TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-04-20 (42 회차)

강사: Innova Lee(이상훈)
gcccompil3r@gmail.com
학생: 정유경
ucong@naver.com

1. 시스템 콜 fork() 개요

시스템 호출 이 발생하면 인텔프로세서 내부에서 인터럽트 벡터번호 0x80(128)을 이용하여 system call 루틴을 호출하고 커널로 진입한다.

시스템 호출 fork()
사용자 프로그램에서 시스템콜인 fork()함수를 호출하면 라이브러리(libc.a)의 fork()함수 내부에 mov 2 %eax , int 0x80 을 실행한다
즉, fork()함수번호 2 번을 eax 레지스터에 저장, IDT 의 엔트리번호 0x80(128)트랩게이트를 호출
fork 함수번호는 sys_call_table 의 엔트리 2 번을 의미
0x80 트랩게이트의 서비스루틴(system_call)에서는 sys_call_table 의 엔트리 2 에 정의된 sys_fork 함수를 호출
커널에서 제공하는 모든 시스템호출 함수는 sys/_call_table 에 정의되어 잇다
함수명은 모두 “sys_”

2. 커널 분석

```
#ifdef __ARCH_WANT_SYS_FORK
SYSCALL_DEFINE0(fork)
{
#ifdef CONFIG_MMU // MMU-based Paged Memory Management Support 가 정의되어 있는가
	return _do_fork(SIGCHLD, 0, 0, NULL, NULL, 0); //  SIGCHLD 는 17 로 정의(x86)
#else
	/* can not support in nommu mode */
	return -EINVAL;
#endif
}
#endif
```

sys_fork() 함수는 do_fork()함수를 호출한다.

```
/*
 *  Ok, this is the main fork-routine.
 *
 * It copies the process, and if successful kick-starts
 * it and waits for it to finish using the VM if required.
 */
long _do_fork(unsigned long clone_flags,
	      unsigned long stack_start,
	      unsigned long stack_size,
	      int __user *parent_tidptr,
	      int __user *child_tidptr,
	      unsigned long tls)
{
	struct task_struct *p;
	int trace = 0;
	long nr;
```

```
// Determine whether and which event to report to ptracer
 if (!(clone_flags & CLONE_UNTRACED)) { // !(17 & 0x00800000 24 번째 비트가 1 인가) 들어감
            if (clone_flags & CLONE_VFORK) // 17 &  0x00004000 (15 번째 비트가 1 인가 )패스
                  trace = PTRACE_EVENT_VFORK;
            else if ((clone_flags & CSIGNAL) != SIGCHLD) // 17 & 0x000000ff = 17 이므로 패스
                  trace = PTRACE_EVENT_CLONE;
            else
                  trace = PTRACE_EVENT_FORK; // trace 는 1

            if (likely(!ptrace_event_enabled(current, trace))) // ptrace 는 디버깅에 사용 지금은 안씀
                  trace = 0;
      }

      p = copy_process(clone_flags, stack_start, stack_size, // 들어가면
                  child_tidptr, NULL, trace, tls);
```

```
// This creates a new process as a copy of the old one,
static struct task_struct *copy_process(unsigned long clone_flags,
                                        unsigned long stack_start,
                                        unsigned long stack_size,
                                        int __user *child_tidptr,
                                        struct pid *pid,
                                        int trace,
                                        unsigned long tls)
{
       int retval;
      struct task_struct *p;
      void *cgrp_ss_priv[CGROUP_CANFORK_COUNT] = {};

      /* if 문 모두 패스함 하나도 안걸림*/

      retval = security_task_create(clone_flags); // 들어가면
// 들어가면 2 가지 있다. 하나는 0 을 반환하고 하나는 아래와 같다. 둘중 어떤거?
```

2, call_int_hook →list_for_each_entry → 3, list_first_entry → 3, list_entry → 13, container_of

```
// container_of - cast a member of a structure out to the containing structure
#define container_of(ptr, type, member) ({                      \
      const typeof( ((type *)0)->member ) *__mptr = (ptr);    \
      (type *)( (char *)__mptr - offsetof(type,member) );})
```

파고들어왔지만…?? retval 에 반환되는 값이 뭔지 잘 모르겠습니다.

retval 에 값을 리턴받았다면, 계속 진행.

```
       if (retval)
```

```
        goto fork_out;

    retval = -ENOMEM; // retval = -12
    p = dup_task_struct(current); // 들어가면
```

```
static struct task_struct *dup_task_struct(struct task_struct *orig) // orig : 현재수행중 태스크의 포인터
{
    struct task_struct *tsk;
    struct thread_info *ti;
    int node = tsk_fork_get_node(orig); // 들어가면
```

```
/* called from do_fork() to get node information for about to be created task */
int tsk_fork_get_node(struct task_struct *tsk)
{
#ifdef CONFIG_NUMA
    if (tsk == kthreadd_task) // *kthreadd_task 가 현재 수행중인 태스크이면
        return tsk→pref_node_fork; // tsk 구조체의 pref_node_fork 멤버를 리턴
#endif
    return NUMA_NO_NODE;
}
```

```
리턴한 값을 node 에 저장하고 계속 진행합니다
    int err;
    tsk = alloc_task_struct_node(node); // 들어가면
```

2, alloc_task_struct_node→ 3, kmem_cache_alloc_node → ?, slab_alloc_node

- static struct kmem_cache *task_struct_**cachep**;
: kmem_cache 는 슬랩 할당자(5 번)이고, cachep 는 슬랩할당자 포인터
- GFP_KERNEL : 메모리 할당 성공을 요구, 메모리가 충분하지 않을 경우는 호출한 프로세스를 멈추고 동적 메모리 할당할 수 있는 상태가 될때까지 대기(계속 시도)

```
/* Allocate an object on the specified node
 * @cachep: The cache to allocate from.
 * @flags: See kmalloc().
 * @nodeid: node number of the target node. */

void *kmem_cache_alloc_node(struct kmem_cache *cachep, gfp_t flags, int nodeid)
{
    void *ret = slab_alloc_node(cachep, flags, nodeid, _RET_IP_);
// 1,2 번중 어디로 들어가야할지 모르겠습니다??
// trace 는 디버깅, 볼필요 없다
```

```
        return ret;
}
```

슬랩할당자로 새로운 태스크에 메모리를 할당해준다

```
tsk 에 리턴받은 후 나머지를 계속 진행.
if (!tsk)
                return NULL;
        ti = alloc_thread_info_node(tsk, node); // 들어가면
```

3, alloc_thread_info_node → alloc_pages_node → 2 , __alloc_pages → __alloc_pages_nodemask → get_page_from_freelist

```
// Allocate pages if THREAD_SIZE is >= PAGE_SIZE
# if THREAD_SIZE >= PAGE_SIZE // 16K >= 4K
// #define THREAD_SIZE            (1 << THREAD_SHIFT) , 1<<14 이므로 16K
static struct thread_info *alloc_thread_info_node(struct task_struct *tsk,
int node)
{
        struct page *page = alloc_kmem_pages_node(node, THREADINFO_GFP,
                                        THREAD_SIZE_ORDER); // 들어가면
```

```
/*
 * This is the 'heart' of the zoned buddy allocator.
 */
struct page *
__alloc_pages_nodemask(gfp_t gfp_mask, unsigned int order,
                        struct zonelist *zonelist, nodemask_t *nodemask)
{
        …
        page = get_page_from_freelist(alloc_mask, order, alloc_flags, &ac);
// 버디알고리즘에 의해 16KB 메모리를 할당받는다. // 더 들어가지 않고 진행
        …
        return page; // 가상주소 리턴
}
```

정리하면, tsk 는 Slab 을 통해 할당받은 것이고

ti 는 Buddy 를 통해 할당받은 것이다.

```
 ti 에 값을 반환받고 계속 진행한다.
    if (!ti)
                goto free_tsk;
        err = arch_dup_task_struct(tsk, orig); // 들어가면
```

```
// copy the current task into the new thread.
int arch_dup_task_struct(struct task_struct *dst, struct task_struct *src)
```

```
{
	memcpy(dst, src, arch_task_struct_size); //  새로만든 task_struct 인 tsk 에 orig 를 복사한다
#ifdef CONFIG_VM86
	dst->thread.vm86 = NULL;
#endif
	return fpu__copy(&dst->thread.fpu, &src→thread.fpu); //  fpu: 부동소수점 처리용 cpu
}
```

```
err 에 값을 반환받고 계속 진행
	if (err)
		goto free_ti;

	tsk->stack = ti; // thread_info 를 자식 프로세스의 stack 에 저장?

	setup_thread_stack(tsk, orig); // 들어가면
```

```
static inline void setup_thread_stack(struct task_struct *p, struct task_struct *org)
{
	*task_thread_info(p) = *task_thread_info(org); // 부모의 thread_info 를 자식의 thread_info
	task_thread_info(p)->task = p; // thread_info 의 task 멤버를 자식프로세스로 지정
}
```

다시 돌아가서

```
	clear_user_return_notifier(tsk); // 무슨일을 하는지 잘 모르겠습니다…?!
	clear_tsk_need_resched(tsk);
	set_task_stack_end_magic(tsk);

#ifdef CONFIG_CC_STACKPROTECTOR
	tsk->stack_canary = get_random_int(); // 스택 보호…?!
#endif

// One for us, one for whoever does the "release_task()" (usually parent)
	atomic_set(&tsk->usage, 2);
#ifdef CONFIG_BLK_DEV_IO_TRACE
	tsk->btrace_seq = 0;
#endif
	tsk->splice_pipe = NULL; // …??
	tsk->task_frag.page = NULL;
	tsk->wake_q.next = NULL;

	account_kernel_stack(ti, 1);

	return tsk;

free_ti: // 동적할당을 해체한다.
```

```
        free_thread_info(ti);
free_tsk:
        free_task_struct(tsk);
        return NULL;
}
```

다시 돌아가서

```
        if (!p)
                goto fork_out;
         retval = copy_creds(p, clone_flags);  //  copy_creds: 프로세스의 보안관련 설정 을 복사한다
```

copy_creds 로 들어가보면

```
…??
```

```
     if (retval < 0)
                goto bad_fork_free;
/*VIRT 가상화 관련 코드들*/

        delayacct_tsk_init(p);  /* Must remain after dup_task_struct() */ ..????

        INIT_LIST_HEAD( &p→children); //자식있으면 리스트 초기화
        INIT_LIST_HEAD(&p→sibling); // siblings 있으면 리스트 초기화
        rcu_copy_process(p); //  rcu 있으면 setting

        init_sigpending(&p→pending); // 시그널 지연, 초기화
      // → sigemptyset()

#ifdef CONFIG_NUMA // NUMA 이면 mempolicy 복사
        p->mempolicy = mpol_dup(p→mempolicy);


/*이후로 많은 설정들이 있는 듯 하다?!*/
        /* Perform scheduler related setup. Assign this task to a CPU. */
        retval = sched_fork(clone_flags, p); // 실제 cpu 에서 구동되도록 스케쥴러에 배치함
      // 들어가보면
      // → dl_prio(p→prio),rt_prio(p→prio) , fair_sched 가 보인다


  /* copy all the process information */
        retval = copy_files(clone_flags, p);
        retval = copy_sighand(clone_flags, p);
        retval = copy_signal(clone_flags, p); // 자식도 시그널 공유
        retval = copy_mm(clone_flags, p); // task_struct, mm_struct
        retval = copy_thread_tls(clone_flags, stack_start, stack_size, p, tls);     // 스레드 지역변수가
```

위치

```
        pid = alloc_pid(p→nsproxy→pid_ns_for_children); // 자식 프로세스에 PID 할당
    /*sigaltstack should be cleared when sharing the same VM*/
    /* ok, now we should be set up.. */
```

여기까지 p=copy_process()가 끝났다.