ArcSys1 : Systèmes et Réseaux

Examen final du 17 décembre 2018 (2 h)

Tous documents interdits à l'exception d'un A4 recto-verso manuscrit de votre main et du pense-bête du C fourni en cours. Calculatrices, téléphones, tablettes, ordinateurs et montres connectées interdites. La correction tiendra compte de la qualité de l'argumentaire et de la présentation. Un code illisible et/ou incompréhensible est un code faux.

★ Exercice 1 : Questions de cours (3 pts).

▷ **Question 1 :** Expliquer brièvement ce qu'est un serveur DNS et son fonctionnement.

▷ **Question 2 :** L'utilisation de sockets TCP se fait typiquement à l'aide des fonctions `accept`, `bind`, `close`, `connect`, `listen`, `read`, `socket`, `write` (ordre alphabétique). Retrouvez l'ordre typique d'appel de ces fonctions dans un serveur TCP, puis dans un client TCP.

▷ **Question 3 :** Dessinez l'état de la pile en mémoire lorsque la fonction `triple` se termine à la ligne 5 du code suivant (les deux blocs de code forment un seul programme).

```
1  #include <stdio.h>
2
3  void triple(int* a) {
4     *a = *a * 3;
5  }
6
```

```
7  int main(void) {
8     int x = 14;
9     triple(&x);
10    printf("x=%d\n",x);
11    return 0;
12 }
```

★ Exercice 2 : Chaînes de caractères et fichiers (4 pts).

On souhaite écrire un programme convertissant les majuscules en minuscules, et inversemment.

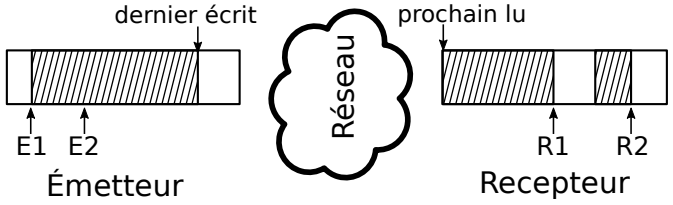
▷ **Question 1 :** Écrivez un programme complet prenant une chaîne de caractères à analyser en argument de la ligne de commande, de la manière suivante.

```
1 $ ./min_maj bOnNE nUIt LES petits ...
2 BoNne NuiT les PETITS ...
3 $ ./min_maj lE c, tROp 3l13T!
4 Le C, TroP 3LL3t!
```

▷ **Question 2 :** Modifiez votre programme afin qu'il lise le texte à analyser depuis un fichier dont le nom est passé en argument de la ligne de commandes.

Indication : le décallage entre les majuscules et les minuscules est donné par l'expression 'A'-'a'.

★ Exercice 3 : TCP (2 pts).



Le schéma ci-contre représente l'état des buffers émetteur et récepteur d'une connexion TCP. Les zones hachurées sont des parties contenant des données que TCP doit absolument concerver (reçues de l'application émettrice, pas encore lues par l'application réceptrice). On suppose que les buffers se remplissent de gauche à droite.

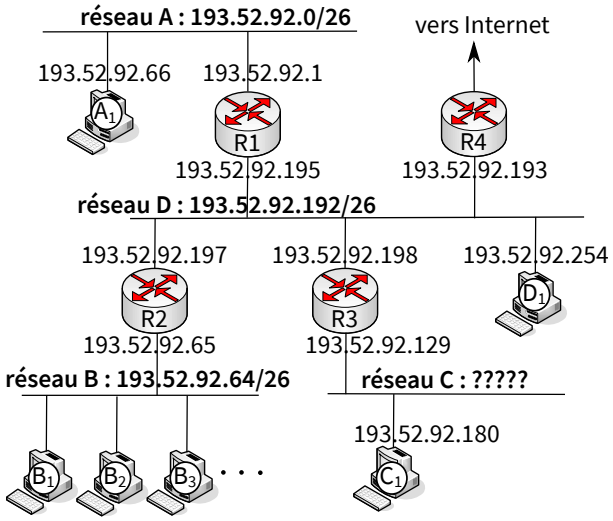
Coté émetteur, la fin de la zone hachurée représente le dernier caractère écrit par l'application : au delà, le buffer ne contient pas encore de données utiles. Coté récepteur, le début de la zone hachuré représente le prochain caractère lu par l'application. Ici, l'application réceptrice n'a encore rien lu des données reçues, puisque la zone hachurée commence au début du buffer.

▷ **Question 1 :** Sachant qu'il s'agit de pointeurs importants pour le fonctionnement de TCP, expliquez à quoi correspondent E1, E2, R1 et R2 ? Donnez une cause possible pour la zone blanche à droite de R1.

▷ **Question 2 :** Refaites le schéma sur votre copie, en représentant graphiquement W, la taille de la fenêtre TCP dans ce cas.

★ **Exercice 4 : Routage IP (4pts).**

Le schéma ci-dessous représente une petite partie de l'internet, composée de quatre sous-réseaux notés A, B, C et D. Le routeur R1 connecte les réseaux A et D ; R2 connecte B et D ; R3 connecte C et D tandis que R4 connecte D au reste de l'internet.



▷ **Question 1 :** L'une des machines A_1 ou D_1 a une adresse invalide. Laquelle ? Pourquoi ? Proposez une adresse valide pour cette machine.

▷ **Question 2 :** Quel est le masque réseau du réseau B ? Combien de machines peuvent se connecter à ce réseau en même temps ?

▷ **Question 3 :** Connaissant les adresses de R3 et C_1 et celles des réseaux environnants, quels sont l'adresse du réseau C et son masque ?

▷ **Question 4 :** Donnez une table de routage fonctionnelle pour le routeur 4, en précisant à chaque ligne le réseau, le masque et le next hop. Sa passerelle vers Internet est 193.52.88.1.

★ **Exercice 5 : C et ... Réseaux (questions relativement indépendantes – 7 pts).**

On souhaite coder en C une version très simplifiée du protocole à vecteurs d'état de Bellman-Ford. Notre version ne tiendra pas compte des réseaux et de leurs masques : chaque machine existante sera explicitement listée dans la table de routage. Le nombre de machines est constant et connu à l'initialisation. Les distances sont mesurées en millisecondes.

```
1 struct table {
2    int *next_hop; // next_hop[i]: identifiant de l'hôte à utiliser pour joindre i
3    int *distance; // distance[i]: nombre de millisecondes pour joindre i
4    int size;
5 };
```

▷ **Question 1 :** Écrivez les fonctions suivantes `table_new()` et `table_free()` (prototypes ci-dessous), qui permettent respectivement d'allouer et libérer la mémoire nécessaire pour stocker une table. `table_new()` reçoit en paramètres le nombre de machines connectées sur le réseau (qui est la taille de la table de routage), ainsi que l'identifiant de la machine sur laquelle cette fonction est appelée. Les distances sont initialisées à `MAXINT` pour les machines distantes et à 0 pour la machine locale.

```
1 struct table* table_new(int taille, int ID);
2 void table_free(struct table* t);
```

▷ **Question 2 :** Écrivez la fonction `table_seen()`, utilisée par la machine locale quand elle voit sur le réseau local un paquet venant de la machine *neighb*. Deux modifications sont apportées à la table de routage : la distance à *neighb* est fixée à *millis*, et la distance aux éléments qui passent par *neighb* est mise à jour de façon adéquate.

```
1 void table_seen(struct table* t, int neighb, int millis);
```

▷ **Question 3 :** Écrivez la fonction `table_failed()`, utilisée pour indiquer qu'un lien vers la machine *neighb* ne fonctionne plus. La distance à cette machine est fixé à `MAXINT` de façon à ce qu'une autre route soit trouvée par les mises à jour futures.

```
1 void table_failed(struct table* t, int neighb);
```

▷ **Question 4 :** Écrivez la fonction `table_send()`, utilisée pour envoyer les informations de la table de routage sur une socket fd déjà ouverte. Là où Bellman-Ford envoie des couples <ID, distance>, nos simplifications nous permettent de n'envoyer que le vecteur de distances.

```
1 void table_send(struct table* t, int fd);
```

▷ **Question 5 :** Écrivez la fonction `table_recv()`, utilisée pour recevoir les informations de routage d'une machine distante depuis une socket fd déjà ouverte (le format des données envoyées est celui de votre fonction `table_send`).

La fonction reçoit également en paramètre *neighb* (identificateur de la machine émettrice) et *millis* (distance entre la machine émettrice et la machine locale). Il faut modifier la table de routage pour tenir compte des informations reçues. Entre autres, le `next_hop` de certaines destinations peut passer à *neighb* si cette solution est plus avantageuse.

```
1 void table_recv(struct table* t, int fd, int neighb, int millis);
```

▷ **Question 6 :** On s'intéresse maintenant à la dynamique de cet algorithme au cours du temps, en supposant que chaque machine envoie régulièrement ses informations de routage à tous ses voisins.

Tout comme Bellman-Ford, notre solution n'est pas très efficace pour propager les mauvaises nouvelles quand un lien tombe en panne et que `table_failed()` est utilisée par l'une des machines.

Proposez une modification de `table_send()` permettant d'accélérer la propagation de ces informations sans modifier la taille du vecteur envoyé (faites une copie de votre vecteur avant de le modifier). Comment s'appelle cette modification dans le cours ?