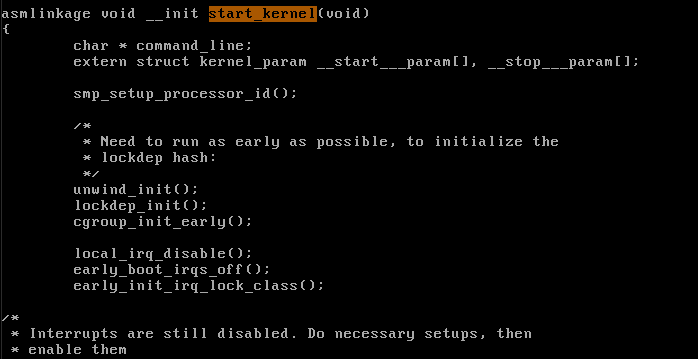
start\_kernel파일을 찾기 위해 cd를 활용하였다. init 디렉토리로 이동한 뒤 ls를 통해 main.c가 존재함을 확인하였다.



vi 편집기를 통해 main.c 파일을 열어보았다.



수많은 코드 중 32번째 라인에 #include <linux/start\_kernel.h> 코드가 존재함을 확인하였다.



/start\_kernel을 통해 찾아주고 ctrl+b와 ctrl+f를 통해 자유롭게 start\_kernel을 찾을 수 있었다.

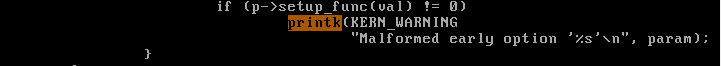
4) start\_kernel() is the first C function run by Linux. Predict what will be the first message appearing on the screen by analyzing start\_kernel(). Note that "printk()" (not "printf") is the function to print something in the kernel.

이번엔 /printk를 통해 출력문이 어떤 것들이 있는지 파악해본다.

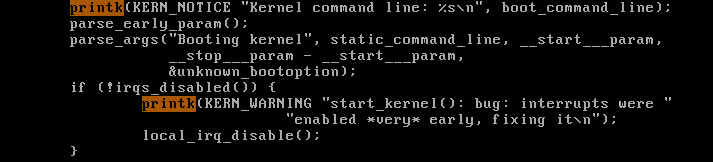






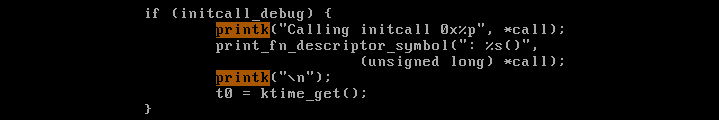


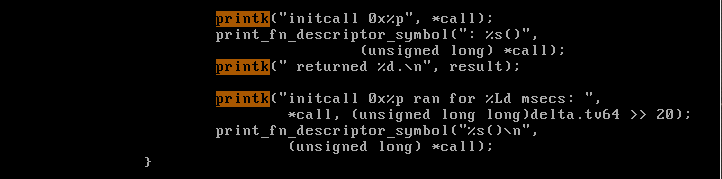


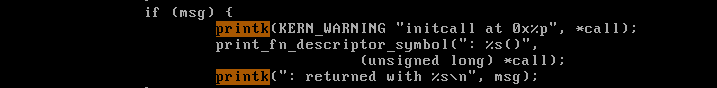




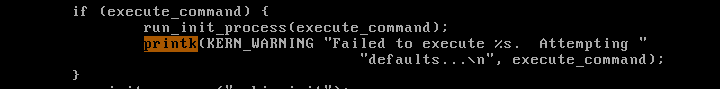










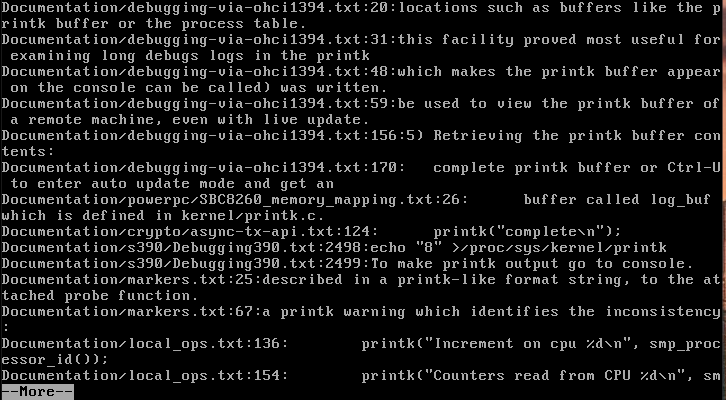


모든 printk를 찾아보았다. start\_kernel이 리눅스가 시작될 때 가장 먼저 실행되는 함수이므로, printk 뒤의 문장들이 부팅할 때 실행되는 문장임을 짐작해볼 수 있다. 하지만 if문 안에 있는 문장들은 조건에 맞지 않으면 실행되지 않고, main.c 내부의 호출된 함수 내부에 printk 가 있을 수 있으므로 이는 정확한 예측이 아니다.

정확한 예측을 위해서는 모든 함수를 모두 따라가고, 그 함수가 호출하는 함수까지 모두 따라가서 printk를 추적해야한다.



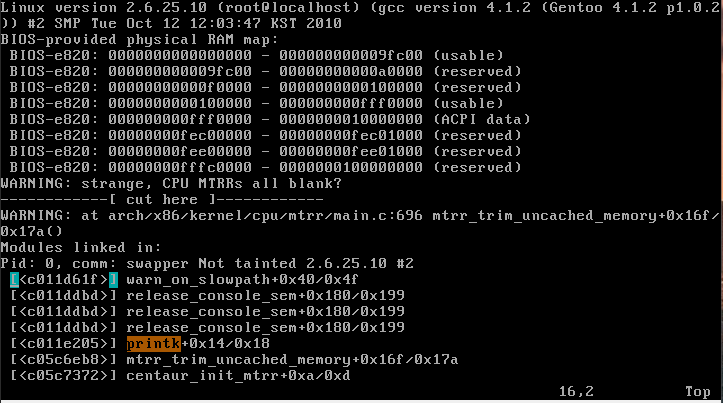
이를 위해 상위 디렉토리로 이동하여 linux-2.6.25.10에 존재하는 모든 printk(함수 내부에 있는 printk까지) 찾아가야 한다. grep -nr printk \* | more을 통해 printk라는 문자열이 들어간 모든 코드를 확인한다.

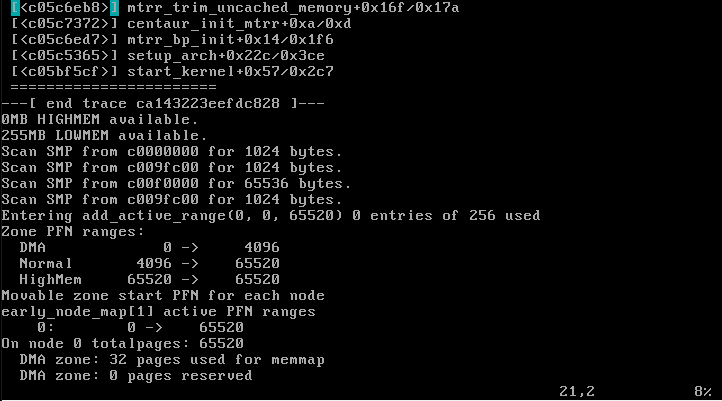


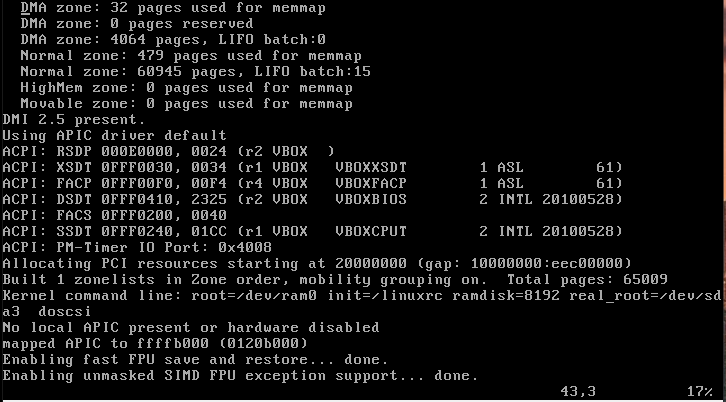
Confirm your prediction with "dmesg > x" and "vi x". The kernel remembers the booting message in the system buffer and dmesg displays the content of this buffer to the screen. "dmesg > x" will send the booting message to file x. With "vi x" you can look at the file x.

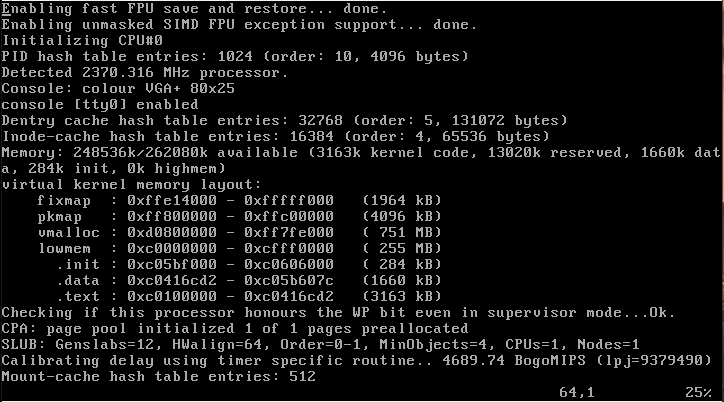
dmesg > x 를 통해 부팅 메시지를 파일 x에 저장하고, vi x를 통해 파일 x의 내부 코드를 확인한다.

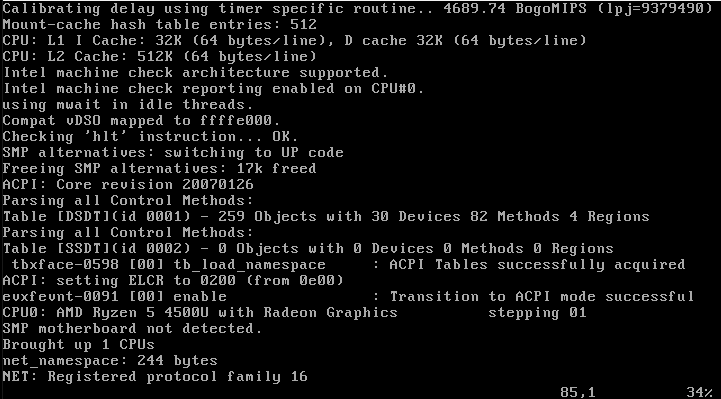


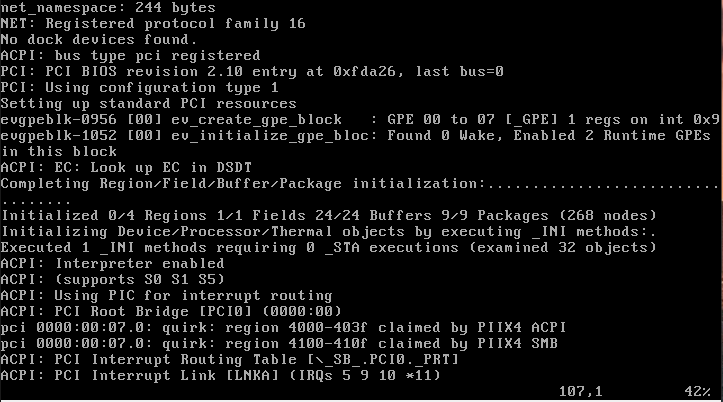












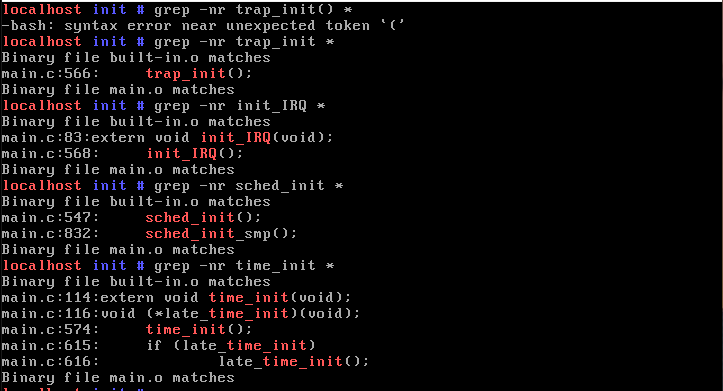
(부팅 메시지 매우 길어 절반 정도를 캡쳐하여 첨부하였습니다.)

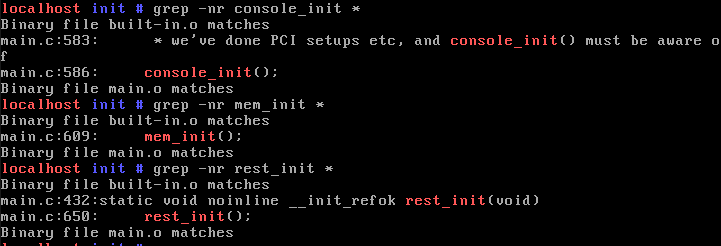
5) Find the location of the following functions called in start\_kernel() and briefly explain your guessing about what each function is doing (but do not copy the whole code).

trap\_init(), init\_IRQ(), sched\_init(), time\_init(), console\_init(), mem\_init(),

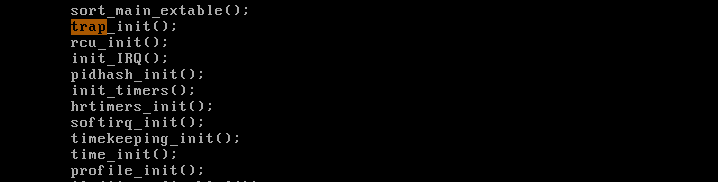
rest\_init()

gerp -nr 을 활용하여 위 함수들의 위치를 파악하였다.



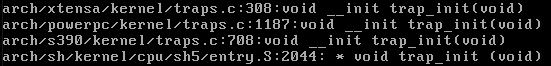


아래 모든 코드들은 vi main.c에서 “/” 기능을 통해 찾은 것들이다.



trap\_init()과 init\_IRQ()를 확인할 수 있다.

grep -nr trap\_init을 통해 \* | more을 찾아본다.







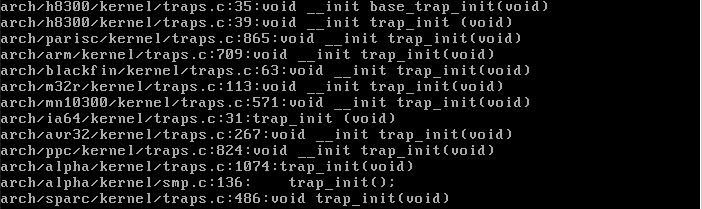






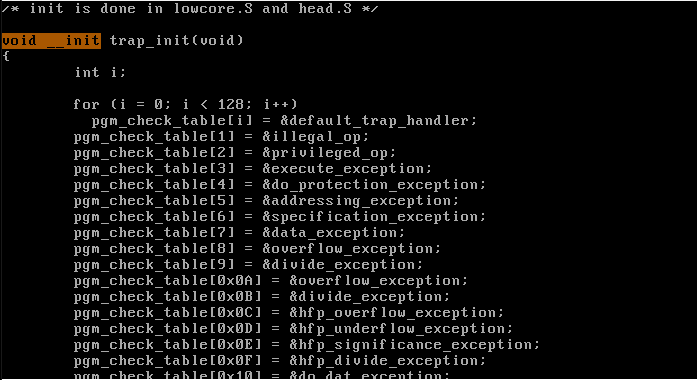






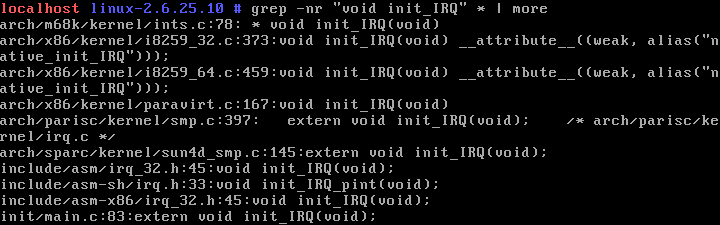
정의를 하는 것 같은 모든 경로를 캡쳐하였다. 경로의 끝은 대부분 traps.c임을 알 수 있다.

경로 끝의 traps.c를 vi 편집기를 통해 확인해본 결과 대부분의 trap\_init() 정의가 비어있었지만 s390 디렉토리 내의 kernel 내 traps.c 함수가 정의됨을 확인하였다.

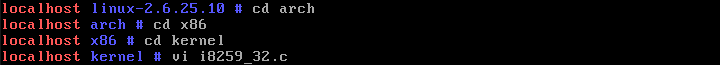


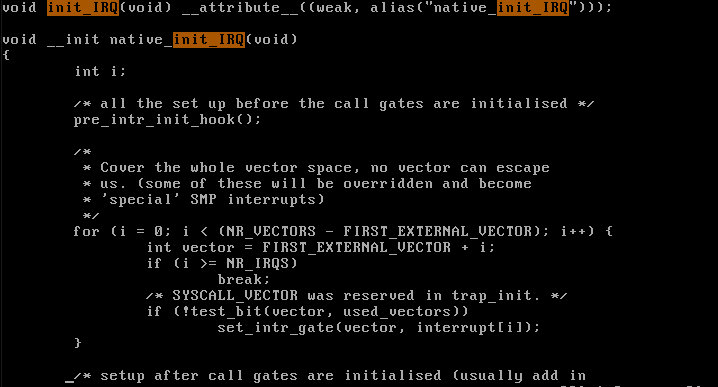
traps.c 함수 내 trap\_init의 정의를 확인해본 결과 프로그램 시작 전 프로그램을 체크하는 코드인 것으로 짐작된다.

grep -nr “void init\_IRQ” \* | more을 통해 찾아본다.



함수 정의여 보이는 경로를 찾아본다.



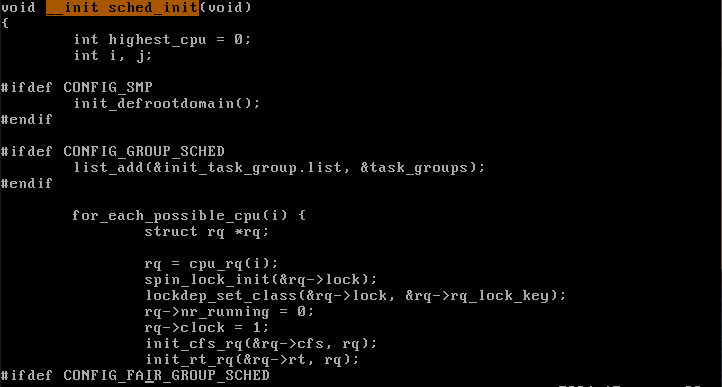


init\_IRQ()에서 IRQ는 Interrupt를 나타내고 이를 초기화할 것으로 예상된다.

grep -nr sched\_init \* | more을 통해 찾아본다.

함수 정의여 보이는 경로를 찾아본다.

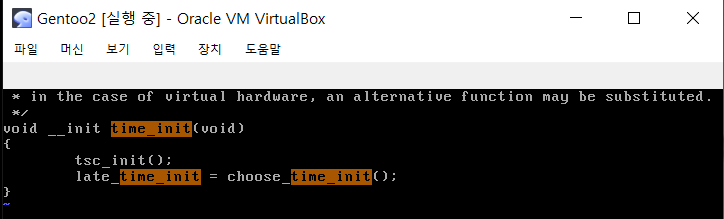
/linux-2.6.25.10/kernel에 sched.c 파일이 있는 것을 확인하였다.



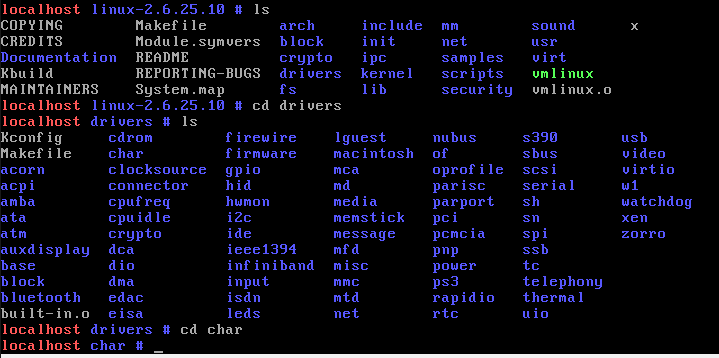
sched\_init()은 interrupt가 실행되는 스케쥴을 초기화 할 것으로 예상된다.

위와 같은 방법으로 아래 함수들도 찾아본다.

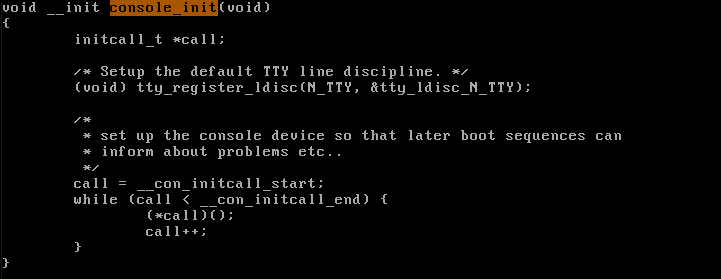
/linux-2.6.25.10/arch/x86/kernel에 time\_32.c 파일을 확인하였다./



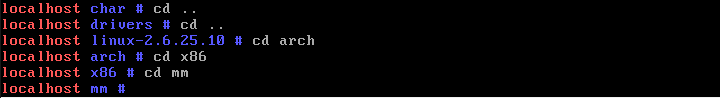
time\_init()은 타이머를 초기화할 것으로 예상된다.



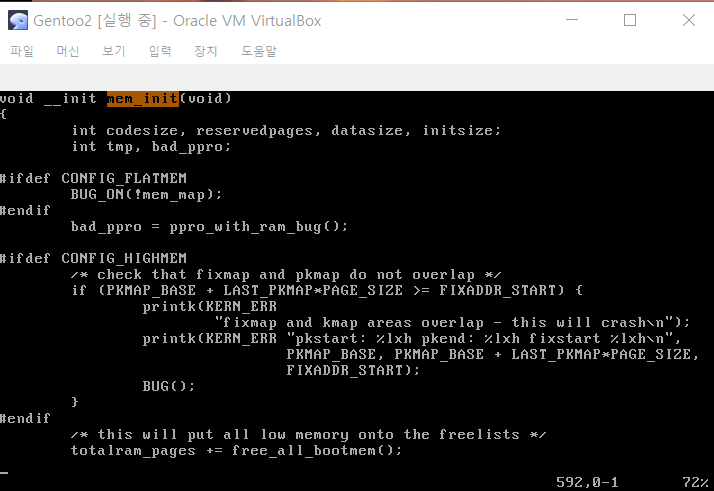
/linux-2.6.25.10/drivers/char 에 tty\_io.c 파일을 확인하였다./



console\_init()은 입력한 콘솔창을 초기화할 것으로 예상된다.



/linux-2.6.25.10/arch/x86/mm 에 init\_32.c 파일을 확인하였다./



mem\_init()은 main.c에 구성된 member를 초기화할 것으로 예상된다.

6) Why can't we use "printf" instead of "printk" in Linux kernel?

리눅스 커널은 libc를 사용하지 못하기 때문에 라이브러리가 없다고 볼 수 있다. 단지, 커널 이미지 내에 포함된 함수들만 사용 가능하다고 볼 수 있다.