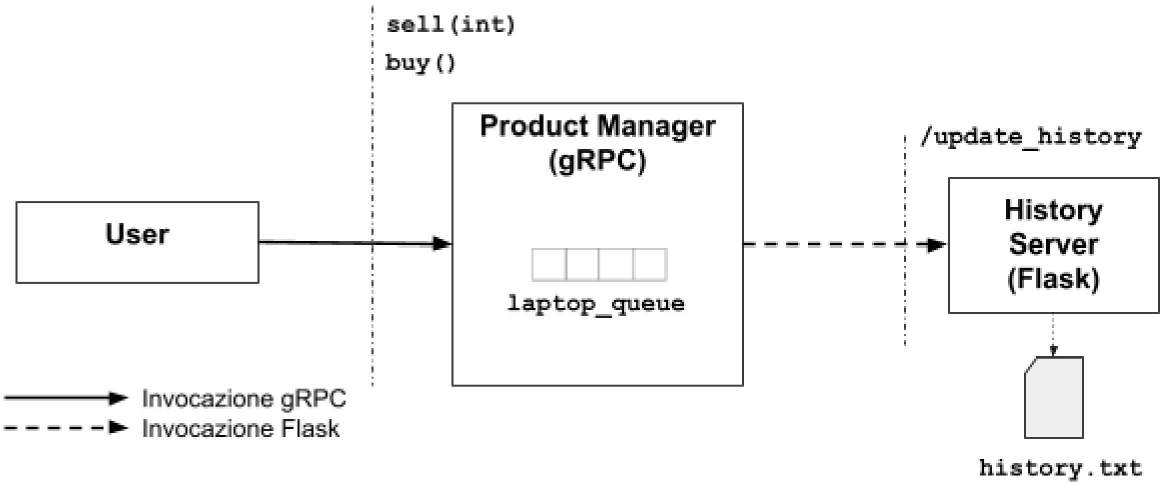
Università degli Studi di Napoli Federico II
Esame di Advanced Computer Programming

Proff. De Simone, Della Corte

Prova pratica del giorno 26/06/2024
Durata della prova: 120 minuti

Lo studente legga attentamente il testo e produca il programma ed i casi di test necessari per dimostrarne il funzionamento. Al termine della prova lo studente dovrà far verificare il funzionamento del programma ad un membro della Commissione.

Testo della prova



Il candidato implementi un sistema distribuito in **Python** per la compravendita di laptop basato su **gRPC** e **Flask**. Il sistema è caratterizzato dai seguenti componenti.

User. E’ un client utilizzato per richieste di acquisto/vendita verso il **Product Manager**. L'invio di una richiesta di acquisto consiste nella invocazione del metodo *buy( )*, che non prevede parametri. L'invio di una richiesta di vendita consiste nella invocazione del metodo *sell(int)*. La richiesta è caratterizzata dal **serial_number** (int) che identifica il laptop da vendere. User avvia 10 thread: ogni thread genera o una richiesta di vendita, invocando il metodo *sell* e generando casualmente il *serial_number* del laptop (intero tra 1 e 100), oppure una richiesta di acquisto invocando il metodo *buy*.

Product Manager. E’ un server gRPC che espone i metodi *sell* e *buy*. Il metodo *sell* inserisce l’ID nella coda *laptop_queue*. Il metodo *buy* consuma un prodotto dalla coda *laptop_quque*. N.B: E’ necessario utilizzare una lista per implementare la coda, prevedendo i meccanismi di sincronizzazione per il problema produttore/consumatore; la coda ha dimensione pari a 5. Prima di ritornare, i metodi *sell* e *buy* generano un **richiesta di tipo POST** verso l’**History Server**, inserendo nel body il *tipo di operazione effettuata* (cioè sell o buy) ed il *serial_number del laptop*, in formato json, e.g., {"operation":"sell", “serial_number”:10}, attendendo la risposta prima di ritornare. Il metodo *buy* ritorna al chiamante l’ID del prodotto estratto dalla coda, mentre il metodo *sell* un semplice ack (valore booleano).

History Server. Implementa un server Flask che espone una REST API con l’endpoint *update_history*. Tale endpoint accetta richieste di tipo POST, con payload in formato *json* (descritto in precedenza). Ricevuta una richiesta, l’History Server scrive (in append) sul file *history.txt* una stringa che è la concatenazione dei due campi ricevuti tramite il payload della POST, cioè operation ed serial_number, e.g., sell-20.