UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Ingegneria

Corso di Laurea in Ingegneria Informatica

Software Engineering for Embedded Systems
Project work

Titolo

Edoardo Sarri

7173337

Data

Indice

Elenco delle figure

1 Introduzione

L'obiettivo è creare un sistema (in Java) eseguibile da linea di comando che permetta di generare tracce di un'esecuzione. Ogni traccia è definita come una sequenza di coppie $< tempo, evento >$.

Un *evento* può essere: rilascio di un job di un task; acquisizione/rilascio di un semaforo da parte di un job di un task; completamento di un chunk; completamento di un job di un task.

1.1 Capacità

A partire da un taskset, il sistema ha le capacità di:

- Osservare possibili fallimenti
  I possibili fallimenti che si voglio osservare sono: deadline miss; violazione del tempo di computazionde del chunck (sia in eccesso che in difetto).

2 Analisi

In questo capitolo analiziamo la struttura del progetto, partendo dai suoi componenti e definendo la loro relazione.

2.1 Componenti

2.1.1 Task

Un task è definito da: un un insieme di Chunk; la deadline; la priorità nominale e dinamica; il pattern di rilascio.

Non ci interessa definire un activation time perché vogliamo considerare il caso pessimo: l'activation time sarà l'istante inziale per tutti i task.

2.1.2 Chunk

Un chunk, cioè una computazione atomica del task. È definito da: una distribuzione del tempo di esecuzione; una eventuale richiesta di risorse da usare in mutua esclusione (da acquisire prima dell'esecuzione e rilascaire subito dopo).

2.1.3 Taskset

È un insieme di task. È l'oggetto principale gestito dallo scheduler.

2.1.4 Risorse

Sono le risorse da utilizzare in mutua esclusione. Ogni risorsa è gestita da un semaforo binario, quindi può essere posseduta da un solo task alla volta.

2.1.5 CPU

È l'unità di elaborazione. Supponiamo essere unica.

2.1.6 Scheduler

È il componente che assegna un task al processore. Al momento abbiamo implementato solo Rate Monotonic (RM).

2.1.7 Protocollo di accesso alle risorse

È il meccanismo che garantisce la mutua esclusione di una risorsa. Al momento abbiamo implementato solo Priority Ceiling Protocol (PCP).

2.2 Class diagram

Per capire meglio la struttura del progetto, analizziamo il diagramma delle classi.

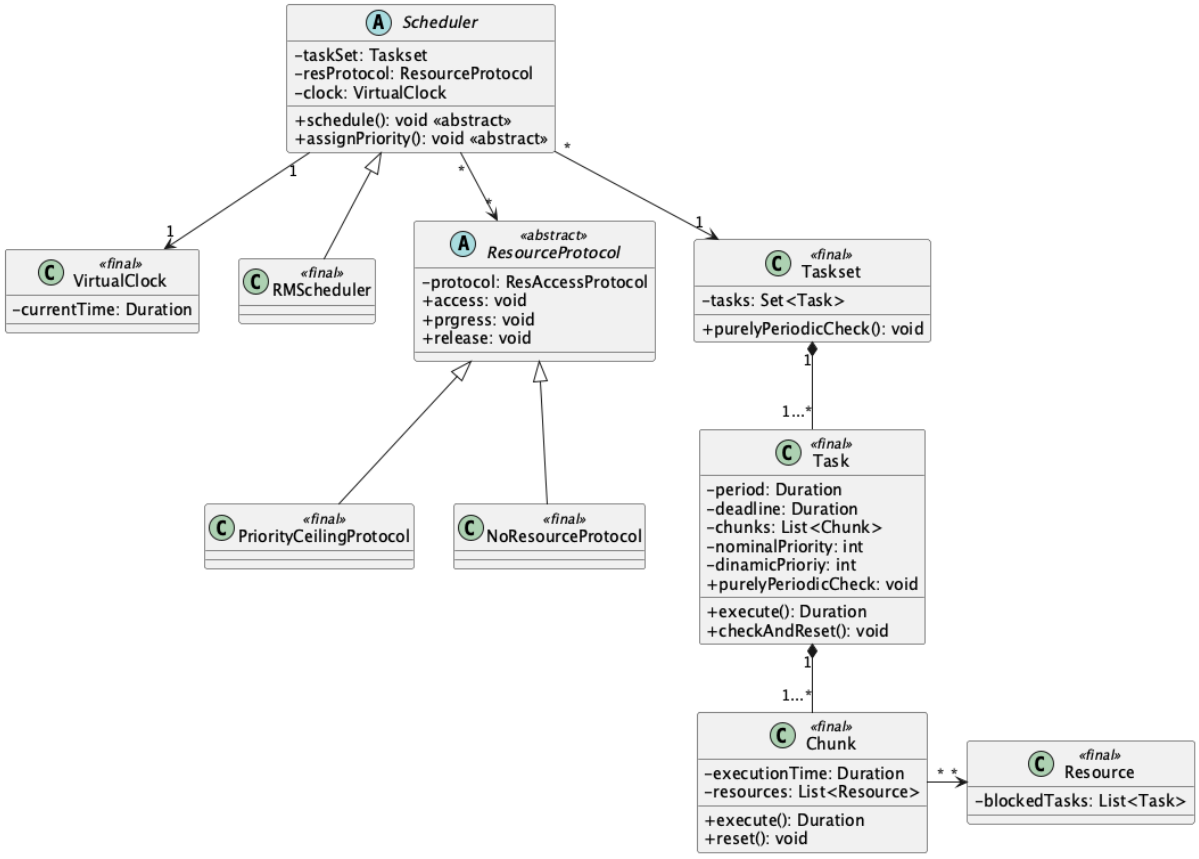

Figura 2.1: Class diagram.

3 Implementazione

3.1 Scheduler

Come prima osservazione specifichiamo che la classe `RMScheduler` è una sotto classe di `Scheduler`. questo permette in futuro di implementare altri tipi di scheduler e di dataarli facilmente al sistema.

Ogni oggetto che estende `Scheduler` ha, oltre a ciò che definisce uno scheduler (e.g. taskSet e l'eventuale protocollo di accesso alle risorse), un oggetto di tipo `VirtualClock` che rappresenta il clock globale del sistema e una lista di task bloccati `blockedTask`.

Ogni scheduler deve associare una priorità; ad esempio RM la assegna in modo opposto rispetto alla durata del periodo. Questo compito è delegato dal metodo `assignPriority`. Ovviamente poi abbiamo il metodo `schedule` che viene chiamato per avviare la simulazione.

3.1.1 Rate Monotonic

Descriviamo brevemente l'idea di implementazione di RM e osserviamo il relativo Sequence Diagram in Figura 3.1.

Durante la creazione dello scheduler si fa un controllo per valutare che tutti i task del taskSet siano puramente periodici.

La simulazione si basa su due strutture principali:

- `taskReady`
  Ospita i task che si contendono l'accesso alla CPU. Al suo interno si usa un'ordinamento inverso rispetto alla durata del periodo dei vari task.

- `events`
  Gestisce gli eventi importnati, cioè i momenti in cui finisce il periodo di un task. Nell'intervallo tra un perido e il successivo infatti lo scheduler non fa altro che mandare in esecuzione uno dopo l'altro il task a priorità maggiore. Quando arriva il momento di un evento, vengono controllati i task il cui periodo è finito; questo controllo serve per valutare se una deadline è stata mancata e per rilasciare nuovamente il task nel caso non ci siano errori. Gli eventi sono l'unione ordinata dei multipli di ciascun periodo fino al minimo comune multiplo dei periodi oppure fino a 10 volte il periodo maggiore (per semplicità del caso sia molto oneroso gestire questa lista).

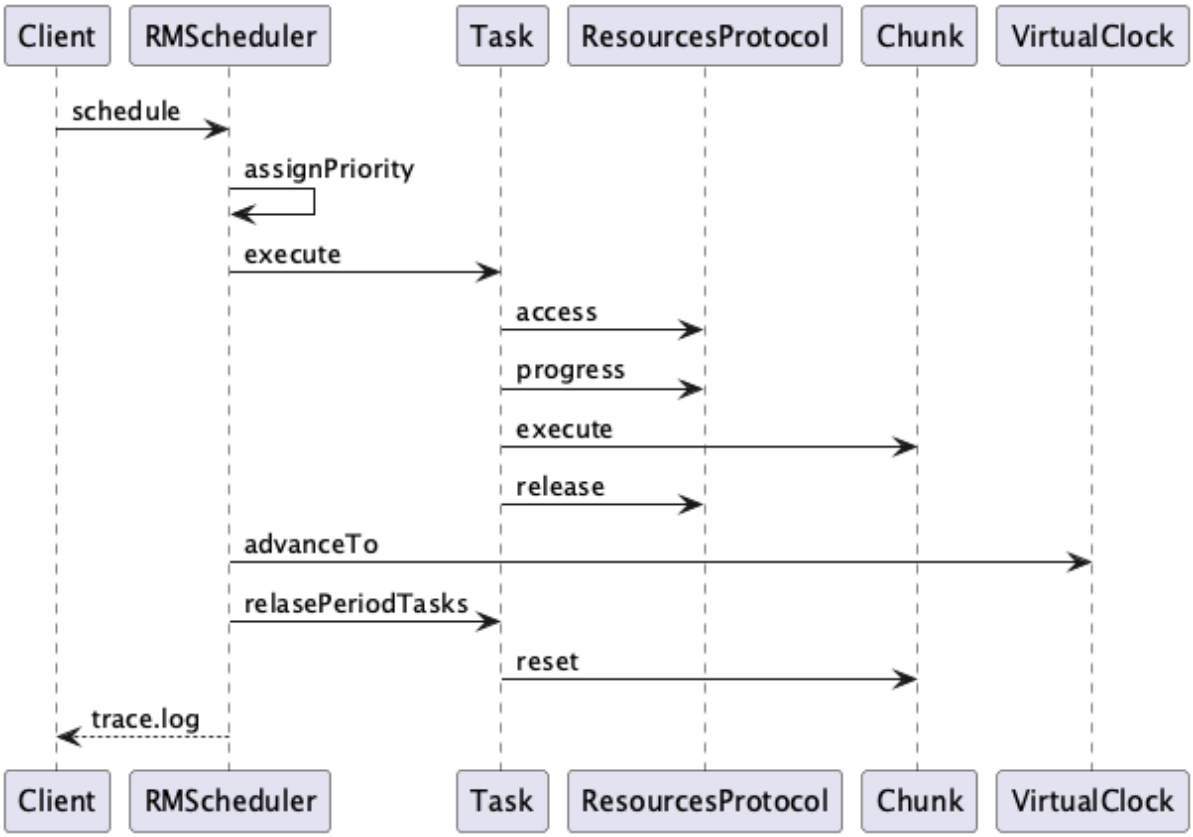

Figura 3.1: Sequence Siagram RM

Il metodo `schedule` poi delega la gestione dei chunk di ogni task alla classe `Task`, la quale a sua volta rimanda alla calsse `Chunk` la loro esecuione (e.g. il logging).

3.2 Resource Access Protocol

Ogni implementazione di un protocollo di accesso alle risorse deve estendere la classe `ResourceProtocol`.

I metodi definiti da questa classe astratta sono le operazioni che devono essere svolte da un procotollo di questo tipo: deve gestire la fase di accesso, progresso e rilascio. Definisce anche il metodo `initStructures` che ha il compito di inizializzare le strutture dati usate dal protocollo.

3.2.1 Priority Ceiling Protocol

Tralasciando quello che fanno i metodi di accesso, progresso e rilascio, che riflettono quanto ci dice la teoria, in questo classe le strutture usate sono prevalentemente due:

- `ceiling`
  È una mappa che associata ad ogni risorsa il suo ceiling, cioè la massima priorità nominale dei task che usano quella risorsa.

- `busyResources`
  È una losta delle risorse che sono occupate da un qualche task.

3.3 Utilità

In questo capitolo sono brevemente descritte le scelte di alcuni componenti di utilità.

3.3.1 Loggin

Per il logging è stato implementato un semplice logging su un file e viene rappresnetato come una sequenza di coppie $< evento, tempo >$.

Il file di destinazione delle tracce loggate è `trace.log`.

3.3.2 Clock

Il clock del sistema è rappresnetato dalla classe `VirtualClock`. Questa non fa altro che mantenere il tempo assoluto ed esporre due metodi che permettono di avanzare di un dato intervallo temporale e avanzare fino a un determinato tempo.

Il tempo è gestito tramite oggetti di tipo `Duration`, che implementa oggetti immutabili e che permetto una facile gestione del tempo.

3.3.3 Sampling dei tempi

Quando si deve definire i tempi che definiscono i vari componenti del sistema, cioè come il periodo, la dealine, l'execution time di un chunk, si usa un campionamento da un data distribuzione.

Le distribuzioni sono data dalla libreria `Sirio`; oltre a quelle definite dalla libreria è stata implementata la classe `ConstantSampler`, che permette di gestire tempi costanti, mantenendo l'astrazione della libreria Sirio.

4 Dubbi

4.1 Domande

- Nei fallimenti osservati che vuol dire valutare la violazione del tempo di computazione di un chunk (troppo basso o troppo alto)? Se non viola la deadline allora esegue per il suo execution time altrimenti di più.
- Manca EDF e la possibilità di iniettare fault.

Bibliografia

[1] Laura Carnevali. *Appunti slides Software Engineering for Embedded System*.

[2] *chatGTP*.

[3] *documentazione Java, oracle*. https://docs.oracle.com/javase/8/docs/api/overview-summary.html.