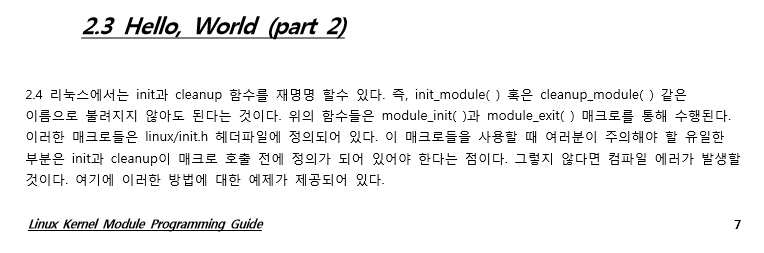
키로거 코드 분석

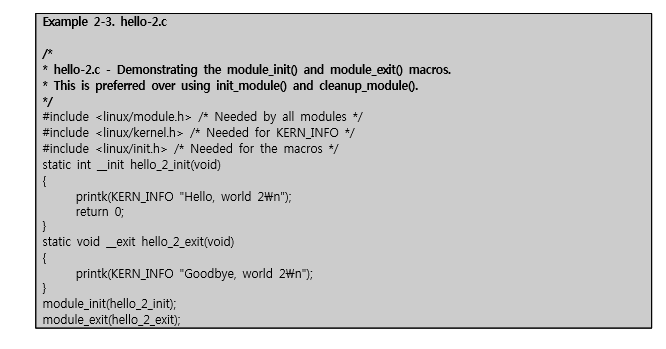
1. module\_init( ) 을 통해 init\_keylogger를 프로그램을 처음 시작하게 함.

모듈 프로그래밍 처음 시작하는 부분은 그 pdf 준거 참고해서 앞부분 읽어보고 이해한담에

아래 진행할 것!!!

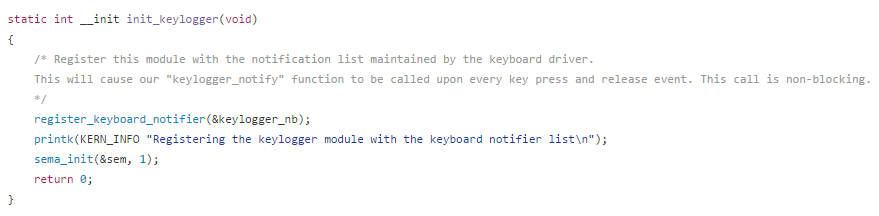






(모듈 프로그래밍 시작은 이거 참고 ㄱㄱ )

2. init\_keylogger 동작



(주석을 보면) 이 코드를 통해 모듈을 키보드 드라이버에 의해 관리되는 notification list에 등록한다고 한다. 이 결과 모든 키가 눌릴 때나 키를 뗄 때의 이벤트가 발생할 때마다 keylogger\_notify 함수가 호출된다. 이 호출은 non-blocking이다.

(blocking / non-blocking 관련 설명 ) : <http://devsw.tistory.com/142>

-> 응답이 올 때 까지 마냥 아무것도 못하고 대기하는 상태로 대기하는 것이 아니라

할 꺼 하면서 응답이 올 때 마다 그때그때 처리한다는 말인 듯.

아직은 이해가 잘 안되는데 결국 키보드 드라이버에 의해 관리되는 notification list라는 녀석한테

키를 읽어 들이는 모듈을 등록해주면 키를 누르고 떼는 동작(이벤트)을 실시간으로 받아올 수 있다는 말인 듯

그 다음 커널 로그에 해당 메시지 출력해주는듯

3. 세마포어 사용(자원에 대한 접근제어 할 때 사용된다고 함.)



세마포어를 1로 세팅해줘서 다른 프로세스가 자원을 사용 가능한 상태로 만들어준다.

(0이면 사용 불가. Up() / down() 를 통해 변경)

아래꺼 보고 공부하자!

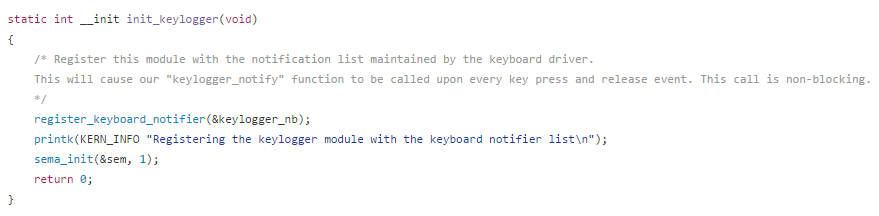
(세마포어 관련 설명) : <http://www.joinc.co.kr/w/Site/system_programing/IPC/semaphores>

<http://egloos.zum.com/agrumpy/v/346057>

4.

메인 코드는 이렇게 동작하는 것 같고 이제





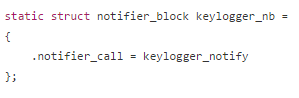
키보드 드라이버에 의해 관리되는 notification list라는 녀석한테

키를 읽어 들이는 모듈을 등록해주면 키를 누르고 떼는 동작(이벤트)을 실시간으로 받아올 수 있는 기능을 살펴보기 위해



이 녀석에 대해 분석해보자

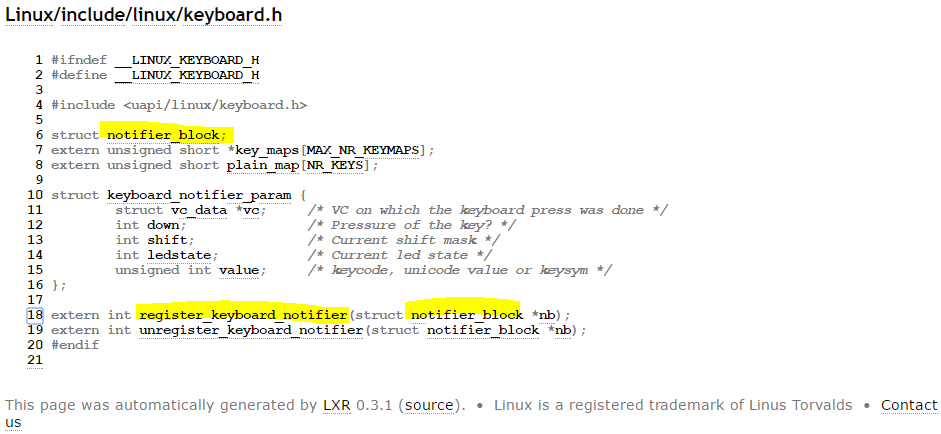
먼저 전달인자로 준 keylogger\_nb 부분을 보면



구조체 형식으로 .notifier\_call에 keylogger\_notify 함수의 주소를 주는 것 같다.

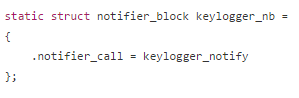
Notifiey 부분은 찾아봤는데 좀 어려운거 같아서 좀 더 찾아보고 공부해 봐야 할 듯

<http://lxr.free-electrons.com/source/include/linux/notifier.h>



기본 라이브러리 함수 같이 키보드 notifier에 등록할 때 그냥 가져다 쓰면 되는 녀석일 듯

5. keylogger\_notify 함수 분석



이 결과 keylogger\_notify 함수가 notifier\_call에 등록이 되어서

키를 누르고 떼는 동작(이벤트)이 일어낼 때 마다

Keylogger\_notify 함수가 호출되는 것 같음.

실시간으로 받아올 수 있는 기능을 아래 keylogger\_notify 에 구현해서 키로거 기능이

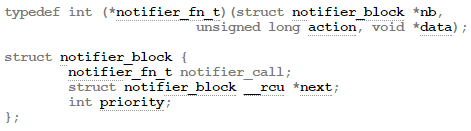
돌아간다.

이제 keylogger\_notify에서 어떻게 키로깅 기능이 구현되어 있는지 살펴보자

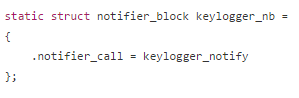


먼저 전달인자들을 살펴보니 아래와 같은 형식으로 정의되어있다.

<http://lxr.free-electrons.com/source/include/linux/notifier.h>



위에서 코드 분석한 결과 아래 notifie call에 keylogger\_notify를 등록하게 되면서 키 입력이 일어날 때 마다 이 정보를 받아올 수 있는 걸 알 수 있었다.

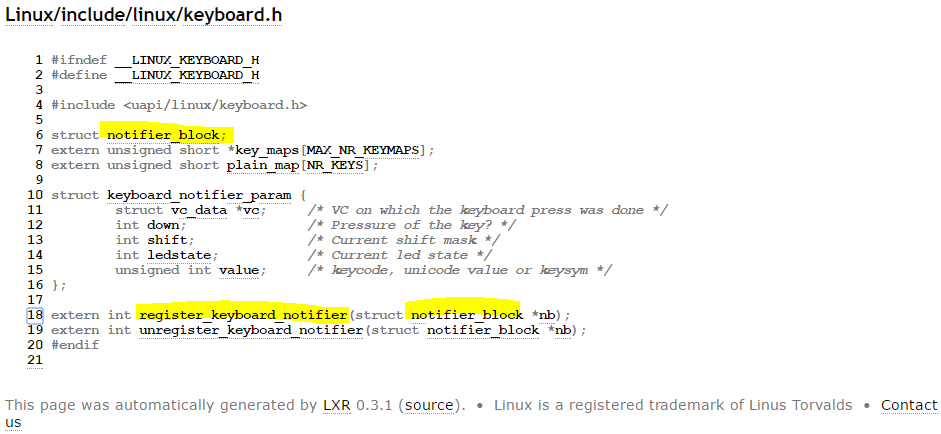


이렇게 받아온 정보는 아래 keyboard\_notifier\_param 구조체에 저장이 되는데

저장될 때

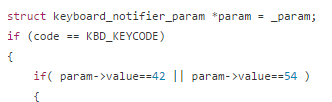


아래 구조체의 형식대로 저장된다.



우리는 키 입력이 일어날 때 마다 이 구조체에 자동 저장되는 정보를 가져다가

커널 로그로 보내주면 된다.



먼저 전달인자 중 code 중에 키보드와 관련된 KBD\_KEYCODE 코드가 있을 때

아래의 코드를 진행하게 된다.

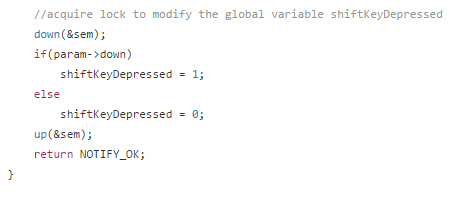
먼저 param->value 가 42와 54일 때는 키가 shift인 경우이다.

이거는 keymap을 확인해 보면 알 수 있는데

Shift의 경우를 고려를 먼저 하는 이유는 shift가 눌려져있는 상황에서 알파벳이 입력될

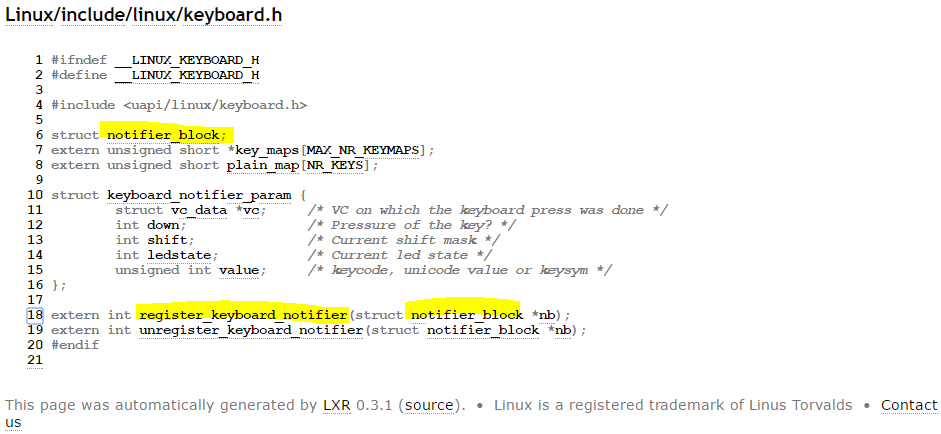
때 대소문자의 차이가 생기기 때문이다.

Shift가 눌려져 있으면 아래의 코드를 계속해서 진행하게 된다.



먼저 세마포어를 0으로 만들어 세마포어가 up으로 다시 1이 되기 전까지

다른 녀석들이 이용하지 못하게 만들어 주고



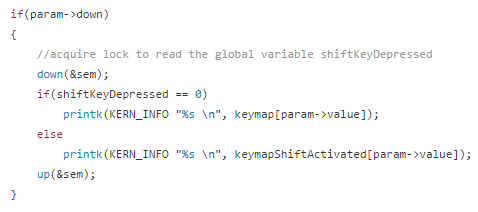
위에서 알 수 있듯 Param->down인 경우는 키를 누르고 있는 상태이고

위의 조건문의 param->value가 42, 54인 경우일 때는 shift가 눌린 상태이니

다시 말해 shift키가 눌려져 있는 상태이므로

Shiftkeydepressed 변수를 1로 만들어준다.

아닌 경우는 0으로 만들어주고 아래 코드를 진행한다.



위에서 shift 유무를 판별해서 변수에 저장했으니

이제 다른 키가 눌려져 있는지를 판별해야한다.

먼저 세마포어 down -> up 사이의 조건문에서 쉬프트 키 유무를 변수에 저장된

값을 통해 구분하고 눌려져 있지 않으면 keymap 문자열 배열에서

param->value에 해당하는 값에 해당하는 인덱스에 저장된 문자열을

KERN\_INFO(커널 로그 :  /var/log/kern.log ) 에 작성해준다.

Shift가 눌려져있으면 마찬가지로 shiftkeyactivated 문자열 배열에서 해당 부분을

찾아 커널 로그에 써준다.



이부분은 잘 모르겠는데 추측해보면 아마 키보드로부터 입력이 일어나게 되면

해당 정보를 커널에 갖다주게 되는데 이 커널에서 입력이 정상적으로

일어났다는 것을 확인하기 위해 에러가 발생하지 않았다고 리턴해주는

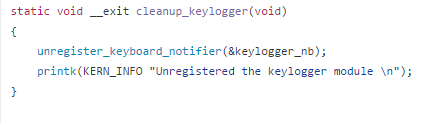
것이라고 생각된다. 따라서 이 값이 리턴되어야 정상적으로 키 입력이

처리되었다고 운영체제가 인식하지 않을까 라고 생각해본다.

<https://lab.nexedi.cn/kirr/linux/commit/b957e043ee557ca9b6bc451755ecd849b28852a4>

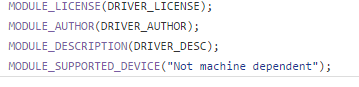


마지막으로 종료!!



Rmmod keylogger 를 입력해서 모듈을 커널에서 빼 주면 해당 과정 처리하고

커널에 메시지 써주는듯!!



요건 키로거 분석에서 별로 중요한건 아닌듯