

Université de Bretagne Occidentale
Licence Informatique option informatique

Examen de Réseaux

Jeudi 22 mai 2003, 13h30-15h30

Documents autorisés

Notes de cours, de travaux dirigés et de travaux pratiques.

Remarques

La présentation des réponses est un élément déterminant tant pour la conception de celles-ci que pour leur compréhension par le correcteur. Les réponses doivent être justifiées.

Le barème et le temps sont donnés à titre indicatif.

Lire le sujet en entier avant de commencer l'examen.

1 Taille des buffers (5 points - 20 min.)

1.1 Cas TCP

Soient deux programmes `serveur_tcp` et `client_tcp` communiquant à l'aide du protocole TCP. Le programme `serveur_tcp` envoie de manière régulière des informations au programme `client_tcp`, après la phase initiale de connexion.

1.1.1 extrait de code de `serveur_tcp`

```
#define BUFSIZE_SERVEUR 20
...

char buf[BUFSIZE_SERVEUR];
...

strcpy(buf, "01234567899876543210");
if (write(sock, buf, strlen(buf)+1) != strlen(buf)+1)
    { ... }
...

```

1.1.2 extrait de code de `client_tcp`

```
#define BUFSIZE_CLIENT 20
...

char buf[BUFSIZE_CLIENT];
...

if (read(socket_service, buf, BUFSIZE_CLIENT) < 0)
    { ... }
printf("j'ai reçu %s\n", buf);
...

```

Que se passe-t-il dans le cas où la variable `BUFSIZE_CLIENT` a une valeur plus grande que la variable `BUFSIZE_SERVEUR` et réciproquement? Peut-on gérer facilement les éventuels problèmes? Si oui comment?

1.2 Cas UDP

Soient deux programmes `emetteur_udp` et `receveur_udp` communiquant à l'aide du protocole UDP. Le programme `emetteur_udp` envoie de manière régulière des informations au programme `receveur_udp`.

1.2.1 extrait de code de `emetteur_udp`

```
#define BUFSIZE_EMETTEUR 20
...

char test[BUFSIZE_EMETTEUR];
...

strcpy(test,"01234567899876543210");
if ((envoye = sendto(sock, test, strlen(test), 0, &adr, lgadr)) != strlen(test))
    { ... }
...
```

1.2.2 extrait de code de `receveur_udp`

```
#define BUFSIZE_RECEVEUR 20
...

char test[BUFSIZE_RECEVEUR];
...

if ((recu = recvfrom(sock, test, BUFSIZE_RECEVEUR, 0, &adr, &lgadr)) == -1)
    { ... }
printf("j'ai recu %s\n", test);
...
```

Que se passe-t-il dans le cas où la variable `BUFSIZE_RECEVEUR` a une valeur plus grande que la variable `BUFSIZE_EMETTEUR` et réciproquement? Peut-on gérer facilement les éventuels problèmes? Si oui comment?

2 Fenêtres d'émission en TCP (5 points - 20 min.)

Le protocole TCP est un protocole d'émission de données qualifié de fiable. Sa fiabilité repose en partie sur la gestion des acquittements de message : les messages sont numérotés et chaque message envoyé doit être acquitté par le receveur. Si au bout d'un temps donné, un message n'est pas acquitté, il est alors réémis. Les acquittements ne sont pas eux-même acquittés.

Les parties ci-dessous présentent, de manière simplifiée, le mécanismes des fenêtres d'émission qui permettent tout à la fois de gérer les problèmes d'acquittement et de débit.

2.1 Fonctionnement simple

En supposant qu'un émetteur doit envoyer 40 messages, que le temps d'envoi d'un message (émetteur \rightarrow receveur) et que le temps d'envoi d'un acquittement (receveur \rightarrow émetteur) sont tous les deux de 0,5 secondes, que les temps de traitement des messages sont négligeables, combien de temps faut-il à l'émetteur pour envoyer tous ses messages? Un schéma sera apprécié.

2.2 Fenêtres d'émission

Pour optimiser les communications, un mécanisme de fenêtres d'émission a été mis en place. Si la fenêtre a une taille M , l'émetteur peut envoyer les messages numérotés n , $n+1$, $n+2$, \dots , $n+M-1$ sans attendre l'acquittement du message n . Dès que l'acquittement du message n est arrivé, il peut envoyer le message $n+M$. Si l'acquittement du message $n+1$ est déjà arrivé, il peut envoyer le message $n+M$, sinon il doit attendre l'acquittement avant de poursuivre et ainsi de suite.

En utilisant les mêmes suppositions que dans la question précédente et en supposant M fixé à 8, combien de temps faut-il à l'émetteur pour envoyer 40 messages? Un schéma sera apprécié.

2.3 Fenêtres d'émission en cas de perte de message

Si un message envoyé ne reçoit pas d'acquittement (perte du message ou perte de l'acquittement), il est automatiquement réémis au bout d'un temps déterminé (`timeout`).

En se plaçant dans les mêmes conditions que ci-dessus, en supposant que la valeur du `timeout` est de 2,5 secondes, que les messages 12 et 24 sont perdus lors de leur première émission, combien de temps faut-il pour émettre 40 messages? Un schéma sera apprécié.

2.4 Adaptation dynamique des fenêtres d'émission

En cas de congestion sur le réseau, la perte de message peut devenir un phénomène important. La perte entraînant des réémissions celles-ci peuvent entraîner un sur-engorgement du réseau. Pour résoudre ce problème, il est nécessaire que l'ensemble des émetteurs diminuent leurs envois de messages. Une méthode assez simple est basée sur l'adaptation dynamique de la taille de la fenêtre. En cas de perte d'un message, la taille de la fenêtre est divisée par 2 (arrondi à la valeur entière inférieure) et ceci éventuellement jusqu'à obtenir la taille minimale c'est à dire 1.

Inversement, si la communication se déroule correctement (on peut envoyer la totalité des messages contenus dans une fenêtre), la taille de la fenêtre est augmentée de 1, jusqu'à sa valeur maximale.

En supposant la valeur initiale et maximale de N à 8, combien de temps faut-il pour émettre 40 messages dans les conditions expérimentales précisées précédemment? Un schéma sera apprécié.

3 Problème (10 points - 60 min.)

Dans le cadre d'une application particulière comprenant un serveur et des clients, on souhaite que chaque client connaisse à tout moment la liste des clients connectés au serveur (adresse IP).

A chaque arrivée d'un client, le serveur lui enverra donc la liste des clients connectés et avertira ces derniers de l'arrivée d'un nouveau client. De même, le serveur avertira l'ensemble des clients du départ d'un client particulier.

Point de vue du client

1. Initialisations
2. Préparation de la demande de connexion
3. Demande de connexion au serveur : si la connexion échoue, aller au point 6.
4. Réception d'un message en provenance du serveur comportant la liste des clients connectés. Décodage du message et remplissage d'un tableau de clients.
5. Attente, soit d'un message du serveur, soit d'une interaction utilisateur (utilisation par exemple d'une primitive `select` sur la socket et sur le clavier)
 - Si le déblocage correspond à une demande utilisateur, la traiter. Si cette demande ne correspond pas à une demande de fin d'utilisation, se remettre en attente au point 5 sinon aller au point 6.
 - Si le déblocage correspond à un message du serveur, le traiter. Il s'agit soit de la connexion, soit de la déconnexion d'un client.
6. Terminer

Questions

1. Rédiger le "Point de vue du serveur" en vous inspirant du point de vue du client.
2. Donner une description détaillée du serveur.
On supposera disponible, les fonctions classiques de haut niveau permettant de gérer des listes (`(liste) creerl()`, `inserer(liste L, position P, elt-liste x)`, `(position) chercher (liste L, elt-liste x)`, etc...). Vous pouvez également définir d'autres fonctions si nécessaire.