# TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-04-23 (43 회차)

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

**fork 커널 분석**

**[1] 후반부: Copy process 이후 부분이다.**

```
	/*
	 * Do this prior waking up the new thread - the thread pointer
	 * might get invalid after that point, if the thread exits quickly.
	 */

	if (!IS_ERR(p)) {
		struct completion vfork;
		struct pid *pid;

		trace_sched_process_fork(current, p); //?

		pid = get_task_pid(p, PIDTYPE_PID);
/*task pid 를 가져온다(이때 pid 는 tgid!) */
		nr = pid_vnr(pid);
/* 들어가보면 2 가지 기능을 하는 것 같다
1. pid_vnr() : virtual PID 번호를 반환
2. pid_nr_ns() : 지정된 ns(namespace)에 속한 PID 번호를 반환*/

		if (clone_flags & CLONE_PARENT_SETTID)
			put_user(nr, parent_tidptr);

		if (clone_flags & CLONE_VFORK) {
			p->vfork_done = &vfork;
			init_completion(&vfork);
			get_task_struct(p);
		}
/*조건문은 둘다 들어가지 않는다
CLONE_PARENT_SETTID 0x00100000
CLONE_VFORK 0x00004000
*/


		wake_up_new_task(p);
/*프로세스가 새로 생성되었다면 프로세스를 깨워서 실행시켜야 한다.
함수 내부의 주요 부분은 다음과 같다.
1. rq = __task_rq_lock(p); //  해당 task_struct 에 대한 Runqueue 를 가져온다
2. activate_task(rq, p, 0); // Runqueue 에 해당 프로세스(struct task_struct *p)를 Enqueue 한다.


		/* forking complete and child started to run, tell ptracer */
		if (unlikely(trace))
			ptrace_event_pid(trace, pid);

		if (clone_flags & CLONE_VFORK) {
			if (!wait_for_vfork_done(p, &vfork)) /*들어가지 않는다*/
				ptrace_event_pid(PTRACE_EVENT_VFORK_DONE, pid);
		}


		put_pid(pid);
	} else {
		nr = PTR_ERR(p);
	}
```

```
        return nr;
}
```

[2] 전반부

- sys_fork() 함수는 do_fork()함수를 호출한다.

```
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;
 /* 리눅스의 fork()는 clone() 시스템콜을 이용하여 구현된다. 이 함수는 또한 어떤 자원을 부모와 자식 프로세스가 공휴
할 것인가를 지정하는 여러 플래그들을 사용한다
- 라이브러리함수인 fork(), vfork(), __clone()은 적절한 플래그를 사용해서, clone()를 호출하고, clone() 시스템콜은 다
시 do_fork()함수를 호출한다.
프로세스 생성의 대부분의 작업은 do_fork()에서 처리되는데, 이것은 kernel/fork.c 에 정의되어 있다. 이 함수는
copy_process()를 호출한다음, 프로세스를 시작시킨다.  */
 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); // 자식 프로세스 권한검사
         if (retval) // 권한 오류일경우 forkout
```

```
        goto fork_out;

    retval = -ENOMEM; // retval = -12 (ENOMEM: 메모리가 부족합니다. 예를 들어, 자식 프로세스를 실행하기에 메모리가 부족함)
    p = dup_task_struct(current);
/*dup_task_struct()를 호출하여 새 커널 스택과 thread_info 구조체, 그리고 새 프로세스를 위한, 현재 태스크와 동일한 값들을 갖는 task-struct 구조체를 생성한다. 이 시점에서 자식과 부모의 프로세서 서술자는 완전히 동일하다.*/
```

```
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
```

```
/* 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 진행한다
    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)

```
void *kmem_cache_alloc_node(struct kmem_cache *cachep, gfp_t flags, int nodeid)
{
    void *ret = slab_alloc_node(cachep, flags, nodeid, _RET_IP_);
// 슬랩할당자로 새로운 태스크에 메모리를 할당해준다
    return ret;
}
```

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

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
```

```
__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.
      …
      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
// fpu 정보 복사
}
```

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

      tsk->stack = ti; // thread_info 를 task_struct 와 연결

      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

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

```
      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.. */
```

p=copy_process()는 여기까지