TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
43일차

sys_fork() 분석

```
#ifdef __ARCH_WANT_SYS_FORK
SYSCALL_DEFINE0(fork)
{
#ifdef CONFIG_MMU
    return _do_fork(SIGCHLD, 0, 0, NULL, NULL, 0);
#else
    /* can not support in nommu mode */
    return -EINVAL;
#endif
}
#endif
```

SYSCALL_DEFINE0(fork)를 통해서 do_fork()로 들어간다. 아래와 같은 코드로 가게 되며 인자는 위와 마찬가지로 (SIGCHLD, 0, 0, NULL., NULL, 0)을 가지고 들어오게 되었다.

```
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.  When
     * called from kernel_thread or CLONE_UNTRACED is explicitly
     * requested, no event is reported; otherwise, report if the event
     * for the type of forking is enabled.
     */
    if (!(clone_flags & CLONE_UNTRACED)) {
        if (clone_flags & CLONE_VFORK)
            trace = PTRACE_EVENT_VFORK;
        else if ((clone_flags & CSIGNAL) != SIGCHLD)
            trace = PTRACE_EVENT_CLONE;
        else
            trace = PTRACE_EVENT_FORK;

        if (likely(!ptrace_event_enabled(current, trace)))
            trace = 0;
    }

    p = copy_process(clone_flags, stack_start, stack_size,
             child_tidptr, NULL, trace, tls);
    /*
     * 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);
        nr = pid_vnr(pid);

        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);
        }

        wake_up_new_task(p);

        /* 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))
                ptrace_event_pid(PTRACE_EVENT_VFORK_DONE, pid);
        }

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

여기서

```
    if (!(clone_flags & CLONE_UNTRACED)) {
        if (clone_flags & CLONE_VFORK)
            trace = PTRACE_EVENT_VFORK;
        else if ((clone_flags & CSIGNAL) != SIGCHLD)
            trace = PTRACE_EVENT_CLONE;
        else
            trace = PTRACE_EVENT_FORK;

        if (likely(!ptrace_event_enabled(current, trace)))
            trace = 0;
    }
```

이 부분은 fork(), vfork(), clone(), pthread_create() 중 어떤 함수를 썼는가 구분해주는 부분이다. system call로 커널로 제어권이 넘어갔을 때, 결국 구동시켜주는 함수는 do_fork()이기 때문이다.

```
14      61  arch/sparc/include/uapi/asm/signal.h <<SIGCHLD>>
            #define SIGCHLD 20
15      39  arch/x86/include/uapi/asm/signal.h <<SIGCHLD>>
            #define SIGCHLD 17
16      51  arch/xtensa/include/uapi/asm/signal.h <<SIGCHLD>>
            #define SIGCHLD 17
17      27  include/uapi/asm-generic/signal.h <<SIGCHLD>>
            #define SIGCHLD 17
```

수식의 각각의 값을 알아야 한다. 우선 clone flags를 보자면, do_fork로 들어올 때, 인자로 가져온 값이 SIGCHLD로 17을 가진다. 이런 식으로 값을 확인하여

```
        /*
         * Determine whether and which event to report to ptracer.  When
         * called from kernel_thread or CLONE_UNTRACED is explicitly
         * requested, no event is reported; otherwise, report if the event
         * for the type of forking is enabled.
         */
         /*CLONE_UNTRACED = 0x00800000 */
        if (!(clone_flags & CLONE_UNTRACED)) {
                /*CLONE_VFORK = 0x00004000 */
                if (clone_flags & CLONE_VFORK)
                        trace = PTRACE_EVENT_VFORK;  //PTRACE_EVENT_VFORK = 2
                /* CSIGNAL = 0x000000ff
                clone_flags로 들어온 것에 SIGCHLD가 있는지 없는지에 따라
                trace가 PTRACE_EVENT_FORK = 1 혹은 PTRACE_EVENT_CLONE = 3이 됨*/
                else if ((clone_flags & CSIGNAL) != SIGCHLD)
                        trace = PTRACE_EVENT_CLONE; // = 3
                else
                        trace = PTRACE_EVENT_FORK; // = 1

                if (likely(!ptrace_event_enabled(current, trace)))
                        trace = 0;
        }
```

로 채워지며 fork를 사용했을 때는 if와 else if가 0이 되어 통과 되기에 trace는 1를 갖는 것을 알 수 있다. ptrace는 디버깅 상태를 확인하기 위한 것이다. current는 현재 구동중인 태스크를 뜻하고 fork를 사용한 경우 trace는 1을 인자로 갖는다. ptrace 는 유닉스 계열 운영체제에서의 시스템 콜이다. process trace 의 약자로 컨트롤러가 대상의 내부 상태를 조사하고 조작하게 함으로써, 한 프로세스가 다른 프로세스를 제어할 수 있다. 디버거와 다른 코드 분석, 특히 소프르웨어 개발을 도와주는 도구들에서 사용된다. Ptrace_event_enabled 는 ptrace event 가 가능한지 확인하는 것이다. 가능하면 true, 불가능하면 false 리턴한다.

```
/**
 * ptrace_event_enabled - test whether a ptrace event is enabled
 * @task: ptracee of interest
 * @event: %PTRACE_EVENT_* to test
 *
 * Test whether @event is enabled for ptracee @task.
 *
 * Returns %true if @event is enabled, %false otherwise.
 */
static inline bool ptrace_event_enabled(struct task_struct *task, int event)
{
        return task->ptrace & PT_EVENT_FLAG(event);
}


#define PT_OPT_FLAG_SHIFT       3
/* PT_TRACE_* event enable flags */
#define PT_EVENT_FLAG(event)    (1 << (PT_OPT_FLAG_SHIFT + (event)))
#define PT_TRACESYSGOOD         PT_EVENT_FLAG(0)
```

p = copy_process(clone_flags, stack_start, stack_size, child_tidptr, NULL, trace, tls);는 새로운 프로세스를 생성하고 이전 것을 복사한다. 하지만 아직 실제로 시작되지는 않고 레지스터와 프로세스의 환경을 복사한다.

```
/* copy_process(clone_flags, stack_start, stack_size,
                         child_tidptr, NULL, trace, tls)*/
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] = {};
        /* CLONE_NEWNS = 0x00020000
           CLONE_FS = 0x00000200  */
        if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
                return ERR_PTR(-EINVAL);
        /*CLONE_NEWUSER = 0x10000000*/
        if ((clone_flags & (CLONE_NEWUSER|CLONE_FS)) == (CLONE_NEWUSER|CLONE_FS))
                return ERR_PTR(-EINVAL);

        /*
         * Thread groups must share signals as well, and detached threads
         * can only be started up within the thread group.
         */

        /* CLONE_THREAD = 0x00010000
           CLONE_SIGHAND = 0x00000800*/
        if ((clone_flags & CLONE_THREAD) && !(clone_flags & CLONE_SIGHAND))
                return ERR_PTR(-EINVAL);

        /*
         * Shared signal handlers imply shared VM. By way of the above,
         * thread groups also imply shared VM. Blocking this case allows
         * for various simplifications in other code.
         */

        /* CLONE_VM = 0x00000100 */
        if ((clone_flags & CLONE_SIGHAND) && !(clone_flags & CLONE_VM))
                return ERR_PTR(-EINVAL);

        /*
         * Siblings of global init remain as zombies on exit since they are
         * not reaped by their parent (swapper). To solve this and to avoid
         * multi-rooted process trees, prevent global and container-inits
         * from creating siblings.
         */

        /* CLONE_PARENT = 0x00008000
           SIGNAL_UNKILLABLE = 0x00000040*/
        if ((clone_flags & CLONE_PARENT) &&
                                current->signal->flags & SIGNAL_UNKILLABLE)
                return ERR_PTR(-EINVAL);

        /*
         * If the new process will be in a different pid or user namespace
         * do not allow it to share a thread group with the forking task.
         */
                                                              1243,0-1      59%
```

```
        /*
         * If the new process will be in a different pid or user namespace
         * do not allow it to share a thread group with the forking task.
         */

        /* CLONE_NEWPID = 0x20000000 */
        if (clone_flags & CLONE_THREAD) {
                if ((clone_flags & (CLONE_NEWUSER | CLONE_NEWPID)) ||
                    (task_active_pid_ns(current) !=
                                current->nsproxy->pid_ns_for_children))
                        return ERR_PTR(-EINVAL);
        }

        retval = security_task_create(clone_flags);
        if (retval)
                goto fork_out;

        retval = -ENOMEM;
        p = dup_task_struct(current);
        if (!p)
                goto fork_out;

        ftrace_graph_init_task(p);

        rt_mutex_init_task(p);
```

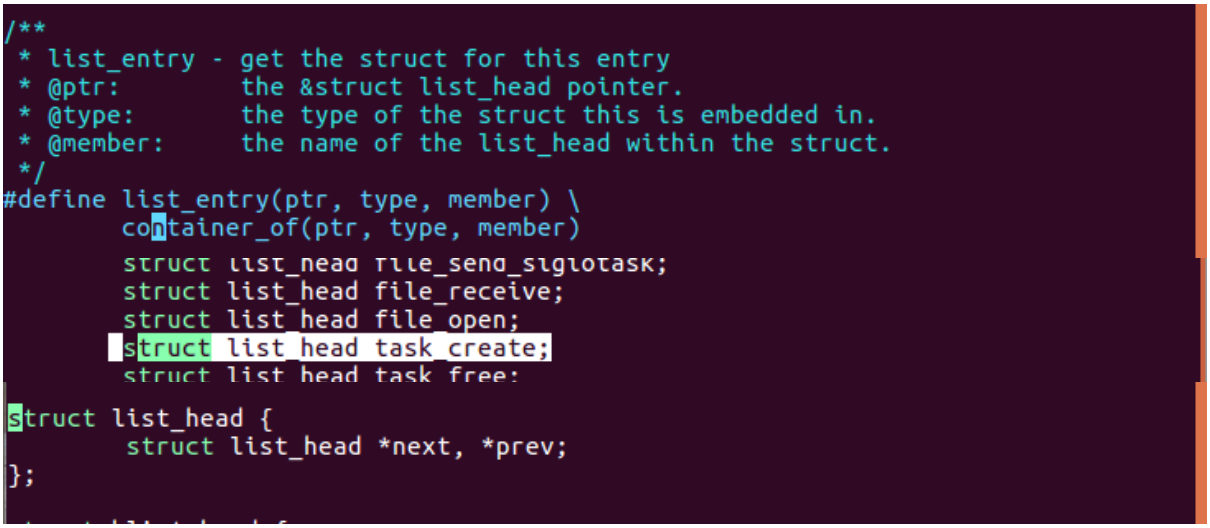
여기서 clone_flags = 17 = 0x00000011 이므로 if문은 0이 되어 통과한다. Retval 값을 구하기 위해선 security_task_create를 봐야한다.

```
int security_task_create(unsigned long clone_flags)
{
        return call_int_hook(task_create, 0, clone_flags);
}

#define call_int_hook(FUNC, IRC, ...) ({                        \
        int RC = IRC;                                           \
        do {                                                    \
                struct security_hook_list *P;                   \
                                                                \
                list_for_each_entry(P, &security_hook_heads.FUNC, list) { \
                        RC = P->hook.FUNC(__VA_ARGS__);         \
                        if (RC != 0)                            \
                                break;                          \
                }                                               \
        } while (0);                                            \
        RC;                                                     \
})
```

이 부분부터 해석이 잘 안 됩니다..

```
#define list_for_each_entry(pos, head, member)                          \
        for (pos = list_first_entry(head, typeof(*pos), member);        \
             &pos->member != (head);                                    \
             pos = list_next_entry(pos, member))
```



으로 연결리스트를 사용한다.