[syscall_64.tbl]
(linux)/arch/x86/entry/syscalls로 이동

syscalls 디렉토리 안에 system call 함수들의 이름에 대한 심볼정보를 모아놓은 파일이 syscall_64.tbl 파일이다.
system call push와 pop의 number를 저장하기 위한 코드를 추가한다.

```
335  common  my_push  _x64_sys_my_push
336  common  my_pop   _x64_sys_my_pop
```

gedit을 이용해 syscall_64.tbl push와 pop의 고유번호 335와 336를 추가

[syscalls.h]
(linux)/include/linux로 이동

linux 디렉토리 안에 system call 함수들의 전체적인 기능을 간략한 형태로 정의한 파일이 syscalls.h 파일이다.

syscall은 assembly 코드로 작성되어있는 int 80 interrupt handler에서 호출된다.
assembly에서 C함수 호출이 가능하도록 asmkinkage를 함수 앞에 선언하여 syscalls.h에 함수 원형을 등록.

```
asmlinkage  void  sys_my_push(int);
asmlinkage  int  sys_my_pop(void);
```

[my_stack_syscall.c]

(linux)/kernel로 이동

kernel 디렉토리는 system call이 실제로 할 일이 구현되어 있다.
my_stack_syscall.c 파일을 생성, push와 pop함수를 구현.

SYSCALL_DEFINEx : 매크로를 이용하여 parameter의 개수에 따른 system call 구현.

parameter 0 - SYSCALL_DEFINE0 (syscall name)
parameter 1 - SYSCALL_DEFINE1 (syscall name, data_type 1, variable 1)
parameter 2 - SYSCALL_DEFINE2 (syscall name, data_type 1, variable 1, data_type 2, variable 2)
parameter n - SYSCALL_DEFINEn (syscall name, data_type 1, variable 1,...data_type n, variable n)

```
#include<linux/syscalls.h>
#include<linux/kernel.h>
#include<linux/linkage.h>

#define LENGTH 5                        // 원하는 길이를 상수로 표현
int stack[LENGTH];                      // int array 형태의 stack을 전역 variable 로 선언
int top = -1;                       // int 형의 stack top index를 전역 variable 로 선언



SYSCALL_DEFINE1(my_push,int,data){      // parameter 개수가 1개인 system call 구현을 위한 매크로
   if( top < LENGTH ){                  // stack top이 5를 넘지 않는 경우
      stack[++top] = data;              // array값이 늘어난 후, data값이 들어옴
   }
   int i;
```

```
    printk("[System call] my_push(): Push %d\n",data);
    printk("Stack Top----------\n");
    for(i = top; i>=0; i--){              // 현재 stack top에서 bottom까지
        printk("%d\n", stack[i]);          // stack의 data값을 출력
    }
    printk("Stack Bottom----------\n");
}

SYSCALL_DEFINE0(my_pop){                 // parameter 개수가 0개인 system call 구현을 위한 매크로

    if(top != -1){                       // stack이 비어 있지않다면
    int tmp = stack[top--];              // stack에 data값을 tmp에 옮긴 후, stack 한 칸 감소
    int i;
    printk("[System call] my_pop(): Pop %d\n",tmp);
    printk("Stack Top----------\n");
    for(i = top; i>=0; i--){             // 현재 stack의 top의 마지막부터 bottom까지
        printk("%d\n", stack[i]);          // stack의 data값을 출력
    }
    printk("Stack Bottom----------\n");

    return tmp;                          // pop의 prototype은 asmlinkage int sys_my_pop(void)이므로
    }                                    // pop된 data값을 반환
}
```

[Makefile]
(linux)/kernel로 이동

kernel 디렉토리 커널 컴파일 시 컴파일을 위한 조건과 타겟을 명시한 파일이 있는데 그것이 Makefile 파일이다.

MakeFile에 obj-y에 구현했던 my_stack_syscall의 빌드 파일을 위해 .o를 붙여준다.

my_stack_syscall.o

[커널 컴파일 및 재시동]
sudo make -j 4
sudo make install

두 명령어 실행 후, 아래 명령어를 통해 재시동.
sudo reboot

[사용자 응용 프로그램 작성]

이제 system call의 push와 pop이 정상적으로 추가 되었는지 확인하기 위해 응용프로그램을 작성.

```
#include<linux/unistd.h>

#define my_stack_push 335     // my_stack_push == 335
#define my_stack_pop 336      // my_stack_pop == 336

void push(int data){
   printf("Push: %d\n",data);            // push함수로 data값을 인자로 받아 출력
   syscall(my_stack_push,data);       // 커널의 system call 호출
}

void pop(){                  // pop함수로 stack data값을 출력, system call 호출
   printf("Pop: %d\n", syscall(my_stack_pop));
}

int main(void){

   push(1);                // 1 push
   push(2);                // 2 push
   push(3);                // 3 push
   push(4);                // 4 push
   push(5);                // 5 push

   pop();                  // pop
   pop();                  // pop
   pop();                  // pop
   pop();                  // pop
   pop();                  // pop

   return 0;               // 종료
}
```

아래 명령어를 이용하여 작성한 코드를 gcc로 컴파일
gcc call_my_stack.c –o call_my_stack

실행파일 ./call_my_stack을 터미널에서 실행.
dmesg 명령어로 my_stack_syscall.c의 printk로 출력 확인.

```
[   63.549646] [System call] oslab_push(): Push 1
[   63.549648] Stack Top-------------
[   63.549649] 1
[   63.549649] Stack Bottom-----------
[   63.549652] [System call] oslab_push(): Push 2
[   63.549652] Stack Top-------------
[   63.549653] 2
[   63.549653] 1
[   63.549654] Stack Bottom-----------
[   63.549656] [System call] oslab_push(): Push 3
[   63.549656] Stack Top-------------
[   63.549657] 3
[   63.549657] 2
[   63.549658] 1
[   63.549658] Stack Bottom-----------
[   63.549660] [System call] oslab_push(): Push 4
[   63.549661] Stack Top-------------
[   63.549661] 4
[   63.549661] 3
[   63.549662] 2
[   63.549662] 1
[   63.549663] Stack Bottom-----------
[   63.549665] [System call] oslab_push(): Push 5
[   63.549665] Stack Top-------------
[   63.549666] 5
[   63.549666] 4
[   63.549666] 3
[   63.549667] 2
[   63.549667] 1
[   63.549668] Stack Bottom-----------
[   63.549669] [System call] oslab_pop(): Pop 5
[   63.549669] Stack Top-------------
[   63.549669] 4
[   63.549670] 3
[   63.549670] 2
[   63.549671] 1
[   63.549671] Stack Bottom-----------
[   63.549673] [System call] oslab_pop(): Pop 4
[   63.549674] Stack Top-------------
[   63.549674] 3
[   63.549674] 2
[   63.549675] 1
[   63.549675] Stack Bottom-----------
[   63.549677] [System call] oslab_pop(): Pop 3
[   63.549678] Stack Top-------------
[   63.549678] 2
[   63.549679] 1
[   63.549679] Stack Bottom-----------
[   63.549681] [System call] oslab_pop(): Pop 2
[   63.549682] Stack Top-------------
[   63.549682] 1
[   63.549682] Stack Bottom-----------
[   63.549685] [System call] oslab_pop(): Pop 1
[   63.549685] Stack Top-------------
[   63.549685] Stack Bottom-----------
```