ArcSys1 : Systèmes et Réseaux

Examen final du 17 décembre 2020 (2 h)

Tous documents interdits à l'exception d'un A4 recto manuscrit de votre main et du pense-bête de C fourni 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. **Le sujet est divisé en deux parties qui doivent être rendues sur des copies séparées.**

Partie 1 : Programmation en C (11 points).

À rendre sur une copie séparée.

★ Exercice 1 : Chaînes de caractères et fichiers (3 pts).

On cherche le caractère le plus fréquent dans une chaîne de caractères donnée.

▷ **Question 1 :** Écrivez une fonction `char plus_frequent(char* str)`, qui prend en paramètre la chaîne de caractères à analyser, et retourne le caractère le plus fréquent. Par exemple, `plus_frequent("Bon courage!")` doit retourner `'o'`.

▷ **Question 2 :** Écrivez un programme complet prenant une chaîne de caractères à analyser en argument de la ligne de commande, et affichant le résultat de la manière suivante.

```
$ ./plus-frequent "Bon courage!"
Le caractère le plus fréquent est: o
```

▷ **Question 3 :** 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 : Mémoire statique en C (3 pts).

Les six programmes de cet exercice compilent et ne provoquent pas d'erreur à l'exécution. Les `#include` et les fonctions `main` ont parfois été omis par souci de concision. Pour chacun des programmes, indiquez la sortie affichée après leur exécution, en justifiant par un schéma mémoire représentant les variables et le cadre de pile lorsque nécessaire. L'affichage de la sortie du fichier `fichier6.c` est en bonus.

fichier1.c

```c
int i = 5;

int f() {
    static int i = 2;
    return ++i;
}

int g() {
    return ++i;
}

int main(void) {
    printf("%d\n", f());
    printf("%d\n", g());
    printf("%d\n", f());
    printf("%d\n", g());
}
```

fichier2.c

```c
void triple(int* a) {
  *a = *a * 3;
}
int main(void) {
  int x = 14;
  triple(&x);
  printf("%d", x);
}
```

fichier3.c

```c
char s[] = "hello";
char* p = s + 1;
*(++p) = *s;
printf("%s", p-1);
```

fichier4.c

```c
int a[] = {4, 3, 2, 1, 0};
int* p[2] = {&a[1], &a[2]};
p[0] += *p[1];
printf("%d", **p);
```

fichier5.c

```c
int a[] = {2, 6, 3};
int* p = &a[0];
*(p+1) *= (*p)+1;
printf("%d", a[1]);
```

fichier6.c

```c
int i = 42;
int* pi = &i;
char* pc = pi;
*(pc++) = *(pi--);
printf("%d", i); // en bonus
```

★ Exercice 3 : Programmation C (5 pts).

▷ **Question 1 :** Soient les déclarations ci-dessous. Comment accède-t-on à l'année du 3ième timbre de la collection statique ?

a `collection[2,2]`
b `collection[2].annee`
c `collection.annee[2]`
d `collection[2][2]`
e `(collection+2)->annee`
f `collection.annee`

```
1 struct timbre {
2   int prix;
3   int annee;
4   char description[20];
5 };
6 struct timbre collection[5];
```

La solution précédente ne permet de stocker que 5 timbres dans la collection, et le champ `description` est extrêmement limité. On souhaite maintenant développer une solution pouvant stocker un nombre arbitraire de timbres, chacun doté d'une description de longueur arbitraire.

▷ **Question 2 :** Proposez une nouvelle version de la structure adaptée à ce souhait, ainsi que l'implémentation des fonctions de création et destruction de cette structure. La chaîne de caractères passée en paramètre à `timbre_new()` devra être copiée ailleurs en mémoire, et pourra être de taille arbitraire.

```
struct timbre* timbre_new(int prix, int annee, const char* desc);
void timbre_free(struct timbre* timbre);
```

On veut maintenant définir un type `album`, permettant de stocker un nombre variable de timbres. La structure de données et le prototype des fonctions associées sont donnés ci-dessous.

```
1 struct album {
2   ??? timbres ???; // (tableau de pointeurs) Q3
3   int taille;      // Nombre de timbres stockés dans le tableau
4 };
5 struct album* album_new();             // supposée connue
6 void album_free(struct album* album); // supposée connue
7
8 char* album_desc_max(struct album* album); // Q4
9 void album_add_timbre(struct album* album, int prix,int annee, const char* desc);//Q5
```

▷ **Question 3 :** Complétez cette structure pour que le champ `timbres` puisse être utilisé comme un tableau dont chaque case est un pointeur vers l'un des timbres stockés dans l'album.

On suppose données les fonctions de création et destruction d'un album de timbres. À la création d'un album, le champ `taille` vaut 0, et le champ `timbres` ne contient aucun contenu.

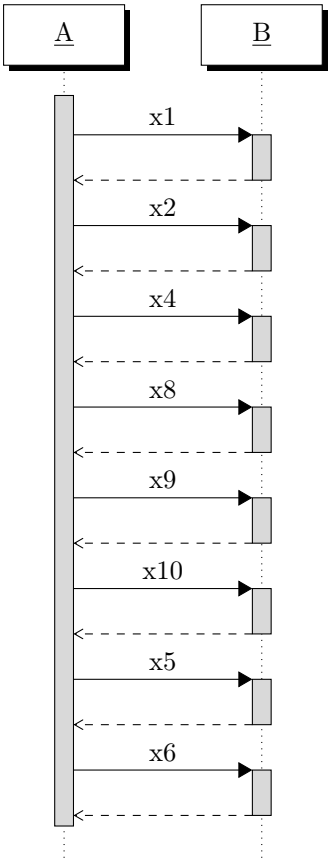
▷ **Question 4 :** Implémentez la fonction `album_desc_max()` de la ligne 8. Elle retourne la description du plus cher des timbres de l'album.

▷ **Question 5 :** Implémentez la fonction `album_add_timbre()` de la ligne 9. Elle crée un nouveau timbre à partir des paramètres, puis l'ajoute à l'album.

Partie 2 : Réseau (9 points).

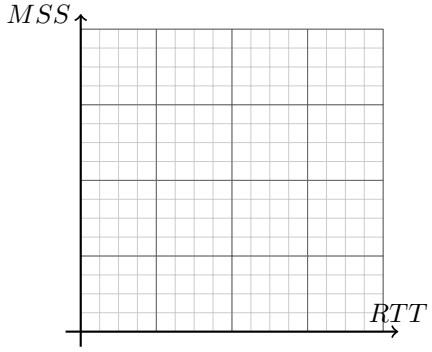
À rendre sur une copie séparée.

★ **Exercice 4 : TCP (3 pts).** Le diagramme ci-dessous représente les envois de paquets et les ACK échangés entre deux machines A et B. Rappelons que le Maximum Segment Size (MSS) est la quantité maximale d'octets pouvant être envoyés par un segment (équivalent au MTU moins la taille des en-têtes IP et TCP) et que le Round-Trip Time (RTT) est le délai de transmission pour envoyer un paquet plus le délai pour recevoir le ACK.



▷ **Question 1 :** Indiquez les intervalles de temps (en RTT) où le protocole TCP est en mode Slow Start, et ceux où le protocole est en mode Congestion Avoidance.

▷ **Question 2 :** Complétez le graphe ci-dessous représentant la taille de la fenêtre de congestion `cwnd` et le seuil slow start `ssthreshold` (en MSS) en fonction du temps (en RTT).
Vous pouvez répondre sur la feuille d'énoncé, ou refaire un schéma comparable sur votre copie.



▷ **Question 3 :** Un Timeout se produit après l'envoi du dernier segment représenté sur le schéma ci-contre. En supposant qu'aucune autre perte de paquet ne se produise par la suite, comment va évoluer la fenêtre de congestion après cet évènement ? Justifiez votre réponse, puis complétez le graphe.

★ **Exercice 5 : TCP (2 pts).** Questions indépendentes.

▷ **Question 1 :** 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 2 :** Une connexion est établie entre deux machines A et B, et a une latence de 200ms, un débit de 500Kb/s, et un MSS de 1460b. Combien de temps faut-il au protocole TCP pour transférer un fichier de 10Kb de A à B. On suppose qu'il n'y aucune erreur, aucune perte de paquets, aucune duplication de paquets. Justifiez la réponse.

★ **Exercice 6 : IP (4 pts).**

Un routeur a la table de routage suivante :

entrée	préfixe	port
1	0.0.0.0/0	1
2	34.110.0.0/16	2
3	129.231.20.0/24	2
4	128.0.0.0/2	3
5	67.30.0.0/16	3
6	67.0.0.0/10	3
7	0.0.0.0/1	4
8	128.10.87.0/24	4
9	70.10.0.0/16	4
10	70.10.10.0/16	4
11	129.0.0.0/8	5
12	130.0.0.0/8	5
13	50.20.0.0/18	6
14	50.30.0.0/18	6
15	50.30.0.0/16	6

▷ **Question 1 :** Pour chacune des adresses de destinations ci-dessous, indiquez le port de sortie, et listez les entrées de la table de routage correspondantes. Vous pouvez répondre sur la feuille d'énoncé, ou refaire un tableau comparable sur votre copie.

destination	port	entrées correspondantes
67.31.10.20		
128.138.242.5		
129.231.99.120		
131.80.70.33		
31.110.44.21		
34.120.44.21		
192.23.140.239		
50.20.24.162		

▷ **Question 2 :** La table de routage du routeur est-elle minimale ? Justifiez la réponse. Si elle ne l'est pas, proposez une table de routage équivalente mais minimale.