

실시간시스템 활용 사례 조사 및 코드분석

컴퓨터공학과
5171515 권태희

목차

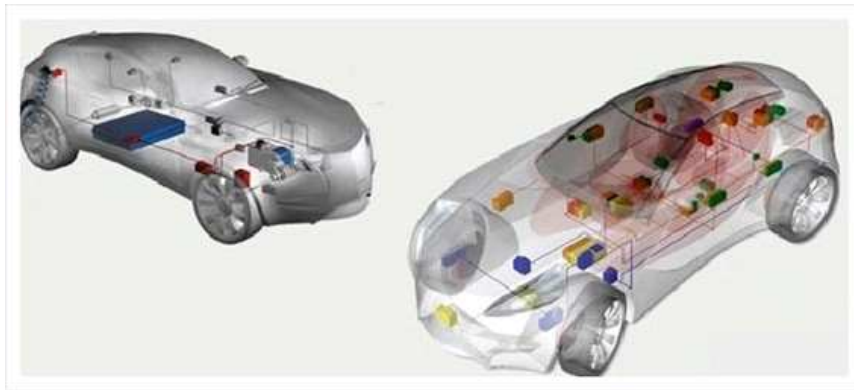
1. 주제 선정 이유
2. 주제 설명
 - 2-1. ECU 정의
 - 2-2. ECU 기능
 - 2-3. ECU 종류
3. OSEK/VDX OS
 - 3-1. OSEK 정의
 - 3-2. OSEK 특징
 - 3-3. OSEK 개요
4. 소스코드 분석
 - 4-1. Trampoline OS
 - 4-2. 주요 개념 및 API
5. 흐름도
6. Source code
 - 6-1. car_simulation.oil
 - 6-2. car_simulation.c
7. 참고

1. 주제 선정 이유

- 자동차에 사용되는 전자 장치의 증가
- 이를 제어하기 위한 소프트웨어의 양과 복잡도의 증가
- 자동차용 소프트웨어의 표준화의 필요성의 증가

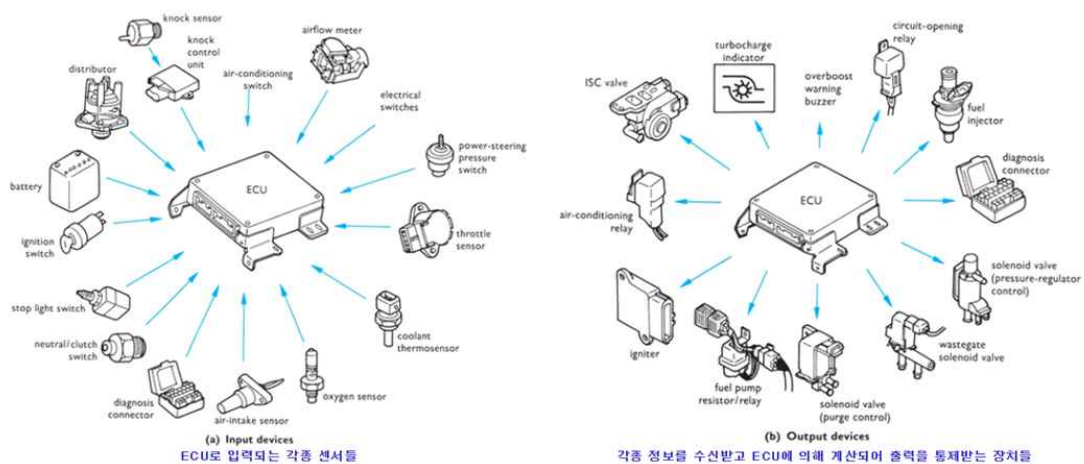
2. 주제 설명

2-1. ECU 정의



- ECU(Electronic Control Unit)는 자동차의 엔진, 자동변속기, ABS와 같은 차량의 전자 시스템을 제어하는 자동차 전자 장치의 모든 내장 시스템. 컴퓨터와 비교했을 때 CPU와 유사하다. Electronic Control Unit, Engine Control Unit, Powertrain Control Unit 등 여러 이름을 가지고 있다. 일부 자동차에는 최대 80 개의 ECU가 있으며, 각 ECU는 그 목적된 기능에 따라 하나의 자동차 안에 분산되어 위치해 있다.

2-2. ECU 기능



- 애초의 개발 목적은 당시에는 점화시기와 연료분사, 공회전, 한계값 설정 등 엔진의 핵심 기능을 정밀하게 제어하는 것이었다. 그러나 차량과 컴퓨터 성능의 발전과 함께 자동변속기 제어를 비롯해 구동계통, 제동계통, 조향계통 등 차량의 모든 부분을 제어하는 역할까지 하고 있다.

2-3. ECU 종류

명칭	. Full name . 주요 기능 * 자동차 제작사에 따라 그 명칭이 상이할 수 있음
ACU	. Airbag Control Unit . 자동차 충돌 상황의 센서 신호를 받아 에어백을 제어하는 ECU
BCM	. Body Control Module . 자동차의 각종 경고, 도난방지 기능 등을 제어하여 자동차 상태를 사용자에게 알려주며 사용자 요구에 맞게 자동차를 제어함
ECU	. Engine Control Unit . 엔진의 상태를 센서를 통해 모니터링하면서 연료의 양, 점화시기 등의 제어 값을 결정하는 ECU
PCM	. Powertrain Control Module . 엔진과 파워트레인(기관에서 발생된 동력을 구동바퀴까지 전달하는 일련의 모든 장치)의 각종 센서로부터 정보를 받아 각종 제어신호를 출력하는 장치
TCU	. Transmission Control Unit . 자동차의 속도, 바퀴 속도, 스로틀 위치 등의 센서값을 받아 운전자가 원하는 주행조건이 되도록 변속기를 제어해주는 ECU
ABS	. Anti-lock Braking System . 자동차의 Lock-up 현상(자동차는 진행 중인데 바퀴가 잠겨있는 현상)을 감지하여 자동차가 미끄러지지 않도록 브레이크의 on/off 상황을 짧은 시간 반복 제어하여 운전자가 원하는 제동을 도와주는 ECU
ESC	. Electronic Stability Control . ABS 등의 주행 및 제동 조건을 제어하는 ECU의 기능을 포함하기도 하며 필요한 바퀴를 선택적으로 제어하여 자동차의 안정된 자세를 유지하게 도와주는 ECU
HPCU	. Hybrid Power Control Unit . 하이브리드 자동차의 경우 내연기관과 전기동력장치를 주행상황에 맞게 상호연동하는 ECU
BMS	. Battery Management System . 구동을 목적으로 하는 배터리의 전압, 온도등의 상태를 모니터링 및 제어하며 배터리의 충전량 등을 구동 제어 ECU에 전송하는 ECU
MCU	. Motor Control Unit . 전기모터의 회전수, 토크, 냉각수 온도 등을 모니터링 및 제어하는 ECU

- ECU의 임베디드 소프트웨어는 라인 수, 복잡성 및 정교함이 계속 증가하고 있다. 차량의 복잡성과 수의 ECU를 관리하는 것은 OEM (Original Equipment Manufacturer)의 핵심 과제가 되었다.

3. OSEK/VDX

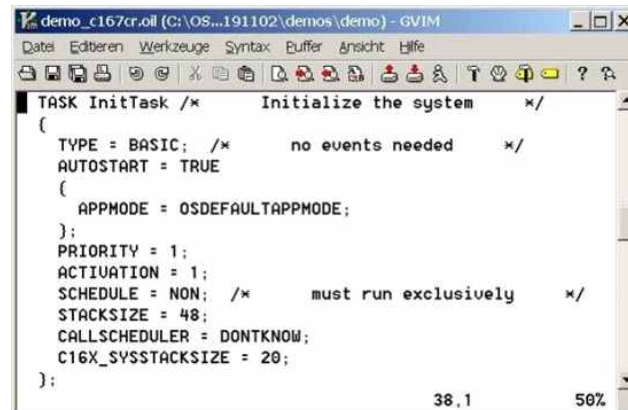
3-1. OSEK/VDX 정의

- 자동차 ECU 응용 소프트웨어 제어, 자원 관리 및 최적화를 지원하는 차량용 실시간 운영체제 표준 규격

3-2. 특징

- OSEK은 Static Operating System이다. 사전에 Task 갯수, Task 우선순위, 스케줄링 방법, 스택 크기 등 OS 리소스 관리에 필요한 모든 정보를 개발자가 정해줘야 한다.

- OIL은 OSEK Implementation Language의 약어로서, OSEK을 구현하기 위한 언어이며, OS 리소스 관리에 필요한 객체들을 OIL언어로 미리 정의한다. 이 OIL을 이용하면, C언어로 되어 있는 소스 코드를 직접 수정하지 않고도, OSEK을 재구성 할 수 있다.



```
demo_c167cr.oil (C:\OS...191102\demos\demo) - GVIM
Datei Editieren Werkzeuge Syntax Buffer Ansicht Hilfe
TASK InitTask /* Initialize the system */
{
    TYPE = BASIC; /* no events needed */
    AUTOSTART = TRUE
    {
        APPMODE = OSDEFAULTAPPMODE;
    };
    PRIORITY = 1;
    ACTIVATION = 1;
    SCHEDULE = NON; /* must run exclusively */
    STACKSIZE = 48;
    CALLSCHEDULER = DONTKNOW;
    C16X_SYSSTACKSIZE = 20;
};
38,1 50%
```

- OSEK은 최소의 기능만을 가지고 있는 아주 작은 OS 이다. 경우에 따라 다르겠지만, OSEK은 1KB 미만까지도 빌드가 가능하다.

3-3 OSEK 개요

○ 운영체제 구조

- OSEK의 3가지 Processing levels

- * Interrupt level
- * Logical level for scheduler
- * Task level

- Task level에서 Task는 사용자가 지정한 우선순위에 따라 비선점, 선점, 혼합선점으로 스케줄링 된다. 실행 문맥(runtime context)은 실행 시간의 시작을 차지하거나 Task가 종료되면 해제된다.

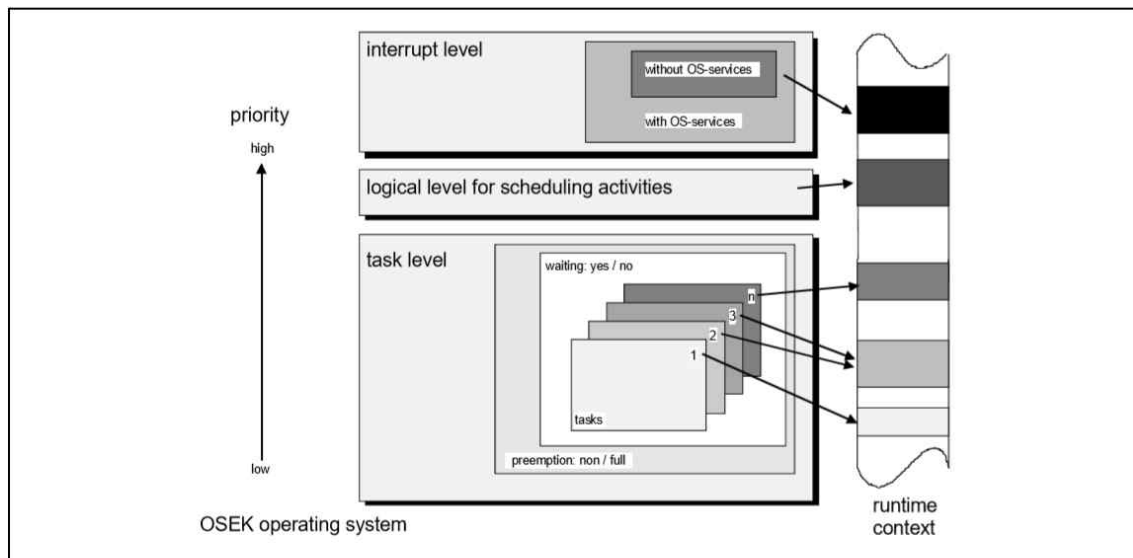


Figure 3-1 Processing levels of the OSEK operating system

- 우선순위 규칙(Priority Rule)

- * Interrupt가 Task보다 우선순위가 높다.
- * ISR(Interrupt Service Routine)은 고정적으로 부여된 Interrupt 우선순위가 있다.
- * ISR에 Interrupt 우선순위를 부여하는 것은 수행 및 하드웨어 구조에 의존적이다.
 - Task의 우선순위는 숫자가 클수록 우선순위가 높다.
 - Task의 우선순위는 사용자에게 의해 고정적으로 할당된다.

○ Task 관리

- OSEK에서는 두 가지 타입의 Task 상태에 대한 model을 제공하며, 각 Task별로 둘 중 하나를 선택하여 사용한다.

- Basic Task

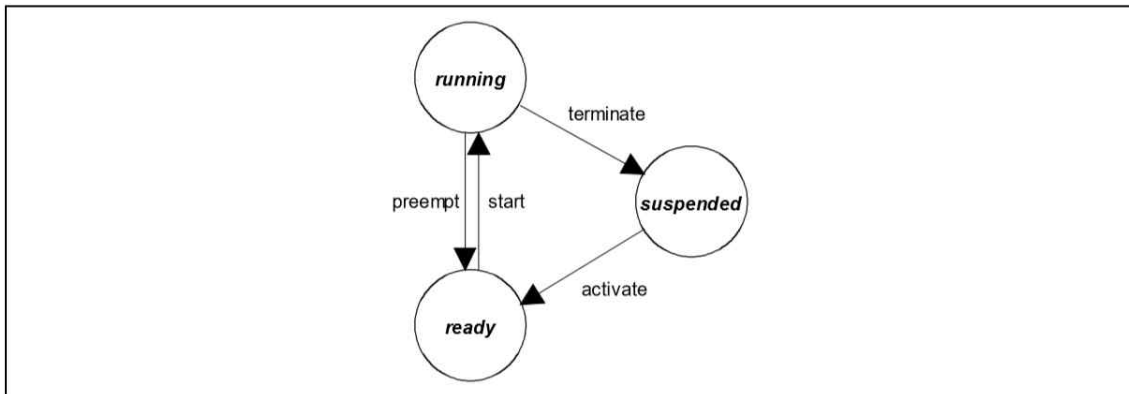


Figure 4-3 Basic task state model

- * 불필요한 자원사용을 방지하기 위해 waiting state가 없다.
- * 다음의 경우 processor를 release한다.
 - 스스로 종료되었을 경우
 - OSEK OS가 높은 우선순위 Task로 전환될 때
 - processor를 ISR로 바꾸는 Interrupt가 일어날 때
 (인터럽트가 일어나 프로세서가 인터럽트 서비스 루틴으로 전환 되었을 경우)

Transition	Former state	New state	Description
activate	<i>suspended</i>	<i>ready</i> ⁴	A new task is set into the <i>ready</i> state by a system service. The OSEK operating system ensures that the execution of the task will start with the first instruction.
start	<i>ready</i>	<i>running</i>	A <i>ready</i> task selected by the scheduler is executed.
preempt	<i>running</i>	<i>ready</i>	The scheduler decides to start another task. The <i>running</i> task is put into the <i>ready</i> state.
terminate	<i>running</i>	<i>suspended</i>	The <i>running</i> task causes its transition into the <i>suspended</i> state by a system service.

Figure 4-4 States and status transitions for basic tasks

* State

- **running** : CPU가 Task에 할당되어 명령을 수행하며,
한 번에 하나의 Task만이 이 상태에 있을 수 있다.
- **ready** : running 상태에 들어가기 위해 우선 거쳐야 하는 state이다.
scheduler는 ready state에 있는 Task 중에서 실행할 Task를 결정한다.

- *suspended* : 휴식상태이며 Scheduler에 의해 다시 활성화 될 수 있다.

* State transition

- *start* : *ready* state에서 scheduler가 Task를 선택하였을 때 실제 동작상태로 전환
- *preempt* : scheduler가 다른 Task를 시작하기로 결정하여 *running* 에서 *ready*로 전환
- *activate* : system service에 의해 Task가 실행 준비되었을 때 발생
- *terminate* : Task가 종료되거나 에러로 인하여 Task가 강제로 종료될 때 발생

- Extended Task

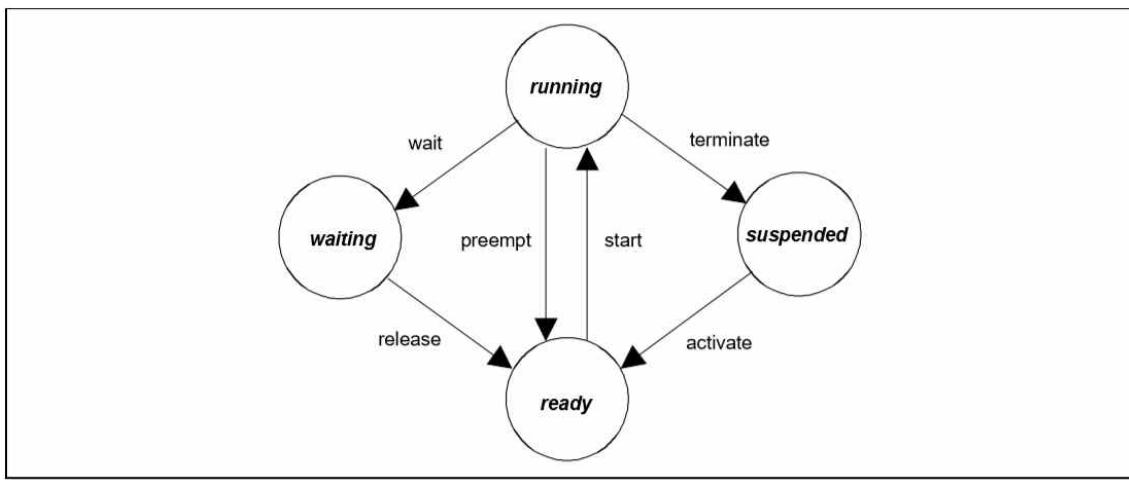


Figure 4-1 Extended task state model

* wait event가 있다.

- *waiting* state는 processor가 release되도록 하고 실행중인 *running* extended Task를 terminate하지 않고 더 낮은 우선순위로 재할당한다.

Transition	Former state	New state	Description
activate	<i>suspended</i>	<i>ready</i>	A new task is set into the <i>ready</i> state by a system service. The OSEK operating system ensures that the execution of the task will start with the first instruction.
start	<i>ready</i>	<i>running</i>	A <i>ready</i> task selected by the scheduler is executed.
wait	<i>running</i>	<i>waiting</i>	The transition into the waiting state is caused by a system service. To be able to continue operation, the <i>waiting</i> task requires an event.
release	<i>waiting</i>	<i>ready</i>	At least one event has occurred which a task has <i>waited</i> for.
preempt	<i>running</i>	<i>ready</i>	The scheduler decides to start another task. The <i>running</i> task is put into the <i>ready</i> state.
terminate	<i>running</i>	<i>suspended</i>	The <i>running</i> task causes its transition into the <i>suspended</i> state by a system service.

Figure 4-2 States and status transitions for extended tasks

* State

- Basic Tasks의 state를 포함 (*running*, *suspended*, *ready*)
- *waiting* : 특정 event가 발생하기 전까지 동작하지 않는 상태

* State transition

- Basic Task의 State transition를 포함(start, preempt, terminate, activate)
- wait : system service에 의해 발생하며, 동작을 이어가려면 event가 발생해야한다(Context 저장이 필요).
- release : waiting state에 있는 Task에 event가 발생하였을 때 Ready로 전이된다.

* Activating a Task

- Task activation은 'ActivateTask'나 'ChainTask'를 이용해서 수행
- 'Parallel implementation and Multiple requesting of Task activation'이 가능

* Task priority

- Task priority는 고정적으로 부여되며 실행 중에 변화시킬 수 없음
- 같은 우선순위를 가진 Task는 먼저 들어온 순서대로 시작된다.

* OS에서 복잡도를 줄이기 위한 제한 항목

- Task의 종료는 Task 자신만이 terminate시킬 수 있다.
- Suspended state에서 waiting state로 바로 갈 수 없다.

* Scheduler (order of event)

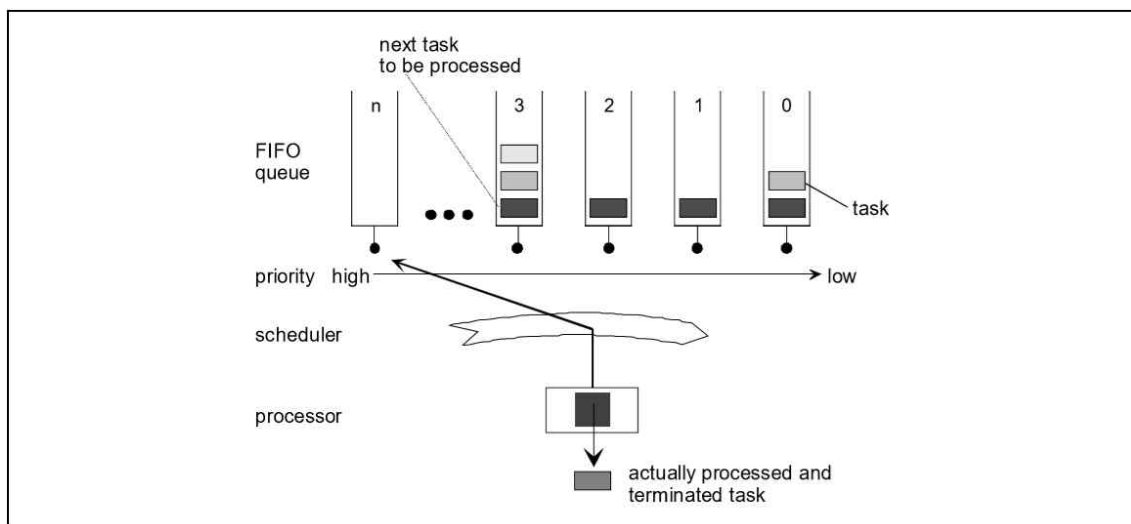


Figure 4-5 Scheduler: order of events

- 우선순위 기반의 scheduling, 숫자가 높을수록 높은 우선순위이다.
- 우선순위별로 큐가 있으며 큐에 들어온 순서대로 Task가 실행된다.
- 스케줄링 과정
 - ① ready/running state 상의 모든 Task 검색
 - ② Highest priority를 가진 Task 집합 결정

- ③ 그 중에서 가장 먼저 들어온 Task 찾기
- ④ 찾아낸 Task를 running 상태로 전환

* Scheduling policy

· Full preemptive scheduling

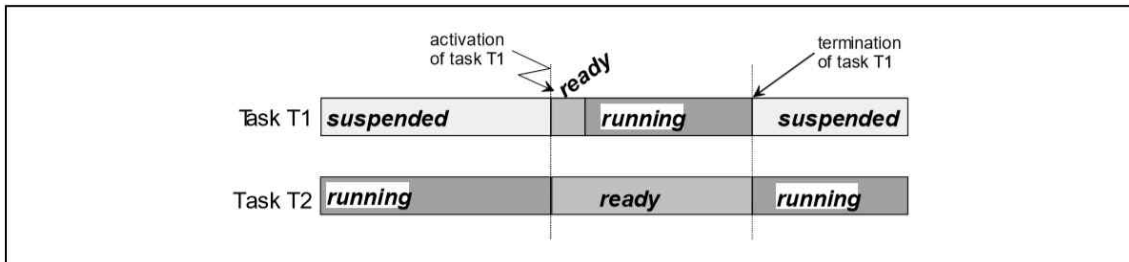


Figure 4-6 Full preemptive scheduling

Task가 Activate 되었을 경우 다시 Scheduling하여 더 높은 우선순위를 가진 Task가 먼저 수행된다. 위의 그림에서는 Task T2가 실행되는 도중에 높은 우선순위를 가지는 T1이 선점하여 Task를 먼저 수행한다. 더 높은 우선순위의 Task가 'ready'가 되자마자 'running' Task를 'ready' Task로 전환한다.

· Non Preemptive scheduling

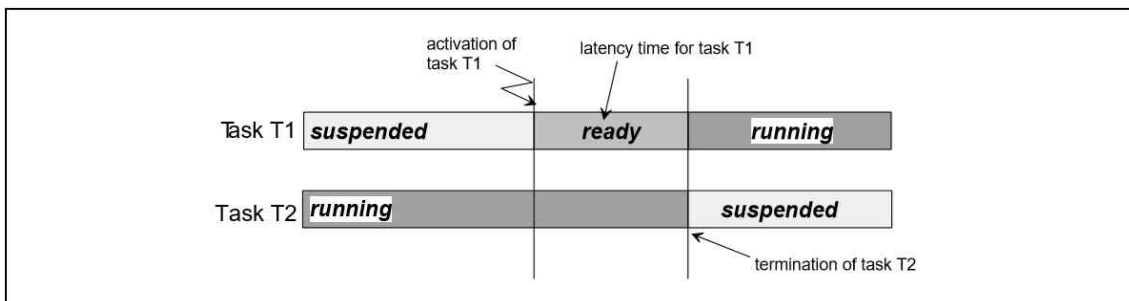


Figure 4-7 Non preemptive scheduling

Task가 Activate되더라도 Scheduling이 일어나지 않고 현재의 Task를 끝까지 수행한다. 우선 순위가 높아도 Task T2가 종료 될 때까지 ready 상태에 머무르게 된다.

○ 인터럽트 처리

- Interrupt 정의

- * 주변 장치에서 발생한 상태 변화
- * e.g. CAN signal receive, Timer expired

- Interrupt 특징

- * Interrupt는 Task들을 선점하며 Context switching이 일어난다.
- * Interrupt는 Category1과 Category2로 분류
- * 우선순위 : Interrupt level > Logical level for scheduler > Task level
- * ISR(Interrupt Service Routine: Interrupt 처리를 위한 기능) 중에는 rescheduling이 발생하지 않는다.

- OSEK OS는 두 가지 Type(Category 1, Category 2)에 따라 인터럽트 처리 기능 제공한다.

* ISR category 1

- ISR(Interrupt Service Routine)에서 OS 서비스를 사용하지 않는 인터럽트로, ISR 구현 시 사용자가 ISR frame까지 구현해야한다.
- ISR이 끝난 후 직전에 처리 중이던 구문을 계속 처리한다.
- ISR이 최소한의 오버헤드를 가진다.

* ISR category 2

- ISR에서 API를 호출하여 OS 서비스를 사용할 수 있는 인터럽트로, OS가 지정된 사용자 routine를 위해 런타임 환경을 제공하는 ISR frame를 제공한다.
- system generation 중에 사용자 루틴이 Interrupt에 할당된다.

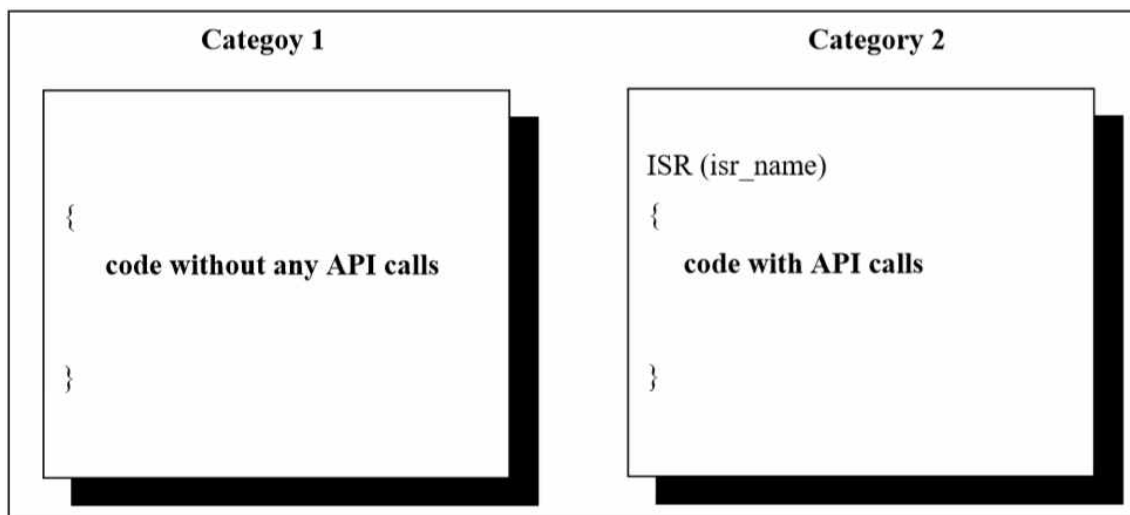


Figure 6-1 ISR categories of the OSEK operating system

○ Resource 관리

- 다른 우선순위를 가진 여러 Task가 scheduler, program sequences, memory, hardware 등의 공유된 Resource에 동시에 접속할 수 있도록 공유

- 주의사항

- * 두 개의 Task와 ISR은 동시에 같은 Resource를 차지할 수 없다.
- * Priority inversion(도치,역전)은 일어날 수 없다.
- * 이런 Resource들의 사용에 의한 deadlock은 일어나지 않는다.
- * waiting state에서는 Resource에 접근할 수 없다.

- General problems with synchronization mechanisms(동기화 메커니즘의 일반적인 문제)

* Explanation of priority inversion(우선순위 역전의 설명)

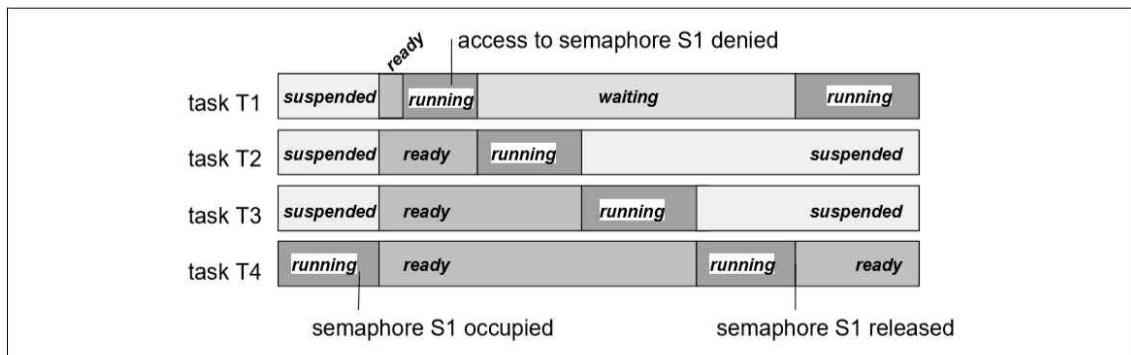


Figure 8-1 Priority inversion on occupying semaphores

· Task T1이 4개의 Task들 중에 가장 우선순위가 높으며 semaphore S1을 획득하려하나 Task T4가 이미 차지하고 있어서 거부당한다. 그리고나서 스케줄러에 의해 자신보다 우선순위가 낮은 T2, T3, T4가 먼저 실행되며 자신은 우선순위가 높음에도 불구하고 역전이 일어나 엄청나게 긴 대기시간이 지나서야 자신이 실행된다. 이러한 우선순위 역전을 피하기 위해서 OSEK는 OSEK Priority Ceiling Protocol을 사용한다.

- Deadlocks

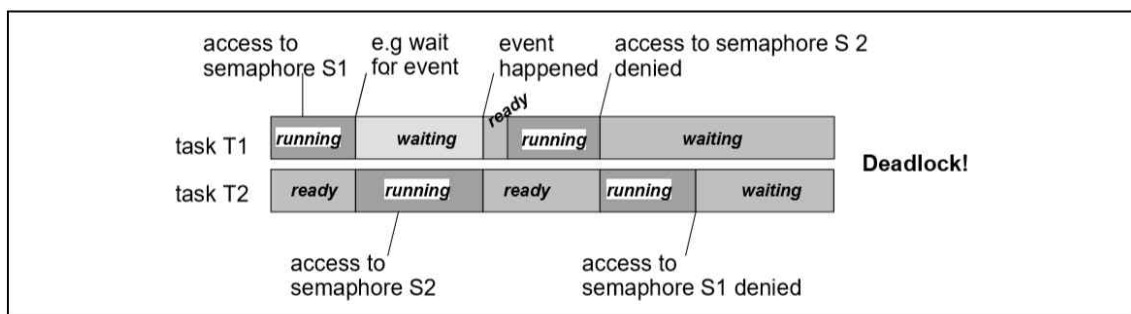


Figure 8-2 Deadlock situation using semaphores

* T1은 T2가 가지고 있는 S2에 접근하려 하고, T2는 T1이 가지고 있는 S1에 접근하려고 하여 무한정 waiting 상태에 빠지게 된다. 이를 해소하기 위해 역시나 OSEK Priority Ceiling Protocol을 사용한다.

- OSEK Priority Ceiling Protocol(우선순위 상한 프로토콜)

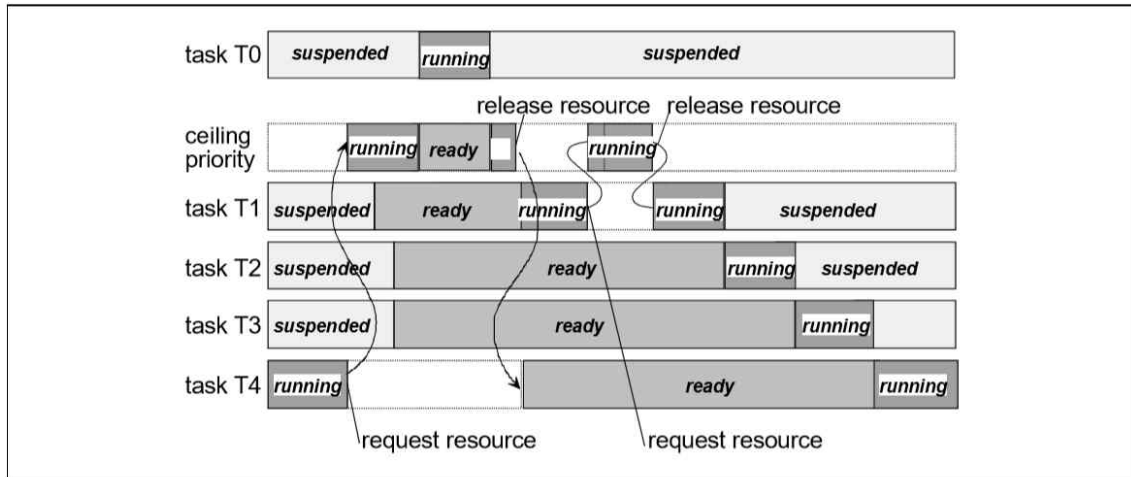


Figure 8-3 Resource assignment with priority ceiling between preemptable tasks.

* 위에서 소개한 우선순위 역전과 교착 상태를 해소하기 위해 OSEK/VDX OS에서 지원하는 프로토콜

· 자신(Task)의 우선순위보다 Resource의 우선순위가 더 높을 때, 자신의 우선순위를 일시적으로 높인다. Resource를 얻는 순간에 Resource에 지정된 우선순위로 Task의 우선순위가 상승하며 Resource를 release 하는 순간에 Task는 원래의 우선순위로 돌아간다.

- Internal Resources

* User에게 보이지 않는 Resource

* System Function인 GetResource와 ReleaseResource 에 의해 호출이 불가능하다.

* System generation 동안에 하나의 Internal Resource가 호출 될 수 있다.

· Resource는 running Task로 진입할 때 자동적으로 획득한다(이미 취했을 경우 제외), 그 결과 자동적으로 Task의 priority는 자동적으로 Resource의 ceiling priority로 변한다. 또한 자동적으로 release된다.

○ 알람

- 지정된 시간(alarm expired)에 지정된 기능을 수행(일회성 혹은 주기적) 하도록 하는 서비스이다. 예를 들어, 주기적으로 Task를 activation할 수 있다.

- Counter가 미리 정의된 값에 도달하였을 경우 expired 되어 지정된 동작을 반복적으로 수행할 수 있도록 하는 OS의 객체이다.

- 주요 용도

* Call Alarm Callback

* Activate Task : OS 내 구성된 Task를 주기적으로 실행시키기 위하여 사용할 수 있다.

* Set Event

- Counter

* Counter는 소스(e.g. Timer)의 값을 TICK 단위의 상수 값으로 변경시킨다(e.g. 1ms -> 1tick).

* OSEK OS 는 S/W 혹은 H/W Timer 와 연결된, 적어도 하나의 Counter를 제공한다.

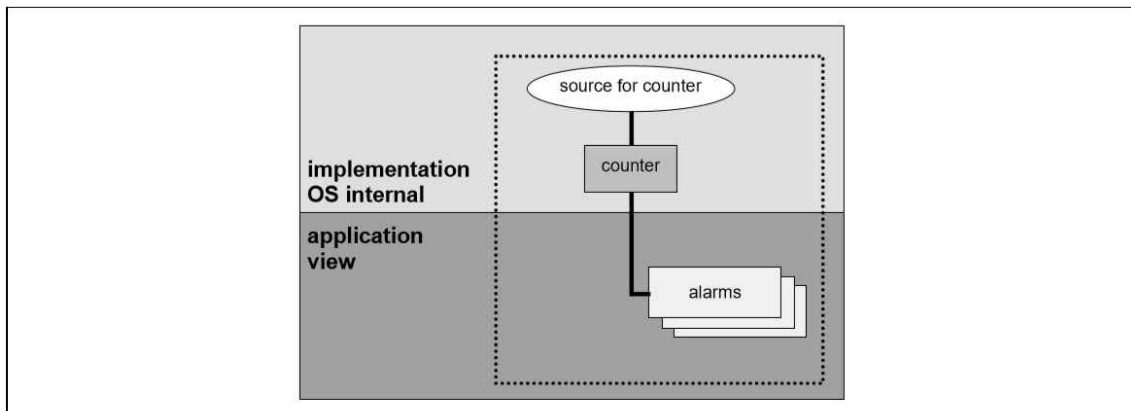


Figure 9-1 Layered model of alarm management

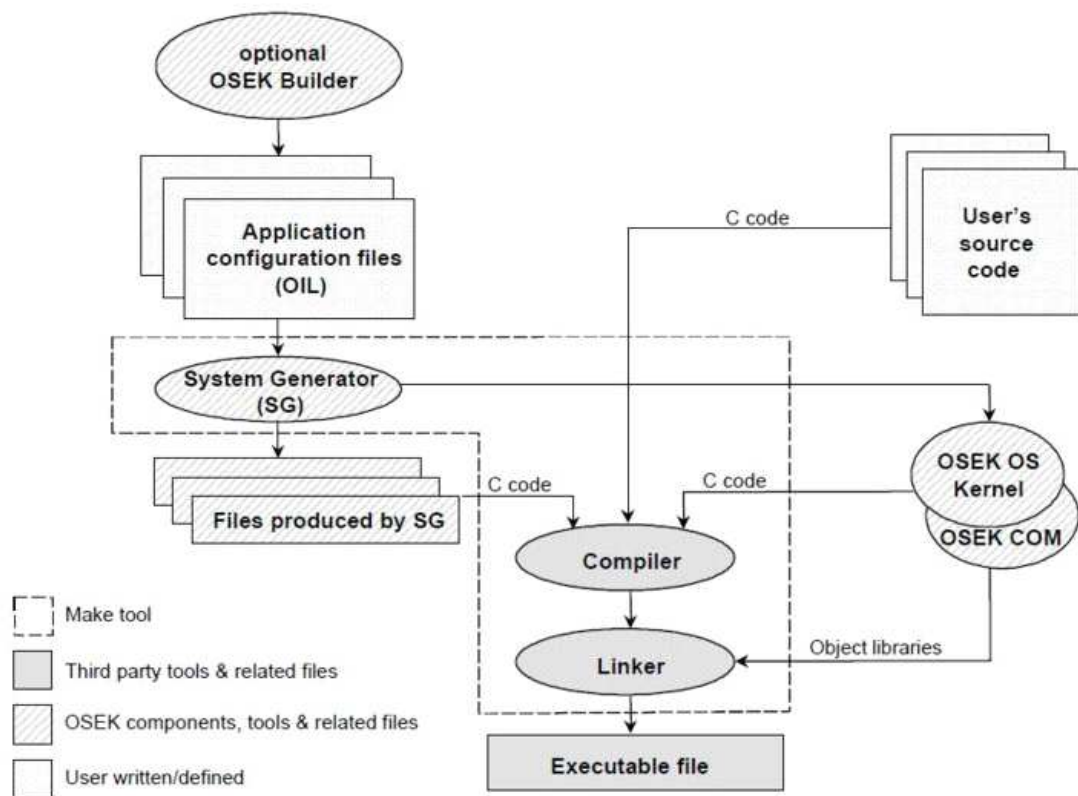
4. 소스코드 분석

4-1. Trampoline OS

○ 특징

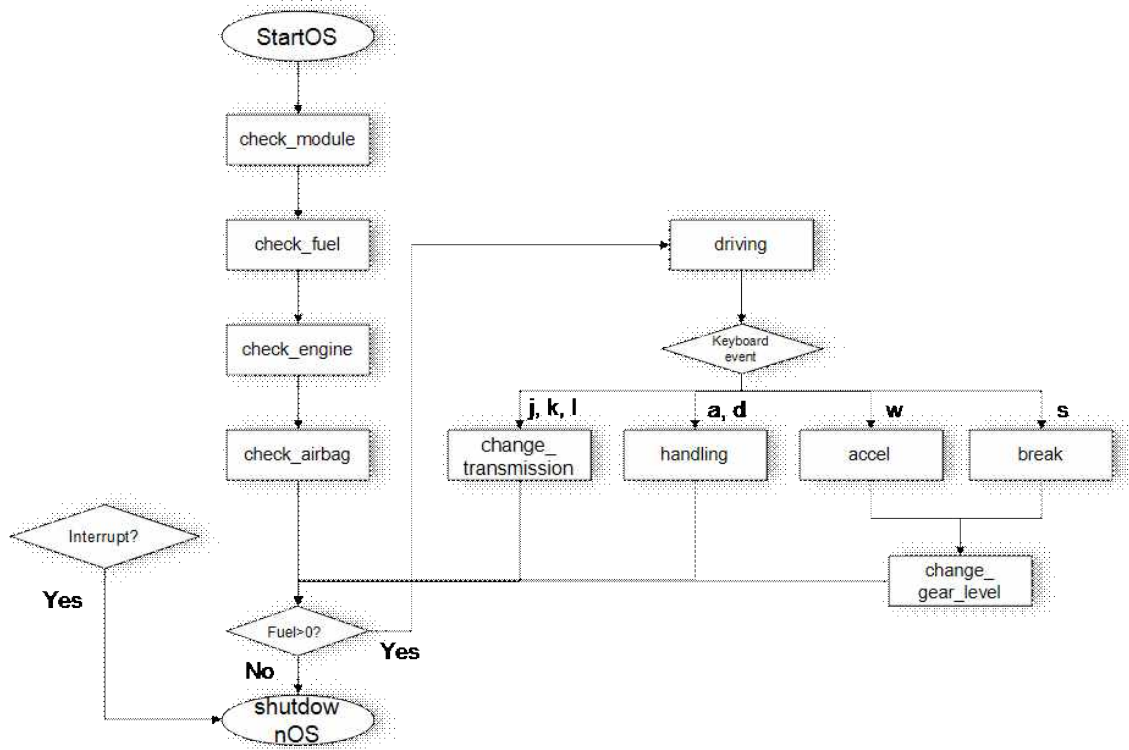
- 자동차 ECU 응용 소프트웨어 제어, 자원 관리 및 최적화를 지원하는 차량용 실시간 운영체제인 OSEK/VDX OS 규격을 지원하는 RTOS
- POSIX, Arduino, Raspberry pi 2, Teensy board, ARM 기반 보드 등 다양한 플랫폼 지원
- blink, periodic 등의 간단한 예제 제공

- Trampoline OS Application build 순서



* OS 리소스에 필요한 정보를 담은 OIL 파일을 goil이라는 컴파일 툴을 사용하여 C언어로 변환된다. 그리고 이를 사용자가 작성한 c언어 파일을 컴파일과 링크과정을 거쳐 실행가능한 파일을 생성한다.

5. 흐름도



- StartOS

- * OS가 oil 파일에 설정된 Task 및 Alarm을 자동 시작한다.
- * 순서
 - StartOS Call
 - OS에서 초기화 코드 실행
 - StartupHook routine을 제공하여 사용자가 device driver 등을 초기화
 - OS 커널 실행
 - 가장 처음 순서 사용자 Task 실행

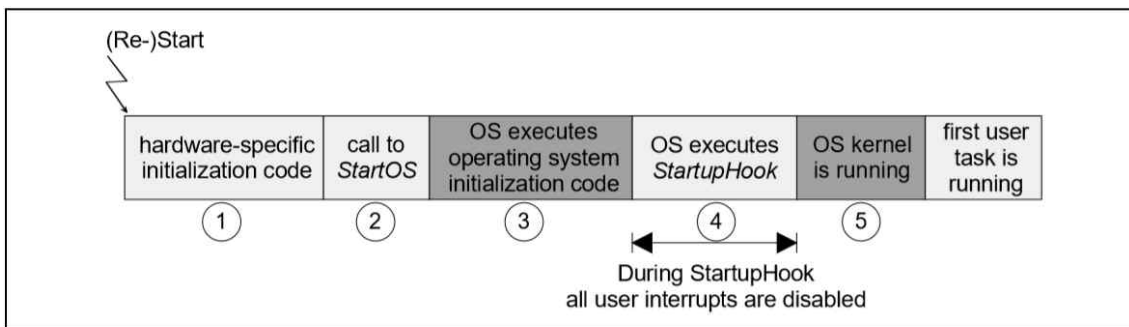


Figure 11-2 System start-up

- Task 목록

- * check_module: 자동차 내부 모듈 점검 및 초기화 작업 수행
- * check_fuel: 남은 연료 확인

- * check_engine: 자동차 엔진 점검
 - * check_airbag: 자동차 에어백 점검
 - * driving: 주행 시작, 터미널 내의 입력키에 따른 자동차 주행 작업 수행
 - * change_transmission: 자동차 변속기 제어
 - * handling: 자동차 핸들 제어
 - * accel: 자동차 액셀 제어
 - * break: 자동차 브레이크 제어
 - * change_gear_level: 자동차 기어 단수 제어
 - * end_car: 자동차 시동 꺼짐
-
- check_module, fuel, engine, airbag
 - * 자동차의 각종 모듈 점검, 시뮬레이션 코드에서는 연료 체크, 엔진 점검, 에어백 점검만 다루고 있지만 실제 자동차는 훨씬 더 많은 모듈들을 점검함

 - driving
 - * 연료전부 연소될 때까지 driving Task를 실행, POSIX 환경에서 시뮬레이션하므로 터미널 상에서 키보드 입력에 따라 기어 변속, 핸들링, 액셀, 브레이크 Task를 실행함

 - ShutdownOS
 - * Fatal error 발생시 application이나 OS에서 Shutdown을 요청(ShutdownOS 호출)
 - * ShutdownOS 수행시 ShutdownHook를 호출하고 이 Hook이 종료되면 shutdown 진행

6. Source code

6-1. car_simulation.oil

```
car_simulation.oil x C car_simulation.c
1 // goil Tool을 사용하여 oil파일 컴파일 필요
2 // usage: goil --target-posix --templates=../../goil/templates/ car_simulation.oil
3
4 /*
5 ** OIL 버전 설정
6 ** 2.5: OSEK/VDX Application
7 ** 3.1: AUTOSAR 3.1 Application
8 ** 4.0: AUTOSAR 4.0 Application
9 */
10 OIL_VERSION = "2.5";
11
12 /*
13 ** IMPLEMENTATION 파트: application 속성 지정
14 ** 주로 TASK나 ISR의 스택 사이즈 정적으로 결정
15 */
16 IMPLEMENTATION trampoline {
17
18     /* This fix the default STACKSIZE of tasks */
19     TASK {
20         | UINT32 STACKSIZE = 32768 ;
21     } ;
22
23     /* This fix the default STACKSIZE of ISRs */
24     ISR {
25         | UINT32 STACKSIZE = 32768 ;
26     } ;
27 };
28
29 /*
30 ** CPU 파트: application의 각각의 객체에 대한 속성 지정
31 ** OS의 속성, TASK, ISR, RESOURCE,
32 ** ALARM, COUNTER, APPMODE, EVENT, MESSAGE 속성 정적으로 지정 필요
33 */
34 CPU car_simulation {
35     OS config {
36         /*
37         ** 확장 상태 사용(Default: STANDARD), 상태에 따른 오류 검사 반환값 다름
38         ** 작업이 표준 상태에서 활성화되었을 시, "E_OK" 혹은 "Too many Task activations" 에러 반환 가능
39         ** 작업이 확장 상태에서 활성화되었을 시, "invalid Task" 혹은 작업이 "Task still occupies Resources" 에러 반환 가능
40         ** 확장 상태에서는 OSEK application에서 에러 반환하지 않음
41         */
42         STATUS = EXTENDED;
43         BUILD = TRUE {
44             APP_SRC = "car_simulation.c"; // application SOURCE
45             TRAMPOLINE_BASE_PATH = "../../"; // trampoline base 경로
46             APP_NAME = "car_simulation_exe"; // 컴파일 시 생성되는 실행가능 파일
47             LINKER = "gcc";
48             SYSTEM = PYTHON;
49         };
50     };
51
52     APPMODE stdAppmode {}; // 적어도 하나의 어플리케이션 모드가 CPU 파트 내에 정의되어 있어야됨
53
54     ALARM alarm_end_car {
55         COUNTER = SystemCounter; // Trampoline default counter
56         ACTION = ACTIVATETASK { TASK = task_end_car; }; // Alarm 타이머 만료될 때의 동작과 내용 정의, 만료시 task_end_car 수행
57         AUTOSTART = TRUE { // Alarm 자동시작 여부
58             APPMODE = stdAppmode; // 어플리케이션 모드 지정
59             ALARMTIME = 100; // 처음 알람 만료시간 지정
60             CYCLETIME = 0; // 처음 알람 만료후 순환 주기, 0이면 한번만 알람 만료됨
61         };
62     };
63 }
```

```

64 TASK task_start_car {
65     PRIORITY = 2;                                // Task 우선순위, 숫자가 높을수록 높은 우선순위
66     AUTOSTART = TRUE {                            // Task 자동시작 여부
67         APPMODE = stdAppmode;                    // Task가 실행될 어플리케이션 모드 지정
68     };
69     ACTIVATION = 1;                                // 동시에 실행될 수 있는 Task 개수
70     SCHEDULE = FULL;                              // 스케줄링 모드 - FULL: preemptive, NON: non-preemptive
71 };
72
73 TASK task_check_module {
74     PRIORITY = 2;
75     AUTOSTART = FALSE;
76     ACTIVATION = 1;
77     SCHEDULE = FULL;
78     RESOURCE = module_initialize;
79 };
80
81 TASK task_check_fuel {
82     PRIORITY = 3;
83     AUTOSTART = FALSE;
84     ACTIVATION = 1;
85     SCHEDULE = FULL;
86 };
87
88 TASK task_check_engine {
89     PRIORITY = 3;
90     AUTOSTART = FALSE;
91     ACTIVATION = 1;
92     SCHEDULE = FULL;
93 };
94
95 TASK task_check_airbag {
96     PRIORITY = 3;
97     AUTOSTART = FALSE;
98     ACTIVATION = 1;
99     SCHEDULE = FULL;
100 };
101
102 TASK task_driving {
103     PRIORITY = 2;
104     AUTOSTART = FALSE;
105     ACTIVATION = 1;
106     SCHEDULE = FULL;
107 };
108
109 TASK task_change_transmission {
110     PRIORITY = 3;
111     AUTOSTART = FALSE;
112     ACTIVATION = 1;
113     SCHEDULE = FULL;
114 };
115
116 TASK task_handling {
117     PRIORITY = 3;
118     AUTOSTART = FALSE;
119     ACTIVATION = 1;
120     SCHEDULE = FULL;
121 };
122
123 TASK task_accel {
124     PRIORITY = 3;
125     AUTOSTART = FALSE;
126     ACTIVATION = 1;
127     SCHEDULE = FULL;
128 };
129

```

```

130 TASK task_break {
131     PRIORITY = 3;
132     AUTOSTART = FALSE;
133     ACTIVATION = 1;
134     SCHEDULE = FULL;
135 };
136
137 TASK task_change_gear_level {
138     PRIORITY = 4;
139     AUTOSTART = FALSE;
140     ACTIVATION = 1;
141     SCHEDULE = FULL;
142 };
143
144 TASK task_end_car {
145     PRIORITY = 2;
146     AUTOSTART = FALSE;
147     ACTIVATION = 1;
148     SCHEDULE = FULL;
149 };
150
151 ISR isr_airbag {    // 인터럽트 서비스 루틴(Interrupt Service Routine) 정의
152     /*
153     ** ISR 카테고리 지정
154     ** category 1: OS 시스템 서비스를 사용할 수 없음
155     **             사용자가 인터럽트 전후처리를 직접 해줘야하며 약간의 오버헤드 발생
156     ** category 2: OSEK OS 시스템에서 ISR frame 제공 및 전용 ISR API 사용 가능
157     */
158     CATEGORY = 2;
159     PRIORITY = 10;    // ISR 우선순위, 숫자가 높을수록 높은 우선순위
160     /*
161     ** 인터럽트 발생 트리거 지정
162     ** 실행환경이 리눅스 기반이므로, POSIX keyboard interrupt signal 지정
163     ** 지정한 인터럽트 발생시 ISR 호출됨
164     ****
165     ** POSIX 환경에서 사용가능 signal
166     **
167     ** SIGPIPE: 신호를 읽는 사용자가 없는 상태에서 파이프에 기록
168     ** SIGQUIT: Ctrl+\(backslash), 프로세스를 종료시킨 뒤 코어 덤프
169     ** SIGTERM: 종료 신호
170     ** SIGTRAP: 트레이스/브레이크포인트 트랩
171     ** SIGUSR2: 사용자 정의 신호 2.
172     */
173     SOURCE = SIGQUIT;
174     RESOURCE = res_airbag;
175 };
176
177 RESOURCE module_initialize {    // Resource 정의
178     /*
179     ** Resource 특성 지정
180     ** STANDARD: Resource 관련 API 사용 가능
181     ** INTERNAL: Resource 관련 API 사용 불가, Task가 Running 상태에 진입할 때마다
182     **             OSEK Priority Ceiling Protocol에 의해 자동으로 internal resource 획득
183     */
184     RESOURCEPROPERTY = STANDARD;
185 };
186
187 RESOURCE res_speed {
188     RESOURCEPROPERTY = STANDARD; // enable get&release resource
189 };
190
191 RESOURCE res_airbag {
192     RESOURCEPROPERTY = INTERNAL; // disable get&release resource
193 };
194 };
195

```

6-2. car_simulation.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <fcntl.h>
5  #include <errno.h>
6  #include <linux/input.h>
7  #include <string.h>
8  #include "tpl_os.h"
9
10 /*
11  ** POSIX 키 입력 이벤트 구조체
12  ** 키가 입력되면 다음 구조체에 이벤트 관련 정보 저장됨
13  **
14  ** struct input_event {
15  **     struct timeval    time;        // 입력 이벤트가 발생한 시간에 대한 스탬프
16  **     __u16             type;        // 키 이벤트 형식 - 0: key released, 1: key pressed, 2: key repeated
17  **     __u16             code;        // 키 이벤트 코드
18  **     __s32             value;       // 키 이벤트 값
19  ** }
20  */
21 struct input_event ev;
22
23 // c언어 논리자료형 구현
24 typedef enum
25 {
26     false,
27     true
28 } bool;
29
30 /*
31  ** 변속기 상태 열거형
32  ** GEAR_PARKING: 주차
33  ** GEAR_REVERSE: 후진
34  ** GEAR_NEUTRAL: 중립
35  ** GEAR_DRIVE: 진행
36  */
37 typedef enum transmission
38 {
39     GEAR_PARKING,
40     GEAR_REVERSE,
41     GEAR_NEUTRAL,
42     GEAR_DRIVE
43 } Transmission;
44
45 // 핸들링 상태 열거형
46 typedef enum handling
47 {
48     LEFT,
49     RIGHT
50 } Handling;
51
52 /*
53  ** 변수 목록
54  ** fuel:          연료량
55  ** speed:         현재 자동차 속도
56  ** RPM:           자동차 엔진 분당회전수(Revolutions Per Minute)
57  ** gear_level:    기어 단수
58  ** airbag:        에어백 터짐 유무
59  */
60 int fuel = 100;
61 int speed;
62 int RPM;
63 int gear_level;
64 bool airbag;
65
66 Transmission tm = GEAR_PARKING;
67 Handling h;
68
```

```

69 int main(void)
70 {
71     /*
72     ** void StartOS(AppModeType AppModeID);
73     ** 운영체제가 oil파일에 설정된 Task 및 Alarm을 자동 시작해주도록 함
74     ** 순서
75     **
76     ** 1. StartOS Call
77     ** 2. OS에서 초기화 코드 실행
78     ** 3. StartupHook 함수(초기화 절차 진행 가능) 실행
79     ** 4. OS 커널 실행
80     ** 5. 가장 처음 순서 사용자 Task 실행
81     */
82     StartOS(OSDEFAULTAPPMODE);
83     return 0;
84 }
85
86
87 /*
88 ** DeclareTask(TaskType TaskID);
89 **
90 ** 각각의 Task는 .oil 파일 상에 정적으로 선언되며
91 ** 컴파일시 TaskType이라는 식별자를 가짐
92 ** C 프로그램 내에서 사용하기 위해 선언이 필요함
93 ** C 매크로로 정의되어 있음
94 ****
95 ** task_start_car:          자동차 시동 켜짐
96 **
97 ** task_check_module:       자동차 내부 모듈 점검 및 초기화 작업 수행
98 ** L task_check_fuel:       남은 연료 확인
99 ** L task_check_engine:     자동차 엔진 점검
100 ** L task_check_airbag:     자동차 에어백 점검
101 **
102 ** task_driving:            주행 시작, 터미널 내의 입력키에 따른 자동차 주행 작업 수행
103 ** L task_change_transmission: 자동차 변속기 제어
104 ** L task_handling:         자동차 핸들 제어
105 ** L task_accel:            자동차 액셀 제어
106 ** L task_break:            자동차 브레이크 제어
107 ** L task_change_gear_level: 자동차 기어 단수 제어
108 **
109 ** task_end_car:            자동차 시동 꺼짐
110 */
111 DeclareTask(task_start_car);
112
113 DeclareTask(task_check_module);
114 DeclareTask(task_check_fuel);
115 DeclareTask(task_check_engine);
116 DeclareTask(task_check_airbag);
117
118 DeclareTask(task_driving);
119 DeclareTask(task_change_transmission);
120 DeclareTask(task_handling);
121 DeclareTask(task_accel);
122 DeclareTask(task_break);
123 DeclareTask(task_change_gear_level);
124 DeclareTask(task_end_car);
125

```

```

126  /*
127  ** DeclareResource(ResourceType ResourceID);
128  **
129  ** 각각의 Resource 또한 .oil 파일 상에 정적으로 선언되며
130  ** 컴파일시 ResourceType이라는 식별자를 가진
131  ** 역시나 c 프로그램 내에서 사용하기 위해 선언이 필요함
132  ** c 매크로로 정의되어 있음
133  ****
134  ** module_initialize: 자동차 모듈 초기화 접근에 사용할 Resource
135  ** res_speed:      속도값 접근에 사용할 Resource
136  ** res_airbag:     에어백 접근에 사용할 Resource
137  */
138  DeclareResource(module_initialize);
139  DeclareResource(res_speed);
140  DeclareResource(res_airbag);
141
142  /*
143  ** oil 파일 상에서 AUTOSTART 옵션을 TRUE로 설정해서 Task들 중 가장 처음 실행되는 Task
144  ** task_check_module Task를 활성화시키고 자신을 Terminate 시킴
145  */
146  TASK(task_start_car)
147  {
148      printf("Starting car simulation..\n");
149      ActivateTask(task_check_module);
150      TerminateTask();
151  }
152

```

```

153  /*
154  ** StatusType GetResource(ResourceType ResourceID);
155  **
156  ** 다음 함수를 통해 ResourceID에 해당하는 Resource를 얻어 critical section에 진입함
157  ** 함수의 결과로 다음 StatusType의 결과 코드를 반환함
158  **
159  ** E_OK      : 에러 없음
160  ** E_OS_ID   : 유효하지 않은 Resource id, Resource가 없거나 이미 존재함
161  ** E_OS_ACCESS: 다른 Task 혹은 ISR에서 해당 Resource 점유 중
162  ****
163  ** StatusType ReleaseResource(ResourceType ResourceID);
164  **
165  ** GetResource() 함수로 획득한 Resource를 반환함
166  ** 함수의 결과로 다음 StatusType의 결과 코드를 반환함
167  **
168  ** E_OK      : 에러 없음
169  ** E_OS_ID   : 유효하지 않은 Resource id, Resource가 없거나 이미 존재함
170  */
171  TASK(task_check_module)
172  {
173      /*
174      ** initialize module(fuel, engine, airbag, etc.)
175      */
176
177      // fuel check
178      printf("fuel check.. ");
179      GetResource(module_initialize);
180      ActivateTask(task_check_fuel);
181      ReleaseResource(module_initialize);
182
183      // engine check
184      printf("engine check.. ");
185      GetResource(module_initialize);
186      ActivateTask(task_check_engine);
187      ReleaseResource(module_initialize);
188
189      // airbag check
190      printf("airbag check.. ");
191      GetResource(module_initialize);
192      ActivateTask(task_check_airbag);
193      ReleaseResource(module_initialize);
194
195      printf("initialize result: \nfuel=%d, RPM=%d, speed=%d, gear_level=%d, airbag=OK\n", fuel, RPM, speed, gear_level);
196
197      ActivateTask(task_driving);
198
199      TerminateTask();
200  }
201

```



```

202 TASK(task_check_fuel)
203 {
204     printf("\nremaining fuel: %d\n", fuel);
205     TerminateTask();
206 }
207
208 TASK(task_check_engine)
209 {
210     int i, j;
211     RPM = 1000;
212     speed = 0;
213
214     // something working simulation
215     for (i = 0; i < 10000; i++)
216     {
217         for (j = 0; j < 10000; j++)
218         {
219             }
220     }
221     printf("success\n");
222     TerminateTask();
223 }
224
225 TASK(task_check_airbag)
226 {
227     int i, j;
228     airbag = FALSE;
229     // something working simulation
230     for (i = 0; i < 10000; i++)
231     {
232         for (j = 0; j < 10000; j++)
233         {
234             }
235     }
236
237     printf("success\n");
238     TerminateTask();
239 }
240

```



```

241  /*
242  ** 자동차 주행 Task
243  ** 시뮬레이션을 위해 POSIX keyboard event을 다루는 procedure 포함되어 있음
244  **
245  ** 기본 조작
246  ** w:      accel 제어
247  ** s:      break 제어
248  ** a, d:   핸들 제어
249  ** j, k, l: 변속기 제어
250  */
251  TASK(task_driving)
252  {
253      // 다음 경로의 파일에 키보드 이벤트 정보가 있음
254      const char *dev = "/dev/input/by-path/platform-i8042-serio-0-event-kbd";
255      ssize_t n;
256      int fd;
257
258      // 읽기 모드로 파일 오픈
259      fd = open(dev, O_RDONLY);
260      if (fd == -1)
261      {
262          fprintf(stderr, "Cannot open %s: %s.\n", dev, strerror(errno));
263          return EXIT_FAILURE;
264      }
265
266      // 연료가 전부 연소될 때까지 주행
267      while (fuel > 0)
268      {
269          // 키 입력 이벤트 구조체에 저장
270          n = read(fd, &ev, sizeof ev);
271          if (n == (ssize_t)-1)
272          {
273              if (errno == EINTR)
274                  continue;
275              else
276                  break;
277          }
278          else
279          {
280              if (n != sizeof ev)
281              {
282                  errno = EIO;
283                  break;
284              }
285
286              // 이벤트 타입이 키보드 입력이며, 단일 혹은 반복 입력일 시,
287              if (ev.type == EV_KEY && ev.value >= 1)
288              {
289                  // pressed 'j' or 'k' or 'l'
290                  if (ev.code == 36 || ev.code == 37 || ev.code == 38)
291                      ActivateTask(task_change_transmission);
292                  // pressed 'a' or 'd'
293                  if (ev.code == 30 || ev.code == 32)
294                      ActivateTask(task_handling);
295                  // pressed 'w'
296                  if (ev.code == 17)
297                      ActivateTask(task_accel);
298                  // pressed 's'
299                  if (ev.code == 31)
300                      ActivateTask(task_break);
301              }
302          }
303          fflush(stdout);
304      }
305      TerminateTask();
306  }
307

```

```

308 TASK(task_change_transmission)
309 {
310     // key 'j'
311     if (ev.code == 36)
312     {
313         tm = GEAR_REVERSE;
314         printf("\ntransmission: Reverse");
315     }
316     // key 'k'
317     else if (ev.code == 37)
318     {
319         tm = GEAR_NEUTRAL;
320         printf("\ntransmission: Neutral");
321     }
322     // key 'l'
323     else
324     {
325         tm = GEAR_DRIVE;
326         printf("\ntransmission: Drive");
327     }
328
329     TerminateTask();
330 }
331
332 TASK(task_handling)
333 {
334     if (ev.code == 30)
335     {
336         h = LEFT;
337         printf("\nleft handling..");
338     }
339     else
340     {
341         h = RIGHT;
342         printf("\nright handling..");
343     }
344     TerminateTask();
345 }
346
347 TASK(task_accel)
348 {
349     fuel--;
350     RPM += 100;
351
352     if (RPM > 4000)
353         RPM = 4000;
354
355     switch (tm)
356     {
357     case GEAR_DRIVE:
358         speed += 2;
359         break;
360     case GEAR_REVERSE:
361         speed -= 2;
362         if (RPM > 2000)    RPM = 2000;
363         if (speed < -20)  speed = -20;
364         break;
365     }
366     printf("\nCurrent speed: %d, RPM: %d", speed, RPM);
367     ActivateTask(task_change_gear_level);
368     ActivateTask(task_check_fuel);
369     TerminateTask();
370 }
371

```

```

372 TASK(task_break)
373 {
374     fuel--;
375     if (speed >= 0)
376     {
377         speed -= 1;
378         RPM -= 100;
379         if (speed < 0) speed = 0;
380         if (RPM < 1000) RPM = 1000;
381     }
382     printf("\nCurrent speed: %d, RPM: %d", speed, RPM);
383     ActivateTask(task_change_gear_level);
384     ActivateTask(task_check_fuel);
385     TerminateTask();
386 }
387
388 TASK(task_change_gear_level)
389 {
390     if (tm == GEAR_DRIVE)
391     {
392         if (RPM >= 2500)
393             RPM -= 500;
394         if (speed > 10 && speed <= 20)
395             gear_level = 2;
396         else if (speed > 20 && speed <= 30)
397             gear_level = 3;
398         else if (speed > 30 && speed <= 40)
399             gear_level = 4;
400         else if (speed > 40 && speed <= 50)
401             gear_level = 5;
402         else if (speed > 50 && speed <= 60)
403             gear_level = 6;
404     }
405     else
406         gear_level = 1;
407     printf("\ngear_level: %d", gear_level);
408     TerminateTask();
409 }
410
411 /*
412 ** void ShutdownOS(StatusType Error);
413 **
414 ** OS를 종료하고 에러코드를 알려줌
415 ** 임베디드 플랫폼에서는 인터럽트 비활성화 및 청지
416 ** POSIX 환경에서는 application 종료
417 */
418 TASK(task_end_car)
419 {
420     printf("\nEnding car simulation..\n");
421     ShutdownOS(OSDEFAULTAPPMODE);
422     TerminateTask();
423 }
424
425 /*
426 ** 에어백 인터럽트 서비스 루틴
427 ** Ctrl+C로 인터럽트 발생시키면 oil에 정의된 ISR 실행
428 */
429 ISR(isr_airbag)
430 {
431     airbag = TRUE;
432     printf("\nwarning!! car crash, inflate airbag\n");
433     CallTerminateISR2();
434     ActivateTask(task_end_car);
435 }
436

```

7. 참고

- 개요(주제 선정 이유)

<https://blog.naver.com/bongkwankim/150107616214>

- ECU

<https://m.post.naver.com/viewer/postView.nhn?volumeNo=8967430&memberNo=32594659&vType=VERTICAL>

<http://blog.naver.com/PostView.nhn?blogId=whdgh3121&logNo=220561274243>

https://en.wikipedia.org/wiki/Electronic_control_unit

- OSEK

<https://blog.naver.com/bycho211/220956759038>

<http://autosar4.tistory.com/125>

<https://m.blog.naver.com/PostView.nhn?blogId=hsis6&logNo=161040530&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F>

<http://bmfrog.tistory.com/entry/OSEKVDX-%EB%9E%80>

- OIL

<https://blog.naver.com/hsis6/161040243>

- etc

https://www.oss.kr/oss_use?use_field_id=151

<https://blog.naver.com/h111922/220693943311>

- API

<http://www.irisa.fr/alf/downloads/puaut/TPNXT/SummaryOSEK.htm>

- Trampoline OS

<https://github.com/TrampolineRTOS/trampoline>