TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/4/20
수업일수	42 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

1. sys_fork()분석

1. sys_fork()분석

vi -t SYSCALL_DEFINE0(fork)로 sys_fork 찾음

```
#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
}
```

_do_fork(SIGCHLD, 0, 0 ,NULL, NULL, 0) 내부 (1) - 크기가 커서 잘랐음

인자인 SIGCHLD = 17 로 정의되어 있음

```
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;
        }
```

/*
clone_flags = 17
stack_start = 0
stack_size = 0
parent_tidptr = NULL
child_tidptr = NULL
tls =0

CLONE_UNTRACED 0x00800000
CLONE_VFORK 0x00004000
CSIGNAL 0x000000ff
PTRACE_EVENT_VFORK 2
PTRACE_EVENT_CLONE 3
PTRACE_EVENT_FORK 1
*/

if(!(clone_flags & CLONE_UNTRACED)를 만족하고 else 조건을 만족하여 trace = 1 이 된다.

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 - 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);
}
```

Ptrace_event_enabled 는 ptrace event 가 가능한지 확인함.
첫 번째 인자는 현재 실행되는 태스크,
두 번째 인자는 trace = event = 1

가능하다면 true, 불가능하다면 false 리턴.

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

PT_EVENT_FLAG(event) = 1<<(3+1) = 10000
likely(!ptrace_event_enabled(current, trace))이므로 ptrace_event_enabled(current, trace)가 0 일 것임. 따라서 trace = 0

_do_fork(SIGCHLD, 0, 0 ,NULL, NULL, 0) 내부 (2)

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

인자 (0, 0, 0, NULL, NULL, 0, 0)

copy_process 내부 (1)

```
/*
 * This creates a new process as a copy of the old one,
 * but does not actually start it yet.
 *
 * It copies the registers, and all the appropriate
 * parts of the process environment (as per the clone
 * flags). The actual kick-off is left to the caller.
 */
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)
{
```

새로운 프로세스를 생성하고 이전 것을 복사한다. 하지만 아직 실제로 시작되지는 않음.
레지스터와 프로세스의 환경을 복사.

copy_process 내부 (2)

```
{
        int retval;
        struct task_struct *p;
        void *cgrp_ss_priv[CGROUP_CANFORK_COUNT] = {};

        if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
                return ERR_PTR(-EINVAL);

        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.
         */
        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.
         */
        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.
         */
        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.
          */
         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);
         }
```

clone_flags = 17 = 0x11

CLONE_FS 0x00000200

CLONE_NEWNS 0x00020000

CLONE_NEWUSER 0x10000000

CLONE_NEWPID 0x20000000

clone_flags = SIGCHLD = 17

CLONE_THREAD 0x00010000

CLONE_SIGHAND 0x00000800

CLONE_VM 0x00000100

*쓰레드 그룹은 시그널을 공유해야 분리된 쓰레드는 쓰레드 그룹 내에서만 동작 가능하다.
*시그널 핸들러를 공유 했다는 것은 가상 메모리를 공유했다는 것과 같다. 위의 방법으로 쓰레드 그룹 또한 가상메모리를 공유했다는 것을 암시한다. 이 경우를 차단하면 다른 코드에서 다양한 단순화가 가능하다.

*만약 새로운 프로세스가 다른 pid 나 유저네임을 허용하지 않으면 쓰레드 그룹을 포크작업과 공유할 수 없다.

if 문들 만족하지 않고 넘어간다.

copy_process 내부 (3

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

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

retval = 0
if 문 안들어감
retval = -ENOMEM = -12

```
static inline int security_task_create(unsigned long clone_flags)
{
        return 0;
}
```

dup_task_struct 내부(1)

```
static struct task_struct *dup_task_struct(struct task_struct *orig)
{
         struct task_struct *tsk;
         struct thread_info *ti;
         int node = tsk_fork_get_node(orig);
         int err;
```

현재의 task_struct 를 인자로 가짐. fork 하기 이전의 부모 태스크로 생각하고 있다. = orig

tsk_fork_get_node 내부(1)

```
/* 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)
                return tsk->pref_node_fork;
#endif
        return NUMA_NO_NODE;
}
```

부모 태스크를 인자로 가져오고 x86cpu 는 NUMA 구조 이므로 kthread_task 를 찾을 수 없지만 tsk->pref_node_fork 를 리턴 할 것이다.

dup_task_struct 내부(2)

```
         tsk = alloc_task_struct_node(node);
```

alloc_task_struct_node 내부

```
#ifndef CONFIG_ARCH_TASK_STRUCT_ALLOCATOR
static struct kmem_cache *task_struct_cachep;

static inline struct task_struct *alloc_task_struct_node(int node)
{
        return kmem_cache_alloc_node(task_struct_cachep, GFP_KERNEL, node);
}
```

kmem_cache 는 slab 할당자와 관련되어 있음

kmem_cache_alloc_node (task_struct_cachep, 0, node)

kmem_cache_alloc_node 내부

```
static __always_inline void *kmem_cache_alloc_node(struct kmem_cache *s, gfp_t flags, int node)
{
        return kmem_cache_alloc(s, flags);
}
```

kmem_cache_alloc 내부

```
/**
 * kmem_cache_alloc - Allocate an object
 * @cachep: The cache to allocate from.
 * @flags: See kmalloc().
 *
 * Allocate an object from this cache.  The flags are only relevant
 * if the cache has no available objects.
 */
void *kmem_cache_alloc(struct kmem_cache *cachep, gfp_t flags)
{
        void *ret = slab_alloc(cachep, flags, _RET_IP_);

        trace_kmem_cache_alloc(_RET_IP_, ret,
                               cachep->object_size, cachep->size, flags);

        return ret;
}
```

(cachep ,0)

slab_alloc 내부(1)

```
#endif /* CONFIG_NUMA */

static __always_inline void *
slab_alloc(struct kmem_cache *cachep, gfp_t flags, unsigned long caller)
{
        unsigned long save_flags;
        void *objp;

        flags &= gfp_allowed_mask;

        lockdep_trace_alloc(flags);
```

flags = 0

```
# define lockdep_trace_alloc(g)                 do { } while (0)
```

lockdep_trace_alloc 는 아무런 동작을 하지 않음

```
void lockdep_trace_alloc(gfp_t gfp_mask)
{
        unsigned long flags;

        if (unlikely(current->lockdep_recursion))
                return;

        raw_local_irq_save(flags);
        check_flags(flags);
        current->lockdep_recursion = 1;
        __lockdep_trace_alloc(gfp_mask, flags);
        current->lockdep_recursion = 0;
        raw_local_irq_restore(flags);
}
```

위의 lockdep_trace_alloc 인지 아래 lockdep_trace_alloc 인지 잘 모르겠다 ㅠㅠ

unlikely 이므로 특수한 경우가 아닌 이상 if 문을 만족하지 않을 것이다.

raw_local_irq_save

```
#define raw_local_irq_save(flags)                       \
        do {                                            \
                typecheck(unsigned long, flags);        \
                flags = arch_local_irq_save();          \
        } while (0)
```

long 타입인지 체크하고 flags 에 arch_local_irq_save 를 동작

```
#define raw_local_irq_save(flags) ((flags) = 0)
```

```
static inline unsigned long native_save_fl(void)
{
        unsigned long flags;

        /*
         * "=rm" is safe here, because "pop" adjusts the stack before
         * it evaluates its effective address -- this is part of the
         * documented behavior of the "pop" instruction.
         */
        asm volatile("# __raw_save_flags\n\t"
                     "pushf ; pop %0"
                     : "=rm" (flags)
                     : /* no input */
                     : "memory");

        return flags;
}
```

첫번째 변수 flags 에 pushf 하여 rm 을 넣음?
no input 뭡니까

결국 flags 리턴하고 값 갱신 된 것..?

ㅠㅠㅠ 다음은 나중에 마무리 짓도록 하겠습니다…