TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 최대성
c3d4s19@naver.com

2018.04.10. – 034일차 수업

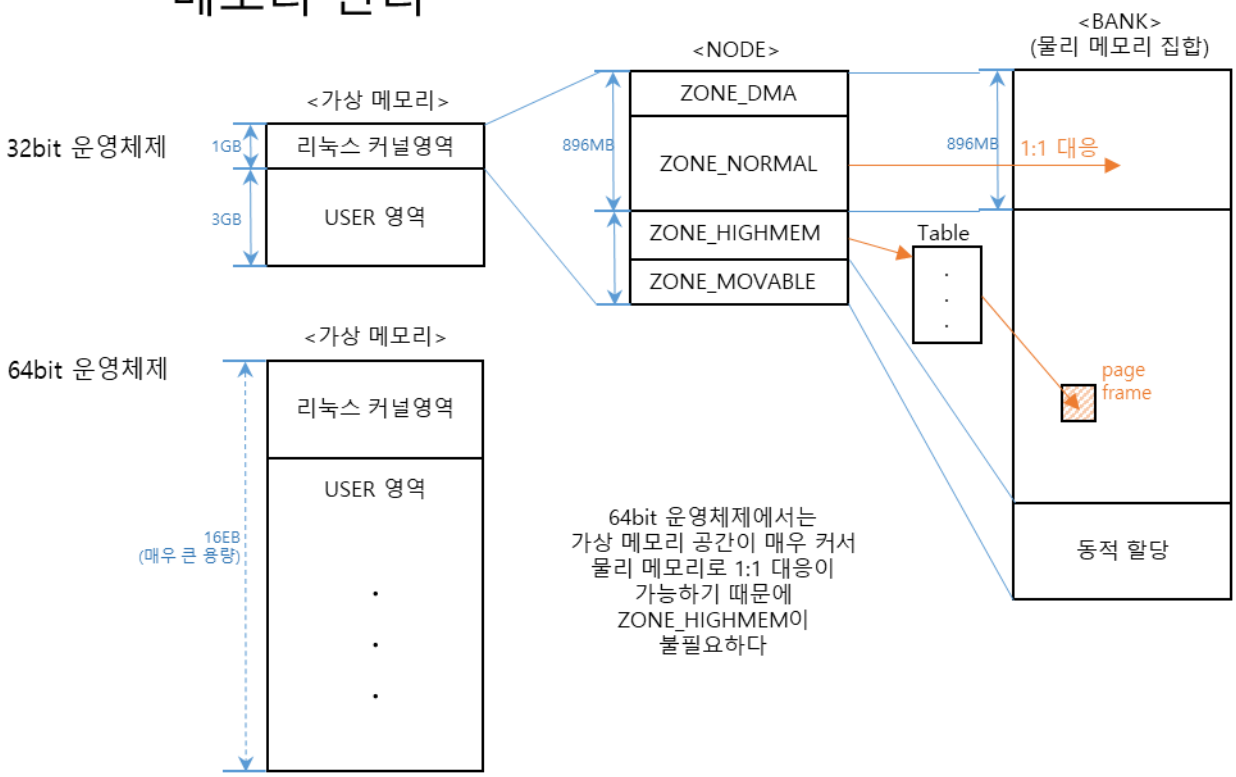
---

* 리눅스 커널 task_struct 구조체 드라이빙

```
    /* Kernel Driving */

task_struct // Task마다 생성됨
    state // Task의 현재 상태
    flag //
    file_struct
        file
            inode // super_block정보 가지고 있음
            path
                dentry // 디렉토리 관련 진입 정보
                    inode
                        super_block // 디스크 관리
                            s_dev // 디바이스
                            s_root // 루트 위치
    mm_struct // 메모리 관리
        rb_root mm_rb // vm_area_struct들은 RB트리로 관리되는데 그 트리의 시작주소 변수
        pgd // 페이지 테이블 관리, 페이징, 페이지 디렉토리 시작주소 들어있음(물리 메모리)
        vm_area_struct // 세그먼트(가상 메모리) 관리, (연속적으로 할당된 같은 속성의 세그먼트)
            vm_start, vm_end // 이 세그먼트의 시작과 끝 주소
            vm_flags // 권한이 유저인지 커널인지
            vm_pgoff , (file*)vm_file // 이 세그먼트가 실제 실행 파일의 어느 위치에 있는지 정보, page fault 발생시 어디 읽어야 할지 정보
        start_code, end_code // TEXT의 시작과 끝
        start_data, end_data // DATA의 시작과 끝
        start_brk, brk       // HEAP의 시작과 끝
        start_stack          // STACK의 시작
        arg_start, arg_end, env_start, env_end // 환경변수와 초기인자(main(argc, argv, env)함수가 전달될 때 인자를 의미함)
    list_head // 이중 연결 리스트 children, sibling, parent 등 다른 task_struct(Task)들을 가르킨다
    on_rq // 런큐에 있는지 없는지 확인(있으면 1, 없으면 0)
    policy // 이 Task가 어떤 스케줄링 정책을 사용하는지(FIFO, RR, DEADLINE)
    prio, static_prio, normal_prio // 우선순위
    sched_entity se // 노멀(동적 우선순위) 100 ~ 139 가상적으로 공평한 시간을 주고 우선순위에 따라 실제 런타임이 달라짐
        cfs_rq                  // cfs 스케줄러에 대한 런큐
        vruntime // 우선순위에 따라 실제 running time이 가상시간보다 크거나 작아진다
    sched_rt_entity rt          // 리얼타임(정적 우선순위) 0 ~ 99
        rt_rq                   // real time 스케줄러에 대한 런큐
    pid_t pid // process id 값
    pid_t tgid // thread group id 값
    start_time // Task의 시간 정보
    real_start_time // Task의 시간 정보
    signal_struct // 시그널 공유
    sighand_struct
        sigaction // signal handler 보관
    blocked // 특정 시그널 블록(받지 않음)
    sigpending // 공유하지 않는 signal 보관 (특정 쓰레드에만 signal 보내는 경우)
    audit_context // 이 Task에 대한 사용자 접근 권한 제어
        uid, euid, suid, fsuid // 사용자 ID, 유효 사용자 ID, 저장된 사용자 ID, 파일시스템 사용자 ID
        gid, egid, sgid, fsgid // 위와 마찬가지로 사용자 그룹에 대한 접근 제어 ID들
    thread_struct thread // 지금은 디버깅 용도로만 사용함 (context_switching은 다른 위치에 있음)
        address
        trap_no
        error_code // fault number
        debug
thread_union
    thread_info // current의 task_struct의 정보를 가르키는 포인터 pd(process descriptor)라고 부르기도 함
        flags // 스케줄링 할지 말지 결정
        task // task_struct 포인터
        cpu_context_save // cpu의 context switching 정보를 저장
            __u32 r4, r5, r6, r7, r8, r9    // 이건 arm의 경우임
            __u32 sl, fp, sp, pc, extra[2]
    syscall
```

메모리 관리



BANK -> 접근 속도가 같은 물리 메모리들의 집합, Bank 하나 당 Node 하나씩 존재

Node -> Bank를 관리하는 자료구조, pg_data_t로 Node 접근 가능, Node로 전체 물리메모리 접근 가능

ZONE -> 노드의 일부분을 따로 관리할 수 있는 자료구조, 메모리 특정 영역(물리메모리 16MB이하 부분)에 존재

ZONE_DMA (Direct Memory Access)

-> 데이터량이 방대한 경우 CPU거치지 않고 메모리 사용(영상, 비디오, 음성, Network 정보 등)

ZONE_NORMAL

-> 가상 메모리가 1:1로 대응하여 물리 메모리를 관리한다

ZONE_HIGHMEM

-> Table index를 이용한 간접참조방식으로 넓은 범위의 물리 메모리를 관리한다.