Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 장성환
redmk1025@gmail.com

```
Cscope 태그: read_cr0
   #   줄  파일 이름 / 콘텍스트 / 줄
   1     59  arch/x86/include/asm/paravirt.h <<read_cr0>>
             static inline unsigned long read_cr0(void )
   2    114  arch/x86/include/asm/paravirt_types.h <<read_cr0>>
             unsigned long (*read_cr0)(void );
   3    113  arch/x86/include/asm/special_insns.h <<read_cr0>>
             static inline unsigned long read_cr0(void )
   4    350  arch/x86/kernel/paravirt.c <<read_cr0>>
             .read_cr0 = native_read_cr0,
   5   1211  arch/x86/xen/enlighten.c <<read_cr0>>
             .read_cr0 = xen_read_cr0,
숫자 입력후 <엔터> (숫자없으면 취소):
```

3 번

```
static inline unsigned long read_cr0(void)
{
        return native_read_cr0();
}
```

read_cr0 함수

```
static inline unsigned long native_read_cr0(void)
{
        unsigned long val;
        // asm volatile(구동시킬 명령어 : 출력 : 입력 : 어셈블러 지시어)
        asm volatile("mov %%cr0,%0\n\t" : "=r" (val), "=m" (__force_order));
        return val;
}
```

native_read_cr0 함수

```c
/* Lookup the address for this symbol. Returns 0 if not found. */
unsigned long kallsyms_lookup_name(const char *name)
{
        char namebuf[KSYM_NAME_LEN];
        unsigned long i;
        unsigned int off;

        for (i = 0, off = 0; i < kallsyms_num_syms; i++) {
                off = kallsyms_expand_symbol(off, namebuf, ARRAY_SIZE(namebuf));

                if (strcmp(namebuf, name) == 0)
                        return kallsyms_addresses[i];
        }
        return module_kallsyms_lookup_name(name);
}
EXPORT_SYMBOL_GPL(kallsyms_lookup_name);
```

kallsyms_lookup_name() 함수

```
static unsigned int kallsyms_expand_symbol(unsigned int off,
                                           char *result, size_t maxlen)
{
        int len, skipped_first = 0;
        const u8 *tptr, *data;

        /* Get the compressed symbol length from the first symbol byte. */
        data = &kallsyms_names[off];
        len = *data;
        data++;

        /*
         * Update the offset to return the offset for the next symbol on
         * the compressed stream.
         */
        off += len + 1;

        /*
         * For every byte on the compressed symbol data, copy the table
         * entry for that byte.
         */
        while (len) {
                tptr = &kallsyms_token_table[kallsyms_token_index[*data]];
                data++;
                len--;

                while (*tptr) {
                        if (skipped_first) {
                                if (maxlen <= 1)
                                        goto tail;
                                *result = *tptr;
                                result++;
                                maxlen--;
                        } else
                                skipped_first = 1;
                        tptr++;
                }
        }

tail:
        if (maxlen)
                *result = '\0';

        /* Return to offset to the next symbol. */
        return off;
}
```

kallsyms_expand_symbol() 함수

/* 압축되지 않은 문자열로 압축된 기호 데이터를 확장, 압축되지 않은 문자열이 너무 길면 (>=maxlen) 잘린다, 심볼이 압축된 스트림에 있는 위치에 대한 오프셋이 제공

된다.

/* Get the compressed symbol length from the first symbol byte. */
```
    data = &kallsyms_names[off]; //초반 off 값은 무조건 0 즉 칼심네임 배열의 시작주소를 저장한다.
    len = *data; //char 형 데이터의 값을 int 형 len 에 저장한다.
    data++; // data = data+1
```
* 첫번째 기호 바이트에서 압축 된 기호 길이를 가져온다.

```
/*
 * Expand a compressed symbol data into the resulting uncompressed string,
 * if uncompressed string is too long (>= maxlen), it will be truncated,
 * given the offset to where the symbol is in the compressed stream.
 */

// 압축되지 않은 문자열로 압축된 기호 데이터를 확장, 압축해제된 문자열이 너무 길면 잘린다.
// 심볼이 압축된 스트림에 있는 위치에 대한 오프셋이 제공된다.

static unsigned int kallsyms_expand_symbol(unsigned int off,
                                           char *result, size_t maxlen)
{ //인자로 off =0, char *result = 128크기의 char형 배열, result 배열의 길이를 전달.

        int len, skipped_first = 0;
        const u8 *tptr, *data; //8bit data 즉, 1바이트 크기의 포인터를 선언

        /* Get the compressed symbol length from the first symbol byte. */
        // 첫번째 심볼 바이트로부터 압축된 심볼의 길이를 얻는다.

        data = &kallsyms_names[off]; // 초반 off값은 무조건 0 즉, kallsyms_name 배열의 시작주소를 저>
장
        len = *data; // kallsyms_name[0]의 값을 int형 len에 저장
        data++; // kallsyms_name[1]로 변함

        /*
         * Update the offset to return the offset for the next symbol on
         * the compressed stream.
         */
        //오프셋을 업데이트 하여 압축 된 스트림의 다음 심볼에 대한 오프셋을 반환.

        off += len + 1; //off에 kallsyms_name[0]의 값 +1을 넣어준다.

        /*
         * For every byte on the compressed symbol data, copy the table
         * entry for that byte.
         */
        // 압축된 심볼 데이터의 모든 바이트에 대하여 해당 바이트에 대한 테이블 항목을 복사.

        while (len) {
                tptr = &kallsyms_token_table[kallsyms_token_index[*data]];
                data++;
                len--;

                while (*tptr) {
                        if (skipped_first) {
                                if (maxlen <= 1)
                                        goto tail;
                                *result = *tptr;
                                result++;
                                maxlen--;
                        } else
                                skipped_first = 1;
                        tptr++;
                }
        }

tail:
        if (maxlen)
                                                                    85,0-1        14%
```