Funzionamento Scheduler

Per prima cosa occorre capire quando e come uno scheduler viene chiamato dall'infrastruttura sottostante. Per fare ciò si sono creati due scheduler. Uno che estende direttamente la classe Scheduler e uno che estende la classe PriorityScheduler, ossia lo scheduler utilizzato a default dal sistema. Si sono estesi tutti i metodi della classe madre e si sono inseriti in essi dei breakpoint per osservare quando lo scheduler viene chiamato.

Esperimenti con lo scheduler vuoto

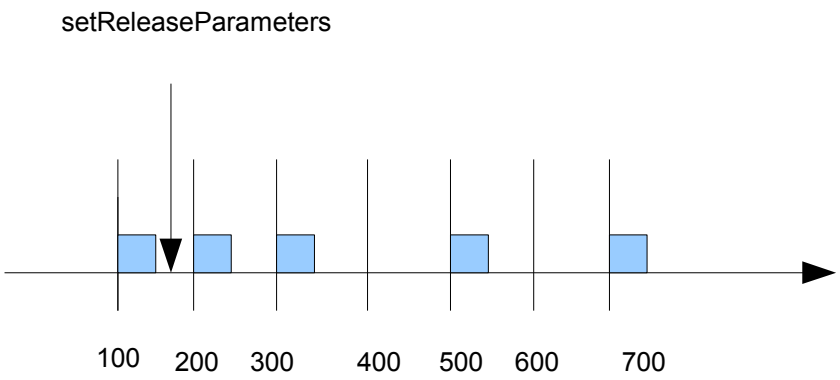
In prima battuta si è associato allo scheduler un solo processo periodico e lo si è lanciato. Sorprendentemente lo scheduler non è mai stato chiamato, neanche quando è stato eseguito il metodo setScheduler del thread. Dal momento che lo scheduler non entra in azione bisogna ipotizzare che ci sia un'entità associata al thread (o ai PeriodicParameters ad esso associato) che provvede a sbloccare il thread dalla waitForNextPeriod() quando giunge il nuovo periodo. Si può ipotizzare che sia lo stesso ServerThread che si occupa di controllare il rispetto di una eventuale deadline e di chiamare l'handler, se presente, in caso di missed deadline. Un'altra ipotesi è che, nel momento in cui il thread chiama il metodo waitForNextPeriod() viene calcolato il tempo rimanente al prossimo istante di realease e che il thread esegua una sleep pari fino al nuovo periodo.

In un secondo tempo ho preparato due threads periodici in modo che quello con priorità maggiore avesse l'istante di release in nel bel mezzo dell'esecuzione dell'altro. Il thread con priorità maggiore esercita preemption su quello a priorità minore, ma nessun metodo dello scheduler viene richiamato.

Analogamente, ho provato a creare un thread che sfora la deadline, con relativo handler associato. Tutto funziona come a defalut, senza che lo scheduler venga richiamato.

Ho provato a cambiare durate l'esecuzione gli scheduling parameters: qui viene lanciata un'eccezione(Exception in thread "launcher" java.lang.IllegalArgumentException at javax.realtime.RealtimeThread.setSchedulingParameters(RealtimeThread.java:1115)) dal metodo della classe RealTimeThread e non dallo scheduler. Provando a mettere delle interruzioni si fa fatica a capire il flusso dell'esecuzione e quali parametri il thread ha a disposizione.So solo che il processo che ci entra dentro il codice che lancia l'eccezione per primo si chiama cyclicCleaning thread, il quale è controllato dall'istanza di PriorityScheduler. Guardando con il debugger all'interno del threa d ci sono mille campi, ma è difficile capire se, e quali, concorrono nell'opera di gestione dei thread.

Ho provato a cambiare anche i ReleaseParameters, non vengono sollevate eccezioni né vengono eseguiti i metodi dello scheduler. Il comportamento del thread non cambia immediatamente, come ci si potrebbe aspettare. La cosa accade anche al caso in cui si usi il PriorityScheduler, anziche lo scheduler vuoto .



Il thread in esempio ha periodo 100; all'istante 180 gli vengono cambiati i parametri di release, imponendogli un periodo doppio: di 200 ms. La stranezza è che il nuovo comportamento è osservabile solo dall'istante 500, ossia ci sono due release con il vecchio periodo. L'analisi dettagliata di cosa avviene quando si cambiano i parametri di un Thread durante la sua esecuzione verrà affrontata a parte, in un documento separato.

Ho provato, infine a settare lo Scheduler vuoto come scheduler di defalut prima della creazione dei processi realTime, alla prima creazione di un oggetto schedulabile(processo o handler) ho un'eccezione nel costruttore della classe madre: per RealtimeThread ho

```
Exception in thread "launcher" java.lang.NullPointerException
        at
javax.realtime.RealtimeThread.setSchedulingParameters(RealtimeThread.java:1124)
        at javax.realtime.RealtimeThread.setScheduler(RealtimeThread.java:1093)
        at javax.realtime.RealtimeThread.<init>(RealtimeThread.java:455)
        at javax.realtime.RealtimeThread.<init>(RealtimeThread.java:433)
        at javax.realtime.RealtimeThread.<init>(RealtimeThread.java:355)
        at javax.realtime.RealtimeThread.<init>(RealtimeThread.java:340)
        at realtimeLibrary.schedulables.CpuDespotTrhead.<init>(CpuDespotTrhead.java:23)
        at prova18.Launcher18.run(Launcher18.java:47)
```

Nel costruttore c'è la chiamata al metodo setScheduler, dove presumibilmente gli si attribuirà come scheduler quello di default. E' molto strano il fatto che l'eccezione sia su un null poiter nel metodo setSchedulingParameters, quasi che gli scheduling parameters passati al metodo siano null.

E' ancora più interessente l'eccezione ottenuta quando si prova a creare l'handler:

```
Exception in thread "launcher" java.lang.ClassCastException: prova18.EmptyScheduler
cannot be cast to javax.realtime.PriorityScheduler
        at
javax.realtime.RealtimeThread.setSchedulingParameters(RealtimeThread.java:1126)
        at javax.realtime.RealtimeThread.setScheduler(RealtimeThread.java:1093)
        at javax.realtime.RealtimeThread.<init>(RealtimeThread.java:455)
```

```
        at javax.realtime.RealtimeServerThread.<init>(RealtimeServerThread.java:17)
        at javax.realtime.AsyncEventHandler.<init>(AsyncEventHandler.java:247)
        at javax.realtime.AsyncEventHandler.<init>(AsyncEventHandler.java:65)
        at
realtimeLibrary.schedulables.DeadlineMissedHandler.<init>(DeadlineMissedHandler.java:34)
        at prova18.SleepingHandler18.<init>(SleepingHandler18.java:30)
        at prova18.Launcher18.run(Launcher18.java:44)
```

Esaminandola osserviamo che è generata nel tentativo di creare un RealtimeServerThread. E' logico ipotizzare che sia il thread che ha il compito di monitorare il rispetto della deadline e di chiamare l'handler, se presente. Ipotizzo che nel costruttore del thread si associno ad essi dei PriorityParameters e che si voglia ottenere la priorità dallo scheduler di default, ipotizzando che sia il PriorityScheduler. Inutile sottolineare quanto questa scelta sia limitativa.

Prime Conclusioni

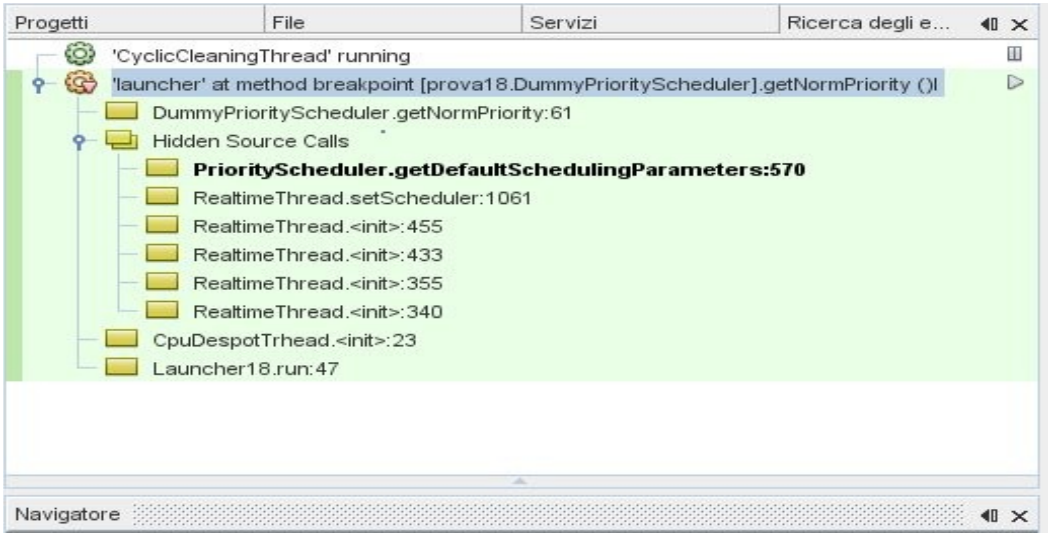
Possiamo concludere ipotizzando che tutta la parte di sincronizzazione sia fatta, in realtà, dai metodi statici della classe RealTimeThread e che lo scheduler venga sfruttato solo nella costruzione dei vari thread. Inoltre appare evidente come la classe RealTimeThread (il cui codice sorgente non è disponibile) abbia cablato all'interno dei suoi costruttori l'utilizzo di un PriorityScheduler. Per questo motivo si provvederà a creare uno scheduler che esporrà tutti i metodi di PriorityScheduler per vedere quando questi vengono utilizzati.

Estensione di PriorityScheduler

Per proseguire nell'analisi di ciò che accade quando si crea un oggetto schedulabile, come già detto si è creato uno scheduler, DummyPriorityScheduler, che estende tutti i metodi non privati della classe madre. Si è inserito un breakpoint in tutti i metodi in modo da vedere quando questi vengono chiamati e ricostruire lo stack dei metodi.Non è stato possibile esaminare direttamente l'utilizzo dei metodi non estendibili della classe PriorityScheduler, quali quelli con visibilità di package, ad esempio dei metodi accept. Si può tuttavia avanzare l'ipotesi che questi analizzino la validita “sintattica” dei paramtri ad esso passati: ad esempio,

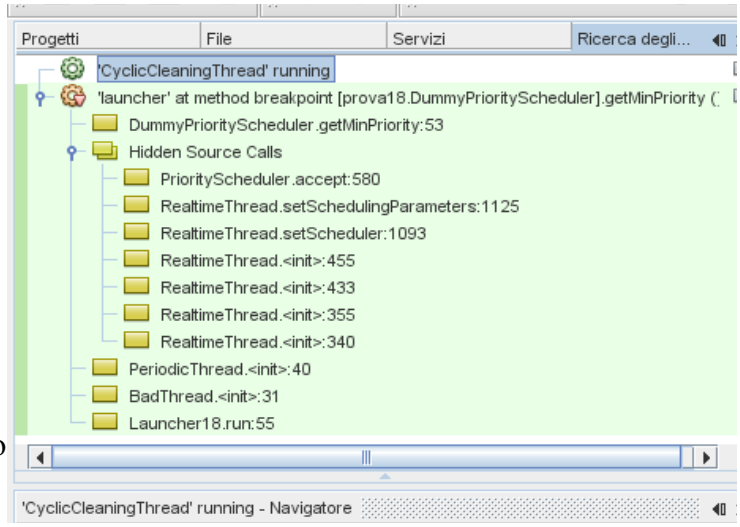
Costruzione Di Un RealTimeThread

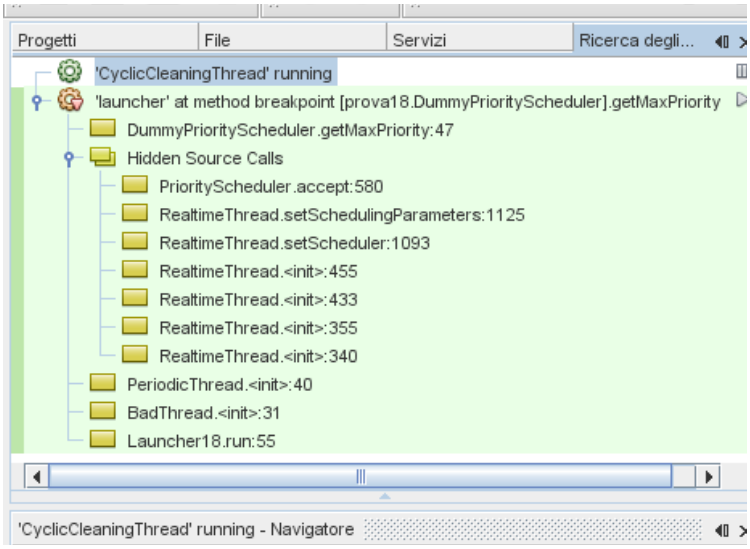
Per un RealtimeThread si ha una serie di interruzioni durante la costruzione dello stesso. Si mostrano li stack di chiamate e li si commenta



1. Come si può osservare il costruttore di default di RealTimeThread chiama il suo metodo setScheduler, il quale invoca getDefaultSchedulerParameters. A quanto pare questo metodo richiede all'istanza di PriorityScheduler qual'è la priorità di default. La cosa fa ipotizzare che i parametri di Scheduling di default di un RealTimeThread siano, in realtà, dei PriorityParameters e che, di conseguenza, ogni scheduler dovrebbe supportarli

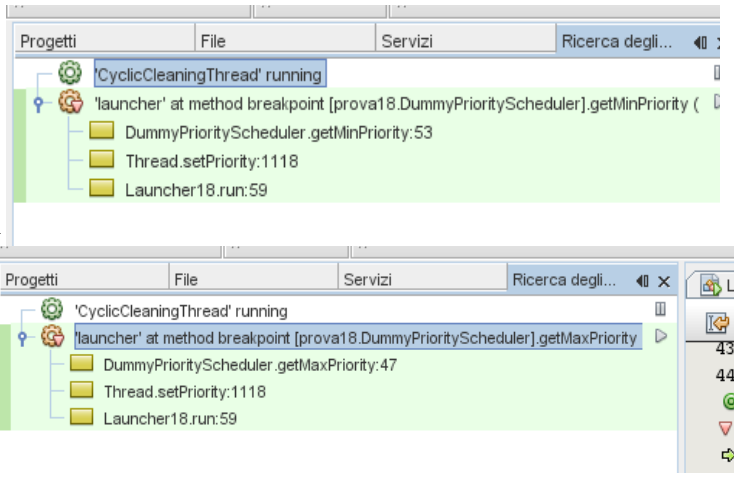
2. Successivamente il metodo setScheduler setta gli SchedulingParameters del thread realtme, probabilmente usando quelli di default ottenuti grazie alla chiamata mostrata sopra. Il metodo setSchedulingParameters chiama a sua volta il metodo accept di PriorityScheduler. La responsabilità di questo metodo sembra essere quella di controllare la validità dei nuovi parametri, difatti sembra che confronti la priorità dei parametri di scheduling con quella massima e minima per verificare che sia nel range di valori validi.

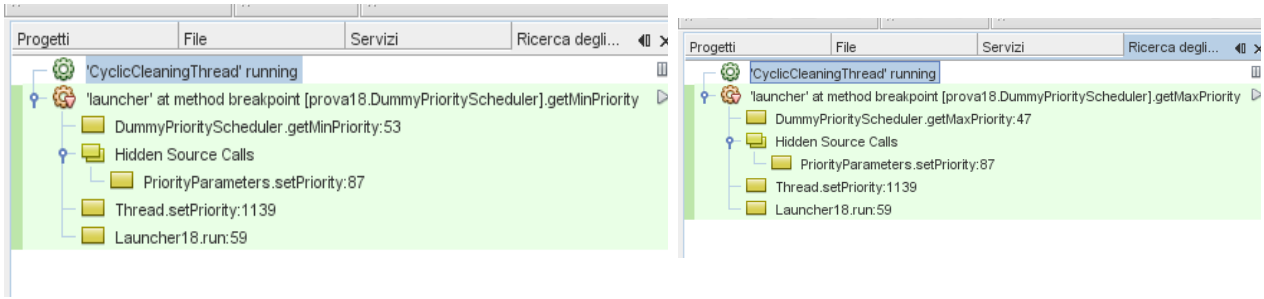




SetPriority

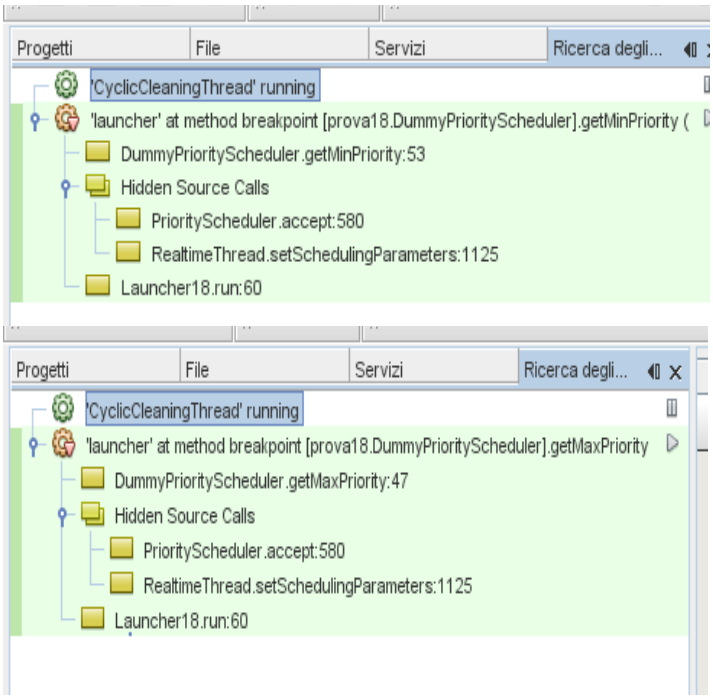
Nel caso in cui si modifichi la priorità di un RealtimeThread la nuova priorità viene controllata verificando che sia compresa tra la priorità minima e quella massima del sistema. Tale operazione viene fatta due volte, in quanto il sistema modifica sia la priorità del thread (attributo ereditato dalla classe java.lang.Thread) sia i gli SchedulingParameters associati al thread stesso.





SetSchedulingParameters

Nel caso in cui si modifichi la priorità del thread attraverso il metodo setSchedulingParameters, ossia si cambino i PriorityParameters del thread l'interazione tra RealTimeThread è leggermente diversa. Come si può osservare dagli stack di chiamate riportati in figura, viene invocato il metodo accept che controlla una sola volta che la nuova priorità sia nel range di quelle di sistema.



Creazione di un Handler e gestione di un'evento

Come già evidenziato dagli esperimenti eseguiti con lo scheduler vuoto, lo scheduler viene chiamato per la creazione del RealTimeServerThread associato alla gestione dell'evento in maniera analoga a quanto già documentato per un normale RealtimeThread. Lo scheduler, ed in particolare il metodo fireSchedulable, non vengono interpellati per la gestione dello sforamento della deadline.

Conclusione

La classe Scheduler non sembra avere grandi responsabilità nel sistema JavaRealTime.

Essa non sembra intervenire nella schedulazione a real-time dei vari processi.

Dagli esperimenti effettuati si piò ipotizzare che tutte le responsabilità di coordinamento siano sulle spalle dei metodi statici della classe RealtimeThread e della classe RealTimeServerThread.

Emerge che la classe Scheduler ha il compito di fornire i valori numerici delle priorità, oltre che controllare che le priorità associate ai vari thread siano nel range delimitato dalla priorità minima e quella massima del sistema.

Il costruttore di default della classe RealtimeThread associa alla nuova istanza dei

PriorityParameters anziche dei generici SchedulingParameters. Ciò impone, di fatto, che questi siano supportati da qualunque altro scheduler che si vorrà sviluppare il futuro.