로그인 유지
회원 가입
아이디/비밀번호 찾기

강좌 & 팁
EZ-보드
그래픽 & 멀티미디어
네트워크 프로그래밍
디바이스 드라이버
ImJa 솔루션
임베디드 FLEX
안드로이드
iOS 개발
FAIDE
하드웨어
C/C++
go언어
GNU C 레퍼런스 메뉴얼
ARM 루아
응용 프로젝트
이클립스
지그비
개발자 노트
권장도서
FPGA
VIVADO
DSP
HSIO
Logic
Zynq

최근 글
보드에 iptables nat/... [1]
리눅스에서 파일이나 ...
리눅스 페도라 커널설...
[DOCKER] LINUX에서 ...
윈도우 임베디드 vs ...
라즈베리파이3 마이크...
[DOCKER] 부팅 시 자...
리눅스 'PortAudio'의...
RTL8198 에서 GDB를 ...
안녕하세요~ 인사드립...

최근 댓글
내용을 다시 수정했습니다.
범죄자들이 요구하는 몸값은 0...
보다! 랜섬 사랑 비트 코인 htt...
PortAudio 한번 검색해보시죠
자료 잘 봤습니다. 궁금한...
콘솔 결과값중 질문입니다. ...
좋은 정보 감사합니다^^ 덧붙...
아... platform중 flag를 잘못 ...

강좌 & 팁
글 수 2,406
회원 가입 로그인...

arm kernel 에서 task_struct, thread_info, switch_to 그리고 stack

박문식

http://forum.falinux.com/zbxe/index.php?document_srl=808363 8833 **2015.01.21** 16:57:48 (*.134.169.166)

제가 매번 내용 찾기가 귀찮아서 정리해서 올려 봅니다.

arm kernel 에서 task_struct, thread_info, switch_to

위의 세가지 주제에 대한 이야기는 많이 있지만 제가 언급하고 싶은 것은

프로세스 혹은 쓰레드라고 부르는 실행 단위에서 다른 프로세스로 전환될때

일어나는 일들과 스택의 사용에 대한 것들입니다.

먼저 커널에서 thread 와 task 는 어떤 관계냐?

task_struct 는 아키텍쳐와 무관하게 generic 하게 다루어지는 자료구조입니다.

선언은 include/linux/sched.h 에 선언되어 있습니다.

thread_info 는 아키텍쳐별로 다르게 구성됩니다.

당연히 아키텍쳐에서 선언됩니다.

arm 의 경우 arch/arm/include/asm/thread_info.h 에 선언되어 있습니다.

우리가 쉽게 상요하는 current 라는것이 있죠?

어떻게 가져올까요? arm 에서는 아래와 같은 경로를 통해서 구해 옵니다.

다른 아키텍쳐도 비슷하고 arm 의 예를 들면

arch/arm/include/asm/current.h

```
 8 static inline struct task_struct *get_current(void)
 9 {
10     return current_thread_info()->task;
11 }
12
13 #define current (get_current())
```

arch/arm/include/asm/thread_info.h

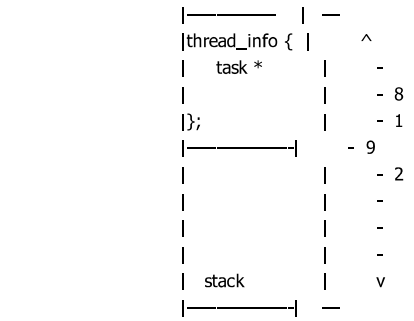
```
94 static inline struct thread_info *current_thread_info(void)
95 {
96     register unsigned long sp asm ("sp");
97     return (struct thread_info *)(sp & ~(THREAD_SIZE - 1));
98 }
```

thread 의 경우 크기가 8092 사이즈 입니다.
해당 구조안에 union 형태로 두가지가 선언됩니다.
include/linux/sched.h

```
2052 /*
2053  * The default (Linux) execution domain.
2054  */
2055 extern struct exec_domain   default_exec_domain;
2056
2057 union thread_union {
2058     struct thread_info thread_info;
2059     unsigned long stack[THREAD_SIZE/sizeof(long)];
```

```
2060 };
```

결국 현재의 task 는 sp 포인터를 구해서 8192 aling mask 를 통해서
thread_info 의 자료구조를 찾아서 thread_info 안에 있는 task 포인터를 가져오는 것입니다.



위의 구조와 같이 되어 있다고 보면 sp 를 구해서 8192 mask 를 통해서 thread_info 를 구하고
task * 를 보면 현재의 task_struct 를 구하게 되는 것입니다.

여기서 한가지 thread_info 가 아키텍쳐별로 구현된다는 것은
당연히 아키텍쳐별로 필요한 기능에서 필요하다는 것이겠죠?
바로 문맥전환이라고 부르는 context switch 에 필요한 내용들을 thread_info 에서 갖고 있습니다.
내용을 볼까요?

arch/arm/include/asm/thread_info.h

```
32 struct cpu_context_save {
33     __u32   r4;
34     __u32   r5;
35     __u32   r6;
36     __u32   r7;
37     __u32   r8;
38     __u32   r9;
39     __u32   sl;
40     __u32   fp;
41     __u32   sp;
42     __u32   pc;
43     __u32   extra[2];       /* Xscale 'acc' register, etc */
44 };
45
46 /*
47  * low level task data that entry.S needs immediate access to.
48  * __switch_to() assumes cpu_context follows immediately after cpu_domain.
49  */
50 struct thread_info {
51     unsigned long       flags;      /* low level flags */
52     int         preempt_count;  /* 0 => preemptable, <0 => bug */
53     mm_segment_t        addr_limit; /* address limit */
54     struct task_struct  *task;      /* main task structure */
55     struct exec_domain  *exec_domain;   /* execution domain */
56     __u32           cpu;        /* cpu */
57     __u32           cpu_domain; /* cpu domain */
58     struct cpu_context_save cpu_context;    /* cpu context */
59     __u32           syscall;    /* syscall number */
60     __u8            used_cp[16];    /* thread used copro */
61     unsigned long       tp_value;
62     struct crunch_state crunchstate;
63     union fp_state      fpstate __attribute__((aligned(8)));
64     union vfp_state     vfpstate;
65 #ifdef CONFIG_ARM_THUMBEE
66     unsigned long       thumbee_state;  /* ThumbEE Handler Base register */
67 #endif
68     struct restart_block    restart_block;
69 };
```

arm 에서 thread_info 의 모습입니다.
54라인에 task 를 가리키는 포인터가 있네요.
또한 58라인에는 문맥전환에 필요한 cpu 의 레지스터들을 위해서 저장할 공간이 있습니다.
그렇다면 실제 문맥전환이 어떻게 일어나는지 볼까요?

커널에서는 schedule 함수가 불리게 되면 현재 실행해야할 프로세스를 실행해주게 됩니다.

kernel/sched.c

```
4344 asmlinkage void __sched schedule(void)
4345 {
4346     struct task_struct *tsk = current;
4347
```

```
4348     sched_submit_work(tsk);
4349     __schedule();
4350 }
4351 EXPORT_SYMBOL(schedule);
```

__schedule 함수가 불리는데 그 함수안에서는 스케쥴이 일어나야 하는지 정책에 따라 검사를 하고
최종적으로 아래와 같이 호출이 됩니다.
kernel/sched.c

```
4308     if (likely(prev != next)) {
4309         rq->nr_switches++;
4310         rq->curr = next;
4311         ++*switch_count;
4312
4313         context_switch(rq, prev, next); /* unlocks the rq */
4314         /*
4315          * The context switch have flipped the stack from under us
4316          * and restored the local variables which were saved when
4317          * this task called schedule() in the past. prev == current
4318          * is still correct, but it can be moved to another cpu/rq.
4319          */
4320         cpu = smp_processor_id();
4321         rq = cpu_rq(cpu);
4322     } else
4323         raw_spin_unlock_irq(&rq->lock);
```

결국 4313 라인에서처럼 context_switch 가 불리게 되는데
그 안에서 불리는 함수가 switch_to 함수가 호출되고 이 부분은 아키텍쳐 별로 구현된 코드가 호출됩니다.

```
switch_to(prev, next, prev);
```

사용되는 자료구조는 rq 라고 불리는 run-queue, 그리고 task_struct 입니다.

arm 에서는 아래와 같이 되어 있죠.
arch/arm/include/asm/system.h

```
241 #define switch_to(prev,next,last)                   \
242 do {                                    \
243     last = __switch_to(prev,task_thread_info(prev), task_thread_info(next));    \
244 } while (0)
```

여기서 호출되는 rq, prev, next 의 경우 모두 task_struct 인데
task_thread_info(prev) 는 매크로로 아래와 같이 선언됩니다.
include/linux/sched.h

```
2379 #define task_thread_info(task)  ((struct thread_info *)(task)->stack)
```

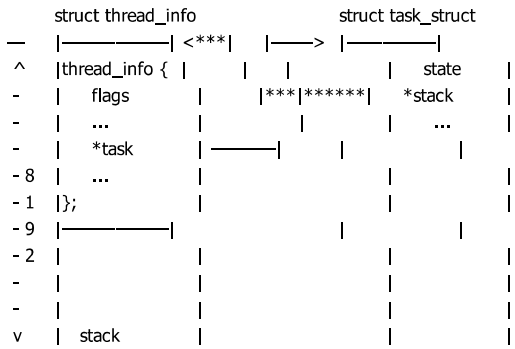
가만히 보면... task_struct 의 멤버인 stack 을 thread_info 로 사용하네요?
어떻게 이게 가능할까요?
task_struct 의 stack 멤버는 실제로는 thread_info 를 가리키고 있다.

kernel/fork.c

```
254 static struct task_struct *dup_task_struct(struct task_struct *orig)
255 {
256     struct task_struct *tsk;
257     struct thread_info *ti;
258     unsigned long *stackend;
259     int node = tsk_fork_get_node(orig);
260     int err;
261
262     prepare_to_copy(orig);
263
264     tsk = alloc_task_struct_node(node);
265     if (!tsk)
266         return NULL;
267
268     ti = alloc_thread_info_node(tsk, node);
269     if (!ti) {
270         free_task_struct(tsk);
271         return NULL;
272     }
273
274     err = arch_dup_task_struct(tsk, orig);
275     if (err)
276         goto out;
277
278     tsk->stack = ti;
```

마지막 278 라인을 보면 thread_info 를 task->stack 에 값을 넣고 있다.
이름만 stack 이다.
물론... 거기에 stack 이 존재하는 것은 사실이다.

아래 그림을 보면 쉽게 이해가 갈거 같다.



위와 같은 구조로 연결이 되어 있다.

자 그러면 마지막으로...
arm 에는 대표적으로 7가지 모드가 있다고 합니다.
뭐... user, system, irq, fiq, abort, svc, undef 이런 것들인데...
각각의 모드별로 stack 을 고유하게 설정할수 있도록 되어 있다.
당연히 위에서 본 것들은 커널 모드에서 즉 svc 모드에서 사용되는 스택이다.
유저모드에서는 어플리케이션이 실행될때 스택을 세팅할것이다.
그럼.... fiq, irq, abort 등의 stack 은 어디에 있을가요?

arm 에서는 cpu_init 시에 해당하는 처리를 하도록 되어 있습니다.
arch/arm/kernel/setup.c

```
118 struct stack{
119     u32 irq[3];
120     u32 abt[3];
121     u32 und[3];
122 } ____cacheline_aligned;

399 void cpu_init(void)
400 {
401     unsigned int cpu = smp_processor_id();
402     struct stack *stk = &stacks[cpu];
403
404     if (cpu >= NR_CPUS) {
405         printk(KERN_CRIT "CPU%u: bad primary CPU number\n", cpu);
406         BUG();
407     }
408
409     /*
410      * Define the placement constraint for the inline asm directive below.
411      * In Thumb-2, msr with an immediate value is not allowed.
412      */
413 #ifdef CONFIG_THUMB2_KERNEL
414 #define PLC "r"
415 #else
416 #define PLC "I"
417 #endif
418
419     /*
420      * setup stacks for re-entrant exception handlers
421      */
422     __asm__ (
423     "msr    cpsr_c, %1\n\t"
424     "add    r14, %0, %2\n\t"
425     "mov    sp, r14\n\t"
426     "msr    cpsr_c, %3\n\t"
427     "add    r14, %0, %4\n\t"
428     "mov    sp, r14\n\t"
429     "msr    cpsr_c, %5\n\t"
430     "add    r14, %0, %6\n\t"
431     "mov    sp, r14\n\t"
432     "msr    cpsr_c, %7"
433         :
434         : "r" (stk),
435           PLC (PSR_F_BIT | PSR_I_BIT | IRQ_MODE),
436           "I" (offsetof(struct stack, irq[0])),
437           PLC (PSR_F_BIT | PSR_I_BIT | ABT_MODE),
438           "I" (offsetof(struct stack, abt[0])),
439           PLC (PSR_F_BIT | PSR_I_BIT | UND_MODE),
440           "I" (offsetof(struct stack, und[0])),
441           PLC (PSR_F_BIT | PSR_I_BIT | SVC_MODE)
```

```
442         : "r14");
443 }
```

stack 의 크기는 u32 타입으로 (64비트에서는 달라지겠죠?) 3개씩 배열이 되고
해당 주소의 첫번째 주소가 인라인으로 cpu 의 모드를 바꿔가면서 세팅되게 됩니다.
왜 3개면 될까요?

arm 에서 인터럽트 벡터의 처리 부분은 아래와 같이 구성됩니다.
arch/arm/kernel/entry-armv.S

```
1041    .macro  vector_stub, name, mode, correction=0
1042    .align  5
1043
1044 vector_\name:
1045    .if \correction
1046    sub lr, lr, #\correction
1047    .endif
1048
1049    @
1050    @ Save r0, lr_<exception> (parent PC) and spsr_<exception>
1051    @ (parent CPSR)
1052    @
1053    stmia   sp, {r0, lr}        @ save r0, lr
1054    mrs lr, spsr
1055    str lr, [sp, #8]        @ save spsr
1056
1057    @
1058    @ Prepare for SVC32 mode.  IRQs remain disabled.
1059    @
1060    mrs r0, cpsr
1061    eor r0, r0, #(\mode ^ SVC_MODE | PSR_ISETSTATE)
1062    msr spsr_cxsf, r0
1063
1064    @
1065    @ the branch table must immediately follow this code
1066    @
1067    and lr, lr, #0x0f
1068  THUMB( adr r0, 1f         )
1069  THUMB( ldr lr, [r0, lr, lsl #2]   )
1070    mov r0, sp
1071  ARM(   ldr lr, [pc, lr, lsl #2]   )
1072    movs    pc, lr          @ branch to handler in SVC mode
1073 ENDPROC(vector_\name)
```

간단히 설명하면... 세개의 값을 저장합니다. r0, lr, spsr
그리고 모드를 svc 모드로 바꿔 커널의 스낵을 사용하고 모든 처리가 끝나면 복귀를 하게 됩니다.
1053 라인에서 stmia 를 사용하는 이유와 ! 가 없는 이유는 각각 해당 모드에서 stack 은
커널 스택처럼 자라나는 구조가 아니라 고정된 값에 저장하기 때문이고
! 없는 이유는 그것을 이용한 리턴코드가 없기 때문입니다.
리턴은 svc 모드에서 수행하니까요.
결국 아래 과정을 거치게 됩니다.

1. exception 진입
2. r0, lr, spsr 저장
3. svc 모드 전환
4. irq 처리
5. spsr 을 이용한 모드 복귀
6. 스택에 저장된 데이타를 이용한 exception 으로 부터 복귀

목록

번호	제목	글쓴이	날짜	조회 수
1966	iptables 에 대하여 3	박진호	2015-01-31	5959
1965	[Ubuntu]우분투14.04설치후 삼성 노트북 9시리즈 TouchPad마우스 동작 안할 경우 1	이병복	2015-01-31	11050
1964	Youtube 동영상 다운로드하는 방법	김재남	2015-01-30	7447
1963	링크할 때 심볼 충돌 방지 하기	김현기	2015-01-30	6650
1962	tooltip 확장 프로그램에 대하여	송기석	2015-01-30	5568
1961	# of parameters of macro function	홍성흔	2015-01-30	5829
1960	[AngularJS] templateUrl 사용하기	김민수	2015-01-30	15323
1959	윈도우7에서 외장하드디스크에 자료를 옮길때 chkdsk라는 문구가 나올 경우	김희래	2015-01-30	6963
1958	미티어(Meteor) 를 알게된 계기와 약간의 정보	유영창	2015-01-29	7106
1957	debootstrap으로 만드는 Debian 루트파일시스템 구축하기	이상민	2015-01-29	9233
1956	iptables에 대하여 2	박진호	2015-01-24	5938
1955	[Java]EZ-S3C6410에서의 Java성능	이병복	2015-01-24	6390
1954	[AngularJS] $scope? scope?	김민수	2015-01-24	11418
1953	Android studeio 우분투 설치	유지원	2015-01-23	7349
1952	[우분투 12.04] username 변경하기	김현기	2015-01-23	7974
1951	USB 랜 카드 사용 방법! (KY-RD9700)	김희래	2015-01-23	7488
1950	버추얼박스 네트워크 문제관련 팁.	송기석	2015-01-23	9998
1949	[우분투14.04]"Failed to create the VirtualBox COM object" 라는 메시지로 버추얼 박스 실행이 안될때	이상민	2015-01-22	8967
1948	orcad bom 추출시 pcb footprint 항목 넣는 방밥	김재남	2015-01-21	10862
	arm kernel 에서 task_struct, thread_info, switch_to 그리고 stack	박문식	2015-01-21	8833

목록 쓰기...

제목 ▼ 검색 취소

부탁드립니다. ^^

- 글을 작성하시려면 회원으로 가입하시고 **로그인 하셔야** 합니다.
- 회원 가입은 **주민등록번호가 필요 없으며**, 메일 주소만 있으면 간단하게 가입하실 수 있습니다.
- 본 포럼에 등록된 모든 글의 저작권은 작성하신 분께 있으며, 저작자의 허락 없이 다른 곳에 펌이나 도용하시면 안 됩니다.
- 또한, 전자우편 수집 프로그램이나 그 밖의 기술적 장치를 이용하여 무단으로 이메일 주소 및 자료 수집되는 것을 거부하며,
- 이를 위반 시 정보통신망 법에 의해 형사 처벌됨을 유념하시기 바랍니다.