Progetto a.a. 2010-2011: realizzazione di un sistema per l'update del software

Sistemi per l'installazione e l'aggiornamento del software sono ormai disponibili per tutti i principali sistemi operativi. Compito di questi sistemi è facilitare le operazioni mensionate mantenendo, in un sistema a versioni, i pacchetti software disponibili e le loro dipendenza da altri pacchetti software. Di norma la realizzazione di tali sistemi prevede la presenza di un database centralizzato e di un demone atto a gestirlo. La gestione comprende sia la possibilità di aggiornare l'archivio (aggiungendo nuovi pacchetti o aggiornando pacchetti già presenti, sia la possibilità di rispondere a richieste da parte dei client. Per client si intendono i computer mantenuti aggiornati grazie al sistema in questione. Per poter essere aggiornato un client deve utilizzare un apposito programma in grado di interfacciarsi con il repository. Ogni client mantiene un elenco (locale) dei pacchetti software installati, delle loro versioni e dipendenze.

Scopo del progetto sarà realizzare un sistema (semplificato) per il supporto dell'installazione e l'aggiornamento del software. Tale sistema sarà costituito da tre programmi principali:

(1) il server che gestisce il repository centralizzato,
(2) il client di upload,
(3) il client di download.

Per semplicità ***non*** gestiremo le dipendenze di un pacchetto software da altri pacchetti software.

IL SERVER

Compito del server è gestire l'archivio centrale e rispondere alle richieste dei diversi client.

Caratteristiche del server:

1. l'implementazione deve prevedere una **fase di avvio** nella quale vengono allocate le risorse di IPC necessarie e vengono inizializzati i dati del repository;
2. le **richieste** vengono inviate al server sotto forma di **messaggi**, inseriti in un'unica coda, accessibile a tutti i processi;
3. ogni richiesta prevede la **gestione** da parte di un **processo figlio del server**. ***ATTENZIONE***: Per evitare sovraccarichi in ogni istante possono coesistere **al più 5 server-figli** (non si possono gestire più di 5 richieste contemporaneamente). Il controllo sulla generalibilità di un server figlio deve essere regolamentato tramite l'uso di semafori.
4. I dati condivisi saranno realizzati tramite una o più aree di **memoria condivisa**.
5. L'accesso ai dati condivisi da parte dei diversi processi server va regolamentato tramite l'uso di **semafori**.
6. Se la richiesta è di **upload**, il server deve innanzi tutto controllare che il r**ichiedente sia accreditato** (decidete voi come implementare il meccanismo di accreditamento);
7. ogni volta che viene fatto un upload, il server informerà tramite l'invio di un **segnale** i client di download che si sono registrati per un *servizio di tipo "push"* (vedi sotto);
8. se la richiesta è di **download**, verrà fornita l'informazione richiesta.
9. L'implementazione deve prevedere una **fase di shutdown**, nella quale vengono disallocate tutte le strutture condivise e viene rimosso il repository. **NB**: lo shutdown può avvenire solo dopo che tutte le richieste in coda sono state soddisfatte.

10. Tutta l'attività dei processi server (padre e figli) va memorizzata in un **file di log** nel seguente modo: i vari processi scrivono in **mutrua esclusione** in un'area di **memoria condivisa** di dimensioni limitate. Quando queste risulta piena, il processo che in quel momento è scrittore la riversa in un **file di log** del server, aperto in append, e la svuota dimodoché sia possibile continuare a registrare l'attività.

Il client di download

Compito del client di download è di interrogare il repository centrale in merito alla presenza di pacchetti di interesse da installare o di aggiornamenti di pacchetti di interesse e procedere con l'installazione/l'aggiornamento.

Caratteristiche del client di download:

1. si avranno tanti client di download quanti clienti (sono **processi dedicati**).
2. Ogni client mantiene un **repository locale** con le informazioni su tutti i pacchetti software installati (decidete voi come realizzarlo).
3. Come prima operazione, il client scaricherà in locale la **versione corrente** di tutti i pacchetti software presenti nel repository centralizzato.
4. Un client di download può effettuare la **registrazione per un servizio di tipo push**: in questo caso verrà informato dal server, tramite un **segnale** che andrà poi gestito, della presenza di aggiornamenti o di nuovi software da scaricare.
5. I client **non registrati** per il servizio push, a **intervalli di tempo regolari** inviano una **richiesta** al server, sotto forma di **messaggio**, domandando quali pacchetti sono stati aggiornati, attendono la risposta e poi procedono con l'aggiornamento (che sarà solo simulato: verrà aggiornata solo la tabella locale dei client).
6. Deve essere possibile chiedere informazioni su **uno specifico pacchetto**, per installarlo (cioè aggiungerlo al repository locale) o aggiornarlo (cioè modificare le info nel repository locale).
7. Tutta l'attività di richiesta/recezione di ciascun client va salvata in un **file di log** (diverso per ogni client).

Client di upload

I client di upload consentono di modificare le informazioni contenute nel repository centrale.

Caratteristiche dei client di upload:

1. **registrazione del client**: l'upload è consentito solo se il client è autorizzato. Quindi la prima operazione che un client di upload deve compiere è accreditarsi, inviando un'opportuna richiesta al server tramite un **messaggio** inviato nella stessa coda utilizzata dai client di download. Solo dopo aver ottenuto un id opportuno dal server, il client di upload potrà effettuare richieste di modifica del repository.
2. **Richiesta di aggiunta**: richiesta che un nuovo pacchetto venga inserito nel repository. Ogni richiesta di aggiunta richiede un feedback sul risultato dell'operazione da parte del server-figlio che la gestisce. **NB**: Non è possibile aggiungere software con un nome identico a altri software già nel repository.
3. **Richiesta di aggiornamento**: richiesta di modifica della versione di un pacchetto già presente. Ogni richiesta di aggiornamento richiede un feedback sul risultato dell'operazione da parte del

server-figlio che la gestisce. **NB**: Non è possibile inserire un aggiornamento il cui numero di versione è inferiore o identico a quello già associato al medesimo software nel repository.

4. Ogni client di upload registrerà in un **file di log** la propria attività, dimodoché sia ispezionabile.

MAKEFILE

Non saranno considerate accettabili soluzioni che non prevedano un **makefile** che consenta la compilazione efficiente di tutti i programmi realizzati.

Simulazione: generazione dei client

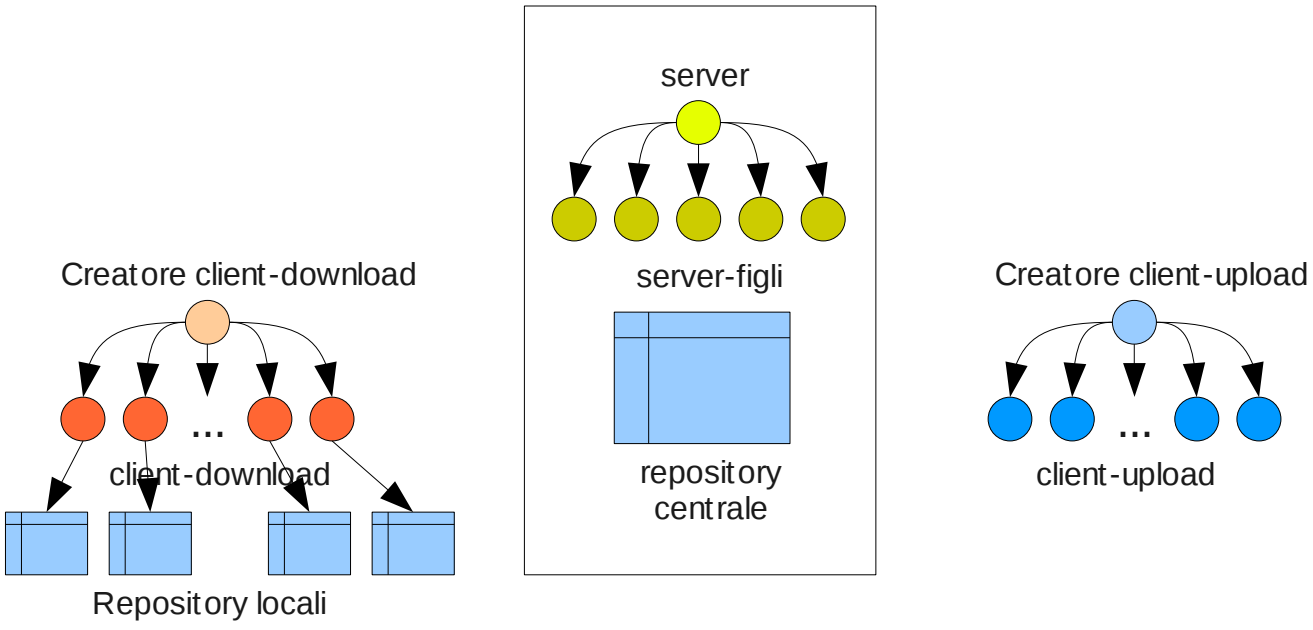
Tutti i client di download devono essere generati da uno stesso processo genitore.
Stessa cosa per utti i client di upload.

Attenzione che la generazione e l'avvio dei client ha senso solo dopo che il server è stato avviato. Quindi gestire anche il caso di inversione dell'ordine di avvio.

Simulare il ciclo di vita di un client

Per rendere realistico il funzionamento di un client, ogni cliente eseguirà un programma di attività specificato da voi in un file opportuno tramite un linguaggio di specifica inventato ad hoc. Scrivete quattro o cinque di tali programmi di attività e poi fate sì che ogni client ne "peschi" uno e si attenga ad esso.

Schema generale dei processi e dei repository



Rissunto delle strutture di IPC da utilizzare

1. Server:
   - una o più aree di memoria condivisa per mantenere il repository centrale
   - un'area di memoria condivisa per realizzare il log buffer
   - un numero adeguato di semafori necessari per sincronizzare l'accesso ai dati condivisi: tutti i semafori vanno allocati in un'unica struttura. Userete quindi sempre un solo "semid"
   - una coda di messaggi per ricevere le richieste
2. Client-download:
   - se non si registra per il servizio push: usa le strutture allocate dal server
   - se si registra per il servizio push: le strutture che servono per realizzare la sospensione in attesa di un segnale
3. Client-upload:
   - usa le strutture allocate dal server

Riassunto dei file di log da utilizzare

1. Server:
   - un file di log unico per tutti i processi di tipo "server"
2. Client-download:
   - un file di log per ciascun client
3. Client-upload:
   - un file di log per ciascun client

Linea guida nella realizzazione del repository

Definire la forma dei dati mantenuti nel repository non deve essere il problema più grosso di questo progetto, il cui scopo è costruire un'adeguata struttura di sincronizzazione e comunicazione fra i processi. Il consiglio è di utilizzare una struttura di minima. Per esempio una tabella contenente: *nome del programma*, *numero di versione*.

Attenzione che le strutture dati condivise non possono essere dinamiche, quindi il repository potrà contenere informazioni su un numero massimo predefinito di pacchetti software.

Le strutture condivise dovranno anche comprendere le informazioni (condivise) necessarie per il loro corretto utilizzo. Per esempio il numero di pacchetti software attualmente memorizzati.