OSEK/VDX 운영체제를 위한 태스크 전환 기법

임성락*, 권오용**, 유영창***
*호서대학교 메카트로닉스 공학과
**호서대학교 컴퓨터 공학과
***(주)에프에이리눅스
e-mail:ssrim@hoseo.edu

Task Switching Mechanism for OSEK/VDX OS

Seong-Rak Rim*, O-Yong Kwon**, Young-Chang Yu***
*Dept. of Mechatronics Engineering, Hoseo University
**Dept. of Computer Engineering, Hoseo University
***Falinux Co.,Ltd

요 약

OSEK/VDX 운영체제는 자동차 전자 제어 장치(ECU)를 위하여 OSEK/VDX 사양을 준수하는 실시간 운영체제로써 다중처리를 위한 태스크 전환 메카니즘이 요구된다. 본 논문에서는 OSEK/VDX 운영체제의 요구사항을 고려하여 ARM 프로세서를 기반으로 한 OSEK/VDX 운영체제의 태스크 전환 메카니즘을 지원하기 위한 기법을 제시한다. 제시한 기법의 타당성을 검토하기 위하여 태스크 전환 루틴과 태스크 전환 관련 API 함수들을 구현하여 EZ-AT7 임베디드 보드에서 이들의 동작 상태를 테스트한다.

1. 서론

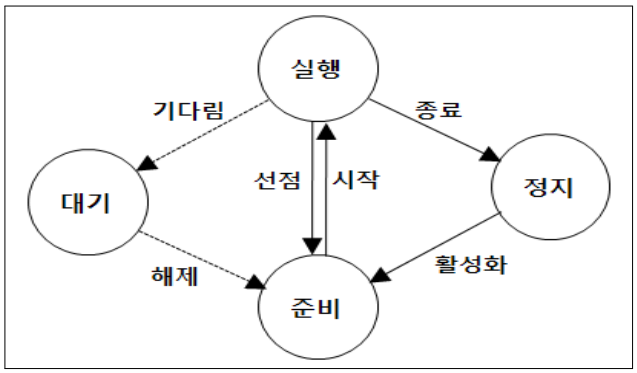
OSEK/VDX 운영체제는 자동차 전자 제어 장치(ECU)를 위하여 OSEK/VDX에서 제안한 사양을 준수하는 실시[1]간 운영체제로써 태스크 전환 메카니즘을 제공한다[1]. OSEK/VDX 운영체제의 기본 메카니즘은 태스크를 우선순위에 따라 처리하기 위한 다중처리(Multitasking), 태스크간의 동기화를 위한 자원 및 사건 관리(Resource and Event Management), 경고(Alarm), 카운터(Counter) 그리고 오류처리(Error Handling)기능을 제공한다[2][3].

본 논문에서는 다중처리를 위하여 OSEK/VDX 운영체제의 요구사항을 고려하여 태스크 전환 메카니즘을 위하여 ARM 프로세서 기반의 기법을 제시한다. 제시한 기법의 타당성을 검토하기 위하여 태스크 전환 루틴을 구현하고, 태스크 관리와 관련된 API 함수들과 함께 ARM 프로세서가 탑재된 EZ-AT7 임베디드 보드[4]에서 이들의 동작 상태를 테스트한다.

2. 태스크 전환 메카니즘

OSEK/VDX 운영체제에서는 태스크 단위로 CPU를 할당하며 2가지 종류(기본, 확장)의 태스크가 존재한다. 태스크의 상태 및 상태 전환도는 (그림 1)과 같다. 기본 태스크와 확장 태스크의 차이점은 오직 확장 태스크에 어떤 사건을 기다리고 있는 대기상태가 추가적으로 존재하는 점이다.[1]

태스크 전환 메카니즘은 다음에 수행되어야 할 태스크를 결정하는 스케줄러와 스케줄러에 의해 선택한 태스크에게 CPU의 제어를 넘겨주는 태스크 디스패처로 구성된다.



(그림 1) 태스크 상태 전환도

2.1.1 태스크 스케줄러

OSEK/VDX 운영체제에서 태스크 스케줄러는 다음에 실행되어야 할 태스크를 결정하기 위하여 기본적으로 다음과 같은 과정을 거치게 된다.

[단계-1] 준비/실행 상태의 태스크 집합을 찾는다.
[단계-2] 준비/실행 상태의 태스크 집합 중에서 최고 우선순위 태스크집합을 선택한다.
[단계-3] 최고 우선순위 태스크 집합에서 가장 먼저 준비 큐에 등록된 태스크를 선택한다.

태스크의 우선순위 값이 클수록 우선순위가 높으며 우선순위 값이 동일할 경우엔 준비 큐에 등록된 순서대로 실행된다.

1) 이 논문은 2010년도 호서대학교의 재원으로 학술연구비 지원을 받아 수행된 연구임(20090523)

(그림 1)에서 선점된 태스크는 준비 큐의 앞에 등록되고, 정지/대기 상태에서 천이된(활성화/해제) 태스크는 준비 큐의 뒤에 등록된다.

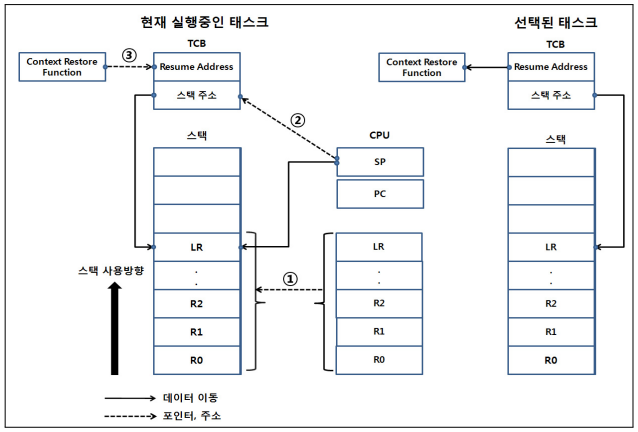
태스크의 우선순위 값은 시스템 생성 시 사용자에 의해 부여되며 초기에 부여된 우선순위 값은 변하지 않는다. 그러나 동기화를 위한 Priority Ceiling Protocol[1]이 적용될 경우에만 일시적으로 변할 수 있다.

2.1.2 태스크 디스패처(Task Dispatcher)

OSEK/VDX 운영체제의 태스크 디스패처는 스케줄러에 의해 선택된 태스크에게 CPU의 제어를 넘겨주기 위하여 다음과 같은 과정을 거치게 된다.

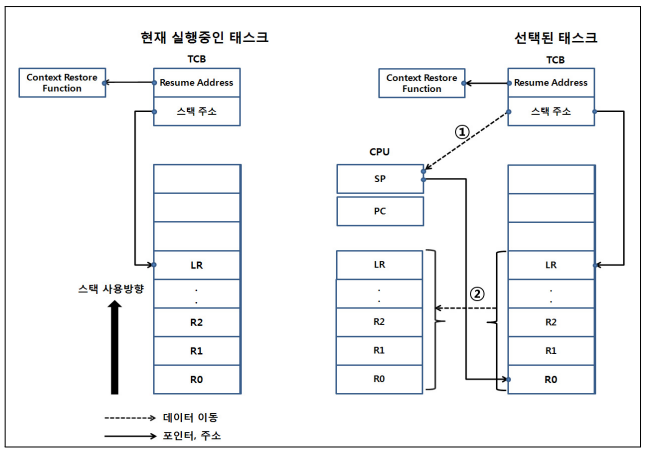
[단계-1] 현재 실행 중인 태스크의 문맥을 저장한다.

[단계-2] 선택된 태스크의 문맥을 복구한다.



(그림 2) 태스크 문맥 저장[단계-1]

(그림 2)에서 현재 실행중인 태스크의 문맥 즉, CPU의 범용 레지스터(R0-R12)와 링크 레지스터(LR)의 값을 현재 실행중인 태스크의 스택에 저장한 후, 스택 레지스터(SP)의 값을 현재 실행중인 태스크의 TCB에 저장한다. 마지막으로 저장된 문맥을 복구하기 위한 문맥복구 함수(Context Restore Function)의 시작주소를 현재 실행중인 태스크의 TCB에 저장한다.



(그림 3) 태스크 문맥 복구[단계-2]

(그림 3)에서 선택된 태스크의 TCB에 저장된 스택 레지스터(SP)값을 복구하여 선택된 태스크의 스택영역을 설정한 후, 선택된 태스크의 TCB에 저장된 문맥 복구 루틴의 시작주소를 PC에 설정함으로써 선택된 태스크에게 CPU의 제어를 넘긴다.

태스크의 문맥을 저장하고 선택된 태스크의 문맥을 복구하여 전환하는 때 까지 소요되는 시간을 디스패치 지연시간(Dispatch Latency Time)이라 한다. OSEK/VDX 운영체제와 같은 RTOS 에서는 디스패치 지연시간의 영향이 매우 크기 때문에 반드시 짧아야 한다.

3. 구현 및 테스트

제시한 메카니즘의 타당성을 검증하기 위하여 어셈블리어와 C언어를 이용하여 인터럽트 처리와 스케줄러 및 디스패처 루틴, 그리고 이를 태스크 하는데 필요한 태스크 관리 관련 OSEK/VDX 운영체제 서비스 함수들(ActivateTask, TerminateTask)을 구현하여 AT91SAM7S256이 탑재된 EZ-AT7 임베디드 보드에서 테스트하였다.

3.1 구현 내용

3.1.1 시스템 시작

CPU가 리셋이 되면 어셈블리어를 이용하여 하드웨어를 초기화한 후, C 언어의 Main()을 호출한다. Main()에서 인터럽트 핸들러를 등록하고 StartOS()를 호출한다. StartOS()에서는 자동실행 태스크들을 준비 큐에 등록한 후, 태스크 스케줄러를 수행하여 준비 큐에 있는 가장 높은 우선순위 태스크를 선정한 후 디스패처를 호출하여 태스크를 실행시킨다.

만약, 자동실행 태스크가 없거나 준비 큐에 태스크가 없을 경우엔 무한 루프를 돌면서 준비 큐를 검색한다.

3.1.2 태스크 스케줄러(Task Scheduler)

태스크 스케줄러가 준비상태에 있는 높은 우선순위 태스크를 찾기 위하여, 준비 상태의 태스크 집합 중 최고 우선순위 집합에서 가장 낮은 우선순위 집합으로 차례대로 탐색하는 선형탐색 알고리즘을 이용한다.

```
task_info_t*   read_highest_priority_task(void)
{
  u32 priority_point;

  for(priority_point = 0 ; priority_point < _max_priority
      ; priority_point++)
  {
    if(mo_ready_queue_main[priority_point] != NULL)
      return  mo_ready_queue_main[priority_point];
  }

  return READY_QUEUEU_EMPTY;
}
```

(그림 4) 태스크 스케줄러

(그림 4)에서 가장 높은 우선순위 준비큐에서 가장 낮은 우선순위 준비큐로 검색하면서 준비큐에 태스크가 존재할 경우 태스크를 선택 태스크로 설정한다.

3.1.3 태스크 디스패처(Task Dispatcher)

태스크 디스패처는 현재 실행 태스크의 문맥을 저장하는 루틴과 선택된 태스크로 전환하기 위해 문맥을 복구하는 루틴으로 구성되어 있다.

```
current_context_save:

    /* 이전에 실행중인 태스크가 있는지 체크한다.*/
    ldr   r0, = current_task
    ldr   r1, [r0]

    cmp   r1, #0

    /* 이전에 실행중인 태스크가 존재하지 않을경우 선택된 태스크로
       CPU 제어를 넘긴다. 존재할경우 실행중인 태스크의 문맥을 저장
      한다.*/
    beq   select_context_restore

    stmfd sp!, {r0-r12,r14}  ------------------------------ ①

    str   sp, [r1,#STACK_ADDRESS]  ---------------------- ②

    ldr   r2, = restore_cpu
    str   r2, [r1,#RESUME_ADDRESS]  ---------------------- ③
```

(그림 5) 태스크 문맥 저장

(그림 5)에서 먼저 현재 실행중인 태스크의 문맥 즉, CPU의 범용 레지스터(R0-R12)와 링크 레지스터(LR)의 값을 현재 실행중인 태스크의 스택에 저장한 후, 스택 레지스터(SP)의 값을 현재 실행중인 태스크의 TCB에 저장한다. 마지막으로 저장된 문맥을 복구하기 위한 문맥복구 함수(Context Restore Function)의 시작주소를 현재 실행중인 태스크의 TCB에 저장한다.

```
select_context_restore :

    /* 선택된 태스크를 현재 실행태스크로 지정한다.*/
    ldr   r0, = selected_task
    ldr   r0, [r0]

    ldr   r1, = current_task
    str   r0, [r1]

    ldr   sp, [r0,#STACK_ADDRESS]  ---------------------- ①

    ldr   r1, [r0,#RESUME_ADDRESS]
    bx    r1                        ---------------------- ②

context_restore_function:
    ldmfd sp!, {r0-r12,pc}          ---------------------- ③
```

(그림 6) 태스크 문맥 복구

(그림 6)에서 선택된 태스크의 TCB에 저장된 스택 레지스터(SP)값을 복구하여 선택된 태스크의 스택영역을 설정한 후, 선택된 태스크의 TCB에 저장된 문맥 복구 루틴의 시작주소를 PC에 설정함으로써 선택된 태스크에게 CPU의 제어를 넘긴다.

3.1.4 운영체제 서비스 함수

ActivateTask(TaskID)

ActivateTask() 서비스는 ID에 해당하는 태스크를 준비상태로 전환한다. 태스크가 이미 준비상태 일 경우에는 활성화 개수를 증가시킨다. 활성화 개수가 최대 활성화 개수를 넘어갔을 경우 에러를 반환한다.

```
StatusType    ActivateTask(TaskType  task_id)
{
  task_info_t get_task, activate_task;

  activate_task = &gen_task[task_id];  ---------- ①

  if(activate_task ->state != SUSPENDED)
  {
    if(activate_task ->activation >= activate_task ->max_activation)
       return E_OS_LIMIT;

    else                                          ---------- ②
    {
      insert_last_ready_queue(activate_task);
      activate_task ->activation = activate_task ->activation + 1 ;
    }
  }
  else
  {
    activate_task ->activation = activate_task ->activation + 1 ;
    activate_task ->state = READY;                ---------- ③
    insert_last_ready_queue(activate_task);
  }

  /* 가장 높은 우선순위 태스크를 선택 */
  get_task = read_highest_priority_task();
  if(IS_CURRENT_TASK_PREEMPTIVE())                ---------- ④
    dispatch_cpu_ready(get_task);

    return E_OK;
}
```

(그림 7) ActivateTask(TaskID)

(그림 7)에서 TaskID를 이용하여 해당하는 태스크의 TCB를 얻어온다. 태스크의 상태가 정지상태가 아닐 경우 최대 활성화 개수를 비교한다. 활성화 개수가 같거나 높을 경우 에러를 리턴하고 반대인 경우에는 활성화 개수를 한 개 증가시킨다. 태스크의 상태가 정지 상태 일 경우 태스크를 준비상태로 전환한다. 마지막으로 함수를 호출한 태스크가 선점형 일 경우 준비상태에 있는 가장 높은 우선순위 태스크를 선택하여 태스크를 전환한다.

TerminateTask()

TerminateTask API 서비스는 현재 실행중인 태스크를 정지 상태로 전환한다. 태스크 활성화 개수가 여전히 남아있을 경우에는 정지 상태로 전환하지 않고 준비상태로 전환한다.

(그림 8)은 TerminateTask API 서비스를 나타낸다.

```
StatusType   TerminateTask(void)
{
  task_info_t    get_task;
  _current_task->activation = _current_task->activation - 1 ;

  if(_current_task->activation == 0 )
  {
    _current_task->pc_address    = _current_task->start_pc;
    _current_task->stack_address = _current_task->start_stack;
    _current_task->state         = SUSPENDED;
    _delete_ready_queue(_current_task);
    _current_task                = NULL;
  }

  else
  {
    _current_task->pc_address    = _current_task->start_pc;
    _current_task->stack_address = _current_task->start_stack;
    _current_task->state         = READY;
    _delete_ready_queue(_current_task);
    _current_task                = NULL;

  }

  /* 가장 높은 우선순위 태스크를 선택 */
  get_task = read_highest_priority_task();
  dispatch_cpu_ready(get_task);
}
```

① ② ③

(그림 8) TerminateTask API 서비스

(그림 8)에서 현재 태스크의 활성화 개수를 1개 감소시킨 후 활성화 개수 체크를 한다. 현재 태스크의 활성화 개수가 0일 경우 태스크 초기화 후 태스크 상태를 정지 상태로 전환한다. 반대로 활성화 개수가 여전히 남아 있을 경우 태스크 초기화 후 앞에 경우와 반대로 준비 상태로 전환한다. 마지막으로 준비상태에 있는 가장 높은 우선순위 태스크를 선택하여 태스크를 전환한다.

3.2. 테스트

본 논문에서 제시한 태스크 전환 메카니즘의 타당성을 검증하기 위해(주)에프에이리눅스에서 제공하는 EZ-AT7 임베디드 보드[4]에서 <표 1>과 같은 태스크를 사용하여 <표 2>와 같은 OSEK/VDX에서 제공하는 테스트 시퀀스를 이용하여 테스트한다.

<표 1> 태스크 속성

스케줄링 정책 : 완전 선점 스케줄링			
태스크	**자동실행**	**우선순위**	**선점/비선점**
태스크 0	예	1	선점형
태스크 1	아니오	2	선점형
태스크 2	아니오	3	선점형

<표 2> 동작 시퀀스

태스크	**실행중인 운영체제 서비스**	**리턴 상태**
태스크 0	ActivateTask(태스크 3)	E_OK
태스크 2	ActivateTask(태스크 2)	E_OK
태스크 2	TerminateTask()	
태스크 1	TerminateTask()	
태스크 0	TerminateTask()	

(그림 9)는 <표 1>의 태스크 속성을 이용하여 <표 2>의 동작 시퀀스에 따라 테스트한 다음과 같은 결과를 통하여 태스크 전환 기법의 타당성을 확인할 수 있었다.

```
----------------------true
0 start\r\n
----------------------true
2 start\r\n
----------------------true
OK\r\n
----------------------true
2 end\r\n
----------------------true
1 start\r\n
----------------------true
1 end\r\n
----------------------true
OK\r\n
----------------------true
0 end\r\n
----------------------true
Enter IDLE\r\n
----------------------true
Test Success!!
```

(그림 9) 테스트 결과

5. 결 론

본 논문에서는 OSEK/VDX 운영체제의 요구사항을 고려하여 ARM 프로세서를 기반으로 한 OSEK/VDX 운영체제의 태스크 전환 메카니즘을 지원하기 위한 기법을 제시하고, 태스크 전환 루틴과 태스크 전환 관련 API 함수들을 구현하여 EZ-AT7 임베디드 보드에서 이들의 동작 상태를 테스트함으로써 제시한 기법의 타당성을 검토 하였다.

참 고 문 헌

[1] OSEK/VDK Operating System Specification 2.2.3, 2005. 2. (http://www.osek-vdx.org)

[2] 유우석, 박지용, 홍성수 “분산형 실시간 차량제어 시스템을 위한 RTOS, 미들웨어 및 결함 허용성 요소기술연구” 2006.

[3] 서영빈, 김상철, 마평수, 최태영 “ROSEK: OSEK 기반 자동차용 운영체제” 정보처리학회지 제15 권 5호 2008. 9

[4] http://www.falinux.com