Monitor

Introduction

Le sujet du laboratoire est l'élaboration d'un *système de monitoring décentralisé*.

L'objectif est de vous *faire découvrir quelques protocoles applicatifs* (HTTP, FTP, …), de *vous permettre d'implémenter un protocole* (avec les différentes étapes comme l'analyse syntaxique des messages, la compréhension et la réponse à ceux-ci), de *vous permettre d'aborder la programmation concurrente* (permettant plusieurs connexions simultanées), de *vous faire programmer différents types de communication* (unicast avec TCP et multicast en UDP).

Le travail sera réalisé, par groupe **de 3 étudiants**. Exceptionnellement, et avec l'accord de votre responsable de laboratoire, des groupes de 4 étudiants (avec des objectifs plus grands) seront possibles si le nombre d'étudiants n'est pas un multiple de 3.

Le laboratoire comporte **28h** dont **24h seront dédiées** à la réalisation de cet exercice. Si vous prenez du retard, il sera nécessaire d'avancer par vous-même.

Pour ce laboratoire, **vous devez utiliser le GitLab HELMo**. Ainsi, on vous demande de déterminer rapidement quel étudiant hébergera le dépôt pour ce travail.

Un planning est proposé à la suite de la présentation de l'énoncé. Veillez à le respecter scrupuleusement.

1. L'architecture

Dans notre système, nous distinguerons (voir figure 1) :

- Les **services** qui sont les éléments que l'on souhaite surveiller. Cela peut être un site web, un répertoire FTP utilisé pour le backup, un service d'envoi de mail (SMTP), un service de réception des mails (POP3 ou IMAP), … Ces services ne doivent pas être programmés, on vous proposera de surveiller des éléments existants.
- Les **sondes** qui sont des petits serveurs capables de monitorer un service particulier. Ainsi, nous avons une sonde spécifique pour surveiller les sites web (protocole HTTP), une autre pour analyser les fichiers présents sur un serveur FTP, … Ces sondes doivent être programmées et respecter le protocole applicatif en question.
- Le **concentrateur** est le collecteur d'information. Il permet *de configurer les sondes lorsque celles-ci deviennent actives*, de *réceptionner les données provenant des sondes*, de *transmettre les informations aux clients*, de *permettre à un client d'ajouter un élément à surveiller*.
- Le **client** est un programme avec une interface graphique simple permettant d'afficher les différents services et leurs états (OK, ALARM, DOWN).

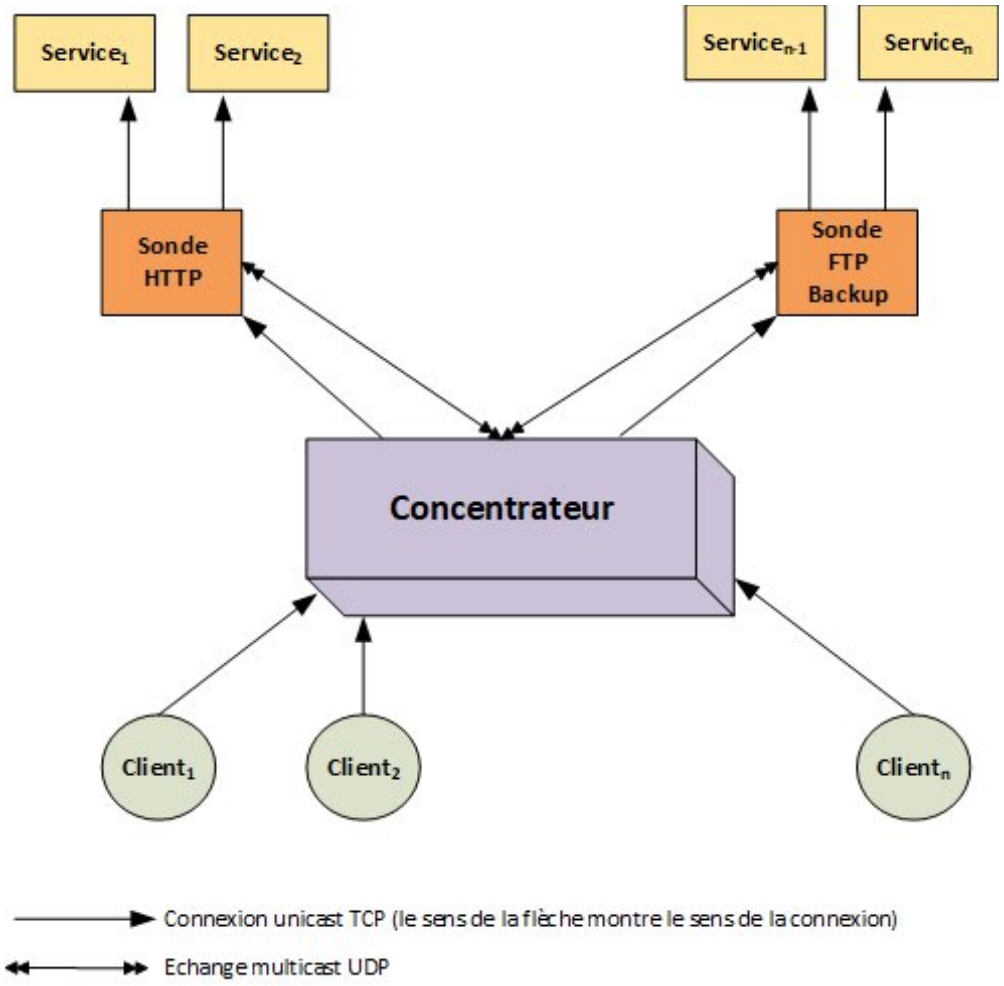

Figure 1 : Architecture du système

Comme montré sur la *figure 1*, l'architecture proposée est variée puisqu'on y trouve des protocoles applicatifs différents (HTTP,FTP, …), des protocoles de transport variés (TCP et UDP) et des modes de communication diversifiés (unicast et multicast).

Les éléments qui devront être implémentés sont : *les clients*, *le concentrateur* et *les sondes*. Les *services* à surveiller seront présents sur le réseau de l'école ou trouvé sur internet.

1.1 Fonctionnement

Les *clients* sont responsables d'afficher l'état des *services* à l'utilisateur. Nous supposerons que tous les clients peuvent visualiser l'état de l'ensemble des *services*. L'état d'un *service* peut être OK (tout est en ordre), ALARM (un problème est rencontré) ou DOWN (aucune connexion possible). L'état ALARM est déterminé par la sonde (fichier inexistant, temps de réponse trop élevé, …). Les *clients* peuvent être géographiquement distants, d'où l'intérêt d'en avoir plusieurs. Enfin, les *clients* n'ont pas qu'une tâche d'affichage, ils doivent aussi permettre d'ajouter des nouveaux services à superviser.

Le *concentrateur* reçoit les demandes des différents clients. Il faut bien mesurer que plusieurs clients peuvent interagir avec le concentrateur en même temps. Lorsqu'un *client* demande la supervision d'un *service*, le *concentrateur* configurera la *sonde* qui est en charge de ce protocole précis[1]. Le *concentrateur* doit ainsi établir la connexion avec la sonde lorsque :

- Celle-ci s'annonce, en multicast, avec le protocole applicatif qu'elle supporte pour lui transmettre la configuration
- Celle-ci mentionne, en multicast, que des nouvelles données sont disponibles. Le *concentrateur* doit alors récupérer celles-ci à destination des *clients* lorsque ceux-ci en effectueront la demande.

Le *concentrateur* est chargé de sauvegarder, sur disque, la configuration reçue (ie. les *services* à interroger). Les informations concernant l'état des services est, quant à elle, gardée en mémoire.

Lors de son démarrage, chaque *sonde* annonce, en multicast, sa présence au *concentrateur*, rapportant le protocole supporté. Comme indiqué, le *concentrateur* pourra alors envoyer à la *sonde* sa configuration. A ce titre, il faut bien remarquer que la *sonde* ne sauvegarde pas les services à interroger, c'est le *concentrateur* qui doit lui transmettre ces informations. La *sonde* interrogera à intervalle régulier les services qu'elle doit superviser. Quand des informations sont disponibles (état d'un service, …), elle signale, en multicast, au *concentrateur* qu'il peut se connecter pour récupérer celles-ci. Il faut donc bien noter que les liens entre la *sonde* et *le(s) services* mais également entre le *concentrateur* et la *sonde* sont **intermittents** : la connexion est effectuée au moment opportun pour récupérer les informations, elle n'est pas maintenue au-delà du temps nécessaire pour rapatrier celles-ci.

Sans compter les **extensions** à prévoir, vous devez supporter 2 types de sonde : la sonde HTTP qui peut déterminer si un site déterminé (ou un fichier) est toujours bien accessible et la sonde FTP qui a pour but de vérifier la présence d'un fichier dans un dossier donné (ie. pour vérifier qu'un backup a bien été effectué, par exemple). Il y a donc plusieurs applications différentes à développer. Le choix du langage est laissé à votre appréciation (avec accord[2] du professeur de laboratoire).

1.2 Scénario

Dans ce paragraphe, nous allons découvrir le scénario le plus courant de cette application :

1. Le *concentrateur* démarre et lit la configuration qu'il avait sauvegardée. Il écoute, en multicast, pour entendre les annonces des différentes *sondes*. Il répertorie, pour chacune, le protocole applicatif supporté. A chaque annonce, il va envoyer la configuration courante à la sonde.
2. Les *sondes* sont démarrées et s'annoncent en multicast au *concentrateur*. Cette annonce a lieu toutes les 60 secondes. Il faut noter que le concentrateur enverra automatiquement, à la réception de cette annonce, la configuration courante de la sonde. A charge de celle-ci de mettre à jour les services qu'elle surveille. En respectant la fréquence demandée, la sonde interroge l'état de chaque service. Elle maintient à destination du concentrateur le dernier état connu. Enfin, quand la sonde a terminé d'interroger les services, elle annonce, en multicast, au *concentrateur* que de nouvelles données sont disponibles.

[1] Nous supposerons, par simplicité, qu'une sonde est en charge d'un seul protocole applicatif : HTTP, FTP, …
[2] Si vous choisissez une autre technologie que Java ou C#

3. Lorsque le *concentrateur* reçoit l'annonce, il peut récupérer, depuis la *sonde*, les différentes données disponibles afin de les transmettre aux clients qui en feraient la demande.
4. Le *client* peut demander de récupérer toutes les données disponibles pour tous les services supervisés (il n'y a pas d'identification des clients à proprement parler). Le *concentrateur* transmet alors les informations qu'il a mémorisée lors de ses échanges avec les *sondes*. Le client peut également demander la surveillance d'un nouveau *service* :
    a. Le *concentrateur* peut accepter / refuser suivant qu'il dispose d'une sonde capable de gérer ce protocole.
    b. Si la demande est acceptée, la configuration est mise à jour et le concentrateur transmettra (en répondant au message d'annonce de la sonde), les éléments à surveiller (incluant les nouveaux ajoutés par le client).

2. Protocole

Dans cet énoncé, il y a plusieurs protocoles applicatifs. Il y a des protocoles standards comme HTTP ou FTP, des protocoles spécifiques comme celui entre le *client* et le *concentrateur*, celui entre le *concentrateur* et les *sondes* ou encore celui utilisé en multicast par les *sondes* pour s'annoncer ou notifier que des informations sont disponibles.

Tous les protocoles sont décrits sous la forme d'une *grammaire ABNF*[3]. Ainsi, la syntaxe est strictement spécifiée et vos applications doivent s'y conformer précisément.

2.1 Définitions standards

```
lettre = %d65-90 / %d97-122      ; [A-Z][a-z]
chiffre = %d48-57                ; [0-9]
lettre_chiffre = lettre / chiffre ; une lettre ou un chiffre
crlf = %d13 %d10                 ; \r suivi de \n
port = 1*5chiffre                ; 1 à 5 chiffre (1 à 65535)
caractere = %d32-255             ; tous les caractères imprimables et l'espace
caractere_pass = %d33-255 ; tous les caractères imprimables
sp = %d32                        ; espace
id = 5*10lettre_chiffre          ; 5 à 10 lettres/chiffres
protocole = 3*15lettre_chiffre ; 3 à 15 lettres/chiffres
utilisateur= 3*50lettre_chiffre ; 3 à 50 lettres/chiffres
mot_passe = 3*50caractere_pass ; 3 à 50 caractères imprimables
site = 3*50(lettre_chiffre / "." / "_" / "-")
chemin = "/" 0*100(lettre_chiffre / "." / "_" / "/" / "-")
url = protocole "://" [utilisateur ":" mot_passe "@"] site [":" port] chemin
frequence = 1*8chiffre           ; 1 à 8 chiffres
etat = "OK" / "ALARM" / "DOWN" ; les différents états
message = 1*200caractere
```

2.2 Protocole entre le client et le concentrateur

Le protocole entre le *client* et le *concentrateur* se décrit comme suit :

```
requete_ajout_service = "SERVICE_ADD" sp id sp url sp frequence crlf
reponse_ajout_service = ("+OK" | "-ERR") [message] crlf
requete_liste_service = "SERVICE_LIST" crlf
reponse_liste_service = "SERVICE_INFO" 0*100(sp id) crlf
```

[3] Augmented Backus-Naur Form (RFC 5234 - https://tools.ietf.org/html/rfc5234)

```
requete_info_service = "SERVICE_RETR" sp id crlf
reponse_info_service = "SERVICE_DATA" sp id sp url sp etat crlf
```

Le message *requete_ajout_service* est envoyé par le *client* pour demander la supervision d'un nouveau *service*. La réponse *reponse_ajout_service* est renvoyée pour confirmer l'ajout du *service*. Le message *requete_liste_service* est envoyé par le *client* pour obtenir la liste des *services* qui sont supervisés. Le message *reponse_liste_service* est retourné par le *concentrateur* au *client* avec la liste des identifiants des *services*. Le message *requete_info_service* permet à un *client* de demander l'état du *service* dont l'identifiant est mentionné. Le message *reponse_info_service* est la réponse à la requête précédente.

La *fréquence* mentionné dans *ajout_service* est le délai entre 2 requêtes successives de la sonde. Il est exprimé en secondes. L'*url* mentionne tous les éléments importants (protocole, site distant, port éventuel, nom d'utilisateur et mot de passe, chemin vers la ressource).

2.3 Protocole entre le concentrateur et la sonde

Le *concentrateur* contacte la *sonde* à 2 moments : après avoir reçu l'annonce de la *sonde* ou après avoir reçu la notification que des données doivent être récupérées.

```
envoi_config = "PROBE_CONFIG" sp 0*100(id "!" url "!" frequence sp) crlf
requete_info_sonde = "PROBE_LIST " crlf
reponse_info_sonde = "PROBE_INFO" sp 0*100(id sp) crlf
requete_donnee_sonde = "PROBE_RETR" sp id crlf
reponse_donnee_sonde = "PROBE_DATA" sp id sp etat crlf
```

Le message *envoi_config* est envoyé par le *concentrateur* à la *sonde* (directement en unicast) pour lui transmettre les *services* que celle-ci doit superviser. Ce message est envoyé en réponse à chaque *annonce* (reçue en multicast). Le message *requete_info_sonde* est envoyé par le *concentrateur* pour obtenir la liste des *services* supervisés. Le message *reponse_info_sonde* est envoyé en réponse au message précédent, il sert à fournir la liste des *services* supervisés. Le message *requete_donnee_sonde*, envoyé par le concentrateur, permet de demander l'état d'un *service* dont l'identifiant est mentionné. Le message *reponse_donnee_sonde* est la réponse au message précédent mentionnant l'état demandé.

2.4 Echanges entre les sondes et le concentrateur

Les *sondes* envoient leurs messages en *multicast* vers le *concentrateur*. En effet, il n'y a pas de connexion de la sonde vers le concentrateur.

Les échanges multicast ont lieu à 2 moments : lorsque la sonde s'annonce (ie. dès que celle-ci est démarrée et toutes les minutes), et lorsque des données sont disponibles.

```
annonce_sonde = "PROBE_ANNOUNCE" sp protocole sp port crlf
annonce_donnee = "PROBE_NEWDATA" sp protocole sp port crlf
```

Le message *annonce_sonde* est envoyé par la sonde en *multicast* sur l'adresse `239.255.40.15` toutes les 60 secondes en mentionnant le protocole applicatif supporté. Le concentrateur répondra par le message *envoi_config*. Le message *annonce_donnee* est envoyé en multicast par la *sonde* pour informer le *concentrateur* que des nouvelles données sont disponibles. Le port mentionné est celui que le *concentrateur* doit utiliser, en unicast, pour se connecter à la *sonde*. Il faut noter que l'adresse IP de la *sonde* est obtenue en analysant la source de l'annonce.

2.5 Les protocoles applicatifs standards

Dans cet énoncé, nous avons mentionné que deux sondes devaient, au moins, être implémentées : une pour le protocole HTTP et une autre pour le protocole FTP.

La sonde HTTP a comme objectif de vérifier qu'un site donné est bien accessible (retour d'un état `2XX` ou `3XX`) à une requête GET. Si l'état est différent, le service doit être positionné en ALARM et si la connexion n'a pu être établie, le service doit être marqué avec un état DOWN. J'attire votre attention sur le fait que le protocole HTTP (en version 1.1) est largement décrit[4]. A priori, dans cette sonde, nous n'utiliserons pas d'authentification (même si cela peut compter comme une extension – voir plus loin).

La sonde FTP a un but un peu différent, elle doit vérifier qu'un fichier déterminé est bien présent sur le site FTP renseigné. Ceci doit permettre, par exemple, de valider qu'une sauvegarde a bien eu lieu. Dans le cas du FTP, il est nécessaire de supporter une authentification. Ici aussi, le protocole FTP est largement défini[5].

J'attire votre attention sur le fait que des libraires existent probablement pour gérer ces protocoles applicatifs, vous êtes libres de les utiliser ou non.

3. Extensions

Il faut noter que le présent énoncé est *une base* pour le laboratoire demandé. En effet, il est nécessaire d'étendre celui-ci en implémentant des *options* laissées à votre choix.

- **Extensions techniques**
  - Implémenter d'autres protocoles applicatifs comme POP3, SMTP, IMAP, DNS, …
  - Supporter les versions sécurisées (HTTPS, FTPS) et vérifier les certificats
- **Apprentissage de langages**
  - Implémenter tout ou une partie du laboratoire dans un langage différent (PERL, Python, C, C++/Qt, NodeJS, …). Avant de vous lancer, discutez-en avec votre professeur de laboratoire !
- **Extension de fonctionnalités**
  - Implémenter une vue spécifique par client (avec authentification et mémorisation des services surveillés pour un client donné)
  - Sécuriser tous les échanges entre le client et le concentrateur
  - Mémoriser les données de manière chiffrée

4. Implémentation

L'implémentation à réaliser :

- Sauf choix discuté avec votre professeur de laboratoire, devra être écrite en C# / Java en respectant les principes de programmations objets. Vous pouvez choisir d'implémenter des éléments dans des langages différents.
- Vérifier tous les messages du protocole par *des expressions régulières*[6]. La plupart des langages proposent des implémentations de celles-ci.

---

[4] Notamment dans le RFC2616 – http://www.ietf.org/rfc/rfc2616.txt
[5] Notamment dans le RFC959 – http://www.ietf.org/rfc/rfc959.txt
[6] Le site https://regexr.com/ vous permet de tester celles-ci facilement.

5. Planning

Voici le planning à respecter pour ce laboratoire :

Phase	Description	Heures
1	Implémentation sondes HTTP & FTP backup. Tester avec un fichier de configuration en local. Implémentation du protocole applicatif	**6**
2	Implémentation du concentrateur et du dialogue direct avec les sondes. Fichier de configuration local pour connaitre les sondes. Implémentation du protocole spécifique §2.3	**4 - 6**
3	Implémentation des échanges multicast et des réponses à apporter aux messages d'annonces et de nouvelles données. Respect du protocole spécifique §2.4.	**2**
4	Implémentation du client avec une interface légère. Implémentation du protocole spécifique §2.2	**4**
5	Extensions	**6**
Eval	Présentation du projet	**4**

Les phases 1 à 4 doivent être terminées pour espérer la réussite dans le cadre de ce laboratoire. Si les phases 1 à 4 **sont terminées et conformes EXACTEMENT à ce qui est demandé**[7], la note obtenue sera de **MAXIMUM** 12/20. Pour espérer obtenir une note supérieure, il faut terminer la 5ème phase.

Une présence **d'au moins à la moitié des laboratoires de tous les membres du groupe** est requise pour être autorisé à défendre le projet.

La version qui sera défendue sera celle **déposée le mardi 15 mai 2018 à 23h55** sur le GitLab HELMo au plus tard.

La dernière séance est réservée, dans chaque groupe, à la présentation orale de votre projet et son évaluation sur base de votre défense.

Une évaluation des pairs vous sera également demandée, individuellement. Celle-ci pourra être utilisée pour ajuster la note de chaque membre du groupe.

[7] Autrement dit, si nous n'avons aucune remarque concernant l'implémentation et le fonctionnement de votre application