ESISAR

NE302 – Projet Réseau
Projet « HTTP Server » - part 2

Table des matières

1 Objectifs de l'étape 2:

Développement d'un parseur de requêtes HTTP.

Vous devrez prendre en compte la grammaire ABNF fournie dans les RFC723x et relatives Cette grammaire définit le format des messages HTTP échangés entre client et serveur.

L'objectif de cette partie est de réaliser un analyseur syntaxique permettant d'extraire certains champs d'une requête HTTP (comme vous réalisez un serveur HTTP, l'analyse des réponses n'a pas d'intérêt ici.)

Les champs d'une requête seront indiqués par les « rulenames» de la grammaire ABNF fournie dans les Annexes des rfc723x.

Exemple :

*HTTP-message = start-line *(header-field CRLF) CRLF [message-body]*
start-line = request-line / status-line
request-line = method SP request-target SP HTTP-version CRLF
header-field = field-name ":" OWS field-value OWS
...
...
…

Ainsi si l'on considère la requête suivante :

GET /index.html HTTP/1.0
Host: www.esisar.grenoble-inp.fr
Transfer-Encoding: gzip,chunked

Si on recherche le champ start-line ou request-line, votre parseur devra trouver :
GET /index.html HTTP/1.0

Si on recherche le champ method, le parseur devra trouver
GET

Si on recherche le champ *request-target*, le parseur devra trouver
/index.html

Si on recherche le champ *field-name*, le parseur devra trouver
Host
Transfer-Encoding

2 Périmètre du travail

La RFC 7230 3.2.4 indique :

> *Messages are parsed using a generic algorithm, independent of the individual header field names. The contents within a given field value are not parsed until a later stage of message interpretation (usually after the message's entire header section has been processed). Consequently, this specification does not use ABNF rules to define each "Field-Name: Field Value" pair, as was done in previous editions. Instead, this specification uses ABNF rules that are named according to each registered field name, wherein the rule defines the valid grammar for that field's corresponding field values (i.e., after the field-value has been extracted from the header section by a generic field parser).*

C'est pourquoi dans notre cas nous utiliserons une autre definition de la règle header-field, cette définition indiquera tous les entêtes que votre parseur devra au minimum être en mesure de comprendre :

```
Cookie-header = "Cookie" ":" OWS cookie-string OWS
Content-Length-header = "Content-Length" ":" OWS Content-Length OWS
Transfer-Encoding-header = "Transfer-Encoding" ":" OWS Transfer-Encoding OWS
Host-header = "Host" ":" OWS Host OWS
Referer-header = "Referer" ":" OWS Referer OWS
User-Agent-header = "User-Agent" ":" OWS User-Agent OWS
Accept-header = "Accept" ":" OWS Accept OWS
Accept-Encoding-header = "Accept-Encoding" ":" OWS Accept-Encoding OWS
Connection-header = "Connection" ":" OWS Connection OWS

header-field = Content-Length-header /
        Transfer-Encoding-header /
        Cookie-header /
        Host-header /
        Referer-header /
        User-Agent-header /
```

Accept-header /
Accept-Encoding-header /
Connection-header /
(field-name ":" OWS field-value OWS)

3 Détail du travail demandé

Vous devrez réaliser un parseur qui effectue une validation syntaxique de votre message.

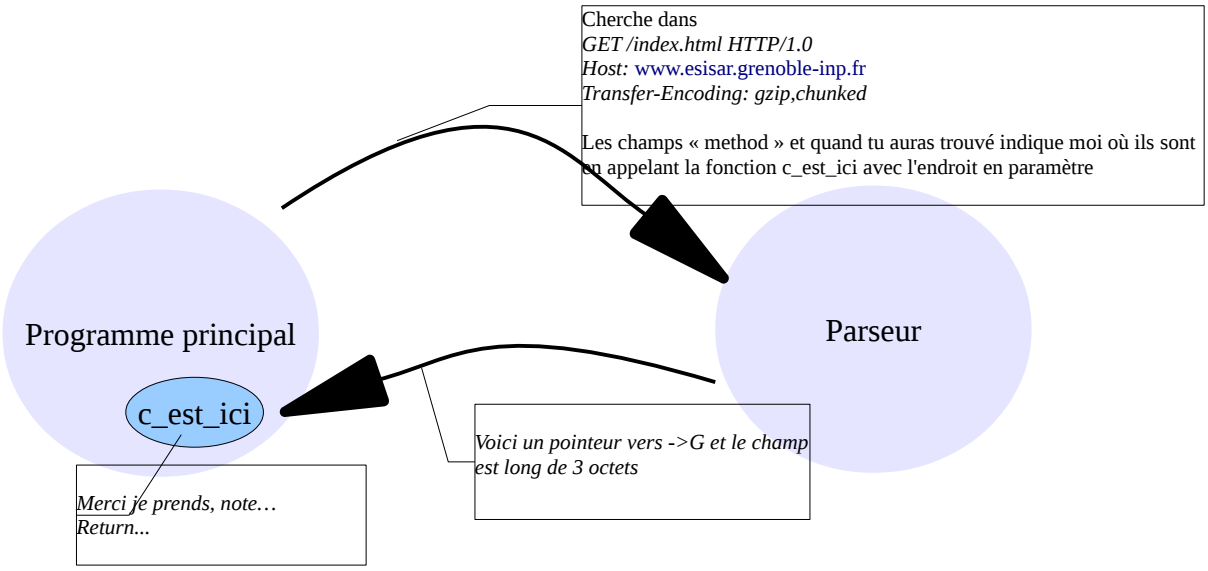
Pensez dès maintenant comment vous pourrez utiliser ce parseur dans l'intégralité de votre projet, notamment quand et comment votre code appelera ce parseur, que devra t-il vous retourner ? IL est necessaire de réaliser l'algorithme de votre programme afin de définir l'interface de votre parseur.

Réfléchissez-y. Réfléchissez-y encore !!

4 API pour la validation et l'évaluation de votre parseur

Afin de pouvoir tester votre parseur, vous devrez respecter cette API de test. Cette API de test peut être complémentaire à l'interface que vous avez définie pour votre code.

Votre parseur devra utiliser la notion de callback (ici une forme dégénérée du design pattern observer), voici comment celui-ci se comporte, nous avons votre parseur et le programme principal:



Cette architecture nous permettra d'automtiser les tests de votre parseur et d'envoyer de nombreux jeux de tests.

Vous devrez vous conformer à l'API suivante :

Les résultats du parseur (champs trouvés) sont toutes les occurrences du champ demandé dans une requête donnée.

Aussi votre parseur prendra en entrée un flux d'octets à analyser.

Le corps du message HTTP pouvant contenir un flux binaire, l'analyse de ce flux d'octet ne peut se faire comme on traiterait une chaine de caractère en C. (algorithme de la sentinelle).

Ce flux d'octets en entrée est donc représenté par :

```
char *buf ;          //   (pointeur vers le premier caractère du flux)
```

```
int len ;                    //    (longueur du flux à traiter)
```

Le parseur doit aussi recevoir le ***nom du champ à trouver dans la grammaire*** (donc chaine de caractère)
Le champ à trouver sera donc :

```
char *search ; //    (champs à chercher)
```

Et la fonction de callback à appeler quand on a trouvé le champ, ici un pointeur vers une fonction.

```
void (*callback)() ;        //       (fonction de callback)
```

Prototype de votre parseur (à respecter) :

```
int parser (char *buf,unsigned int len,char *search,  void (*callback)()) ;
```

Cette fonction retourne la valeur -1 si le message est valide syntaxiquement, sinon un entier corrrespondant à la l'indice dans la chaine buf où votre parseur à détecter une erreur syntaxique.

Prototype de la fonction de callback (à respecter) :

```
void callback (char *found,unsigned int len) ;
```

5 Exemple de code

```
void getField(char *s,int len)
{
       printf("le parseur à trouvé %.*s\n",len,s);
       return;
}
int main(int argc, char *agv[])
{
       char req[]="GET / HTTP/1.0\r\nHost: www.google.com\r\nTransfer-Encoding: gzip\r\n\r\n";
       parseur(req,strlen(req),"method",&getField);
}
```

6 Tests

Un jeu de test vous sera fourni (requêtes, champ à trouver, et résultats) afin de tester votre parseur, un autre jeu de test pourra être utilisé pour l'évaluation.