ArcSys1 : Systèmes et Réseaux

Examen final du 17 décembre 2019 (2 h)

Tous documents interdits à l'exception d'un A4 recto manuscrit de votre main et des pense-bêtes du C et réseau fournis en cours. Calculatrices, téléphones, 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 : Chaînes de caractères et fichiers (4 pts).

On souhaite écrire un programme affichant le décompte des voyelles présentes dans une chaîne de caractères donnée.

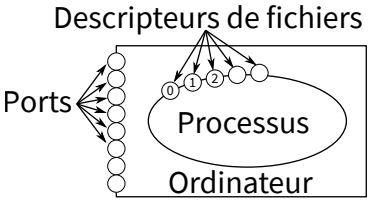
▷ **Question 1 :** Écrivez un programme complet prenant une chaîne de caractères à analyser en argument de la ligne de commande, de la manière suivante.

```
$ ./compte-voyelles "Bonjour. Bon courage pour ce partiel."
Nombre de a: 2
Nombre de e: 3
Nombre de i: 1
Nombre de o: 5
Nombre de u: 3
Nombre de y: 0
```

▷ **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.

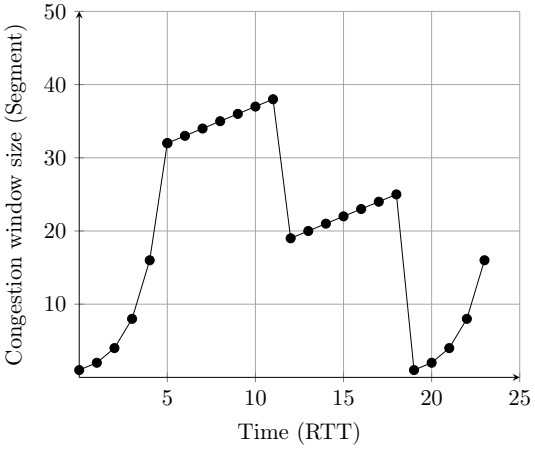
★ Exercice 2 : Sockets BSD (4pts).

On rappelle le formalisme vu en cours pour représenter des processus en cours d'exécution, repris ci-contre. Les machines sont représentées par des rectangles, avec des petits ronds sur le coté pour représenter les ports de la machine. Au sein des machines, chaque processus est représenté par une ellipse, et des petits ronds à la bordure de cette ellipse représente les différents descripteurs de fichier du processus. Les trois premiers sont habituellement `stdin`, `stdout`, `stderr`.



▷ **Question 1 :** Dessinez en respectant ce formalisme les différentes étapes de l'établissement d'une connexion TCP entre deux processus distants. Vous donnerez les appels systèmes utilisés par chaque partie. Vous dessinerez l'état des choses après chaque appel système, en ajoutant les explications adéquates. En revanche, il ne vous est pas demandé de donner le code exact réalisant ces appels systèmes.

★ Exercice 3 : Contrôle de congestion (2pts).



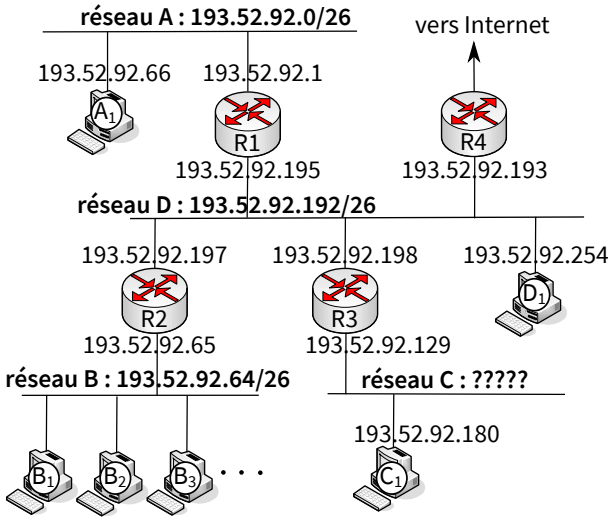
Le schéma ci-contre représente l'évolution de la taille de la fenêtre au cours du temps. Cette taille est mesurée en nombre maximal de segments potentiellement en transit sur le réseau. Le temps est mesuré en multiples de RTT, qui est le le délai de transmission pour envoyer un paquet plus le délai pour recevoir le ACK.

▷ **Question 1 :** Identifiez les intervalles de temps où les algorithmes de *slow start* et de *congestion avoidance* sont en action.

▷ **Question 2 :** Que s'est-il passé coté émetteur aux temps 5, 11 et 18 ?

★ 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 R4, en précisant à chaque ligne le réseau, le masque et le next hop. Le next hop est soit l'adresse de la passerelle à contacter si le réseau destination est distant, soit 0 si le routeur R4 peut contacter directement les machines du réseau destination. Sa passerelle vers Internet est 193.52.88.1.

★ Exercice 5 : Routage dans un réseau . . . et C (6pts).

Dans cet exercice, nous allons nous intéresser à la construction d'une table de routage dans un réseau en utilisant l'algorithme de *link-state routing*. Le principe de l'algorithme est le suivant :

- chaque noeud identifie la liste de ses voisins directs.
- chaque noeud envoie un message Link-State Advertisement (LSA) à TOUS les noeuds du graphe. Un LSA contient son identifiant, la liste de ses voisins ainsi qu'un numéro de séquence permettant d'ordonner les LSA reçus. Quand un noeud reçoit un tel message, il regarde le numéro de séquence du message : si il possède un message provenant de la même source, ayant un numéro de séquence plus grand (i.e. un message plus récent), il ignore le message, sinon il stocke le message et le retransmet à tous ses voisins.
- quand un noeud connaît les LSA de tous les noeuds du réseau, il les utilise pour construire une carte du réseau, applique un algorithme de plus cours chemin vers toutes les destinations du réseau et construit ainsi sa table de routage.

L'objectif de cet exercice est d'écrire les différentes étapes de cet algorithme en C.

Les messages LSA utilisent la structure de donnée suivante :

```
struct LSA {
   int node_id;
   int nb_neighbors;
   int *neighbors;
   int sequence;
};
```

Un noeud du réseau est décrit par la structure suivante :

```
struct node_information {
   int id;                     // ID du noeud courant
   int nb_nodes;               // Nombre de noeuds dans le réseau
   struct LSA *received_LSA;   // Tableau des LSA provenant des différents noeuds
   int nb_neighbors;           // Nombre de voisins directs
   int *neighbors;             // Tableau contenant le noms des voisins directs
   int local_sequence_number; // Dernier n° de séquence utilisé pour l'envoi d'un LSA
};
```

▷ **Question 1 :** Écrire les fonctions `node_new()` et `node_free()` qui initialisent les informations contenues par un noeud. Ces fonctions auront le prototype suivant :

```
struct node_information*
  node_new(int id, int nb_nodes, int nb_neighbors, int *neighbors);
void node_free(struct node_information* node);
```

Attention : l'argument neighbors est une liste locale, qui ne peut pas être référencée. Il est nécessaire de le recopier dans un tableau alloué pour l'occasion.

Dans les questions suivantes, on suppose que toutes les données sont correctement initialisées, et que des sockets réseau sont ouvertes sur chaque nœud pour communiquer avec chacun des autres nœuds. Une fonction `int get_socket(int neighbor_id)` permet de trouver le numéro de la socket permettant de communiquer avec `neighbor_id`.

▷ **Question 2 :** Écrire la fonction `send_LSA()` qui crée une structure LSA et l'envoi à un voisin en utilisant `get_socket()` et l'appel système classique `send()`. La fonction `send_LSA()` aura le prototype suivant :

```
void send_LSA(struct node_information* node, int destination);
```

▷ **Question 3 :** Écrire la fonction `receive_LSA()` dont le prototype est donné ci-dessous. Le paramètre `neighbor_id` identifie le voisin qui envoie les données. On peut l'utiliser avec la fonction `get_socket()`. Le paramètre `node` pointe sur la structure dans laquelle mettre les informations reçues. Cette fonction doit utiliser la/les fonctions habituelle·s pour recevoir des données depuis une socket déjà ouverte.

```
void receive_LSA(struct node_information *node, int neighbor_id);
```

Après réception, cette fonction doit faire suivre l'information à d'autres interlocuteurs comme indiqué au début de l'exercice.

▷ **Question 4 :** Écrire la fonction permettant de construire une matrice d'adjacence à partir des LSA de tous les noeuds du graphe. Cette fonction doit instancier, remplir et renvoyer une matrice de taille `nb_nodes * nb_nodes`. La valeur de la cellule `adj[i][j]` vaut :

- 1 si `j` est voisin de `i` et `i` est voisin de `j`
- 0 si `i` = `j`
- -1 sinon

```
int** build_network_map(struct node_information *node);
```

▷ **Question 5 :** Une fois que vous avez construit la matrice d'adjacence de votre réseau, quel algorithme vous permet de calculer les plus cours chemin vers tous les autres noeuds ? Rappelez rapidement son principe (sans l'implémenter).