TP7: Mini-cloud FaaS, orchestration, load balancing

Matthieu Lemerre Guillaume Girol Grégoire Menguy

IN201 - Cours de système d'exploitation

L'idée de ce TP est de montrer comment on peut créer un mini-cloud de type FaaS, permettant l'instantiation automatique de programmes pour répondre à des demandes provenant de connections internet. Les "fonctions" que nous allons implémenter seront des processus UNIX standards, qui communiqueront à l'aide de leur entrée et sortie standard.

Question 1. *Le but de cette question est d'écrire une "fonction" qui suit le protocole présenté ci-dessus. Écrivez un programme* `toupper.c` *qui fait la chose suivante en boucle:*

1. *Lire un texte sur son entrée standard dans un buffer de taille maximale 1024 octets. Si son entrée standard est coupée (end of file), le programme doit s'arrêter (utilisez la fonction* `fgets`*; celle-ci renvoie NULL si on reçoit "end of file").*
2. *Attendre 3 secondes (utilisez* `sleep(3)`*). Le but de ce rajout est de simuler un temps d'exécution plus important.*
3. *Modifier le buffer pour convertir les minuscules en majuscule (fonction* `toupper`*)*
4. *Affiche sur la sortie standard (à l'aide de* `printf` *et* `fflush`*):*
   - *Le nom de la machine sur laquelle il tourne (fonction* `gethostname`*)*
   - *Suivi du numéro du processus (fonction* `getpid`*)*
   - *Suivi du texte capitalisé.*

Question 2. *Testez votre programme en le lançant ainsi:* `./toupper`*, et vérifiez qu'il sait ainsi prendre plusieurs requêtes à la suite. Testez le également ainsi:* `echo -e "hello22\nbonjour12" | ./toupper`*, et vérifiez ainsi qu'il sait prendre en compte les fins de fichier.*

Question 3. *Nous allons maintenant automatiquement convertir ce programme en un service accessible par le réseau. Le programme suivant ouvre un serveur TCP qui écoute sur le port 8888. Testez son exécution en utilisant l'outil* **nc**, *qui permet d'envoyer des données sur le réseau:* `echo -e "toto\ntutu" | nc localhost 888`

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdlib.h>
#include <strings.h>

void error(char *msg)
{
    perror(msg);
    exit(1);
}

int main(void)
{
     int sockfd = socket(AF_INET, SOCK_STREAM, 0);
     if (sockfd < 0) error("ERROR opening socket");
     struct sockaddr_in serv_addr, cli_addr;
     bzero((char *) &serv_addr, sizeof(serv_addr));
     int portno = 8888;
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     if (bind(sockfd, (struct sockaddr *) &serv_addr,
              sizeof(serv_addr)) < 0)
              error("ERROR on binding");
     listen(sockfd,5);
     int clilen = sizeof(cli_addr);


     int newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
     if (newsockfd < 0)
          error("ERROR on accept");
     //Remplacer à partir d'ici
```

```
    char buffer[256];
    bzero(buffer,256);
    int n = read(newsockfd,buffer,255);
    if (n < 0) error("ERROR reading from socket");
    printf("Here is the message: %s\n",buffer);
    n = write(newsockfd,"I got your message",18);
    if (n < 0) error("ERROR writing to socket");
    return 0;
}
```

Question 4. *La fonction* `accept` *bloque jusqu'à recevoir une nouvelle connection en retournant un file descriptor permettant d'envoyer et de recevoir des données sur cette connection.*

Votre but est de remplacer la fin du programme pour que celui-ci intéragisse avec une instance de `toupper` *à la place.*

Pour cela:

- *Utilisez la fonction* `dup2` *pour remplacer le file-descriptor de l'entrée standard (numéro 0) et de la sortie standard (numéro 1) par le file descriptor de la nouvelle connection* `newsockfd`*.*
- *Fermez le file-descriptor* `sockfd` *dont vous n'avez plus besoin dans la suite de l'exécution.*
- *Utilisez la fonction* `execlp` *pour exécuter toupper.*

Question 5. *On souhaiterait améliorer le service réseau mis en place jusqu'ici de deux manières. Premièrement, on voudrait qu'il ne quitte plus une fois qu'il a traité une connection. Deuxièmement, on voudrait qu'il puisse prendre plusieurs connections simultanément.*

Pour réaliser cela, nous allons procéder de la manière suivante:

- *Le code à partir de* `accept` *doit être mis dans une boucle*
- *Une fois une nouvelle connection reçue (appel à* `accept`*), on duplique le processus avec* `fork()`
- *Le processus fils exécute la requête comme précédemment;*
- *Le processus père ferme la connection ouverte et contine d'attendre une nouvelle connection.*

Essayez d'envoyer plusieurs connection au service réseau simultanément pour voir si votre serveur fonctionne.

Question 6. *On suppose maintenant que les capacités de votre serveur ne suffit pas à exécuter toutes les instances de toupper dont vous avez besoin. Pour pallier à cela, nous allons équilibrer la charge (c'est à dire implanter un load balancer) en utilisant l'ordinateur d'un de vos camarades.*

1. *Récupérez l'adresse IP d'un autre ordinateur de la salle de TP (tapez la commande* `/sbin/ifconfig` *pour la connaitre)*
2. *Une fois sur 2 (en tourniquet), remplacer la ligne où vous exécutez "toupper" par* `system("nc <addresse ip> 2222");`,