Specifiche progetto Sistemi Operativi sessione autunnale A.A. 2010/2011
Docente Emanuele Lattanzi

1. Specifiche generali

Scrivere un programma multithread che simuli il sistema di controllo degli atterraggi e dei decolli in un aeroporto. In particolare, il programma dovrà gestire il traffico aereo evitando collisioni e valutando il tempo medio d'attesa degli aerei per le richieste di decollo e per le richieste di atterraggio.
Il programma dovrà essere costituito da un oggetto "aeroporto" (AEROPORTO), che avrà a disposizione due piste di atterraggio/decollo, e che sarà un oggetto condiviso tra un numero arbitrario di thread (*n*) che rappresentano gli aerei (AEREO) ed un thread gestore del traffico (GESTORE) che rappresenta il sistema di gestione del traffico.

Ogni aereo potrà essere del tipo "privato" o "di linea" ed effettuerà una richiesta di atterraggio o di decollo. Ogni richiesta di atterraggio verrà sottomessa al gestore dell'aeroporto specificando il tipo di aereo "privato" o "di linea" ed un parametro che identifica il peso dell'aereo in tonnellate (espresso come numero intero compreso tra 100 e 200).
Il sistema di gestione dell'aeroporto dovrà evadere le richieste (facendo atterrare e decollare gli aerei secondo la politica di gestione del traffico descritta in seguito.
Il programma terminerà quando ogni aereo avrà effettuato una coppia di richieste decollo/atterraggio o atterraggio/decollo.

2. Politica di gestione del traffico

Secondo la politica di gestione del traffico le richieste di atterraggio avranno sempre precedenza rispetto alle richieste di decollo. All'interno della stessa tipologia di richiesta, gli aerei "privati" avranno sempre precedenza rispetto agli aerei "di linea", a parità di categoria di aerei ("privati" o "di linea") avranno precedenza le richieste fatte da aerei più pesanti rispetto a quelle fatte da aerei più leggeri; ad ulteriore parità di peso le richieste saranno gestire in modalità FCFS.

3. Specifiche dell'oggetto AEROPORTO

- L'oggetto AEROPORTO sarà un oggetto condiviso tra gli *(n)* thread che rappresentano gli aerei ed il thread che rappresenta il gestore del traffico.
- L'oggetto AEROPORTO dovrà emulare un normale aeroporto dotato di due piste di atterraggio/decollo. Chiaramente, vista la presenza di due piste indipendenti, sarà possibile evadere due richieste contemporaneamente.
- Ogni pista sarà occupata per il tempo necessario a far decollare o atterrare un aereo. In particolare, il tempo d'atterraggio sarà determinato secondo la seguente relazione 50+1*peso_aereo (ms); mentre il tempo di occupazione di una pista per un decollo sarà determinato secondo la relazione 20+1*peso_aereo. Saranno gli aerei a simulare il tempo di occupazione di una pista e a notificare al gestore del traffico quando questa sarà di nuovo libera.

4. Specifiche dell'oggetto GESTORE

- L'oggetto GESTORE sarà un thread che avrà il compito di smistare le richieste di atterraggio/decollo secondo la politica descritta in precedenza.
- Quando non ci sono richieste da servire il GESTORE rimarrà in attesa di richieste sospendendo la propria esecuzione (senza occupare la CPU).

- Ogni aereo richiederà di atterrare o di decollare mediante l'invocazione di un metodo *richiediServizio(….)*. Se la richiesta di servizio arriva mentre il GESTORE è inattivo la richiesta stessa attiverà il GESTORE il quale inizierà a gestirla. Tutte le richieste pervenute mentre il GESTORE sta gestendo un'altra richiesta oppure mentre tutte le piste di atterraggio/decollo sono occupate verranno messe in un'apposita coda o code.
- Le richieste di servizio dovranno essere bloccanti sospendendo l'AEREO richiedente per il tempo necessario ad ottenere la pista.
- Le richieste presenti nella coda verranno servite dal GESTORE secondo la politica precedentemente descritta.
- Ogni AEREO comunicherà al gestore di aver liberato la pista che gli era stata concessa attraverso l'invocazione di un metodo *liberaPista(…..)*.

5. Specifiche dell'entità AEREO:

- L'entità AEREO dovrà emulare un aereo che richiede di atterrare o di decollare dall'aeroporto. Ogni AEREO dovrà essere emulato attraverso l'utilizzo di un singolo thread.
- Ogni AEREO dovrà essere inizializzato con almeno i seguenti campi:
  - Identificativo interno numerico ID,
  - Tipo d'aereo {"privato", "di linea"} scelto casualmente al momento dell'inizializzazione con probabilità 0.5 per ogni tipo.
  - Peso aereo scelto casualmente nell'intervallo intero [100,200] tonnellate.
- Il comportamento di ogni entità AEREO può essere descritto dai seguenti passi:
  1. Effettua una richiesta di atterraggio o decollo (scelta randomicamente con probabilità 0.5) specificando il proprio tipo ed il proprio peso.
  2. Attende finché non gli viene concessa una pista per l'atterraggio/decollo.
  3. Simula l'atterraggio/decollo sospendendosi per un tempo pari a 50+1*peso_aereo (ms) se in atterraggio o 20+1*peso_aereo (ms) se in decollo.
  4. Comunica al gestore del traffico di aver liberato la pista.
  5. Effettua la richiesta complementare alla precedente (se aveva richiesto un decollo ora richiederà un atterraggio e viceversa)
  6. Esegue nuovamente i punti 2, 3 e 4
  7. Termina la propria esecuzione

6. Specifiche di sincronizzazione ed implementazione:

- Il progetto deve essere sviluppato in ambiente Linux/Windows/Mac OS utilizzando **esclusivamente il linguaggio java.**
- La muta esclusione, l'accesso a variabili e a metodi condivisi e l'attesa dei thread dovranno essere ottenute utilizzando le primitive di sincronizzazione messe a disposizione dal linguaggio java (ad esempio: package java.util.concurrent.) ad esclusione del costrutto monitor (per chiarezza non sarà possibile utilizzare il costrutto *synchronized*) .

7. Specifiche dati in input:

- Il programma dovrà ricevere in input, tramite riga di comando o tramite file di specifica, i seguenti parametri:
- Numero di aerei *(n)*.

8. Specifiche dati di output:

- Al termine della simulazione il programma dovrà scrivere, su di un file di testo oppure sulla consolle, i tempi medi d'attesa per le richieste di decollo di aerei privati, per le richieste di decollo di aerei di linea, per le richieste di atterraggio di aerei privati e per le richieste di atterraggio di aerei di linea (*tempoAttesa* - tempo che intercorre tra la richiesta di un servizio e l'ottenimento della pista di atterraggio/decollo). Per rilevare i tempi d'attesa è consigliato l'utilizzo del metodo currentTimeMillis() della classe System.

9. Documentazione

- Il progetto deve essere corredato dalla seguente documentazione:
  - codice sorgente commentato nei punti salienti
  - eventuali file di configurazione dell'applicativo
  - relazione in formato PDF o Word contenente:
    - specifiche richieste
    - schema a blocchi (sufficientemente dettagliato) degli elementi progettuali del codice: classi, oggetti e loro connessione logica
    - descrizione della progettazione
    - descrizione dell'implementazione
    - testing e commenti ai risultati ottenuti.

- La relazione, in formato PDF, deve essere inserita in una directory (denominata docs), da allegare al progetto stesso. Vedere il seguente link per una relazione d'esempio: http://www.sti.uniurb.it/lattanzi/OS/Relazione_SO.pdf

10. Modalità di presentazione del progetto:

- Il presente progetto può essere presentato fino ad UNA SETTIMMANA prima dell'appello d'esame in cui s'intende sostenere la prova orale.
- La consegna del progetto dovrà avvenire esclusivamente per posta elettronica all'indirizzo lattanzi@sti.uniurb.it avendo l'accortezza di inserire nel campo subject i seguenti dati:

  Progetto SO/OSys - <numero matricola> - Nome_e_cognome.

  SO: per il corso di sistemi operativi percorso in presenza
  OSys: per il corso di Operatine System percorso on-line

- Per ulteriori informazioni su data e ora di consegna del presente progetto consultare il sito personale del docente al link: http://www.sti.uniurb.it/lattanzi/OS.html
- Il progetto deve pervenire, come allegato, sotto forma di file compresso in modalità .tar.gz o zip.
- Progetti consegnati fuori tempo limite non verranno accettati.
- Progetti consegnati senza relazione oppure accompagnati da relazione che non soddisfi le specifiche richieste non saranno ritenuti sufficienti.

11. Divieto di plagio:

- Ogni progetto, elaborato autonomamente e personalmente da ciascuno studente, deve essere originale. Codice, o porzione di codice che verrà identificato come "copiato" da altre fonti

comporterà l'automatica valutazione insufficiente del progetto. Fanno doverosamente eccezione a questa norma i riferimenti a packages e librerie reperite nei repository pubblici per i quali sarà comunque necessario citarne la fonte d'origine in un'apposita sezione bibliografica della relazione.

- Progetti elaborati da persona o persone diverse dallo studente che li sottomette saranno automaticamente valutati insufficienti.

12. Avvertenze generali:

- Le presenti specifiche sono valide a decorrere dalla loro data di pubblicazione fino alla fine della sessione d'esame esplicitata nell'intestazione del presente documento.
- La presente specifica rende nulla qualsiasi altra specifica pubblicata in data antecedente, e qualsiasi informazione, relativa alla stesura, implementazione, consegna e valutazione del progetto, reperita da qualsivoglia altra fonte ufficiale e non.