Mohamed aziz TAIEB 11927449

Ryan YALA 12000356

Nous déclarons qu'il s'agit de notre propre travail.

Protocole WRDLP

Coté client :

-L’utilisateur entre une proposition quelque soit sa taille et son format (majuscule ou minuscule)

-Le client stocke cette proposition dans une chaîne de caractère de longueur 6 . Ensuite il l’écrit dans la socket .

-Le client lit un entier de la socket et le stocke dans n:

-si n=3 , alors la proposition est trop courte et donc il demande à l’utilisateur d’entrer une nouvelle proposition .

-si n=4 , alors soit la proposition n’est pas dans le dictionnaire soit elle est trop longue et donc il demande à l’utilisateur d’entrer une nouvelle proposition .

-si n=2 , alors la proposition est acceptée , elle fait partie du dictionnaire mais elle n’est pas la bonne . Le client alors lit à nouveau de la socket dans une chaîne de caractère de taille 6 . Il affiche ensuite cette chaîne à l’utilisateur avec les indices .

-si n=1 , alors la proposition est la bonne , la partie est donc finie , le client lit la solution de la socket . Puis il demande à l’utilisateur d’entrer 1 s’il veut rejouer , 0 sinon et le stocke dans la variable rej . Il écrit cet entier dans la socket et refait le même parcours si rej=1 , se déconnecte sinon .

Coté serveur :

-Le serveur stocke une chaîne aléatoire dans la variable a\_deviner .

-Après avoir reçu une connexion , il lit de la socket une chaîne de caractère et la stocke dans la variable prop .

-Il utilise ensuit la valeur de retour de saisir\_prop\_serveur(prop) est la stocke dans la variable ret\_val:

-si ret\_val=3 , alors la proposition est trop courte et donc écrit le contenu de ret\_val dans la socket.

-si ret\_val=4 , alors soit la proposition n’est pas dans le dictionnaire soit elle est trop longue et donc écrit le contenu de ret\_val dans la socket.

-si ret\_val !=3 et !=4 , alors le serveur utilise la valeur de retour de (raiter\_prop(prop, a\_deviner,indic) .

Si cette dernière est inférieur à 5 alors la proposition est acceptée , elle fait partie du dictionnaire mais elle n’est pas la bonne . Le serveur initialise n à 2 et l’écrit dans la socket . Ensuite il écrit la chaîné de caractère indice dans la socket également .

Sinon elle sera égale à 5 alors alors la proposition est la bonne , la partie est donc finie . Le serveur initialise n à 1 et l’écrit dans la socket puis écrit la solution dans la socket aussi . Finalement , il lit un entier de la socket et le stocke dans la variable rej , si rej=1 il refait une nouvelle partie sinon il attend un nouveau client .

AMÉLIORATIONS :

— client et serveur fonctionnent aussi bien en IPv4 qu’en IPv6 et le client peut se connecter à un serveur étant donné un nom de domaine ou ou adresse IP ; Pour ce faire, nous avons décidé que le serveur accepte uniquement les adresse ipv6 et le client s’occupe de mappées les adresses ipv4 grace à la fonction getaddrinfo .

— le serveur est multithreadé et peut servir plusieurs clients en même temps ;

— le client peut faire plusieurs parties à la suite, sans nouvelle demande de connection, le protocole WRDLP est modifié comme il se doit ; Pour ce faire lorsque le client gagne une partie il écrit sur la socket un booléen désignant si le client souhaite rejouer.

— le serveur garde un journal des connections avec les adresses IP des clients et éventuellement d’autres informations (date, heure, durée de la partie, ...).Pour ce faire le serveur ouvre un fichier pour les logs, chaque threads reçoit un mutex pour sécuriser l'écriture sur le fichier , le fichier de logs en question et la socket d'échange.