Master d'informatique | **module PC2R** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2009-2010**

Examen du 26 janvier 2010

Exercice 1 : Répartition du travail en binôme

Deux étudiants (A et B) solidaires doivent faire un devoir ensemble. Chacun a des disponibilités et des contraintes différentes :

- A veut travailler au moins 20% du temps et au plus 50% du temps, et quand il est réveillé il travaille tant qu'il ne s'est pas endormi (il s'endort forcément s'il atteint les 50% du temps).
- B est totalement d'accord avec les contraintes de A. Donc il le réveille s'il a travaillé plus de 79% du temps.

On va modéliser leur travail pour lequel ils partagent une ressource.

Le comptage du travail est effectué par un chef virtuel. Celui-ci peut être un module (OCaml, C) ou une classe (Java ou OCaml) qui possède une fonction (ou méthode) `count` qui prend en paramètre A ou B (et rien d'autre que A ou B) et qui rend son taux de travail.

1. Raffinez la modélisation du problème en précisant le modèle de concurrence et la synchronisation entre les différents acteurs : les étudiants A et B, et le chef.
2. Précisez quel langage (OCaml, java ou C) et quelle bibliothèque de Thread/Mutex/Condition ou Event vous utiliserez pour implanter une simulation du travail des étudiants A et B respectant votre modélisation.
3. Indiquez comment adapter cette simulation dans le cadre coopératif.
4. Implantez votre proposition en utilisant la bibliothèque de Fair Threads (C, Java ou OCaml).

Exercice 2 : Clients/serveur - téléchargement par parties

On veut faire un prototype de téléchargements de fichiers par parties. Le but est que les clients puissent télécharger à l'aide de plusieurs connexions un seul fichier. On décrit tout d'abord le protocole, un serveur gérant un seul client et les clients.

Protocole de communication Le serveur envoie sur une ligne le numéro d'ordre du paquet (morceau de fichier), puis sur une ligne sa taille en nombre d'octets, ensuite le paquet en question. S'il reste des paquets à envoyer, il envoie un nouveau paquet suivant la description précédente. S'il ne reste plus rien à envoyer, il envoie 0 sur une ligne. La numérotation des paquets commence par 1. Attention : les paquets sont envoyés dans l'ordre par le serveur, mais la transmission du paquet 4 peut finir avant la transmission du paquet 2, par exemple.

Le serveur Il attend sur un port (variable globale du numéro : port) des connexions d'un client, et envoie le contenu d'un fichier (variable globale pour le chemin du fichier : fichier) sur l'ensemble des connexions. Les connexions restent ouvertes tant que le fichier en entier n'est pas transmis et sont closes à la fin de la transmission.

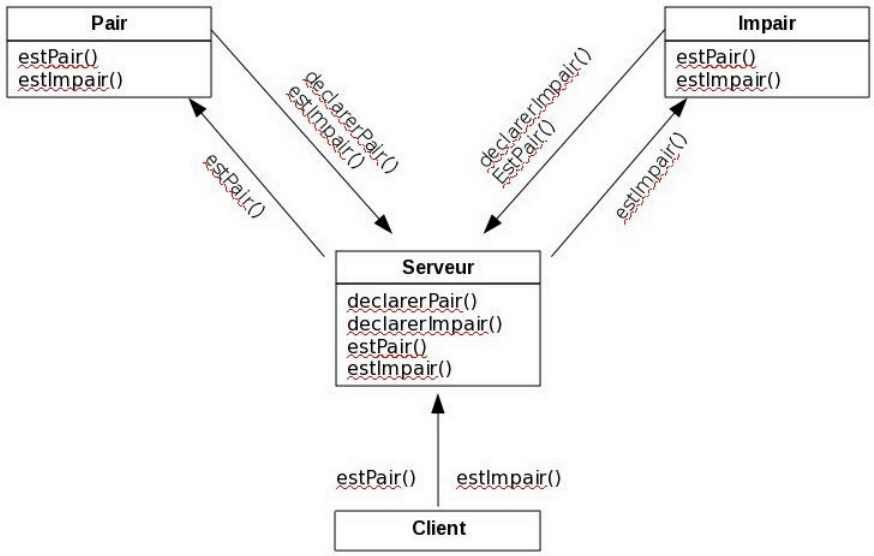
Le client Il ouvre n (variable globale n) connexions vers le serveur pour télécharger le fichier, et l'imprime sur la sortie standard. Attention à l'ordre des paquets ! Le paquet n doit toujours être imprimé entièrement avant le paquet n+1. (Il vous est laissé la liberté de choisir le mode de gestion de données en cours de réception.)

1. Donnez les communications entre le serveur et 1 client qui ouvre 3 connexions pour la transmission d'un fichier qui contient les 26 lettres de l'alphabet dans l'ordre, et qui transmet les données au client par paquets de 4 octets.
2. Implantez le serveur en C, OCaml ou Java.
3. Implantez le client dans un autre langage que celui du serveur.
4. Proposez des modifications (protocole, serveur et client) pour accepter plusieurs clients simultanément ; chaque client pouvant toujours ouvrir plusieurs connexions.
5. **Bonus** : Implantez votre proposition du côté du serveur et des clients si besoin est.

Exercice 3 : Calcul pair-impair en RMI

On propose un dispositif permettant de calculer de manière répartie la parité d'un nombre. Il est composé de (voir dessin ci-dessous)

- Un serveur qui enregistre
  - un client particulier `pair` à qui il va s'adresser pour savoir si un nombre est pair ou non.
  - un autre client particulier `impair` à qui il va s'adresser pour savoir si un nombre est impair ou non.
- Un client `pair` qui renvoie vrai si le nombre `n` proposé est `0`, sinon il demande au serveur si le nombre `n-1` est impair et retourne ce résultat.
- Un client `impair` qui renvoie faux si le nombre `n` proposé est 0, sinon il demande au serveur si le nombre `n-1` est pair et retourne ce résultat.
- des autres clients qui demandent au serveur de calculer la parité d'un nombre.



Lorsque l'objet pair (resp. impair) n'existe pas encore (ne s'est pas encore déclaré), on souhaite que tout appel à la méthode `estPair()` (resp. `estImpair()`) ne provoque pas de levées d'exception.
A la place, le serveur doit se mettre en attente de sa déclaration et relance automatiquement l'appel juste après la déclaration.

1. Définissez le ou les interfaces nécessaires pour le serveur, le ou les classes pour couvrir ces interfaces et la classe `ServeurPairImpair` avec toutes les variables nécessaires à son fonctionnement.
2. Définissez le ou les interfaces nécessaires pour `Pair`, le ou les classes pour couvrir ces interfaces et la classe `Pair`. On souhaite que l'appel de sa méthode `estImpair()` lève une exception.
3. Définissez la classe `Client` en testant dans une boucle de 0 à 9 pour savoir si ces nombres sont pairs ou impairs.
4. Modifiez vos programmes pour que `Pair` et `Impair` discutent ensemble directement sans utilsier le serveur comme intermédiaire (voir dessin ci-dessous).