Trabalho Prático de Sistemas Distribuídos

Professor: Vítor Duarte

Docente: David Navalho

Turno Prático: P3

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Grupo:

João Martins 28194 P3

Pedro Miranda 28289 P2

LEI, FCT-UNL

Introdução

Nos tempos que correm, a maioria dos utilizadores de dispositivos tecnológicos tem informação "espalhada" por vários dipositivos (computadores, portáteis, PDA's, etc). Daí a importância de encontrar uma maneira de sincronizar toda esta informação e de a manter actualizada.

Pretende-se implementar um sistema que permita aceder e manter sincronizados um conjunto de ficheiros, ou repositórios, entre vários dispositivos assim como oferecer para consulta a outros utilizadores do sistema alguns destes ficheiros. Mesmo que não haja um servidor disponível é útil que os clientes se consigam sincronizar entre si. Será utlizado Java RMI na implementação deste sistema.

Descrição da Implementação

A arquitectura que escolhemos para a nossa implementação foi Cliente Pesado / Servidor, assim aliviamos a carga do servidor ao or o cliente a fazer as comparações de datas da útlima modificação dos ficheiros e a pedi-los ao servidor, este limita-se a envia-los para o cliente.

O cliente começa por enviar a sua chave pública(se não existir será criado e guardado um par de chaves) a um servidor com um IP específico (introduzido pelo utilizador), se isto falhar envia a sua chave pública aos servidores multicast, os servidores recebem-na e enviam a referência remota do servidor para cliente (que receberá uma das referências enviadas). Se o cliente não se conseguir ligar a nenhum servidor, lança um thread com um servidor e liga-se ao mesmo.

Para a segurança utilizámos as classes das aulas práticas. Os sockets são criados e caso não tenham um cifra defina(no ínicio não a terão) combinaram usar um chave de sessão, trocando pelos canal seguro com chaves assimétricas, depois passam a usar a chave simétrica. O servidor, quando arranca, obtém um certificado duma instância CertAuthority. O seu IP será dado através da consola, também lança um thread com um ser vidor multicast. Depois de estabelicida a conexão, o cliente pede o certificado ao servidor (classe Certificate) e a chave pública da CertAuthority e verfica o certificado. Se for válido imprime a mensagem "Certified Server", caso contrário imprima "Uncertified Server". Os certificados, nesta implementação não têm um data de validade, por simplicidade. Estas interacções com o servidor é feita por Java RMI.

Extras

Um utilizador pode dar permissões de leitura e escrita sobre um ficheiro que esteja no seu repositório (definido pela pasta "<username>'s repo"). Os ficheiros aos quais tem acessos(escrita ou leitura) serão postos numa pasta com o nome do dono. O utilizador pode fazer upload dos ficheiros que tem permissões de escrita, mas não dos que tem apenas permissão de leitura. Todas as operações de escrita no servidor são "synchronized" utilizando o repositório como monitor, assim só a escrita num repositório é mutuamente exclusiva. Quando os ficheiros são guardados, a data da sua última actualização é escrita com o valor que está também no servidor, assim poupam-se transferências de ficheiros desnecessárias. Um cliente pode ainda reportar um utilizador, esse informação será guardada no servdior e ao fim de 3 "reports" de utilizadores distintos, o utilzador reportado será banido. O servidor guarda os ficheiros do utilizador numa directoria "server/<username>/<username>'s repo". O cliente guarda os ficheiros numa directoria "user/<username>/<username>'s repo". A classe Repository guarda a informação relativa a um repositório (o seu path e o dono). A classe UserInfo guarda o repositório do utilizador mais os repositórios aos quais tem acesso (e os respectivos donos). No servidor as contas dos utilizadores são guardadas em mapas que usam o username do utilizador como chave e que são escritos no disco para o servidor ter a informação dos utilizadores da próxima vez que for executado. Ainda mais, ciframos os ficheiros que são guardados nos servidores com uma chave simétrica, que só é utilizada para isso.

Funções do Cliente:

- Registar-se;
- Fazer login;
- Sincronizar-se com o servidor;
- Dar permissões(leitura ou escrita) a outro utilizador
- Reportar um utilizador
- Pedir os ficheiros dos repositórios a que tem acesso(leitura e escrita);
- Log out .

Funções do Servidor:

- Adicionar um utilizador;
- Fazer o login dado um username e um hash da palavra-passe (introduzidas pelo utilizador);
- Devolver a data da última alteração de um ficheiro;
- Fazer upload/download de ficheiros (em ambas as operações o cliente envia o seu username);
- Registar a informação de clientes reportados (se 3 clientes diferentes reportarem o mesmo utilzador, este é banido).

Como Correr

O programa pode ser corrido no ambiente Eclipse (importando as classes) ou através da consola (utilizando "javac *.java" para compilar as classes).

Em ambos os casos deve ser primeiro lançado o CertAuthority, depois o Server e finalmente o Client. (Com a nossa implementação não é necessário correr os dois primeiros, pois o cliente pode ser um servidor). Para correr as classes compiladas na consola utiliza-se o comando, por exemplo, "java Client". Quando o Server corre é preciso dar o IP para o servidor central, ou uma String vazia para escolher o IP default (basta carregar enter). No Client também é preciso dar o IP do servidor, ou uma String vazia para o IP default).

Os programas pedem o IP quando correm, por isso, não é preciso dá-lo como argumento, ou seja, o IP não está no String[] args.

O sistema tem já 4 utilizadores guardados no ficheiro users:

- mallory – utilizador banido;
- eve – utilizador com 2 ficheiros;
- alice – com 1 ficheiro(alicefile) e permissões de escrita sobre o bobfile de bob;
- bob - com 1 ficheiro(bobfile) e permissões de leitura sobre o alicefile de alice.

Para os 4 utilizadores a password é igual ao seu nome. Os nomes são todos em minúsculas.