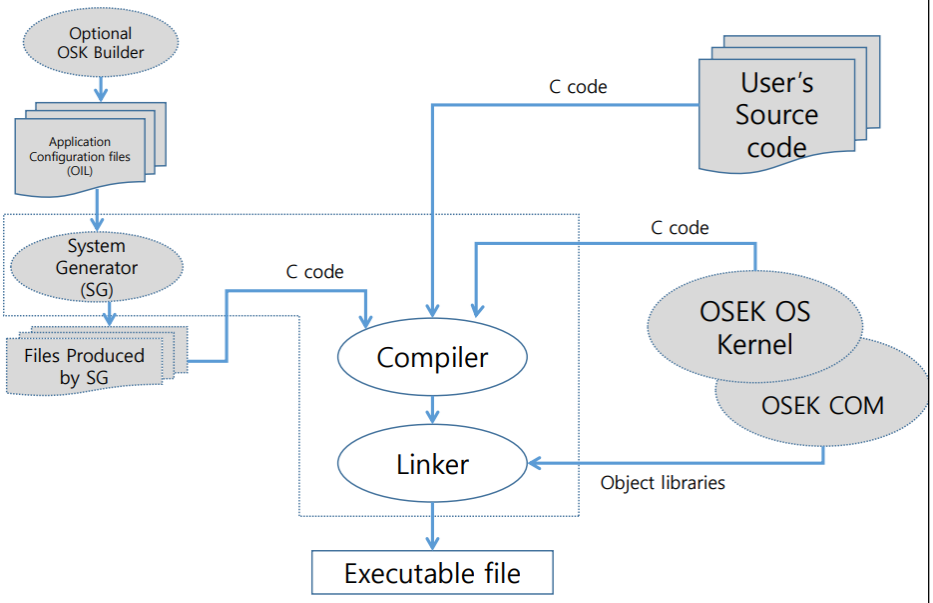
과목명	자동차임베디드 2	담당교수	서석현 교수님
학생이름	김세환	학번	2015146007

(작성유의사항)

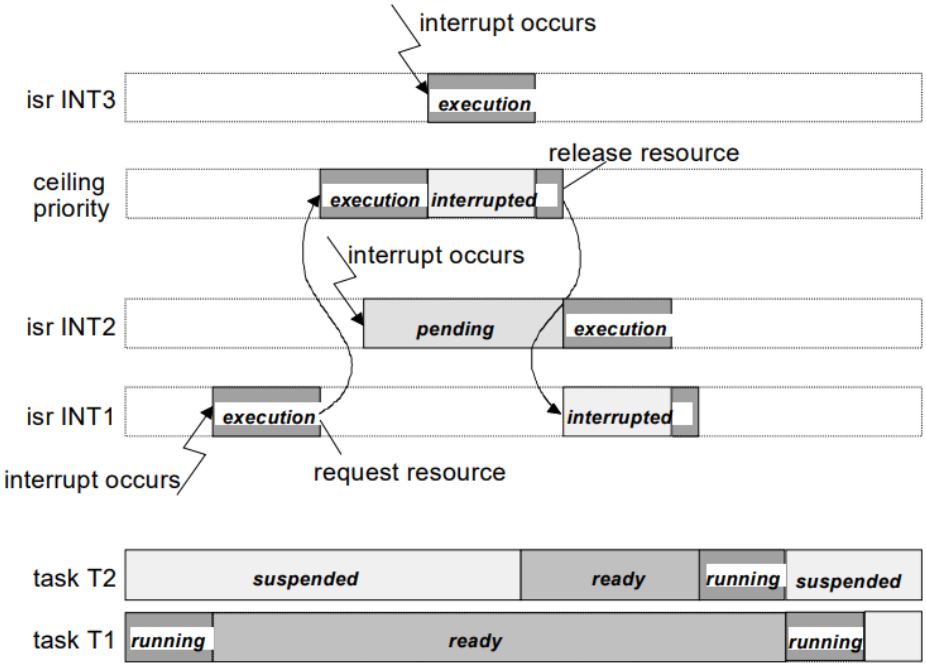
1. 다음 과제에서 요구하는 사항을 정리하여 작성하세요.
2. 수업내용 중 질문사항을 정리하여 작성하세요.

과제

1. 다음의 OSEK/VDX의 개발흐름도를 설명하시오.



2. 다음의 그림을 보고 학습한 이론과 연관지어 실행과정을 순차적으로 설명하시오.



3. TASK와 관련한 API 동작과 사용 용법에 대해 설명하시오.

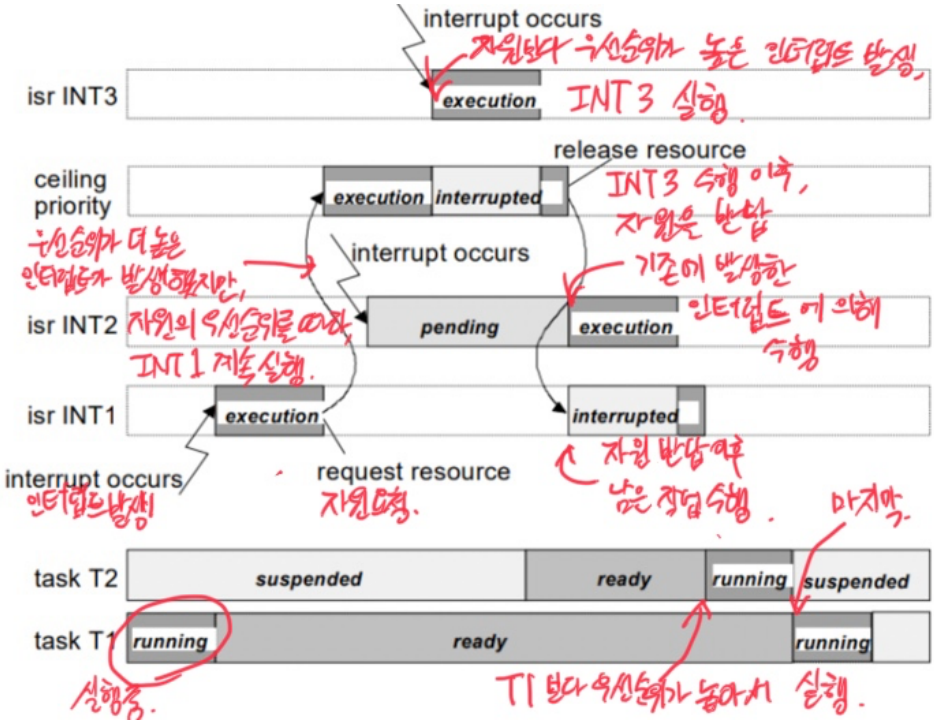
1. OSEK/VDK의 개발 흐름은 다음과 같습니다.

우선 OSEK 빌더를 통해 OIL 파일을 생성합니다. OIL파일은 일종의 설정파일이라고 생각하면 됩니다. 각 OSEK OS에 관련된 셋업이나, 태스크가 어떤 타입의 태스크인지, 우선순위는 얼마인지 등의 여러 환경 정보를 셋업하게 됩니다.

이후 개발자는 OSEK API를 활용하여 C로 어플리케이션 코드를 작성합니다.

이후 최종적으로 개발자의 소스코드 및 System Generator가 OIL파일을 읽어서 자동으로 생성한 .h, c 파일등과 개발자의 소스코드를 모두 컴파일 및 링킹, 즉 빌드합니다.

이후 ECU나 타겟보드에 `퓨징`을 통해 executable (바이너리) 파일을 밀어넣어줍니다.



2. 우선순위는 INT3 > 자원 > INT2 > INT1 > T2 > T1 이다.

따라서 자원 선점의 케이스가 발생하지 않는다면 우선순위에 따라 인터럽트 발생 시, 인터럽트 간 우선순위에 따라 인터럽트가 먼저 수행되고, 모든 인터럽트 작업 이후 태스크의 우선순위에 따라 태스크가 수행된다.

그림과 같은 케이스는 task T1이 running 상태 일 때, INT1에 해당하는 인터럽트가 발생했다. 따라서 T1을 중지하고 INT1을 수행한다.

	이 때 T1은 자원을 요청한다. 자원은 INT3보다는 낮지만, INT2 보다는 높은 우선순위를 가진다. 따라서 INT1은 자원에 해당하는 우선순위를 갖고 작업을 진행한다. 이 때 INT2 에 해당하는 인터럽트가 발생하였지만, INT1의 우선순위는 INT2보다 높은 자원의 우선순위를 갖기 때문에 계속해서 작업을 진행한다. 자원을 갖고 작업하던 중, INT3의 인터럽트가 발생한다. INT3은 자원보다 우선순위가 높으므로, INT1의 작업은 pending되고, INT3을 수행한다. INT3의 작업이 완료 된 이후, INT1가 작업을 수행하다가 자원을 반납한다. 자원을 반납한 경우 INT1의 우선순위는 INT2보다 낮기 때문에, pending되었던 INT2가 실행된다. INT2의 작업이 마무리 된 이후, INT1는 남은 작업을 실행하고 작업을 마무리한다. 이후 T2와 T1이 모두 ready 상태이므로, 우선순위가 높은 T2가 먼저 작업을 진행한다. T2의 작업이 마무리된 이후 남은 T1의 작업을 진행한다. 정리하자면, 태스크보다 인터럽트는 항상 우선순위가 높아서 인터럽트 발생시 인터럽트부터 수행된다. 인터럽트가 수행 중일 때, 자신보다 우선순위가 높은 인터럽트가 발생하면 발생한 인터럽트를 수행한다. 다만, 인터럽트가 자원을 선점하였을 경우, 자원의 우선순위에 따라 다른 인터럽트가 발생하더라도, 발생한 인터럽트를 수행하거나, pending 시킬 수 있다. 3. TASK와 관련된 API API의 형태가 조금 특이하다. 예를들자면 GetTaskState라는 API의 반환값이 state인 것이 아니라, 인자로 넣어주는 변수에 저장해서 전달한다. API들의 리턴값은 사전에 정의된 Status가 리턴된다 **태스크를 정의하는 API : DeclareTask(task_name)**

	태스크 블록 API : TASK(task_name) task_name에 해당하는 작업을 block scope 안에서 작성한다. **태스크를 종료하는 API : TerminateTask()** 이 API는 TASK(task_name) 블록 안에서 호출되어야 하며, task_name에 해당하는 TASK를 종료시키는 역할을 한다 **태스크를 활성화시키는 API : ActivateTask(task_name)** task_name에 해당하는 TASK를 활성화시키는 역할을 한다. 인터럽트 블록 안에서도 호출 가능함. **다른 태스크를 호출하고 현재 태스크를 종료 : ChainTask(task_name)** 다른 Task를 호출하고 현재 Task를 종료한다. task_name에 해당하는 Task를 activate 시킨다. **강제로 스케쥴러를 호출하는 API : Schedule()** 스케쥴링 정책에 상관없이. Schedule()이 호출되는 시점에서 스케쥴링 작업을 진행함. 호출된 Task 블록의 Task의 우선순위보다 다른 ready 상태인 Task의 우선순위가 높다면, 자신을 ready 상태로 만들고, 다른 Task가 실행된다. **현재 실행중인 Task의 TaskID를 얻는 API : GetTaskID(ref_task_name)** 현재 실행중인 TaskID를 ref_task_name 변수에 전달해준다. API 자체의 호출값은 TaskStatus를 반환. TaskID를 리턴해주는 것이 아님을 주의 **특정 Task의 현재 상태를 확인하는 API : GetTaskState(task_name, ref_task_state)** 호출된 시점의 task_name의 상태 정보를 ref_task_state에 저장한다.
질문사항	ActivateTask 호출 시 리턴값 중 Too many task activation of <TaskID> 라는 리턴값이 있습니다. 이러한 경우 사전에 정의된 E_OK (값은 0) 과 같이 특정한 정수를 리턴하는것인가요? Too many task activation of <TaskID>는 생긴 것은 뭔가 error 내용을 반환하는 것처럼 생겨서 궁금합니다. 또한 위 에러와 같은 경우에는 컴파일 / 링킹 즉 빌드 time에서는 에러를 찾을 수 없지만, runtime에서 에러가 발생할 것 같은데, OSEK OS는 런타임 에러가 발생하면 어떻게 처리하나요? (OS가 뻗는다던지 ??) 디버깅은 어떻게 하나요?