



TI DSP, MCU 및 Xilinx Zynq FPGA 프로그램 전문가 과정

Innova Lee(이상훈) 강사
gcccompil3r@gmail.com

은태영 학생
zero_bird@naver.com

clone : 옵션값에 따라 프로세스(fork)가 되기도 하고, 스레드(thread)가 되기도 한다.

`#define _GNU_SOURCE` **//clone** 을 사용하기 위해 선언해야 한다.

```
#include <unistd.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <sched.h>
```

```
int g=2;
```

```
int sub_func(void *arg) {
```

```
    g++;
```

```
    printf("PID(%d) : Child g=%d Wn",getpid(), g);
```

```
    sleep(2);
```

```
    return 0;
```

```
}
```

복습

```
int main(void) {
    int pid;
    int child_stack[4096];
    int l = 3;
    printf("PID(%d) : Parent g=%d,l=%d \n", getpid(), g, l);
    clone(sub_func, (void *) (child_stack+4095), CLONE_VM | CLONE_THREAD | CLONE_SIGHAND, NULL);
    // 인자 1 : 어떤 구동을 할 것인지 나타낸다.
    // 인자 2 : 스레드가 사용할 stack 의 공간. 4095 인 이유는 배열이 0부터 시작하기 때문이다.
    // 물리메모리의 최소단위가 4kb 이기 때문에 4095 로 할당한다.
    // 인자 3 : VM 가상 메모리, THREAD 스레드로 사용, SIGHAND 시그널을 사용한다는 옵션값이다.
    sleep(1);
    printf("PID(%d) : Parent g=%d, l=%d \n", getpid(), g, l);

    return 0;
}
```

복습

vfork() : 프로세스 구성 정보만 잡는다.(메모리 레이아웃은 미 생성.)

exec : 프로세스를 실행한다. 기존에 있던 메모리 레이아웃을 덮어쓰기 때문에 fork 를 하여 실행한다.

fork 의 경우, 메모리 레이아웃을 복사하기 때문에 exec 사용 시, 복사 후 덮어쓰는 불필요한 작업이 발생하게 된다.

이를 해결하기 위하여, 메모리 레이아웃을 생성하지 않고 복제하는 vfork 가 이용된다.

C.O.W 기법을 통해 fork 의 비용이 많이 줄긴 하였지만, 상황에 맞게 사용하도록 한다.

메모리 공유 방식으로는 세어드 메모리, 메시지 큐, 파이프 가 존재한다.

크리티컬 섹션을 방지하기 위하여 : 뮤텍스, 스핀락, 세마포어 등을 이용한다.

프로세스나 스레드 모두 task(task_struct) 이다.

복습

User App 에서 Library(c library) 로 이동한다.

- Library 는 커널 개발자가 유저에게 제공한다.

library 는 int 128(system call) 을 이용하여 system call 로 이동한다.

- 모든 스레드 및 포크 는 kernel_thread 에서 관리한다. o
- 그 안에 있는 do_fork 의 옵션값에 따라서 svr_clone / svr_vfork / svr_fork 로 구분된다.

뒤에 있는것을 붙인다.

```
#define test(name) sys_##name
```

- test(fork) = sys_fork;

```
#define test(이름) 은_##이름
```

- test(태영) = 은_태영

시스템 콜은 어셈블리 어로 구동하기 때문에 asmlinkage 를 이용하여, c 를 호출 할 수 있게 만든다.

grep -rn "text" ./ | grep "text2" : 문자 탐색.

set hlsearch 문자열 : 문자열 탐색.

NPTL

NPTL(Native POSIX Thread Library) 은 LinuxThreads 의 단점을 극복하기 위한 새로운 구현으로 POSIX 요구사항 또한 충족한다. NPTL은 성능과 확장성 측면에서 LinuxThreads 보다 강력한 개선 사항을 제공한다. LinuxThreads 와 같이 NPTL은 1대1 모델을 구현한다.

Ulrich Drepper 와 Ingo Molnar 는 NPTL 설계에 참여한 레드햇 직원이다. 전반적인 설계 목표 중 몇 가지는 다음과 같다.

새로운 스레드 라이브러리는 POSIX를 준수해야 한다.

스레드 구현은 대규모 프로세서를 탑재한 시스템에서도 잘 동작해야 한다.

심지어 작은 작업을 위해 새로운 스레드를 생성하더라도 시작 비용이 낮아야 한다.

NPTL 스레드 라이브러리는 LinuxThreads와 이진 호환이 가능해야 한다. 이런 목적으로 LD_ASSUME_KERNEL을 사용할 수 있다. 이 기사 뒷부분에 다루겠다.

새로운 스레드 라이브러리는 NUMA 지원을 활용할 수 있어야 한다.

NPTL

NPTL의 장점

NPTL은 LinuxThreads에 비해 여러 가지 장점이 있다.

NPTL은 관리자 스레드를 사용하지 않는다. 프로세스의 일부로 모든 스레드에 치명적인 시그널을 보내는 등 관리자 스레드에서 필요한 몇 가지 요구 사항이 존재하지 않는다. 커널 자체가 이런 작업을 신경쓸 수 있기 때문이다. 커널은 또한 각 스레드 스택이 사용한 메모리를 할당 해제한다.

심지어 어버이 스레드를 정리하기 앞서 기다리고 있는 모든 스레드 종료를 관리하므로 좀비를 막을 수 있다.

관리자 스레드를 사용하지 않기 때문에, NPTL 스레드 모델은 NUMA와 SMP 시스템에서 좀 더 나은 확장성과 동기화 메커니즘을 제공한다.

새로운 커널 구현과 함께 NPTL 스레드 라이브러리는 시그널을 사용한 스레드 동기화 기법을 피한다. 이런 목적으로 NPTL은 퓨텍스(futex)라는 새로운 메커니즘을 도입했다. 퓨텍스는 공유 메모리 영역에서 동작하므로 프로세스 사이에 공유가 가능하므로 프로세스 간 POSIX 동기화를 제공한다. 또한 프로세스 사이에서 퓨텍스를 공유할 수도 있다. 이런 행동 양식은 프로세스 간 동기화를 현실로 만든다. 실제로 NPTL은 PTHREAD_PROCESS_SHARED라는 매크로를 포함해 개발자에게 다른 프로세스의 스레드 사이에 뮤텍스를 사용자 수준 프로세스에서 공유하기 위한 핸들을 제공한다.

NPTL이 POSIX 규약을 따르므로 NPTL은 프로세스 단위로 시그널을 처리한다. getpid()는 모든 스레드에서 똑같은 프로세스 ID를 반환한다. 예를 들어 시그널 SIGSTOP을 보내면 전체 프로세스가 멈춘다. LinuxThreads에서는 이 시그널을 받은 스레드만 멈춘다. 이는 NPTL 기반 응용에서 GDB와 같은 디버그 지원을 강화한다.

NPTL에서 모든 스레드에는 어버이 프로세스 하나만 존재하므로, 어버이에게 보고되는 자원 사용(CPU나 메모리 퍼센트와 같은)은 스레드 하나가 아니라 전체 프로세스에 보고된다.

NPTL 스레드 라이브러리에 도입된 중요한 특징 중 하나는 ABI(Application Binary Interface) 지원이다. 이는 LinuxThreads와 하위 호환이 가능하도록 돕는다. 다음에 다룰 LD_ASSUME_KERNEL의 도움을 받아 ABI 지원을 처리한다.

복습

```
tewill@tewill-Z20NH-ASS1BSU: ~/my_proj/kernel0/linux-4.4
# line filename / context / line
1 7 arch/alpha/include/asm/current.h <<current>>
#define current get_current()
2 20 arch/arc/include/asm/current.h <<current>>
#define current (curr_arc)
3 13 arch/avr32/include/asm/current.h <<current>>
#define current get_current()
4 13 arch/cris/include/asm/current.h <<current>>
#define current get_current()
5 15 arch/ia64/include/asm/current.h <<current>>
#define current ((struct task_struct *) ia64_getreg(_IA64_REG_TP))
6 13 arch/m32r/include/asm/current.h <<current>>
#define current (get_current())
7 24 arch/m68k/include/asm/current.h <<current>>
#define current get_current()
8 34 arch/mn10300/include/asm/current.h <<current>>
#define current get_current()
9 13 arch/parisc/include/asm/current.h <<current>>
#define current get_current()
10 28 arch/powerpc/include/asm/current.h <<current>>
#define current get_current()
11 16 arch/s390/include/asm/current.h <<current>>
#define current ((struct task_struct *) const)S390_lowcore.current_task)
12 79 arch/sh/kernel/cpu/sh3/entry.S <<current>>
#define current r7
13 72 arch/sh/kernel/cpu/sh5/switchto.S <<current>>
! the point where the process is left in suspended animation, i.e. current
14 31 arch/sparc/include/asm/current.h <<current>>
#define current _get_current()
15 26 arch/tile/include/asm/current.h <<current>>
#define current get_current()
16 17 arch/x86/include/asm/current.h <<current>>
#define current get_current()
17 25 arch/xtensa/include/asm/current.h <<current>>
#define current get_current()
18 7 include/asm-generic/current.h <<current>>
#define current get_current()
19 62 scripts/kconfig/gconf.c <<current>>
static struct menu *current;
20 30 tools/lib/lockdep/uinclude/linux/lockdep.h <<current>>
#define current (_curr())
21 100 tools/perf/builtins-timeline.c <<current>>
struct per_pidcomm *current;
22 1417 tools/perf/builtins-trace.c <<current>>
struct thread *current;
Type number and <Enter> (empty cancels): 1
```

```
tewill@tewill-Z20NH-ASS1BSU: ~/my_proj/kernel0/linux-4.4
Cscope tag: current_thread_info.h" 9L, 197C 6,24-26 All
# line filename / context / line
1 46 arch/alpha/include/asm/thread_info.h <<current_thread_info>>
#define current_thread_info() __current_thread_info
2 68 arch/arc/include/asm/thread_info.h <<current_thread_info>>
static inline __attribute_const__ struct thread_info *current_thread_info(void )
3 89 arch/arm/include/asm/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void ) __attribute_const__;
4 91 arch/arm/include/asm/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void )
5 74 arch/arm64/include/asm/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void ) __attribute_const__;
6 76 arch/arm64/include/asm/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void )
7 49 arch/avr32/include/asm/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void )
8 67 arch/blackfin/include/asm/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void )
9 68 arch/c6x/include/asm/thread_info.h <<current_thread_info>>
struct thread_info *current_thread_info(void )
10 5 arch/cris/include/arch-v10/arch/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void )
11 5 arch/cris/include/arch-v32/arch/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void )
12 73 arch/frv/include/asm/thread_info.h <<current_thread_info>>
#define current_thread_info() ({ __current_thread_info; })
13 56 arch/h8300/include/asm/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void )
14 96 arch/hexagon/include/asm/thread_info.h <<current_thread_info>>
#define current_thread_info() __current_thread_info
15 50 arch/ia64/include/asm/thread_info.h <<current_thread_info>>
#define current_thread_info() ((struct thread_info *) ((char *) current + IA64_TASK_SIZE))
16 55 arch/ia64/include/asm/thread_info.h <<current_thread_info>>
#define current_thread_info() ((struct thread_info *) 0)
17 62 arch/m32r/include/asm/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void )
18 60 arch/m32r/mm/mmu.S <<current_thread_info>>
ld r1, 0(16, r1) ; current_thread_info->cpu
19 93 arch/m32r/mm/mmu.S <<current_thread_info>>
ld r1, 0(16, r1) ; current_thread_info->cpu
20 47 arch/m68k/include/asm/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void )
21 83 arch/metag/include/asm/thread_info.h <<current_thread_info>>
static inline struct thread_info *current_thread_info(void )
22 93 arch/microblaze/include/asm/thread_info.h <<current_thread_info>>
-- More -- SPACE/d/j: screen/page/line down, b/u/k: up, q: quit
```


복습

```
tewill@tewill-Z20NH-AS51BSU: ~/my_proj/kernel0/linux-4.4
24
25 struct task_struct;
26
27 #include <asm/types.h>
28
29 typedef unsigned long mm_segment_t;
30
31 struct cpu_context_save {
32     __u32 r4;
33     __u32 r5;
34     __u32 r6;
35     __u32 r7;
36     __u32 r8;
37     __u32 r9;
38     __u32 sl;
39     __u32 fp;
40     __u32 sp;
41     __u32 pc;
42     __u32 extra[2]; /* Xscale 'acc' register, etc */
43 };
44
45 /*
46  * low level task data that entry.S needs immediate access to.
47  * __switch_to() assumes cpu_context follows immediately after cpu_domain.
48  */
49 struct thread_info {
50     unsigned long flags; /* low level flags */
51     int preempt_count; /* 0 => preemptable, <0 => bug */
52     mm_segment_t addr_limit; /* address limit */
53     struct task_struct *task; /* main task structure */
54     __u32 cpu; /* cpu */
55     __u32 cpu_domain; /* cpu domain */
56     struct cpu_context_save cpu_context; /* cpu context */
57     __u32 syscall; /* syscall number */
58     __u8 used_cp[16]; /* thread used copro */
59     unsigned long tp_value[2]; /* TLS registers */
60 #ifdef CONFIG_CRUNCH
61     struct crunch_state crunchstate;
62 #endif
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 };
```

current 는 현재 task_struct 구조체를 가리킬 수 있게 해준다.

내부적으로 thread_info 안에 위치한 *task 를 이용하여 구현되어 있다.

즉 *task 는 현재 task_struct 를 나타낸다.

cpu_context_save : arm 레지스트 정보가 들어있다.

thread_union : 커널 스택. ???hardware context???를 관리한다.

컨테스트 스위칭을 도와주는것이 thread_union 과 task_struct 이다.

thread_union

시스템 컨테스트 / 메모리 컨테스트 / 하드웨어 컨테스트

전부다 컨테스트 스위칭에 사용된다.

EUID : 권한을 빌려주는 대상.

인터럽트 : 실행하던 것을 wait queue 로 빼고 동작했다가,
wait queue 에 있는걸 다시 시작한다.

다. 과거 그렸던 것. 부모, 형제.

스케줄링 인포메이션

on_rq : 있는지 없는지 체크.

list_head : 2중 연결 리스트.(과거와 미래 연결) : sibling next, prev

prio, normal, rt_priority.

sched_entity : 일반 유저

sched_rt_entity : 시스템

signal_struct : 시그널이나 시그액션을 관리한다.


sighand_struct : 어떤 행동을 할 것인가 관리한다.

pending : 지연시킨다.

blocked : 시그널을 막고 싶은 것들을 처리한다.

memory : mm

가상메모리, pgb(페이지 디렉토리 시작위치), vm_struct(가상 메모리)



file
path :
super_block : 루트파일 찾는데 사용한다.
s_root : 루트파일 시스템의 위치가 저장됨.
mtd_info :
inode :

CPU(thread)_union