**TI DSP, MCU 및 Xilinx Zynq FPGA**

**프로그래밍 전문가 과정**

**2018-04-20 (42회차)**

강사: lnnova Lee(이상훈)

[gcccompil3r@gmail.com](mailto:gcccompil3r@gmail.com)

학생: 정유경

[ucong@naver.com](mailto:ucong@naver.com)

**1. 시스템 콜 fork() 개요**

시스템 호출 이 발생하면 인텔프로세서 내부에서 인터럽트 벡터번호 0x80(128)을 이용하여system call루틴을 호출하고 커널로 진입한다.

시스템 호출 fork()

사용자 프로그램에서 시스템콜인 fork()함수를 호출하면 라이브러리(libc.a)의 fork()함수 내부에 mov 2 %eax , int 0x80을 실행한다

즉, fork()함수번호 2번을 eax레지스터에 저장, IDT의 엔트리번호 0x80(128)트랩게이트를 호출

fork함수번호는 sys\_call\_table의 엔트리 2번을 의미

0x80트랩게이트의 서비스루틴(system\_call)에서는 sys\_call\_table의 엔트리 2에 정의된 sys\_fork함수를 호출

커널에서 제공하는 모든 시스템호출 함수는 sys/\_call\_table에 정의되어 잇다

함수명은 모두 “sys\_”

**2. 커널 분석**

|  |
| --- |
| #ifdef \_\_ARCH\_WANT\_SYS\_FORK  SYSCALL\_DEFINE0(fork)  {  #ifdef CONFIG\_MMU // MMU-based Paged Memory Management Support가 정의되어 있는가  return \_do\_fork(SIGCHLD, 0, 0, NULL, NULL, 0); // SIGCHLD 는 17로 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 24번째 비트가 1인가) 들어감  if (clone\_flags & CLONE\_VFORK) // 17 & 0x00004000 (15번째 비트가 1인가 )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; // trace는 1  if (likely(!ptrace\_event\_enabled(current, trace))) // ptrace는 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] = {};    /\* if문 모두 패스함 하나도 안걸림\*/  retval = security\_task\_create(clone\_flags); // 들어가면  // 들어가면 2가지 있다. 하나는 0을 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…?? retval에 반환되는 값이 뭔지 잘 모르겠습니다.

retval에 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\_task가 현재 수행중인 태스크이면  return tsk→pref\_node\_fork; // tsk구조체의 pref\_node\_fork멤버를 리턴  #endif  return NUMA\_NO\_NODE;  } |

|  |
| --- |
| 리턴한 값을 node에 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(5번)이고, cachep는 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\_);  // 1,2번중 어디로 들어가야할지 모르겠습니다??  // trace는 디버깅, 볼필요 없다  return ret;  } |

슬랩할당자로 새로운 태스크에 메모리를 할당해준다

|  |
| --- |
| tsk에 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<<14이므로 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 16KB메모리를 할당받는다. // 더 들어가지 않고 진행  …  return page; // 가상주소 리턴  } |

정리하면, tsk는 Slab을 통해 할당받은 것이고

ti는 Buddy를 통해 할당받은 것이다.

|  |
| --- |
| ti에 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\_struct인 tsk에 orig를 복사한다  #ifdef CONFIG\_VM86  dst->thread.vm86 = NULL;  #endif  return fpu\_\_copy(&dst->thread.fpu, &src→thread.fpu); // fpu: 부동소수점 처리용 cpu  } |

|  |
| --- |
| err에 값을 반환받고 계속 진행  if (err)  goto free\_ti;  tsk->stack = ti; // thread\_info를 자식 프로세스의 stack에 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\_info를 자식의 thread\_info  task\_thread\_info(p)->task = p; // thread\_info의 task멤버를 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\_creds로 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); // siblings있으면 리스트 초기화  rcu\_copy\_process(p); // rcu있으면 setting    init\_sigpending(&p→pending); // 시그널 지연, 초기화  // → sigemptyset()  #ifdef CONFIG\_NUMA // NUMA이면 mempolicy복사  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 cpu에서 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 PID할당  /\*sigaltstack should be cleared when sharing the same VM\*/  /\* ok, now we should be set up.. \*/ |

여기까지 p=copy\_process()가 끝났다.