Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 장성환
redmk1025@gmail.com

* do_fork 함수를 분석하여 봅시다.

*NPTL 조사하기.

*clone_pt.c

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <linux/unistd.h>
#include <sched.h>

int sub_func(void *arg){
   printf("TGID(%d), PID(%d): Child\n", getpid(), syscall(__NR_gettid));
   sleep(2);
   return 0;
}

int main(void){

   int pid;
   int child_a_stack[4096];
   int child_b_stack[4096];

   printf("before clone \n\n");
   printf("TGID(%d), PID(%d): Parent\n", getpid(), syscall(__NR_gettid));

   clone(sub_func, (void*)(child_a_stack+4095),
CLONE_CHILD_CLEARTID | CLONE_CHILD_SETTID, NULL);
// 프로세스로 해석될 수 있도록 자원공유가 되지 않는 형태로 생성
   clone(sub_func, (void*)(child_b_stack+4095), CLONE_VM |
```

#include <linux/unistd.h>

syscall(__NR_gettid)

리눅스에서 새로 생성된 스레드에서 getpid() 호출 시, 메인 프로세스의 pid 를 리턴하는 경우가 있다. 이때, 사용하면 해당 스레드의 pid 를 얻을 수 있다.
(실상 tgid 값이 getpid()이며, syscall 리턴값이 pid 로 보면 된다.)
(태스크 생성시 이 태스크를 위한 pid 값을 할당한다. fork() 시 pid 를 tgid 에 넣고, 쓰레드 생성시 tgid 는 부모 태스크와 같은 값이 할당된다.)

int clone(int (*fn) (void*arg), void *child_stack, int flags, void *arg)

fn = sub function name
child_stack = 자식 프로세스에 의해 사용되는 스택의 위치
flags = 플래그 지정 (부모와 자식사이에 무엇이 공유되는가를 명시)

CLONE_VM
부모와 자식 프로세스는 동일한 메모리 공간에서 실행된다.

CLONE_SIGHAND
부모 및 자식 프로세스들은 시그널 처리기의 동일한 테이블을 공유한다.

```
CLONE_THREAD | CLONE_SIGHAND, NULL);
// 스레드로 해석될 수 있도록 자원공유 생성
   sleep(1);

   printf("after clone \n\n");
   return 0;
}
```

* NPTL (Native POSIX Thread Library) 조사

기존에 리눅스에서는 Linux Threads 방식을 썼다. 중요한 특징으로는 각 스레드를 독자적인 프로세스 ID 를 부여한 다른 프로세스로 구현하였고, 관리자 스레드가 존재하였다. 관리자 스레드 방식 설계 때문에 문맥 전환이 많이 일어나며, SMP 나 NUMA 시스템에서 확장성 문제를 초래할 수 있다.

NPTL 은 Linux Threads 의 단점을 극복하기 위한 새로운 구현 방법이다. NPTL 의 설계목표중 몇 가지는 다음과 같다.
1. 새로운 스레드 라이브러리는 POSIX 를 준수한다.
2. 스레드 구현은 대규모 프로세서를 탑재한 시스템에서도 잘 동작해야한다.
3. 시작 비용이 낮은 스레드를 생성한다.
4. Linux Threads 와 이진 호환이 가능해야 한다.
5. NUMA 지원을 활용할 수 있어야 한다.

NPTL 은 Linux Thread 방식에 비해 여러가지 장점이 있다.
1. 관리자 스레드를 사용하지 않는다 (모든 스레드에 시그널을 보내는 등의 관리자 행위를 신경쓰지 않는다.) 따라서 관리자 스레드에 필요한 요구사항이 존재하지 않는다.
   또한 관리자 스레드를 사용하지 않기 때문에 NUMA, SMP 시스템에서 좀 더 나은 확장성과 동기화 메커니즘을 제공한다.

2. 공유 메모리 영역에서 동작하는 뮤텍스를 통하여 프로세스 간 POSIX 동기화를 제공한다.

3. getpid()를 통하여 모든 스레드에서 똑같은 프로세스 ID 를 반환한다. 따라서 시그널을 보내면 전체 프로세스가 멈추게 된다. (Linux Threads 방식은 해당 스레드만 멈춘다.)

4. 모든 스레드에 부모 프로세스 하나만 존재하므로 부모 프로세스에 보고되는 자원사용의 정보는 스레드 하나가 아닌 전체 프로세스에 보고된다.