Compétition Informatique Pratique

des Jeux de Génie 2016

L'objectif de la compétition pratique d'ingénierie logiciel sera de concevoir et développer un programme communiquant avec un gestionnaire de mission et remplir les missions assignées par celui-ci. Ainsi, votre logiciel devra pouvoir résoudre une série de missions différente dans un minimum de temps afin de maximiser son score de performance.

Partie 1 - Communication

Votre première tâche sera d'implémenter un protocole de communication avec le gestionnaire de mission afin de pouvoir recevoir vos tâches et envoyer vos résultats. Le protocole de communication supporte plusieurs extensions afin d'ajouter des fonctionnalités plus avancées et sécuritaires.

La communication se fait par les entrée et sortie standard du programme (stdin et stdout)

Poignée de main

Le premier message envoyé doit se faire par votre client afin de spécifier les fonctionnalités supportées par celui-ci. Le premier message est une série de lettres représentant les fonctionnalités supportées par votre client au niveau du protocole.

- "T": Indique l'utilisation du protocole texte de base.
- "B": Indique l'utilisation du protocole binaire.
- "R": Indique l'utilisation de compression RLE
- "C": Indique l'utilisation de chiffrement
- "S": Indique l'utilisation de signature

Ainsi, si le premier message envoyé est "BRCS\n", le client indique qu'il **reçoit et envoie** des paquets binaires, compressés avec RLE, chiffrés avec AES et signés avec HMAC-SHA256. S'il envoie le message "BC\n", il indique que les messages seront binaires et chiffrés.

L'ordre des opérations est effectué dans le même ordre que la liste lors de l'envoi, c'est-à-dire qu'un message est compressé avant d'être chiffré et finalement signé.

L'utilisation du mode binaire est nécessaire afin de supporter le mode RLE, le chiffrement ou la signature de message. Le protocole binaire est mutuellement exclusif au protocole texte.

Protocole Texte (2 Pts)

La base du protocole consiste en un simple message CSV (Colon Separated Values).

Requête de mission

Le format d'un paquet d'une requête de mission est le suivant:

<Identifiant du message>:*<Type de mission>*:*<Paramètre de la mission><Saut de ligne (\n)>*

- **Identifiant de la requête**: Chaîne de caractère représentant un nombre non signé 32 bits sous sa forme hexadécimale. Ex.: 4f84a03f. Ce nombre est aléatoire, il permet d'identifier la requête et sera utilisé dans la réponse.
- **Type de mission**: Nombre décimal représentant l'identifiant du type de la mission. La liste des identifiants supportés est disponible dans la deuxième section. Ex.: 3.
- **Paramètre de la mission**: Données qui seront utilisées par votre programme afin de résoudre et calculer la réponse à la mission. Le format des données dépend du type de mission et est spécifié dans la deuxième section.
- **Saut de ligne**: caractère ASCII '\n' ou 0xa0

Exemple de requêtes avec deux missions:

4e1945f1:1:ping
9aaa352e:2:99999999

Réponse à une mission

Le format d'un paquet d'une requête de mission est le suivant:

<Identifiant de la requête>:*<Donnée de la réponse><Saut de ligne (\n)>*

- **Identifiant de la requête**: Chaîne de caractère représentant un nombre non signé 32 bits sous sa forme hexadécimale. Ex.: 4f84a03f. Cet identifiant indique la requête à laquelle la réponse s'applique.
- **Donnée de la réponse**: Résultat de la requête sous forme de chaîne de caractère.
- **Saut de ligne**: caractère ASCII '\n' ou 0xa0

Exemple de réponse des deux requêtes précédente:

4e1945f1:ping
9aaa352e:true

Protocole Binaire (6 Pts)

Les champs et leur ordre sont similaires au protocole de base. Par contre, plutôt qu'être envoyé et reçu sous forme texte, les messages sont transmis sous forme binaire et taille fixe dans le cas des nombres. Les nombres sont tous encodés en petit-boutisses (little-endian).

- **Taille du paquet**: Nombre 32 bits non signé indiquant la taille du paquet. La compression, le chiffrement ou la signature s'applique toujours sur le contenu du paquet et exclu les 4 octets de la taille du paquet.
- **Paquet**:
    - Requête
        - **Identifiant de la requête**: Nombre 32 bits non signé.
        - **Type de mission**: Nombre 8 bits non signé.
        - **Paramètre de la mission**: Paramètre sous forme de chaîne de caractère ASCII.
    - Réponse
        - **Identifiant de la requête**: Nombre 32 bits non signé
        - **Donnée de la réponse**: Résultat de la requête sous forme de chaîne de caractère

Ainsi, la requête et réponse "4e0000f1:1:AAAAAA" et "4e1945f1:AAAAAA" seraient encodés sous la forme suivante*:

*Le format hexadécimal est seulement utilisé pour la présentation de ce document, les messages reçus seront en binaire.

```
0b000000 f100004e 01 424242424242
^        ^        ^  ^
|        |        |  |
|        |        |  Paramètres ("AAAAAA")
|        |        Type de mission (1)
|        Identifiant (0x4e0000f1)
Taille du paquet (11 bytes)

0a000000 f100004e 424242424242
^        ^        ^
|        |        Réponse ("AAAAAA")
|        Identifiant (0x4e0000f1)
Taille du paquet (10 bytes)
```

Les messages n'ont pas de caractère séparateur comme le saut de ligne du protocole texte.

Compression RLE (7 Pts)

Afin de sauver de la précieuse bande passante, les paquets peuvent être compressés avec RLE (Run-length encoding). L'algorithme utilisé consiste à remplacer les répétitions d'octets par deux répétitions suivies du nombre de répétitions. Le nombre de répétitions doit être encodé sous forme d'octet non signé. Si le nombre de répétitions est supérieur à 255, simplement répétez les répétitions. Ex.: 'a' * 400 fois serait encodé sous la forme "`6161`**`ff`**` 6161`**`91`**".

Le paquets binaire précédent (`f100004e01424242424242`) serait donc encodé sous la forme: `09000000f1`**`000002`**`4e01`**`424206`** (Notez que la taille du paquet est ajusté)

Chiffrement AES (8 Pts)

Afin de rester sécuritaires et d'éviter que tout le monde puisse observer vos réponses, nous vous invitons à chiffrer vos messages avec AES en mode CBC. Le format des messages chiffré est sous la forme binaire: <IV><Message chiffré>

- **IV**: 16 octets, vecteur d'initialisation (Doit être unique pour chaque message).
- **Message chiffré**: Paquet paddé avec PKCS#7 chiffré avec AES.

Le message est d'abord paddé avec PKCS#7 **et** chiffré **par la suite** avec la clé "`ThisKeyIsSecuree`" et un vecteur d'initialisation aléatoire. PKCS#7 consiste à remplir le dernier bloc de 16 octets avec N bytes de valeur N répété où `N = 16 - (len(message) % 16)`. Ainsi, le message paddé "AA BB CC DD CC EE FF" donnerait: "AA BB CC DD EE FF 0A 0A 0A 0A 0A 0A 0A 0A 0A".

Le paquet compressé précédent: `f10000024e01424206`

chiffré avec l'IV "`22222222222222222222222222222222`**"** et la clé **"**`ThisKeyIsSecuree`**"**

serait
`20000000222222222222222222222222222222220`**`e1fed570638c859b01fbdeabe3b6c29`**

Signature HMAC (4 Pts)

Finalement, pour éviter que le message soit modifié lors de sa transmission, il est important d'y apposer une signature cryptographique. La signature sera un HMAC avec SHA256 en utilisant la même clé utilisée pour le chiffrement, soit "`ThisKeyIsSecuree`". Le format du message signé est: <Paquet><Signature> où

- **Paquet**: paquet signé
- **Signature**: 32 octets représentant la signature HMAC-SHA256 du paquet

Ainsi, le message crypté précédent:
`22222222222222222222222222222220e1fed570638c859b01fbdeabe3b6c29`

devient une fois signé (Signature en gras):

`40000000222222222222222222222222222222220e1fed570638c859b01fbdeabe3b6c29`**`75de836dcc759708350c33de33b39f51512825a58fa16fc62d24b3c6edb17d03`**

Partie 2 - Missions

Tel que défini dans la section précédente, chaque message contient un type de mission ainsi que ses paramètres. Cette section présente la liste des missions que vous avez à résoudre.

Outil de test

Un logiciel est fourni afin d'exécuter votre application et d'évaluer ses performances en la testant à un petit ensemble de données. Pour tester votre application, éditer "run.sh" en ajoutant les lignes de commande nécessaire pour exécuter votre application. Par la suite, exécutez les tests avec "`./jdg2016 run.sh -t tests.json`". Vous pouvez aussi utiliser le paramètre "-i 3" pour exécuter seulement la suite de tests de la mission 3.

Dans le cas où votre soumission ne supporte pas une mission, veuillez répondre à la mission avec un message contenant l'identifiant de mission et une réponse vide.

Notes

- Les données textes sont encodées en UTF-8.
- 30 secondes sont allouées pour répondre aux missions.
- Vous pouvez écrire des messages dans "stderr" afin de les voir affichés dans la sortie de l'outil de test.
- **Il est possible que les sorties de votre programme soient mises en tampon. Assurez-vous d'appeler "flush" sur la sortie standard après l'envoi de chaque message. Vous pouvez aussi désactiver la mise en tampons de stdout ("setbuf(stdout, NULL);" en C/C++).**

Mission 1 - Ping (2 Pts)

La mission est simplement de renvoyer une réponse avec la chaîne de caractère donnée en paramètre.

Exemple

#	Paramètre de Requête	Réponse
1	ping	ping
2	foobar	foobar

Mission 2 - Palindrome (3 Pts)

La mission est d'envoyer "true" si la chaîne donnée en paramètre est un palindrome ou "false" dans le cas contraire.

Exemple

#	Requête	Réponse
1	foobar	false
2	barrab	true
3	xyzazyx	true

Mission 3 - Trie (3 Pts)

La mission est de trier en ordre croissant la liste de nombres séparés par des virgules

Exemple

#	Paramètre de Requête	Réponse
1	6,5,4,3,2,1	1,2,3,4,5,6
2	6345, 234, 945	234,945,6345

Mission 4 - Compter (3 Pts)

La mission est de compter le nombre d'éléments distincts passés en paramètre séparé par des espaces. La réponse doit être sous la forme <Élément>;<Nombre> avec chaque élément séparé par des espaces, en ordre alphabétique.

Exemple

#	Paramètre de Requête	Réponse
1	c b a	a;1 b;1 c;1
2	aa a aa aa bb	a;1 aa;3 bb;1

Mission 5 - Recherche de texte (4 Pts)

La mission est d'envoyer le nombre d'instances d'une chaîne dearactère X dans le texte Y. Les deux paramètres sont séparés par un point virgule (;): <X>;<Y>

Exemple

#	Paramètre de Requête	Réponse
1	a;aabbccdda	3
2	aba;foobababa	2
3	foo;aabbcc	0

Mission 6 - Mathématique (4 Pts)

La mission contient une expression mathématique et vous devez renvoyer la réponse sous forme d'un nombre décimal avec une précision arrondie de deux chiffres après la virgule.

Opérations supportées: (),-,+,*,/,^,sin,cos,tan,e

Exemple

#	Paramètre de Requête	Réponse
1	1+4*(3-5)/7	-0.14
2	2^4	16
3	cos(1)+e	3.26

Mission 7 - Ping II (6 Pts)

La mission est d'envoyer une réponse vide après avoir attendu le temps en milliseconde transmis dans les paramètres de la requête. Une marge d'erreur de plus de 500 millisecondes est acceptée. Notez que le Ping ne doit pas bloquer votre application, celle-ci doit continuer à traiter d'autres missions pendant l'attente du délai du ping.

Exemple

#	Paramètre de Requête	Réponse
1	500	<Réponse vide envoyé 500ms après avoir reçu la requête>

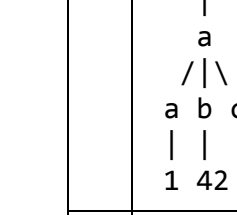
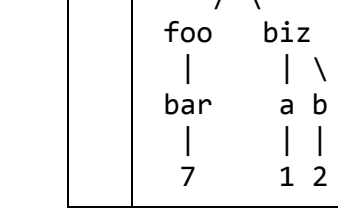
Mission 8 - Parsing (7 Pts)

La mission est de retrouver une valeur précise dans un arbre de propriété N-aire sous un format maison. Le premier paramètre est le chemin à parcourir de la feuille à trouver. Le chemin est représenté sous forme de clés séparé par des points (“.”). Un point virgule (“;”) sépare le chemin des données de l’arbre. Les données de l’arbre sont représentées sous forme de liste de valeurs entre parenthèses séparées par des virgules. Le premier élément de la liste est la clé du noeud et le reste est ses enfants. Si un noeud a un seul enfant, celui-ci est considéré comme une feuille de l’arbre. Par exemple, l’élément “(a, 1)” représente le noeud “a” avec la valeur “1”. Dans le cas où le chemin ne se rend pas à une feuille de l’arbre, la réponse doit être “-1”.

Le format de l’arbre est défini par la grammaire suivante:

Arbre := (Noeud)
Noeud := (Elements)
Elements: Clé, Liste | Clé, Feuille
Liste := Noeud | Noeud, Liste
Clé := [a-zA-Z0-9]
Feuille := [a-zA-Z0-9]

Exemple

#	Paramètre de Requête	Réponse
1	`a.b;(a, (a, 1), (b, 42), (c, 13))` Représente l’arbre: 	42
2	a.b.c;(a, (b,1), (c,2))	-1
2	a.b;(a, (b, (c,2)))	-1
3	`foo.bar;(foo, (bar, 7),(biz, (a, 1), (b, 2)))` Représente l’arbre: 	7

Mission 9 - ASCII Art (10 Pts)

Cette mission consiste à traduire sous forme texte un nombre représenté sous forme de dessin ASCII. Les glyphes utilisés sont prédéfinis et alignés verticalement. Par contre, il est possible que des espaces soient insérés entre les lettres. Les sauts de ligne du dessin ASCII sont remplacés par des points-virgules (“;”). Les glyphes utilisés sont les suivantes:

1	**2**	**3**	**4**	**5**

##
 #
 #
###
	##
	# #
	  #
	####
		###
		  ##
		####
		###
			# #
			# #
			###
			  #
				###
				##
				  #
				###

6	**7**	**8**	**9**	**0**
## # ### ###	### # # #	### # # ### ###	### ### # ##	### # # # # ###

Exemple

#	Paramètre de Requête	Réponse
1	## ## ### ; # # # ##; # # ####; ### #### ### ; (Les sauts de ligne sont ajoutés pour la présentation seulement)	123
2	# # ###; # # # ; ### # ; # # ; (Les sauts de ligne sont ajoutés pour la présentation seulement)	47

Mission 10 - Mathematique II (10 Pts)

Cette mission est similaire à Mathematique I, par contre cette fois il vous est demandé de trouver la valeur de “x” dans une expression d’égalité algébrique. La réponse doit être sous forme d’un nombre décimal avec une précision arrondie de deux chiffres après la virgule.

Opérations supportées: -,+,*,/,^

Exemple

#	Paramètre de Requête	Réponse
1	x=10+1	11
2	1 + x * 1000 = 126 - 2	0.12
3	2 ^ x = 8	3

Mission 11 - Labyrinthe (10 Pts)

La mission est de trouver un chemin pour compléter le labyrinthe reçu en paramètre. Des points additionnels sont donnés si la solution donnée est le chemin le plus rapide. Le labyrinthe est représenté sous la forme:

```
###e#;
#...#;
#s###
```

où ‘e’ est le départ, ‘s’ la sortie, ‘#’ un mur, ‘ ‘ un passage libre et ‘;’ le délimiteur de ligne. Les sauts de ligne sont ajoutés pour la présentation seulement.

La réponse est envoyée sous le format de caractère séparé par des espaces indiquant les directions à prendre à partir du début avec ‘U’, pour haut, ‘R’ pour droite, ‘D’ pour bas et ‘L’ pour gauche.

Exemple

#	Paramètre de Requête	Réponse
1	###e#; # # #; # #; #s### (Les sauts de ligne sont ajoutés pour la présentation seulement)	D D L L D

Mission 12 - Labyrinthe II (13 Pts)

Cette mission est similaire à la **mission 11 - Labyrinthe**. Par contre, le format du labyrinthe envoyé en paramètre est isométrique. Les directions possibles sont ‘UR’ pour la diagonale haut-droite, ‘DR’ pour la diagonale bas-droite, ‘DL‘ pour la diagonale bas-gauche et ‘UL’ pour la diagonale haut-droite.

Exemple

#	Paramètre de Requête	Réponse

1	<pre>/\ ;
	\e\ /\ ;
	 \ \ \s\ ;
	 \ \ \ \/\ ;
	 / / / /\ \ ;
	 \ \/ \/ / ;
	 \ /\ / ;
	 \/ \/ ;</pre>
	(Les sauts de ligne sont ajoutés pour la présentation seulement)
	
	Ce labyrinthe peut être visualisé de la façon suivante:
	
	<pre> ###
	 #e#
	 # #
	## ###
	# #s#
	# ## #
	# ##
	### # #
	 # #
	 #####</pre>
		DR DR DR DL DR DR UR UR UR UL UL

Remise

Votre soumission doit être envoyée via une archive compressée avec le nom de l'université comme nom de fichier à l'adresse informatique@jeuxdegenie.qc.cq.

L'archive doit contenir:

- Les sources de votre programme.
- Votre programme compilé s'il y a lieu.
- Le fichier run.sh qui exécute votre programme.

La correction sera effectuée en exécutant la commande "./jdg2016 run.sh -t correction.json" dans le même environnement donné au début de la compétition.

Pointage

Tâche	**Critère**	**Point**
Protocole	Protocol Texte	2
	Protocol Binaire	4*
	Protocol Compressé	7
	Protocol Chiffré	8
	Protocol Signé	4
Mission 1 - Ping	Exactitude des réponses	2
Mission 2 - Palindrôme	Exactitude des réponses	3
Mission 3 - Trie	Exactitude des réponses	3
Mission 4 - Compter	Exactitude des réponses	3
Mission 5 - Recherche de texte	Exactitude des réponses	4
Mission 6 - Mathématique	Exactitude des réponses	4
Mission 7 - Ping II	Exactitude des réponses	6
Mission 8 - Parsing	Exactitude des réponses	7
Mission 9 - ASCII Art	Exactitude des réponses	10
Mission 10 - Mathématique II	Exactitude des réponses	10
Mission 11 - Labyrinthe	Exactitude des réponses	6
	Chemins optimale	4
Mission 12 - Labyrinthe II	Exactitude des réponses	10
	Chemins optimale	3
Total		100

*Les points du protocol binaire sont ajoutés au protocol texte pour un total de 6 points.