OSPS – TP/Projet Python – M1 FISA

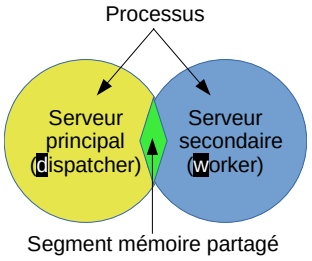
D. Duvivier © 2022-2023 - UPHF-INSA

I. ROADMAP → Début du projet OSPS…

Le but de ce « TP » est de construire les premières étapes du client(s)-serveur(s), à réaliser en Python 3 en guise de projet, en mettant en œuvre un certain nombre d'éléments étudiés dans le cours de système et programmation sécurisée…
Basez-vous sur les exemples placés sur Moodle (des compléments seront fournis en cours de projet).
N'oubliez pas de traiter les codes de retour et exceptions à tous les niveaux.

1 Mise en place d'un segment mémoire partagé entre un serveur principal et un serveur secondaire.

Nous supposons que le serveur secondaire (pour l'instant unique) est lancé via un mécanisme de type fork() depuis le serveurs principal. Un segment mémoire partagé assure la communication, dans un premier temps sans mécanisme de synchronisation. Les codes Python 3 des deux processus (serveur principal « **d** », serveur secondaire « **w** ») contiennent pour l'instant juste une temporisation pour simuler un temps de traitement.



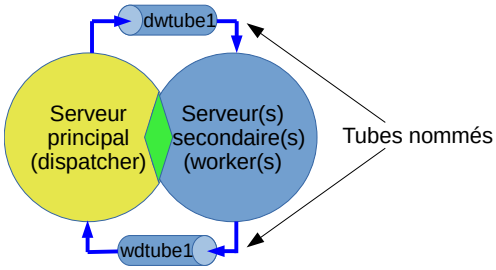
1.a) Quel module utilisez-vous pour manipuler un segment mémoire partagé en Python 3 ?

Veillez à utiliser le module le plus « simple » possible tout en choisissant un module « officiel/maintenu/à jour ».

2 Utilisation d'une paire de tubes nommés pour assurer la synchronisation

Pour assurer la synchronisation via un ensemble de commandes élémentaires, une paire de tubes nommés (pour l'instant unique) est utilisée entre le serveur principal et un serveur secondaire :

- Un tube nommé du serveur principal (d) vers le serveur secondaire (w) identifié par « dwtube1 » ;
- Un tube nommé du serveur secondaire (w) vers le serveur principal (d) identifié par « wdtube1 ».



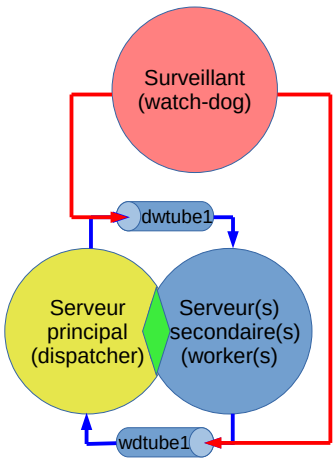
Les codes Python 3 des deux processus (serveur principal « **d** », serveur secondaire « **w** ») contiennent pour l'instant juste un mécanisme de type « ping-pong » :

- Le serveur principal envoie le texte « ping » au serveur secondaire et attend une réponse ;
- Le serveur secondaire, initialement en attente du message du serveur principal (requête à traiter), répond en envoyant le message « pong » dès qu'il reçoit le message « ping » du serveur principal.

2.a) Prévoyez le code nécessaire pour arrêter *proprement* les serveurs après un certain nombre d'échanges (*ping-pong*).
2.b) Quel serveur doit s'arrêter en premier pour éviter les *zombies* ? Qu'est ce qu'un *zombie* au sens informatique / système d'opération ?

3 Mise en place d'un processus (indépendant) de surveillance

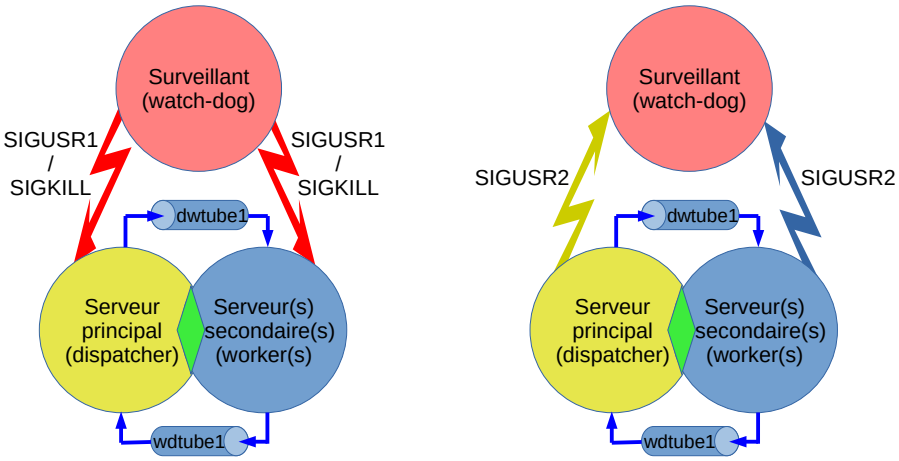
Bien que le(s) serveur(s) secondaire(s) soit sous la responsabilité du serveur principal, ce dernier peut être victime d'un bug ou d'une attaque (déni de service ([D]DoS), etc.) ou ne pas détecter qu'un serveur secondaire est *figé*. Un processus de surveillance (*watch-dog*) est donc ajouté. Il est lancé en premier lieu et lance le serveur principal via un mécanisme de type fork() qui à son tour lance le(s) serveur(s) secondaire(s). Il communique via un seul tube vers le serveur principal (wdtube1) et via d'autres tubes (dwtube1...dwtuben) vers les serveurs secondaires.



3.a) Est-ce une bonne idée de lancer le serveur principal depuis le watch-dog puis les serveurs secondaires depuis le serveur principal ? Sachant que le watch-dog peut être amené à stopper et redémarrer [tous] les autres processus ? Doit-il déléguer le redémarrage d'un serveur secondaire au serveur principal, au risque de lui faire perdre en réactivité ?

3.b) Est-ce une bonne idée d'utiliser les tubes de communication entre les serveurs pour établir une communication entre le watchdog et les serveurs ? Quelle est la conséquence si un serveur secondaire est figé et que le watch-dog lui donne l'ordre de redémarrer via « son » tube nommé (dwtube1..dwtuben) ?

3.c) Comme illustré ci-dessous – sans forcément tout implémenter – ne serait-il pas judicieux d'utiliser des « signaux » (au sens du système d'exploitation) ? Est-ce que le père d'un processus est informé de l'arrêt d'un processus fils par un signal ?



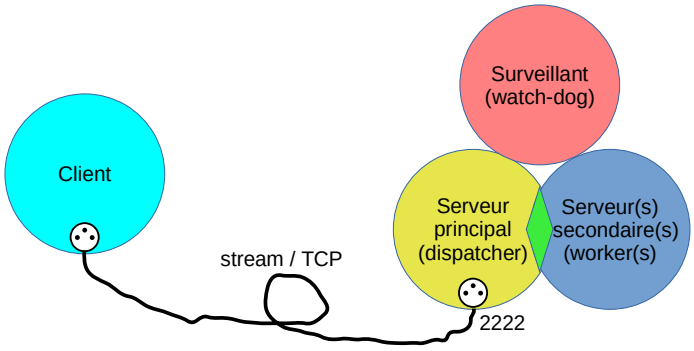
Le fonctionnement sur base de signaux est le suivant : périodiquement le watch-dog envoie un signal SIGUSR1 à chaque serveur (à tour de rôle). Il attend ensuite un certain temps (pendant lequel il peut envoyer un signal à d'autres serveurs) que chaque serveur lui réponde par un signal SIGUSR2. Si un serveur ne répond pas dans les temps par un signal SIGUSR2 (à un signal SIGUSR1), ce dernier est considéré comme figé, un SIGKILL lui est envoyé et, selon votre réponse à la question 3.a) :

- soit le watch-dog est en charge de redémarrer n'importe quel serveur (principal ou secondaire) ;
- soit le watch-dog ne peut redémarrer directement que le serveur principal et c'est le serveur principal qui est en charge de redémarrer un serveur secondaire figé.

Même si vous n'avez pas implémenté la solution par signaux, n'oubliez pas de redémarrer tous les serveurs secondaires si le serveur principal était figé (puis redémarré par le watch-dog).

4 Ajout d'un client

En supposant qu'il y ait pour l'instant un seul serveur principal, un seul serveur secondaire et un seul client, nous souhaitons établir une connexion entre un client et le serveur principal (pour l'instant) en utilisant des sockets en mode connecté (stream / TCP) sur le port 2222.



Supposons le fonctionnement suivant :

1) Le client se connecte via un socket en TCP au port 2222 du serveur principal ;
2) Le client envoie un type de requête au serveur (pour les tests, « requetetype1 » ou « requetetype2 ») ;
3) Le serveur principal retourne un numéro de port > 2222, géré (en TCP) par le serveur secondaire ;
4) Il y a établissement d'une connexion directe en TCP entre le client et le serveurs secondaire chargé de la requête du client ;
5) Le client et le serveur secondaire échangent quelques informations et coupent la connexion ;
6) Le client coupe la connexion avec le serveur principal, sauf s'il a une autre requête à soumettre, auquel cas retour à l'étape 2

Au niveau du serveur (watch-dog + serveur principal + serveur secondaire), la requête du client (« requetetype1 » ou « requetetype2 ») est transmise au serveur secondaire en plus d'informations éventuellement nécessaires pour accepter/établir la connexion directe entre le client et ledit serveur secondaire. De même, le serveur secondaire informe le serveur principal (par un message « debuttraitement » ou « fintraitement ») du début/fin de traitement de la requête. Un message « errtraitement » pourra être prévu en cas de problème lors du traitement de la requête. Dans un premier temps, le traitement de la requête consistera en une simple temporisation inférieure au délai de déclenchement du watch-dog.

Remarque : ceci revient à définir un « mini-protocole » entre le client et le serveur et entre le serveur principal et les serveurs secondaires.