Eviter les threads dans un Serveur Corba

Si dessous est présenté une alternative à la méthode run() de l'ORB en Corba.

Problématique: la méthode run() de l’ORB permet de se placer à l’écoute des requêtes Corba (IIOP). Mais elle est bloquante!

Exemple classique en C++:

{mostip}
//Si on a l'objet ORB obtenu par init
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

// - methode 1: on place orb->run() qui est bloquant
orb -> run();
{/mostip}

Comment faire des traitements en parallèle de l’ORB dans un programme serveur?

- 1/Utiliser des Thread, par exemple JTC de Iona en C++, ou directement avec Java
- 2/Utiliser l’alternative à run() ci dessous

L’alternative à run() :

- boolean work_pending(): Non bloquant, indique si une requête Corba est arrivée
- void perform_work(): Non bloquant, traite les requêtes Corba en attente

il faut utiliser le couple d'appels orb.work_pending()/orb.perform_work() qui remplacent orb.run() et qui peuvent être utilisées dans la boucle principale du programme.

mise en œuvre en C++:

{mostip}

// méthode 2: on utilise orb.work_pending() qui est non bloquant
int x=0;
while (1) {
cout << "je suis dans la boucle de traitement depuis «
<<(x++)<<" secondes" << endl;

//orb.work_pending() fonctionne de paire avec orb.perform_work()
if (orb->work_pending())
orb->perform_work();

//pause 1 second (défini dans dos.h) pour éviter de bloquer le processeur
sleep(1);
}
{/mostip}

Mise en œuvre en Java:

{mostip}

int x=0;
while (true) {
System.out.println("je suis un client qui boucle depuis "
+(x++)+" secondes");
try {Thread.sleep(1000);} catch (Exception e){;};

```
            //methode 2: on utilise orb.work_pending() qui est non bloquant
            //orb.work_pending() fonctionne de pair avec orb.perform_work()
            if ( orb.work_pending() )
                orb.perform_work();
        }
{/mostip}
```

Enfin, et pour information, le must est d'utiliser le service Corba "Event" ou "Notification" qui a été créé précisément pour gérer des messages asynchrones bidirectionnels.