OSEK/VDX 운영체제를 위한 Event 서비스 설계 및 구현

권오용*, 임성락**, 유영창***, 송기석****
*호서대학교 메카트로닉스 공학과
**호서대학교 컴퓨터 공학과
***(주)에프에이리눅스
****충북대학교 컴퓨터 과학과
e-mail:lobogelico@naver.com

Implementation and Design of Event Services for OSEK/VDX Operating System

O-Yong Kwon*, Seong-Rak Rim**, Young-Chang Yu***, Ki-Seok Song***
*Dept of Mechatronics Engineering, Hoseo University
**Dept of Computer Engineering, Hoseo University
***FALINUX co.,Ltd
****Dept of Computer Science, Chungbuk University

요 약

OSEK/VDX 운영체제는 자동차 전자제어장치(ECU)를 위하여 OSEK/VDX에서 제안한 사양을 준수하는 실시간 운영체제이다. 본 논문에서는 OSEK/VDX 운영체제의 Event 관리 메커니즘에 대한 전반적인 설명과 이를 지원하기 위한 4가지의 서비스 함수를 설계하고 구현하였다.

1. 서론

OSEK/VDX 운영체제는 자동차 전자제어장치(ECU)를 위하여 OSEK/VDX에서 제안한 사양을 준수하는 실시간 운영체제이다. OSEK/VDX 운영체제의 기본 메커니즘은 태스크를 우선순위에 따라 처리하기 위한 태스크 관리(Task Management), 태스크간의 동기화를 위한 자원 및 Event 관리(Resource and Event Management), 경고(Alarm), 카운터(Counter) 그리고 오류처리(Error Handling), 디버깅(Debugging) 기능을 제공한다.[2][3]

본 논문에서는 태스크간의 동기화를 위하여 OSEK/VDX 운영체제에서 준수해야할 Event 관리 메커니즘과 이를 지원하기 위한 4가지 서비스 함수에 대해 설계 및 구현을 기술한다.

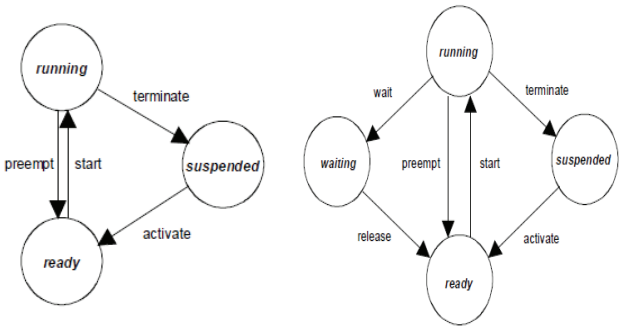
2. Event 메커니즘

Event 메커니즘은 현재 실행중인 태스크보다 낮은 우선순위의 태스크를 위해 대기하는 태스크간의 동기화의 역할을 하며 다음과 같은 특징을 갖는다.

- 확장 태스크를 위하여 제공된다.
- 태스크의 전이가 실행상태에서 대기상태로 대기상태에서 준비상태로 이루어진다.

(그림 1)은 OSEK/VDX에서 지원하는 2가지 태스크(기본태스크, 확장태스크)의 상태 전환도를 나타낸다.

(그림 1)의 상태 전환도에서 running에서 waiting 상태로 전이되는 wait 사건과 waiting에서 ready상태로 전이되는 release사건이 여기에 속한다.



<기본태스크> <확장태스크>

(그림 1) 태스크 상태 전환도

Event 객체들은 운영체제에 의해 관리가 되며, waiting 상태가 존재하는 확장태스크에만 할당이 된다. 따라서 waiting상태가 존재하지 않는 기본태스크에는 할당할 수가 없다.

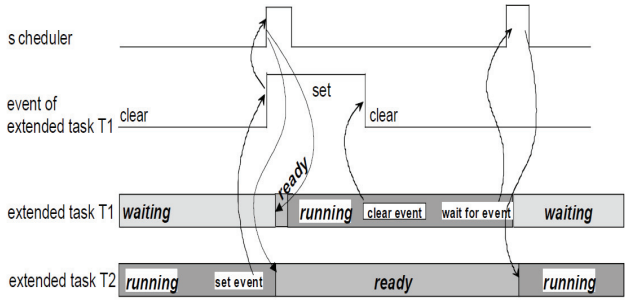
Event를 관리하기 위하여 확장태스크에게 Event의 ID를 할당한다. Event ID를 할당 받은 확장태스크를 Event의 소유자라 한다. Event는 해당 Event의 소유자와 ID에 의하여 구별된다. 확장태스크가 현재 활성화 중일경우에는 운영체제에 의해 Event들은 해제가 된다. 모든 확장태스크는 Event를 설정하여 다른 확장 태스크를 중단 할 수 없으며, 오직 Event의 소유자만이 Event의 해제 및 대기를 위한 설

정이 가능하다.

운영체제는 Event의 설정과 해제 그리고 현재 상태 값을 얻거나 Event 발생을 위한 대기 서비스를 제공한다.[1]

2.1 Event 동작

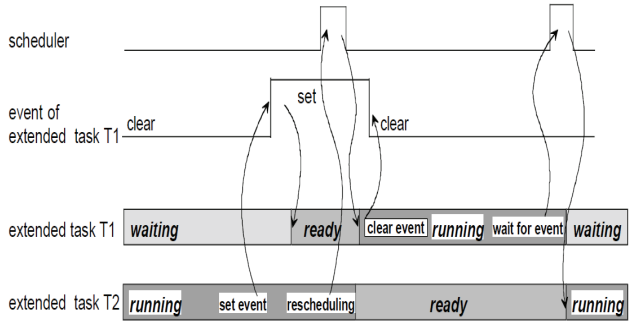
(그림 2)는 Event를 이용한 선점형 확장 태스크간의 동기화를 설명한다.



(그림 2) 선점형 확장 태스크의 동기화

Task T1은 가장 높은 우선순위를 갖는다. 태스크 T1은 Event를 위해 대기한다. 태스크 T2는 T1을 위하여 Event를 설정한다. 스케줄러가 활성화 된다. 이후에 T1은 대기상태로 부터 준비상태로 전환된다. T1의 T2보다 높은 우선순위 때문에 결과적으로 태스크 스위칭이 발생하고, T2는 T1에 의하여 선점 당한다. T1의 Event는 해제된다. 그 후에 T1은 다시 Event를 기다리고 스케줄러는 계속해서 T2를 실행한다.

(그림 3)은 Event를 이용한 비선점형 확장 태스크간의 동기화를 설명한다.



(그림 3) 비선점형 확장 태스크의 동기화

Task T1은 가장 높은 우선순위를 갖는다. 태스크 T1은 Event를 위해 대기한다. 태스크 T2는 T1을 위하여 Event를 설정한다. 이후에 태스크 T2는 대기상태에서 준비상태로 전환한다. T2는 비선점형 태스크 이므로 이벤트 발생 즉시 스위칭이 발생하지 않으며 태스크 T2 계속 실행을 하다가 제어권을 반납하면 스케줄러가 호출되고 T1에 의해 선점 당한다. T1의 이벤트는 해제되며 계속해서 실행이 된다. 그 후에 T1은 다시 Event를 기다리고 스케줄러는 계속해서 T2를 실행한다.

3. 서비스 설계

Event 메커니즘을 위하여 OSEK/VDX 운영체제는 SetEvent(TaskID,Mask), ClearEvent(Mask), WaitEvent(Mask), GetEvent(TaskID, Event) 4가지의 서비스를 제공한다.

3.1 SetEvent(TaskID,Mask)

SetEvent(TaskID,MASK)는 TaskID에 해당하는 태스크의 이벤트를 설정 한다. 설정된 이벤트가 태스크가 기다리던 이벤트일 경우 waiting에서 ready상태로 전이된다.

SetEvent의 알고리즘은 (그림 4)와 같다.

```
SetEvent(TaskID, Mask)
{
  /* TaskID 유효여부 검사 */
  if(IS_INVALID_TASK_ID(TaskID)) return E_OS_ID;

  /*  TaskID에 해당하는 태스크 얻기 */
  current_task = get_task_by_id(TaskId);

  /* 확장 태스크 여부 검사 */
  if(current_task->extended != EXTENDED_TASK)
   return E_OS_ACCESS;

  /*  태스크의 상태 여부 검사 */
  if(current_task->state == SUSPENDED)
    return E_OS_STATE;

  /*  이벤트 설정 */
  current_task->set_event_mask |= Mask;

  /* 설정된 이벤트가 태스크가 기다리는 이벤트인지 검사*/
  if(current_task->wait_event_mask & Mask)
  {
  /* 준비큐에 태스크를 삽입 */
    insert_ready_queue(current_task);

  /* 현재 태스크가 선점형 혹은 비선점형인지 체크. */
   if(current_task->preempt == PREEMPT) call_scheduler();
  }
  return E_OK;
}
```

(그림 4) SetEvent(TaskID,MASK) 알고리즘

SetEvent(TaskID,Mask)는 우선적으로 TaskID의 유효 여부를 검사한다. TaskID가 유효하지 않으면 E_OS_ID를 리턴한다 다음으로 TaskID를 이용하여 태스크 정보를 얻어온다. 태스크 정보에서 태스크가 확장태스크인지 여부를 검사한다. 확장태스크가 아닐 경우 E_OS_ACCESS 리턴한다. 다음으로 태스크의 현재 상태가 suspended일 경우 E_OS_STATE를 리턴한다. 이 후에 이벤트를 설정하고 설정된 이벤트가 태스크가 기다리던 이벤트일 경우 준비큐에 태스크를 넣고 상태를 준비상태로 바꾼다. 현재 태스크의 종류를 검사하여 선점형일 경우 스케줄러를 호출하여 태스크를 스위칭하고 비선점일 경우 E_OK를 리턴한다.

3.2 ClearEvent(Mask)

ClearEvent(Mask)는 서비스를 호출한 태스크에 설정된 이벤트를 해제한다. 알고리즘은 (그림 5)와 같다.

```
ClearEvent(Mask)
{
  /* 확장 태스크 여부 검사 */
  if(current_task->extended != EXTENDED_TASK)
   return E_OS_ACCESS;

  /* 호출 레벨 검사  */
  if(Level != TASK_LEVEL) return E_OS_CALLEVEL

  /* 설정된 이벤트를 해제 */
  current_task->set_event_mask &= (~event_mask);
}
```

(그림 5) ClearEvent(Mask) 알고리즘

ClearEvent(Mask)는 우선적으로 서비스를 호출한 태스크가 확장 태스크인지 여부를 검사하여 확장태스크가 아닐 경우에는 E_OS_ACCESS를 리턴한다. 다음으로 호출 레벨이 TASK_LEVEL인지 검사를 하여 아닐 경우 E_OS_CALLEVEL을 리턴하고 Task레벨일 경우에는 서비스를 호출한 태스크에 설정된 이벤트를 해제한다.

3.3 GetEvent(TaskID, RefEventMask)

GetEvent(TaskID, RefEventMask)는 TaskID에 해당하는 태스에 설정된 이벤트의 현재 상태를 얻어온다. 알고리즘은 (그림 6)과 같다.

```
GetEvent(TaskID, RefEventMask)
{
  mo_task_info_t    *current_task;
  if(IS_INVALID_TASK_ID(task_id))  return E_OS_ID;

  /*  TaskID의 태스크 얻기 */
  current_task = get_task_by_id(task_id);

  /* 확장 태스크 여부 검사 */
  if(current_task->extended!= EXTENDED_TASK)
    return E_OS_ACCESS;

  /*  태스크의 상태 여부 검사 */
  if(current_task->state == TASK_STATE_SUSPENDED)
    return E_OS_STATE;

  /*  현재 설정된 이벤트를 얻는다. */
  *event_mask = current_task->set_event_mask;
  return E_OK;
}
```

(그림 6) GetEvent(TaskID, RefEventMask) 알고리즘

GetEvent(TaskID, RefEventMask)는 우선적으로 TaskID의 유효 여부를 검사한다. TaskID가 유효하지 않으면 E_OS_ID를 리턴한다. 다음으로 TaskID를 이용하여 태스크 정보를 얻어온다. 태스크 정보에서 태스크가 확장태스크인지 여부를 검사한다. 확장태스크가 아닐 경우 E_OS_ACCESS 리턴한다. 다음으로 태스크의 현재 상태가 suspended일 경우 E_OS_STATE를 리턴한다. 마지막으로 RefEventMask에 태스크정보에서 설정된 이벤트 Mask값을 넘겨주고 E_OK를 리턴한다.

3.4 WaitEvent(Mask)

WaitEvent(Mask)는 서비스를 호출한 태스크를 대기상태로 전환한다. 알고리즘은 (그림 7)과 같다.

```
WaitEvent(Mask)
{
  /* 확장 태스크 여부 검사 */
  if(current_task->extended  != EXTENDED_TASK)
    return E_OS_ACCESS;

  /* 자원 사용 여부 검사*/
  if(current_task->using_resource != 0)
    return E_OS_RESOURCE;

  /* 호출 레벨 검사 */
  if(Level != TASK_LEVEL)   return E_OS_CALLEVEL;

  /* 이벤트 설정여부 검사 */
  if(( current_task->set_event_mask & event_mask) == 0)
   {
      delete_ready_queue(current_task);
      current_task->wait_event_mask = event_mask;
      current_task->state = TASK_STATE_WAITING;
      call_scheduler();
   }
   return E_OK;
}
```

(그림 7) WaitEvent(Mask) 알고리즘

WaitEvent(Mask)는 우선적으로 서비스를 호출한 태스크가 확장 태스크인지 여부를 검사하여 확장태스크가 아닐 경우에는 E_OS_ACCESS를 리턴한다. 다음으로 확장 태스크의 자원 사용여부를 판단한다. 자원이 현재 사용 중일 경우 E_OS_RESOURCE를 리턴한다. 호출 레벨을 체크한다. 현재 동작 레벨이 태스크 레벨인지 검사한다. 아닐 경우 E_OS_CALLEVEL을 리턴하고 태스크 레벨일 경우에는 기다릴 이벤트가 현재 설정중인지 검사한다. 설정 되지 않은 이벤트 이면 준비큐에서 태스크를 제거하고 태스크 상태를 대기상태로 전환후 마지막으로 스케줄러를 호출하여 태스크를 스위칭 한다.

4. 구현 및 실험

앞에서 제시한 서비스 알고리즘의 타당성을 확인하기 위해 (주)에프에이리눅스에서 제공하는 EZ-AT7 임베디드 보드[4]에서 <표 1>과 같은 태스크를 사용하여 실험하였다.

<표 1> 실험 태스크

태스크	기능	선점/비선점	우선순위	자동실행
T1	LED-1 ON/OFF	선점	1(Low)	Yes
T2	LED-2 ON/OFF	선점	2(High)	Yes

이벤트	Event1, Event2

T1과 T2를 활성화하여 동작을 하게 되면 T2의 높은 우선순위로 LED-2가 ON/OFF 된다. T2가 WaitEvent(Event1)를 호출한다. Event1를 기다리기 위해 T2는 대기상태로 전환한다. 현재 활성화된 태스크는 T1 밖에 없으므로 T1이 실행되고 LED-1이 ON/OFF된다. T1은 T2를 위하여 .SetEvent(T2,Event1)을 호출하여 이벤트를 설정한다. T2은 대기상태에서 준비상태로 전환된다. T2가 T1보다 우선순위가 높으므로 T2가 실행되고 GetEvent(T2, &EventMask)를 호출하여 Event1 값을 확인 할 수 있다. 다음으로 ClearEvent(Event1)을 호출하여 T2에 설정된 이벤트를 해제하고 GetEvent(T2, &EventMAsk)를 호출한다. 0x0값을 확인 할 수 있다. 이를 통해 올바르게 이벤트가 해제되었음을 확인 할 수 있었고, 결과적으로 4가지 서비스가 올바르게 동작함을 확인할 수 있었다.

5. 결론

본 논문에서 OSEK/VDX 운영체제에서 태스크간의 동기화를 위하여 SetEvent, ClearEvent, GetEvent, WaitEvent 서비스를 설계를 하고, 실험을 통하여 그 타당성을 검증하였다. 우선순위가 낮은 태스크를 위하여 WaitEvent를 호출해 대기를 하고 우선순위가 낮은 태스가 동작을 한다. 동작이 끝난 후, SetEvent를 호출하여 대기상태에 있는 태스크를 다시 실행하여 태스크간의 동기화 문제를 해결하였다.

향후과제는 서비스 구현중 미흡한 에러처리 및 정밀한 테스트 과정을 거쳐 서비스의 무결성을 검증하는데 있다.

참고문헌

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

[2] 유우석, 박지용, 홍성수 “분산형 실시간 차량제어 시스템을위한 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