Projet Programmation système (M1 2014-2015)

Appels de Procédures Distantes, en version « locale »

But du projet

On vous demande de réaliser (en C) un système permettant à diverses applications de partager leurs compétences, en autorisant chaque application à faire appel à des fonctions définies par les autres. Ce mécanisme est appelé *Remote Procedure Call* ou *RPC* dans son cadre général – appel à des procédures définies par une application tournant *a priori* sur un serveur distant. Nous nous contenterons ici d'une version locale, limitée à une seule machine.

Deux modèles sont à envisager :

le modèle client/serveur dans lequel un seul serveur héberge les procédures et a pour unique rôle de répondre aux requêtes de calcul ; les clients s'adressent donc systématiquement au serveur pour obtenir un calcul particulier.
le modèle distribué dans lequel chaque processus fait profiter les autres de ses compétences particulières : chacun déclare publiquement les services qu'il peut offrir, et joue alternativement le rôle du client (s'il a besoin de faire effectuer un calcul par un autre processus) et du serveur (si un autre processus lui demande un service).

Modèle client/serveur

Dans ce modèle, un seul processus répond aux requêtes de calcul des autres processus impliqués ; il doit traiter toutes les requêtes reçues, et renvoyer à chaque demandeur le résultat du calcul correspondant. Le serveur a donc trois types de tâches à effectuer :

- lire les requêtes ;
- exécuter les calculs nécessaires ;
- transmettre le résultat.

Dans l'hypothèse où un client demanderait un long calcul tandis que d'autres ont des requêtes plus raisonnables, il est souhaitable que le serveur ne reste pas bloqué sur le gros calcul. Une solution consiste à déléguer une partie du travail (choisie selon une procédure à définir) à un processus fils ou à un thread pour une exécution *asynchrone*. Il pourrait aussi être envisagé d'interrompre les calculs trop longs.

Le format exact des messages transmis entre les différents processus n'est pas spécifié, mis à part les conventions de sérialisation de certains types (voir ci-dessous). À vous de définir en particulier le format des requêtes.

Modèle distribué

Dans ce modèle, tout processus peut déclarer publiquement quels calculs il sait effectuer, et tout processus peut lui adresser une requête en ce sens. Ce rôle de « serveur » vient en parallèle du rôle premier de ces processus, qui peut nécessiter une interaction avec un utilisateur. Vous pouvez par exemple penser à des petites calculatrices autorisant l'utilisateur à demander d'évaluer n'importe quelle expression arithmétique, alors que chacune ne « sait » effectuer qu'un type d'opération.

Cette déclaration publique nécessite une certaine centralisation, pour que les processus sachent *où* trouver les informations nécessaires à un appel de fonction externe, en particulier :

- quel(s) processus offre(nt) le service souhaité ;
- par quel canal la requête doit être adressée.

Pour cela, une structure de donnée adaptée sera stockée dans un segment de mémoire partagée, qui contiendra l'ensemble des informations utiles. Tout processus voulant offrir un service de calcul devra le faire en enregistrant les informations adéquates dans le segment de mémoire partagée, aussitôt qu'il sera prêt à offrir le service. Il devra prendre soin de supprimer ces informations lorsqu'il ne souhaitera plus offrir ledit service.

Ce modèle pose naturellement des problèmes de concurrence auxquels il faudra trouver des solutions.

Syntaxe d'un appel à une fonction externe

Un appel à une hypothétique fonction `type fonction(type arg, ...)` devra se faire via une fonction *appel_externe* de prototype (à la mode *execl*) :

```
int appel_externe(const char *fonction, int type, void *retour, ... /* int type_i, void *arg_i */, NULL);
```

ou (à la mode *execv*) :

```
int appel_externe(const char *fonction, unsigned short argc, struct arg *argv);
```

avec les notations suivantes :

fonction
: désigne l'identificateur externe (*i.e.* le nom) de la fonction ;

les éventuels *arg_i* et *retour*
: définissent les adresses auxquelles on trouvera, à l'appel, la valeur de ses *ar*guments, et, en retour, sa valeur de *retour ;*

chaque *type*
: définit le type C de l'*ar*gument (ou du *retour*) qui suit ;

le type `struct arg`
: est composé de deux champs `int` *`type`* et `void *`*`arg` et sert à stocker ensemble le type et l'adresse d'un argument (ou du retour) de la fonction ;

argc
: correspond au nombre d'éléments du tableau *argv*.

Constantes à définir

Les différents *type*s supportés sont (au moins) les suivants :

type	type C	Commentaire
`TYPE_VOID`	`void`	ne peut apparaître que pour la valeur de retour
`TYPE_INT`	`int *`	
`TYPE_STRING`	`char *`	

Ainsi un appel à une fonction d'addition de deux nombres entiers pourraît prendre la forme suivante :

```
int i, j , k, r;
/* k = i+j; */
r = appel_externe("plus", TYPE_INT, &k, TYPE_INT, &i, TYPE_INT, &j,NULL);
```

La valeur de retour de la fonction *appel_externe* représente la condition de terminaison de l'appel. On trouvera entre autres :

valeur	sémantique
`APPEL_OK`	tout s'est correctement déroulé
`FONCTION_INCONNUE`	la fonction demandée n'est pas disponible
`MAUVAIS_ARGUMENTS`	un problème concernant les arguments a été détecté (type, nombre)
`PAS_DE_REPONSE`	la fonction externe ne répond pas dans le délai imparti (5 secondes)

Format de sérialisation

Seul le format de sérialisation des types définis ci-dessus vous est imposé :

type	codage	exemple
`TYPE_VOID`	l'octet 0x00	
`TYPE_INT`	l'octet 0x01, suivi d'un octet donnant la longueur *lg* de l'écriture décimale de l'entier, puis des *lg* caractères de cette écriture décimale	123 est sérialisé 0x01, 0x03, '1', '2', '3'
`TYPE_STRING`	l'octet 0x02, suivi par un octet donnant la longueur *lg* de la chaîne, puis des *lg* caractères de la chaîne – caractère nul non compris !	"abc" est sérialisée 0x02, 0x03, 'a', 'b', 'c'

Instructions diverses

Le projet doit être réalisé par groupes de 2 ou 3 étudiants. Les groupes devront être constitués **le 15 mars au plus tard**, et ne plus varier ensuite. Lors de la soutenance (qui devra naturellement être bien préparée), tous les membres du groupe devront intervenir.

Le strict minimum à réaliser est le modèle client/serveur en mode synchrone, et des retours de fonction de type `void` ou `int`. Vous pouvez ensuite améliorer votre projet en autorisant d'autres types de retour, en permettant un fonctionnement asynchrone du serveur, et/ou en implémentant le modèle distribué. Toute application originale sera naturellement appréciée à sa juste valeur.

Votre programme devra être accompagné d'un rapport décrivant le travail réalisé ; en particulier, ce rapport devra expliquer les choix que vous serez amenés à faire concernant les protocoles de communication entre les différents processus.

Vous devrez réaliser différents tests pour vous convaincre que votre réalisation fonctionne. Décrivez-les dans votre rapport, et munissez-vous du nécessaire pour convaincre les examinateurs le jour de la soutenance. N'hésitez pas à mentionner *aussi* les tests négatifs ! Vous pourrez également nous montrer les développements que vous avez essayé de faire sans y parvenir complètement. Cependant, la version de votre projet que vous

présenterez doit être complètement stable.

La date et les modalités de rendu seront communiquées ultérieurement.