08장 서브루틴과 제어 추상화

【8.6.4】 분리적 이벤트 시뮬레이션

예 8.79

복잡한 물리 시스템의 순차적인 시뮬레이션

도시 내의 교통 흐름을 이용한 실험을 원한다고 가정해보자. 충분한 정확도를 가지고 실제 세계를 포착한다면 계산된 교통량 모델은 건설 프로젝트, 사고, 새로운 개발로 인한 증가된 교통량, 도로 배치의 변경 효과 등을 예측하게 할 것이다. 일반적인 순차적 언어로 이런 시뮬레이션을 작성하는 것은 (확실히 불가능하지는 않지만) 어렵다. 자료 구조를 이용해 흥미가 있는 각 객체(자동차, 교차로, 도로 구획 등)를 표현할 것이다. 주 프로그램은 아래와 같을 것이다.

```
while 현재 시간 < 시뮬레이션의 종료 시간
    흥미가 있는 상호작용이 발생할 시간 t를 계산함
    현재 시간 := t
    상호작용을 반영하기 위해 객체의 상태를 갱신함
    원하는 통계치를 기록함
모인 통계치를 출력함
```

이 접근 방법과 관련된 문제는 어떤 객체가 다음에 상호작용해야 하는지를 결정하는 것과 한 상호작용에서 다음 것으로의 상태를 기억하는 것에 있다. 수동적인 자료 구조를 가진 자동차와 같은 활성 객체를 표현하는 것과 시간을 프로그램에서 활성화된 실체로 하는 것은 어떤 의미에서는 부자연스럽다. 좀 더 매력적인 접근 방법은 동시 실행 루틴을 이용해 각 활성 객체를 표현하고 각 객체가 자신의 상태를 기록하게 하는 것일 것이다.

각 활성 객체가 다음에 흥미가 있는 무언가를 할 것인지를 말할 수 있으면 다음 이벤트의 시간이 정렬된 우선순위 큐에 현재 비활성 중인 동시 실행 루틴을 유지함으로써 다음에 어떤 객체가 상호작용할지를 결정할 수 있다. 자동차가 취하는 각 이동에 대한

예 8.80

동시 실행 루틴 기반 교통량 시뮬레이션의 초기화

동시 수행 루틴을 생성하고 이동을 시작하는 시간을 표시하는 "각성" 시간과 함께 각 동시 실행 루틴을 우선순위 큐로 삽입함으로써 하루 동안의 교통량 시뮬레이션을 시작할 것이다.

```
coroutine trip(...)
...
for each trip t
    p := new trip(...)
    schedule(p, t.start time)
```

예 8.81

교통량 시뮬레이션에서 도로 구획을 탐색하기

도로 구획을 수동적으로 간주하고 자료 구조로 표현한다고 가정하자. 주어진 순간에 각 방향으로 지니고 있는 자동차의 숫자로 한 구획을 모델화할 수 있다. 이 숫자는 자동차가 안전하게 운행할 수 있는 속도에 영향을 줄 것이다. 그것이 깨어날 때마다 이동을 나타내는 동시 실행 루틴은 운행할 필요가 있는 다음 도로 구획을 조사한다. 해당 구획상의 현재 교통량에 기반해 그 구획을 가로지르는 데 걸리는 시간을 계산하고 미래의 적절한 시점에 다시 깨어나기 위해 자신을 스케줄링한다.

```
coroutine trip(origin, destination : location)
    origin에서 destination까지의 경로를 계획함
    detach
    그 경로의 각 구획마다
        그 구획에 끝에 도착하는 시간 i를 계산함
        schedule(current_coroutine, current_time + i)
```

예 8.82

추후 실행을 위해 동시 실행 루틴을 스케줄링하기

schedule 연산을 transfer 위에 쉽게 만든다.

```
schedule(p: coroutine, t: time)
    -- p가 자신이나 타인일 것이다.
    우선순위 큐에 (p, t)을 삽입함
    if p = current_coroutine              -- 자신
        가장 빠른 순서쌍 (q, s)를 우선순위 큐에서 뽑음
        current_time := s
        transfer(q)
```

예 8.83

교통 신호에 자동차를 대기 행렬에 넣기

어떤 경우에는 주어진 객체를 다시 스케줄링할 때를 결정하는 것이 어려울지도 모른다. 예를 들어 교차로에서 교통 신호의 효과를 좀 더 정확히 모델화하기 바란다고 가정하자. 각 방향에 기다리고 있는 자동차를 기록하는 자료 구조와 신호가 바뀌면 자동차를 지나가게 하는 동시 실행 루틴을 가지고 각 교통신호를 표현한다.

```
record controlled_intersection =
    EW_cars, NS_cars : queue of trip
    const per_car_lag_time : time
        -- 앞에 있는 차가 출발한 후 차가 출발하는 데 걸리는 시간
```

```
coroutine signal(EW_duration, NS_duration : time)
   detach
   loop
      change_time := current_time + EW_duration
      while current_time < change_time
         if EW_cars not empty
            schedule(dequeue(EW_cars), current_time)
         schedule(current_coroutine, current_time + per_car_lag_time)
      change_time := current_time + NS_duration
      while current_time < change_time
         if NS_cars not empty
            schedule(NS_cars.dequeue(), current_time)
         schedule(current_coroutine, current_time + per_car_lag_time)
```

예 8.84

신호에 기다리기

교통 신호가 제어하는 도로 구획의 끝에 도착했을 때 이동은 교차로를 지나가는 데 걸리는 시간을 계산할 필요가 없다. 오히려 그것은 적절한 대기 차량의 큐에 자신을 넣고 signal 동시 실행 루틴이 미래의 어느 시점에 깨워줄 것임을 알고 "잠들면 된다".

```
coroutine trip(origin, destination : location)
    origin에서 destination까지의 경로를 계획함
    detach
    그 경로의 각 구획마다
       그 구획의 끝에 도착하는 시간 i를계산함
       schedule(current_coroutine, current_time + i)
       if 구획 끝에 교통 신호가 있음
          적절한 큐 Q를 식별함
          Q.enqueue(current_corountine)
          sleep()
```

예 8.85

추후 실행을 기대하고 잠들기

schedule과 마찬가지로 sleep은 transfer 위에 쉽게 만들 수 있다.

```
sleep()
    가장 앞에 있는 순서쌍 (q, s)를 우선순위 큐에서 빼옴
    current_time := s
    transfer(q)
```

사실 schedule 연산은 간단히 다음과 같다.

```
schedule(p: coroutine, t: time)
    우선순위 큐에 (p, t)를 삽입함
       if p = current_corountine
          sleep()
```

명백히 이 교통량 시뮬레이션은 극단적으로 단순화되어서 실제 도시의 자동차의 행태를 포착하지 못하지만 분리적 이벤트 시뮬레이션의 기본 개념을 설명한다. 공학, 계산생물학, 물리학과 우주론, 심지어 컴퓨터 설계까지 포함한 광범위한 응용프로그램 영역에서는 좀 더 세련된 시뮬레이션을 사용한다. 다중 프로세서 시뮬레이션(예를 들면 참고 자료 [VF94]를 보라)을 일반적으로 프로세서를 시뮬레이션하는 "프론트 엔드(front end)"와 메모리 하위 시스템을 시뮬레이션하는 "백 엔드(back end)"로 나눈다. 프론트 엔드의 각 동시 실행 루틴은 그 시스템의 마이크로프로세서 중 하나의 행태를 포착하는 기계어 인터프리터로 구성되어 있다. 백 엔드의 각 동시 실행 루틴은 불러오기나 저장하기 명령어를 표현한다. 프로세서가 불러오기나 저장하기를 수행할 때마다 프론트 엔드는 백 엔드에 새로운 동시 실행 루틴을 생성한다. 백 엔드 내의 자료 구조는 캐시, 버스, 네트워크 연결, 메시지 라우터와 메모리 모듈을 포함한 다양한 하드웨어 자원을 표현한다. 주어진 불러오기나 저장하기에 대한 동시 실행 루틴은 그 위치가 지역 캐시에 있는지를 검사한다. 그렇지 않다면 앞에 제시한 간단한 예에서 자동차가 도로 구획과 교차로로의 접근을 위해 경쟁하는 것과 흡사하게 하드웨어 자원으로의 접근을 위해 다른 동시 실행 루틴과 경합하면서 그 프로세서와 메모리 사이의 상호연결 네트워크를 탐색해야 한다. 해당 자료를 이용하기 전에 한 프로세서가 불러오기를 완료하기를 기다려야 하고 자료를 메모리에 전달하는 속도가 저장하기를 백 엔드로 삽입할 수 있는 속도를 제한하기 때문에 백 엔드 시스템의 행동은 프론트 엔드에 영향을 미친다.

✓ 확인문제

77. 분리적 이벤트 시뮬레이션의 계산 모델을 요약하라. 시간 기반 우선순위 큐의 중요성을 설명하라.

78. 분리적 이벤트 시뮬레이션을 만들 때 어떤 것을 동시 실행 루틴을 가지고 모델화할지와 어떤 것을 자료 구조로 모델화할지를 어떻게 결정하는가?

79. 모든 비활성 중인 동시 실행 루틴이 우선순위 큐에 있음을 보장하는가? 설명하라.