Analisi del codice decompilato di JRTS

Scheduler

- La classe ha un attributo defaultScheduler, con i relativi getter e setter. Questi sono gli unici metodi implementati.
- Come era ipotizzabile, il metodo getDefaultScheduler ritorna l'istanza di PriorityScheduler se il campo defaultScheduler è null.
- La classe come sappiamo è astratta, ma nessun metodo lo è: tutti i metodi, esclusi il getter ed il setter di defaultScheudler, hanno però corpi vuoti in cui non fanno nulla o restituiscono null. Bisogna considerare che ciò potrebbe essere dovuto alla decodificazione.

PriorityScheduler

- I vari metodi accept, come avevo ipotizzato, hanno solo il compito di controllare la correttezza dei parametri che vengono loro passati.
- I vari metodi setIfFeaseble, come scritto nella specifica, non fanno in realtà nessuna analisi di feseability, si limitano a controllare i parametri passati ed a assegnarli.
- I metodi deputati a fornire i valori di priorità restituiscono valori diversi in base al thread che sta invocando i vari metodi, in particolare in base al fatto che sia un oggetto schedulabile (al quale corrispondono priorità real – time) o un thread normale.
- Il metodo fireSchedulable non è supportato, difatti il suo corpo consiste nel lancio di un'eccezione di tipo UnsupportedOperation Exception.
- Il metodo AddToFeasibility , che dovrebbe includere l'oggetto schedulabile nell'insieme di processi di cui valutare la schedulabilità non fa altro che chiamare il metodo addToFeseability dell'oggetto schedulabile. Questo metodo, a sua volta, non fa altro che settare un flag che mostra l'appartenenza o meno all'insieme degli oggetti di cui è valutata la schedulabilità.
- Manca una struttura dati che mantiene i riferimenti degli oggetti gestiti dallo scheduler.
- Esiste una serie di metodi nativi, probabilmente scritti in un linguaggio diverso da Java, due restituiscono la priorità minima e massima di un thread real-time, un altro metodo nativo si chiama registerNatives e viene chiamato nell'inizializzazione statica della classe.
- L'inizializzazione statica della classe prevede che venga eseguito il metodo register natives, successivamente viene inizializzato l'attributo dello scheduler di default con un'istanza di PriorityScheduler (che è un singleton), infine vengono impostati i valori di priorità massima e minima attraverso i due metodi nativi.

RealTimeThread

- Ipotizzando che il lavoro di sincronizzazione dell'esecuzione dei thread realtime sia fatto dai metodi statici della classe RealTimeThread, è da notare come manchi una struttura dati in grado di mantenere traccia dei thread attualmente attivi. Ogni thread realtime tiene traccia con un intero (realTimeThreadInitNumber) di un suo numero identificativo. Questo numero, però, sembra essere usato solo per dare un nome al thread, cosicché si hanno nomi tipo RealtimeThread-1; RealtimeThread-2 ecc

- E' presente, come avevamo, ipotizzato il flag che indica se un processo è deschedulato o meno

- Non ho trovato traccia, al contrario, del contatore di release pendenti.

- E' presente un riferimento ad un handler per eventi asincroni. Non so quando questo viene chiamato: so solo che viene creato con gli stessi parametri (di release, di memoria e di scheduling) del thread.

- Sfortunatemente i 3 metodi che mi interessavano maggiormente: schedulePeriodic, deschedulePeriodic e waitForNextPeriod sembrano offuscati.

- Il metodo schedulePeriodic controlla se il thread è ancora vivo e se è periodico, in tal caso esegue il metodo nativo schedulePeriodic0, se il thread non è vivo si limita ad impostare il flag descheduled a true

- Il metodo deschedulePeriodic è opposto al metodo precedente. Questo, se il thread è vivo ed è periodico richiama il metodo nativo deschedulePeriodic0. Se il thread non è vivo si limita a settare il flag descheduled a false

- waitForNextPeriod, dopo aver controllato che il thread corrente sia periodico chiama il metodo nativo waitForNextPeriod0.

RealtimeServerThread

- La classe RealTimeServerThread, le cui istanze dovrebbero essere i thread deputati ad eseguire gli handler di deadline miss, è praticamente vuota

- Implementa l'interfaccia com.sun.rtsjx.ServerThread. Andando ad aprire quest'interfaccia la troviamo vuota (forse per colpa del decomplilatore)

- La sola cosa presente nella classe è un costruttore che non fa altro che richiamare quello della classe madre

- Tra i parametri del costruttore c'è un oggetto che implementa l'interfaccia java.lang.runnavle. Pensavo che questo fosse l'oggetto contenente la logica da eseguire nel metodo run. Guardando però il corrispondente costruttore della classe si scopre che che il riferimento in questione non viene mai usato.