ENSIIE 1ère Année 21 mai 2012
Durée 1h45; Contrôle de Systèmes Informatiques 2
Calculette non autorisée
Barème indicatif
(le total des points est supérieur à 20, on retiendra *min*(note,20))

Exercice 1 (4 pts) : Fragmentation d'un paquet IP
Un paquet IP de longueur totale 2020 octets et contenant 2000 octets de données est transféré sur un réseau N1 de MTU = 1024. Il traverse ensuite un routeur pour passer du réseau N1 à un réseau N2 de MTU = 512.
Question : représenter les différents paquets issus de la fragmentation du paquet original, qui passeront sur le réseau N1 puis N2. Pour chaque paquet, vous représenterez le bit MF (More Fragments), la valeur du champ *taille* (bits 16-23), et la valeur du champ *fragment offset* (ou *décalage*, à partir du bit 50).

Exercice 2 (3 pts) : Résolution d'adresses
Question : Expliquez rapidement le mécanisme qui permet, dans un réseau local ethernet, à un hôte A de s'adresser à un hôte B en utilisant l'adresse IP de B. Comment A retrouve t-il l'adresse ethernet de B en n'en connaissant que l'IP ?

Exercice 3 (15 pts) : serveur de fichiers

INDICATIONS
- la composition de la liste d'arguments de la ligne de commande doit être vérifiée et traitée en cas d'erreur
- compte tenu de l'absence de documentation technique, l'ordre des paramètres des appels noyaux ne sera pas cause de pénalisation.
- les valeurs de retour des appels noyaux devront être examinées et traitées de façon cohérente. Les traitements répétitifs pourront toutefois être seulement esquissés.
- la question (sauf 1 et 2) reprend le programme d'une question précédente; il n'est pas nécessaire de le recopier, il suffit d'indiquer les modifications à faire et où elles se placent.

On souhaite concevoir un système client-serveur dans lequel un serveur possède sur son disque des fichiers, et les clients demandent à les télécharger. Le client s'identifie, demande un fichier, et en retour, reçoit un flux d'octets qu'il sauvegarde dans un fichier. De son côté, le serveur, à chaque demande de client, créé un nouveau processus qui sera en charge de lire le fichier et en envoyer le contenu vers le client. Décomposons et détaillons l'ensemble des tâches à programmer :

Question 1 (5 pts) : Client
Écrire le client, qui a obligatoirement les quatre arguments suivants sur la ligne de commande:
id, serveur, port, et *f*. Le programme client effectue les tâches suivantes *:*

1. il vérifie que la ligne de commande est correcte, sinon affiche un message d'aide sur les paramètres nécessaires et s'arrête,
2. établit une connexion TCP avec le serveur *serveur* et sur le port *port* indiqués dans la ligne de commande,
3. envoie un identifiant *id* et le nom *f* du fichier à télécharger, sous la forme d'une chaîne de caractères contenant « *id f* » séparées par un espace,
4. se met en lecture sur la connexion TCP,
5. écrit au fur et à mesure de leur réception, ces données dans un fichier local nommé aussi *f*.

Note : pour le point 3. il est conseillé d'utiliser `sprintf(buf,"%s %s",id,f)` pour composer la chaîne à envoyer au serveur. Pour le point 5. , on considère que le fichier local doit être écrasé s'il existe déjà.

Question 2 (7 pts): Serveur
Écrire le serveur qui a un seul argument sur la ligne de commande : le port d'écoute *port*.
Le serveur effectue les tâches suivantes :

1. il vérifie que la ligne de commande est correcte, sinon affiche un message d'aide et s'arrête,
2. initialise et installe un socket TCP sur le port *port*,
3. se met en attente d'une connexion d'un client,
4. quand un client se connecte, lit son identifiant et le nom de fichier *f* dans le flux de données envoyées sur la connexion,
5. créé un nouveau processus fils à l'aide de fork().

   a. Ce fils exécute une fonction qui gère le téléchargement, qui est appelée avec deux arguments : le nom du fichier *f* et la socket de service *s* qu'on peut utiliser pour contacter le client.
   b. La fonction lit *f* et écrit les octets sur *s* à destination du client.
6. le processus père revient immédiatement sur l'attente d'une nouvelle connexion (boucle vers point 3.)

Note : pour le point 4. il est conseillé de lire un nombre d'octet conventionnel d'octet dans le flux (le même que le nombre d'octets écrits au point 3. du client) et d'utiliser `sscanf` pour séparer les chaines *id* et *f* (`sscanf` est identique à `scanf` hormis qu'elle lit les entrées dans la chaîne passée en premier argument au lieu du clavier).

Question 3 (3 pts) : Temporisation des envois

Modifier le programme du serveur en y ajoutant un mécanisme de temporisation sur les envois : on souhaite pouvoir régler le rythme auquel les fichiers sont envoyés aux différents clients. Proposez un mécanisme qui bloque le père et l'empêche d'accepter une nouvelle connexion, tant que 80% du fichier précédemment en cours d'envoi n'a pas été transmis.
Décrivez le principe en quelques lignes puis écrivez le code C.

Vous pourrez utiliser la fonction `stat()` qui permet de connaître la taille d'un fichier. Ex : avec char *f ; et `struct stat s` ; `stat(f,&s);` permet d'obtenir la taille du fichier f par le champ : `f.st_size`