Sujet de TP 9-10 Système et **réseau** - Sockets Licence 3

Programmation réseau avec sockets en C (Linux)

V. Felea & A. Hugeat

Mode connecté - protocole TCP

1. **Dire Bonjour (sens bidirectionnel de communication)**
   Modifier l'application du TP précédent telle qu'une application envoie un nom et l'autre lui répond par "Bonjour *nom*". La taille maximale du nom est connue par les deux applications (donnée par une constante).
   Remarque. Une seule famille de protocoles, correspondant à une communication distante, sera retenue pour cet exercice ainsi que pour ceux qui suivent.

2. **Des centaines de milliers de décimales du nombre π**
   Dans cet exercice, une application souhaite informer une autre application de la valeur du nombre π avec une très bonne précision. La valeur du nombre *pi* à transmettre a 100000 décimales (voir fichier donné). Écrire les deux applications communicantes où cette valeur est transmise sous la forme d'une chaîne de caractères.

   Remarque sur l'échange de données de grande taille.
   Quand des données sont envoyées en connexion TCP, la quantité de données peut dépasser la taille maximale d'octets autorisée par la connexion pour un seul envoi. Il y a deux limites imposées, d'une part par la couche physique, par le type de liaison de transmission (le MTU : Maximum Transmission Unit) et d'autre part, par la couche de transport (le MSS : Maximum Segment Size). Le MTU est la taille maximale d'une trame pouvant être transmise en une seule fois (sans fragmentation) sur une interface. Le MSS désigne la quantité de données en octets que peut contenir un datagramme seul et non segmenté. Le MSS pourra donc garantir une non segmentation des données. Le MSS se calcule de la manière suivante :

   MSS = MTU - en-tête IP - en-tête TCP

   Par exemple avec un MTU de 1500, on aura : MSS = 1500 - 20 - 20 = 1460 octets.

   Le protocole TCP au niveau 4 OSI est responsable du découpage de la donnée chez l'expéditeur (en segments), appelé *segmentation* ainsi que de la reconstitution de la donnée initiale à partir de l'ensemble des segments reçus chez le destinataire (*réassemblage*). Les segments sont numérotés afin d'aider à la reconstitution de la donnée. Le numéro de séquence indique le numéro du premier octet transmis dans le segment. Il peut être visualisé grâce à la commande Linux `tcpdump`.

   Exemple. Cette commande exige de disposer des droits d'administrateur sur la machine :

   ```
   tcpdump -i interf_reseau
   ```

Étant donné que le processus de segmentation produit des segments dont la taille ne dépasse pas celle acceptée par la liaison physique (MSS < MTU), le découpage d'un segment en fragments, appelé *fragmentation* devrait très rarement se produire sur une machine. Activée par le protocole IP, dans la couche internet/réseau, la fragmentation apparaît cependant plus fréquemment sur les routeurs, et plus particulièrement quand le routeur interconnecte des réseaux hétérogènes en terme de liaison physique (par exemple un réseau FDDI de MTU à 4352 octets avec un réseau Ethernet de MTU à 1500 octets).

Une autre commande permettant d'avoir une information globale sur la segmentation et la fragmentation est `netstat`.

Exemple. Le champ TX_OK (ou RX_OK) recense le nombre de transmissions (ou réceptions) correctement effectuées.

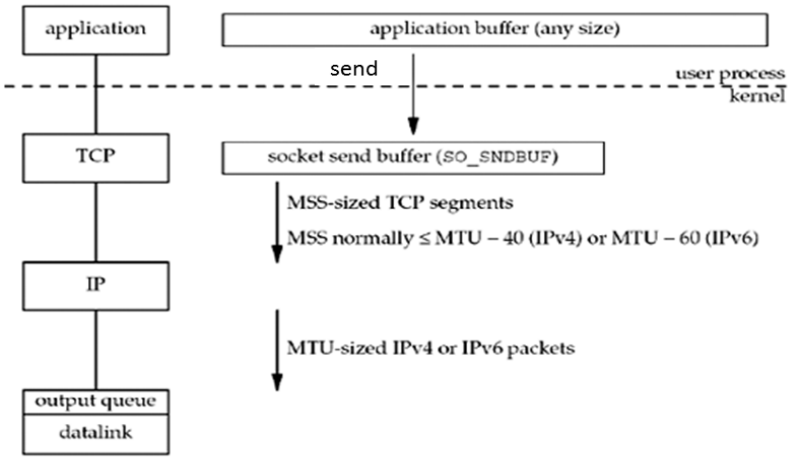
```
netstat -i
```

Tester cette commande pour l'exercice précédent, où un seul envoi est réalisé, avec une taille de données faible (ce qui n'impose pas une segmentation). Ré-effectuer le test pour l'exercice sur la transmission de la valeur du nombre π avec beaucoup de décimales (dépassant largement la limite MSS, engendrant ainsi la segmentation de la donnée).

Rappel. Pour le protocole UDP, la taille d'un datagramme ne peut pas dépasser 2^{16} octets (64 Ko).

3. **Tampons interne associés aux sockets**
Le protocole TCP envoie et reçoit les paquets en passant par des tampons internes (un pour les transmissions, un pour les réceptions) gérés par le noyau, par connexion TCP établie.

L'exploitation de la pile protocolaire TCP/IP lors de l'émission d'une donnée (issue de l'applicatif, par la fonction `send` des sockets) est illustrée dans la figure suivante.



La taille de ces tampons peut être consultée/ajustée par les fonctions `getsockopt`, `setsockopt`.

Programmation C Un extrait de code C permettant d'obtenir la taille du tampon interne d'envoi associé à une socket :

```
int sockbufsize; size = sizeof(int);
err = getsockopt(skt, SOL_SOCKET, SO_SNDBUF, &sockbufsize, &size);
```

Un extrait de code C permettant de changer la taille du tampon interne d'envoi associé à une socket (à eviter à cause d'un principe de modification dynamique de la taille d'un tampon - autotuning activé - voir le fichier `/proc/sys/net/ipv4/tcp_moderate_rcvbuf`) :

```
int skt, sndsize=/* a initialiser */;
err = setsockopt(skt, SOL_SOCKET, SO_SNDBUF, &sndsize, sizeof(int));
```

Remarque. Le noyau double la valeur d'entrée fournie à la fonction `setsockopt`.

La constante `SO_RCVBUF` est à utiliser pour le tampon interne de réception.

Configuration système (extraits des manuels socket(7) et tcp(7)) Des variables système permettent de configurer les tailles par défaut de ces tampons internes :
`net.core.rmem_default`, respectivement `net.core.wmem_default`.
Pour les connexions TCP, les valeurs par défaut peuvent être globalement réécrites par les paramètres système `net.ipv4.tcp_rmem`, respectivement `net.ipv4.tcp_wmem`.
Toujours concernant ces tailles, les valeurs maximales sont données par les variables système `net.core.rmem_max`, respectivement `net.core.rmem_max`.

Visualiser le comportement de la communication TCP Pour obtenir des informations sur le nombre d'octets reçus chez le récepteur dans le tampon interne, mais non encore lus par l'applicatif, ou sur le nombre d'octets envoyés par l'expéditeur non encore acquittés par le récepteur, lors d'une communication TCP, utiliser la commande Linux :

```
netstat --tcp -p -c -v
```

Remarque. Pour des communications UDP, il faudrait sélectionner les sockets UDP.

Modifier le programme de l'exercice précédent afin d'effectuer des envois successifs de la valeur du nombre π, selon une précision croissante, en commençant par 1000 décimales, et avec un pas de 1000 décimales, jusqu'à atteindre les 100000 décimales du nombre π données dans le fichier. Instrumenter le code afin de temporiser (par des saisies clavier) aussi bien les envois que les réceptions. Après un envoi, respectivement une réception, sera affiché le nombre d'octets (applicatifs) envoyés/reçus depuis le début de la communication. Visualiser le résultat et expliquer le fonctionnement grâce aux commandes systèmes précédemment fournies.

4. **Carré d'un nombre entier**
   Écrire une application qui transmet un nombre entier à une autre application qui lui retourne le carré de celui-ci.

Mode non-connecté - protocole UDP

5. **Nombre π**
   Proposer une solution pour la communication de la valeur du nombre π en utilisant le protocole de transport UDP. Quelle précision peut être utilisée dans ce cas ?