velog.io

Real-Time System (1) : Real-Time System, Model

5~6분

사담

학기 중에 수업도 듣고 조교 일도 하고 논문도 쓰면서 바쁜 삶을 보내느라 블로그를 뒷전으로 하고 있었다. 그래도 Real-Time을 연구하는 사람으로 관련 내용을 정리할 필요가 있을 것 같아 Real-Time System에 대해 배운 내용을 간단히 정리해보려고 한다. 이번 글에서는 Real-Time System의 정의와 특징, 그리고 Real-Time System을 일반적으로 어떤 형태로 모델링하는지에 대해 소개하고자 한다.

Real-Time System의 정의

Real-Time System은 아래의 두 constraint을 가지고 있는 system을 의미한다.

- Logical Correctness : 올바른 output을 만들어야 한다.
- Temporal Correctness : 알맞은 시간에 output을 만들어야 한다.

우리가 일반적으로 사용하는 시스템 (General Purpose Computer)들이 프로그램을 최대한 빨리 실행하여 throughput을 높이는 것이 목적이지만, 위의 두 Real-Time constraint는 주어진 deadline을 반드시 만족해야 하는 시스템 (e.g. 사드가 미사일을 발견하고 대응할 때까지 걸리는 시간이 일정 deadline 이상 걸리면 폭격을 맞을 것이다)에 대해 reliability를 제공하는 것이 목적이라고 할 수 있다. 시스템 내의 모든 task가 deadline을 만족한다는 것은 predictability를 높이기 위함이라고도 생각할 수 있다. Real-time system은 일반적으로 우선도에 따라 priority를 주고 priority가 높은 순서대로 task를 스케줄링하는데, 따라서 컴퓨터가 아무리 좋더라도 이러한 스케줄링 기법이 없으면 real-time constraint를 만족하지 못할 수 있다.

일반적으로 Real-Time system은 embedded system에서 많이 사용하는데, Real-time embedded system이 general purposr computer와 비교하여 가지는 특징들은 아래와 같다.

Real-Time embedded system의 특징

Real-Time embedded System	**General Purpose Computer**
특정 목적을 위한 computing을 제공한다	general purpose
일반적으로 정해진 task를 수행한다.	programmable하다. (Dynamic task)
physical world와 상호작용이 많다.	physical world와의 상호작용이 적다.

Real-Time embedded System	General Purpose Computer
Real-time Contraint를 만족해야 한다.	빠르게 돌리는 것이 목적이다.
CPU, memory 등 자원이 제한되어 있다.	자원이 많다.
low level programming에 가깝다. (하드웨어를 직접 코드로 제어)	주로 high level programming을 한다.

Real-Time System의 분류

Real-Time system을 아래와 같은 기준들로 분류할 수 있다.

- Deadline의 특징에 따라
- Hard real-time : deadline을 지키지 못하면 시스템에 치명적 오류가 생긴다.
- Soft real-time : deadline을 지키지 못하면 성능의 저하가 일어난다.
- Workload의 특징에 따라
- Static task : design phase에 만들어지는 task를 의미한다. (아직 프로그램을 실행하기 전인 offline phase라고 생각하면 된다.)
- Dynamic task : runtime에 만들어지는 task를 의미한다.
- system의 분산 여부에 따라
- Centralized : 한 system에 모든 task가 모여있는 경우를 의미한다.
- Decentralized : task들이 분산되어 있어 네트워크 등으로 연결되어 있는 경우를 의미한다.

Reference Model

Embedded system는 전체 시스템에 대한 superloop를 만들고 superloop 안에서 task를 반복하여 실행하고 interrupt 등의 event가 있는 경우 실행한다. 그런데 시스템이 복잡해지게 되면 이런 superloop를 설계하기 어렵고, event가 많아지면 event 간 우선순위를 정하기도 어려워진다. 따라서 Real-Time System의 연구에서는 보통 전체 모델을 abstract한 reference model을 사용한다. (다른 포스트에서 작성했던 system model이 이 reference model에 해당한다.) reference model을 사용하는 경우 아래의 장점이 있다.

- 구체적인 문제 여러 개를 abstraction하여 Model Problem으로 만들고 다시 특정 problem으로 구체화(customizing)할 수 있다.
- 해결하고자 하는 문제가 이미 알려진 문제인지, 해결된 것인지, 쉬운 문제인지 등을 확인할 수 있다.

Reference model의 구성은 크게 아래의 세 가지가 있다.

1. Workload model : 실행할 task들의 특성을 정의한다.
2. Resource model : task를 실행할 processor들의 특성을 정의한다.
3. Scheduling algorithm : task를 어떤 rule에 의해 스케줄링할지를 정의한다.

Workload model

workload model에서의 용어들은 아래와 같다.

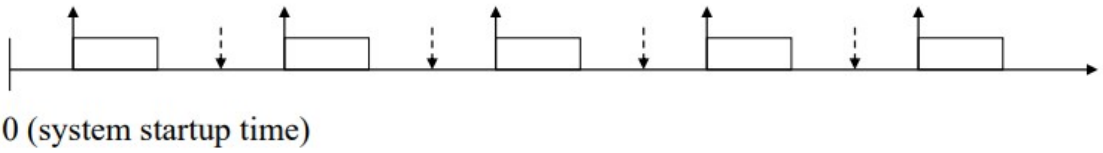
- job : computation의 단위
- task : 같은 종류의 job의 연속이다. 실행할 기능의 단위라고 생각할 수 있다.
- release time : 각 job이 실행할 수 있는 시간 (processor의 ready queue에 들어가는 시간)
- jitter : physical system에서는 release time에 어느 정도 오차가

존재

- periodic task : release time이 주기적임
- sporadic task : 두 release time 사이 간격에 최소값이 있는 경우
- deadline : job이 끝나야 하는 시간
- absolute deadline : wall time 기준 deadline
- relative deadline : release time 기준 deadline
- laxity : deadline까지 남은 시간
- execution time : 실행 시간

Periodic Task Model

superloop에서 각 task를 반복하여 실행하는 것처럼 Real-Time System에서는 task 내의 job이 주기적으로 실행되는 Periodic Task Model을 주로 사용한다.



Periodic Task의 도식화. 위 화살표가 release, 아래 화살표가 deadline을 의미한다.

[periodic task T_i 의 구성]

- phase (offset) ϕ_i : 첫 release time
- period p_i
- j번째 job의 release time $r_{i,j} = r_{i,j-1} + p_i$
- execution time C_i
- relative deadline D_i
- utilization $U_i = C_i / p_i$
- system의 utilization $U = \sum_i C_i / p_i$

(release가 주기적이지 않은 aperiodic job의 경우 inter-arrival time으로 나눈다.)

- temporal distance : period 내에서 실행되는 시간이 다르기 때문에 temporal distance가 0이 되는 bundling 등이 발생할 수 있다.
- task 간 precedence constraint가 존재하는 경우에는 DAG로 나타내기도 한다.

Resource Model

- active resource : job의 크기와 active resource의 **speed**가 execution time, transmission time을 결정 (ex CPU, LAN)
- passive resource : job이 execution을 일으키는 자원들. speed와 관련없음. (ex 공유변수, 세마포어)

Scheduling Model

스케줄링에 관련된 용어는 다음과 같다.

- $\sigma(t)$: t 시간에 할당되어있는 job
- 프로세서는 한번에 한 job을 assign할 수 있다
- feasible : constraint를 만족하면서 complete할 수 있는 경우 schedule을 feasible하다고 한다.
- schedulable : 적어도 한 알고리즘이 feasible schedule을 만들 수 있는 경우 task set을 schedulable하다고 한다.
- preemptive : running task가 실행 중이어도 더 높은 priority의 task에 의해 interrupt될 수 있다. (높은 priority의 task가 선점하여 먼저 실행한다.)
- work conserving : pending job이 있는 경우 CPU가 idle하지 않고 어떤 job을 실행한다.
- optimality : 스케줄링 알고리즘 S가 optimal이라는 뜻은, task set이 특정 알고리즘으로 스케줄링이 가능하다면, S로도 가능하다

는 의미다.

- off-line 스케줄링 : 스케줄 table을 만들고 dispatch만 한다.
- on-line 스케줄링 : ready queue에 있는 job들을 보고 어떤 job을 실행할지 결정한다.
- clairvoyant : 미래의 job들을 모두 알고 스케줄링한다.

  실제 스케줄링 알고리즘은 다음 포스트에 이어서 설명하도록 하겠다.