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

Elenco delle tabelle

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 $< evento, tempo >$.

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

Vogliamo avere la possibilità di:

- Iniettrare fault
  Si vuole avere la possibilità di iniettare fallimenti tramite due tecniche: aggiungere un task che fa cycle stealing (implementato tramite un task a priorità massima che non appartine al task set da schedulare); task programming defect, cioè se abbiamo un'implementazione non funzione di un task scorretta (es: acquisisce il semaforo ma non si alza la priorità).

- 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.

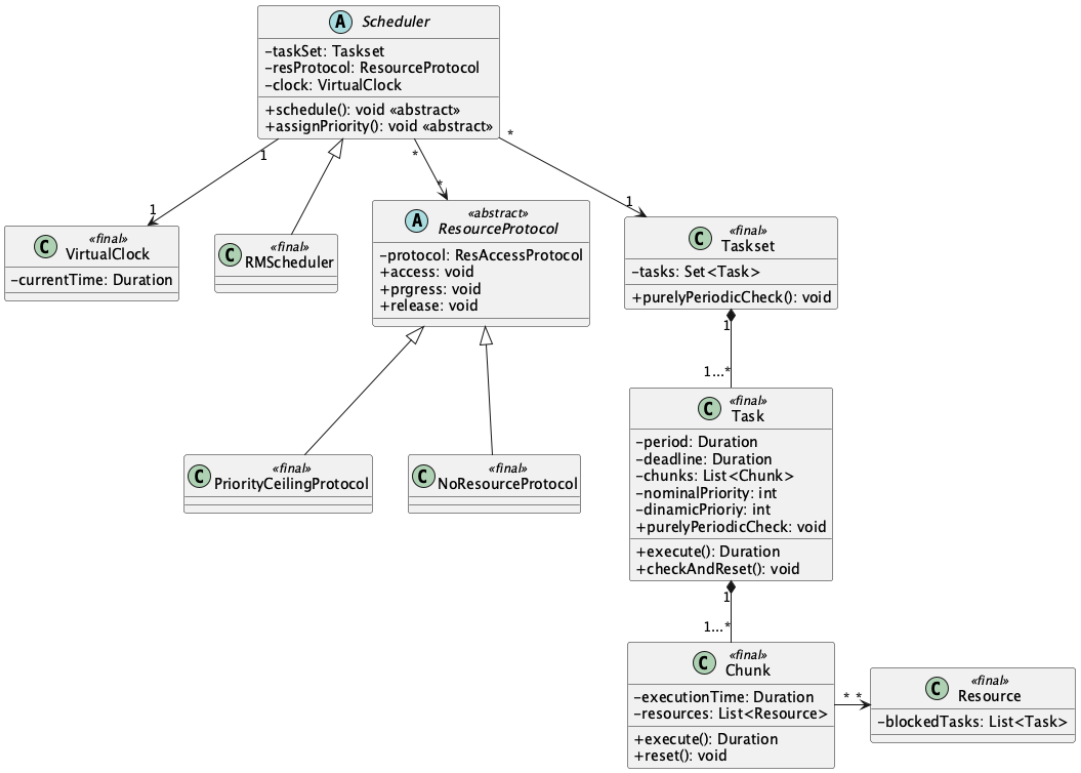

Figura 2.1: Class diagram.

3 Implementazione

3.1 Rate Monotonic

Descrivere come è stato implementato.

3.2 Priority Ceiling Protocol

3.3 Varie

3.3.1 Loggin

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

3.3.2 Sampling dei tempi

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.