Faculté des Sciences
Département d'Informatique

INFO-F-201 – Systèmes d'exploitation

Projet 2 de programmation système

« Quel est ce Pokémon ? » – édition threads & réseau

Alors que votre application de comparaison d'images bat son plein, vous commencez à vouloir proposer celle-ci comme un service en ligne. Cependant, votre application est tellement populaire que vous commencez à avoir peur que votre implémentation actuelle ne tire pas suffisamment partie de ce que votre système d'exploitation a à vous offrir.

En effet, comme vous avez suivi quelques cours et séances d'exercices supplémentaires de votre cours de système d'exploitation, vous vous êtes rendus compte qu'utiliser des threads plutôt que des processus pourrait s'avérer intéressant. Par ailleurs, pour éviter de créer trop de processus, vous avez décidé de transformer *img-dist* en une bibliothèque plutôt que de la garder comme un programme indépendant.

Ni une, ni deux, vous voilà désormais partis avec les membres de votre groupe pour réécrire votre code en utilisant cette fois des threads et des sockets !

1 Détails du projet

Ce projet est à réaliser par groupe de **deux** ou **trois** étudiants (vous avez la possibilité de changer de groupe mais vous devrez gérer ces changements par vous-même) et comporte différentes parties :

— Côté serveur :

1. Une **bibliothèque** C (*libimg-dist.a*) et son en-tête *imgdist.h* qui génère le code de hachage perceptif (le code est fourni, vous ne devez rien faire pour celui-ci à part le compiler) ;
2. Un programme (serveur) C ou C++ (*img-search*) qui reçoit une **image** (pas son chemin) **via un socket** et indique l'image la plus similaire (au choix s'il y en a plusieurs) parmi celles de la banque d'images **situées dans le dossier *./img/***.

— Côté client :

1. Un programme (client) C ou C++ ***pokedex-client*** qui communique par des sockets TCP avec le serveur *img-search* en envoyant l'image (son contenu cette fois, pas le chemin vers celle-ci).

Vous serez pénalisés si vous réalisez le projet seul sans en avoir reçu l'autorisation explicite.

Vous êtes autorisés à utiliser des fonctionnalités du C++ telles que `vector`, `string`, `hashmap`, `new` et `delete`, les classes, etc[1].

Notez que la solution du premier projet est disponible sur la page UV du cours et vous pouvez vous en servir pour ce projet-ci. Vous pouvez également repartir du premier projet que vous avez rendu.

2 Bibliothèque fournie « libimg-dist.a » : comparaison d'images

Pour réaliser ce projet, vous recevez le code C et le Makefile pour créer la bibliothèque appelée **libimg-dist.a**. Vous n'avez rien à coder pour cette bibliothèque mais vous devrez la compiler afin de pouvoir l'utiliser avec le reste du projet. Cette bibliothèque doit être compilée avant de pouvoir

1. Ce projet évalue les compétences en programmation système, vous ne serez donc pas récompensés pour avoir implémenté une magnifique chaîne d'héritage car ce n'est pas le sujet de ce cours.

compiler votre projet (le Makefile fourni s'en assure déjà pour vous). Les options de compilation nécessaires pour lier cette bibliothèque à votre code d'*img-search* sont déjà ajoutées pour vous dans son Makefile.

Le Makefile général qui vous est fourni permet de compiler cette bibliothèque directement.

Le principe pour utiliser la bibliothèque *libimg-dist.a* est simple. Vous devez utiliser une des deux fonctions (`PHash()` ou `PHashRaw()`) afin de générer le code de hachage perceptif correspondant à l'image qui vous intéresse (ce code est conservé dans en entier 64-bit non-signé et est toujours le même pour une même image) puis, pour connaître la distance entre deux images, il suffit de comparer leurs codes de hachage perceptif respectifs avec `DistancePHash()`.

Plus précisément, les 3 fonctions qui vous seront utiles sont :

```
bool PHash(const char imagePath[], uint64_t* hash);
```

Calcule le code de hachage perceptif de l'image stockée dans le fichier indiqué par le paramètre `imagePath`. Le code de hachage perceptif résultant est conservé dans `*hash`. En cas d'erreur, la fonction retourne `false` et, en cas de succès, elle retourne `true`. Un message d'erreur est écrit pour vous sur `stderr` en cas d'erreur.

```
bool PHashRaw(char rawImage[], unsigned int size, uint64_t* hash);
```

Calcule le code de hachage perceptif de l'image dont le contenu a été entièrement stocké dans un tableau `rawImage` fourni en paramètre. Le code de hachage perceptif résultant est conservé dans `*hash`. En cas d'erreur, la fonction retourne `false` et, en cas de succès, elle retourne `true`. Un message d'erreur est écrit pour vous sur `stderr` en cas d'erreur.

```
unsigned int DistancePHash(uint64_t pHash1, uint64_t pHash2);
```

Calcule puis retourne la distance entre deux codes de hachage perceptif *pHash1* et *pHash2*. C'est la même valeur que la valeur de retour d'*img-dist* dans le projet 1 (quand il n'y avait pas d'erreur).

Un code exemple vous est fourni dans le fichier *serveur/main.c*.

3 Programme serveur « img-search » : recherche d'images

3.1 Objectif

Le but de ce programme est de créer un serveur comparant des images au format BMP[2] (.bmp) avec une série d'autres images du même format. Il devra respecter les contraintes suivantes :

- Être écrit en C ou en C++ ;
- Avoir son code dans le dossier *serveur/* (ainsi que *commun/* si jamais vous souhaitez partager du code avec celui du client) ;
- Être compilé à l'aide d'un Makefile (que vous devez donc écrire et qui sera le même que celui de *pokedex-client*) ;
- Utiliser comme banque d'images les images contenues directement dans le dossier *img/* (fourni) – voir Section 3.2 ;
- Comparer les images à l'aide de la bibliothèque *libimg-dist.a* – voir Section 3.3 ;
- Écouter les connexions entrantes sur le port 5555 de toutes les adresses de la machine qui l'exécute (`INADDR_ANY`) – voir Section 3.4 ;
- Utiliser 3 threads par client pour comparer les images reçues et s'assurer d'un travail rapide (en évitant les accès concurrents à l'aide des mécanismes de synchronisation) – voir Section 3.5 ;
- Gérer les signaux SIGPIPE et SIGINT – voir Section 3.6.

2. Pour des raisons de simplicité, toutes les versions d'images BMP ne sont pas supportées et seule la plus rudimentaire l'est. Si vous souhaitez ajouter une image BMP à la banque d'images, assurez-vous qu'elle soit encodée dans une version simple.

3.2 Récupération des images

Il y a deux types d'images : celles de la banque d'images et celles dont on cherche les images similaires dans la banque d'images.

Les images dont on cherche à trouver une image similaire doivent être transmises via un socket. Vous pouvez refuser les images ayant une taille de plus de 20 ko. Un client peut envoyer plusieurs images les unes après les autres et leur transfert doit être fait avec le même socket. Vous devrez justifier votre protocole d'échange de données dans le rapport.

Les images de la **banque d'images** sont contenues **dans un seul dossier fixe**, *img/*, qui vous est fourni. Ce dossier est implicitement côté serveur et donc supposé inaccessible pour le client.

Vous pouvez considérer que le chemin individuel de chaque image ne pourra pas excéder 999 `char` (sans compter le `'\0'` final en C). De plus, vous pouvez considérer dans votre implémentation que le nombre d'images contenues dans la banque d'images dans *img/* sera toujours suffisamment petit pour pouvoir conserver tous leurs chemins en mémoire.

Dans le serveur, vous pouvez réutiliser *list-file* ainsi que les fonctions `popen()` et `pclose()` si cela vous aide. Vous avez aussi la possibilité d'utiliser la bibliothèque *dirent.h*.

3.3 Comparaison de paires d'images

La comparaison de deux images doit être réalisée en utilisant les fonctions fournies par la bibliothèque *libimg-dist.a* décrite Section 2.

Maintenant que vous avez un accès plus complet sur la manière dont les comparaisons sont faites, il vous est demandé de repenser la comparaison d'images pour optimiser un maximum la vitesse de comparaison et donc réduire l'utilisation d'appels système lents.

Expliquez comment vous vous y êtes pris dans le rapport.

3.4 Gestion des sockets

Le programme *img-search* agira comme le serveur écoutant les connexions sur un socket avec la configuration suivante :

- Adresse IP : toutes les adresses de la machine (`INADDR_ANY`) ;
- Port : 5555 ;
- Protocole : TCP ;
- Nombre maximal de connexions en attente : 10 ;
- Nombre maximal de clients connectés simultanément [3] ne sera jamais plus de 1000.

Chaque client (*pokedex-client*) communiquera par l'intermédiaire d'un socket avec le serveur (*img-search*). Cette connexion permettra au client d'envoyer les images à comparer et au serveur de transmettre le résultat (sous forme de texte) des comparaisons.

Vous devez pouvoir traiter plusieurs clients en même temps. Vous ne pouvez donc pas attendre qu'un client se déconnecte pour commencer à prendre en compte les requêtes des autres clients. Il n'y a pas de restriction sur la manière de pouvoir gérer cela (processus, threads, fonctions non-bloquantes, *polling*, ...) mais **expliquez dans le rapport** quel choix vous avez fait et comparez ce choix (efficacité, aisance d'utilisation, impact sur votre implémentation, ...) avec une alternative qui aurait été possible.

Vous devez supposer que le serveur et le client peuvent être sur des machines différentes.

3. Cette hypothèse est là pour vous faciliter la vie mais vous pouvez augmenter cette limite si vous voulez.

3.5 Répartition en threads pour la recherche concurrente

La recherche de la ou des images les plus similaires dans la banque d'images doit se dérouler de manière concurrente dans 3 threads d'*img-search* par client.

La répartition du travail doit être organisée de sorte que le client doive attendre le moins longtemps possible pour avoir une réponse. **Expliquez comment vous vous y êtes pris dans le rapport.**

Lorsqu'une image à comparer avec la banque d'images est transmise par le client sur le socket et que toutes les comparaisons avec la banque d'images ont été effectuées, *img-search* devra écrire le résultat sur le socket associé au client à l'origine de la demande. Le résultat consiste en les mêmes messages (repris plus bas) que pour le projet 1. La connexion avec ce client ne sera pas fermée tant que le client aura encore des images à transmettre.

Le texte de réponse contiendra donc le nom d'une des images de la banque d'images parmi les plus similaires à celle passée en paramètre de *img-search*. Si plusieurs images ont la même similarité minimale, vous pouvez choisir celle à écrire arbitrairement.

Si l'image la plus similaire est, par exemple, l'image « *img/22.bmp* » et avait, par exemple, une distance de 12, votre programme devra écrire sur le socket ceci :

```
Most similar image found: 'img/22.bmp' with a distance of 12.
```

Si aucune image n'a pu être comparée correctement, votre programme écrira sur le socket :

```
No similar image found (no comparison could be performed successfully).
```

Veillez à bien conserver ce format sans quoi les tests automatiques ne fonctionneront pas.

Si un client envoie simultanément plusieurs images à comparer, le serveur devra répondre dans le même ordre.

3.6 Gestion des signaux

Le signal SIGINT devra être géré par *img-search*. Une fois reçu, tous ses threads et lui devront se terminer de manière propre (pas de crash et fermeture la plus rapide des connexions mais sans causer d'accès concurrents).

Le signal SIGPIPE ne doit pas générer le crash de *img-search* mais devra entraîner l'arrêt de manière propre des tâches en cours pour le client concerné par ce signal sans causer d'accès concurrents.

N'oubliez pas que le gestionnaire de signaux doit être le plus court possible et ne peut appeler que les fonctions *async-safe*. N'hésitez pas à vous inspirer de la gestion des signaux dans la solution du projet 1 (un petit exemple est également donné dans le fichier *main.c* du serveur).

La gestion des signaux dans une application multithread est **non triviale**. L'Annexe A est là pour vous aider dans la réalisation du projet.

4 Programme client « pokedex-client »

4.1 Objectif

Le but de ce programme est de transmettre des images au serveur *img-search* pour qu'elles soient comparées avec la banque d'images. Il devra respecter les contraintes suivantes :

— Être écrit en C ou en C++ ;

— Avoir son code dans le dossier *client/* (ainsi que *commun/* si jamais vous souhaitez partager du code avec celui du serveur) ;
— Être compilé à l'aide d'un Makefile (que vous devez donc écrire et qui sera le même que celui d'*img-search*) ;
— Lire sur `stdin` les chemins vers les images dont le **contenu** sera envoyé à *img-search* – voir Section 4.2 ;
— Prendre un seul paramètre optionnel qui sera l'adresse IP du serveur exécutant *img-search* (s'il est omis, ce paramètre vaut le texte "127.0.0.1" par défaut) – voir Section 4.3 ;
— Se connecter au serveur *img-search* sur le port 5555 et à l'adresse IP spécifiée 4.4 ;
— Gérer les signaux SIGPIPE et SIGINT – voir Section 4.5.

4.2 Images à comparer

Les images à comparer avec celles de la banque d'images sont transmises sur l'entrée standard (`stdin`) en indiquant leur chemin.

Le client doit envoyer le plus vite possible chaque nouvelle image qui lui est transmise sur `stdin`. Si des chemins d'images sont transmis sur `stdin` avant d'avoir reçu une réponse pour les images précédentes, vous devez tout de même envoyer les nouvelles images sans attendre la réponse du serveur au préalable. Lorsque `stdin` et fermé (ex. : Ctrl + D), le client doit attendre de recevoir et afficher toutes les réponses du serveur avant de terminer.

Le client doit afficher sur `stdout` les messages envoyés par le serveur décrits à la Section 3.5 (chacun sur une ligne séparée). N'affichez rien d'autre sans quoi les tests automatiques pourraient échouer.

Si *pokedex-client* est fermé avant qu'aucune entrée ne soit donnée, n'affichez rien. Si l'image donnée en entrée sur `stdin` n'existe pas ou qu'il y a eu une erreur quelconque pour le lire, affichez le message suivant sur `stdout` :

```
No similar image found (no comparison could be performed successfully).
```

Vu que le serveur est potentiellement sur une machine distante, vous devez nécessairement transmettre le contenu des images et pas les chemins vers celles-ci. **Expliquez dans le rapport comment vous transférez les images.**

4.3 Paramètres

Le programme prendra un seul paramètre optionnel (aucun autre ne sera autorisé) qui sera l'adresse IPv4 (format "w.x.y.z" avec $w, x, y, z \in \{0, ..., 255\}$) du serveur contenant *img-search*. Si ce paramètre n'est pas spécifié, l'adresse IP sera "127.0.0.1" par défaut.

4.4 Gestion des sockets

Le programme *pokedex-client* agira comme le client se connectant, avec un socket utilisant le protocole TCP, à un serveur distant *img-search*. Ce serveur sera à l'adresse IP spécifiée par l'utilisateur (voir Section 4.3) et écoutera le port 5555.

La connexion servira à transmettre au serveur le contenu des images à comparer puis à recevoir le résulstat pour chacune d'elles.

4.5 Gestion des signaux

Le signal SIGINT devra être géré par *pokedex-client*. Une fois reçu, tous ses threads (s'il y en a plusieurs) devront se terminer de manière propre (pas de crash et fermeture la plus rapide des connexions mais sans causer d'accès concurrents).

Le signal SIGPIPE ne doit pas générer le crash de *pokedex-client* mais devra entraîner l'arrêt de manière propre du programme, comme pour SIGINT.

5 Ce qui est mis à votre disposition

Vous pouvez télécharger la base du projet sur l'Université Virtuelle. Ce répertoire contient :

- le code C de la bibliothèque libimg-dist.a (située dans le dossier img-dist/) ;
- une banque d'images de Pokémon dans le dossier img/ ;
- des tests automatiques utilisables en vous rendant dans le dossier test/ et en exécutant la commande `./tests` ;
- un Makefile à compléter ;
- 3 dossiers (*serveur/*, *client/* et *commun/*) dans lesquels vos codes C devront se trouver ;
- l'en-tête de la bibliothèque (*serveur/imgdist.h*) et un exemple de son utilisation (*serveur/-main.c*).

6 Critères d'évaluation

Si vous écrivez votre code en C, votre projet doit compiler avec gcc version 9.4 (ou ultérieure) et les options ci-dessous (présentes dans le Makefile fourni) :

```
FLAGS=-std=gnu11 -Wall -Wextra -O2 -Wpedantic
```

Si vous écrivez votre code en C++, votre projet doit compiler avec g++ version 9.4 (ou ultérieure) et les options ci-dessous :

```
FLAGS=-std=gnu++17 -Wall -Wextra -O2 -Wpedantic
```

Si votre programme ne compile pas, vous recevrez une note de 0/20.

N'hésitez pas à utiliser les flags des assainisseurs lors du développement de votre projet, ceci vous aidera à détecter vos erreurs plus facilement (ceci n'est pas obligatoire et pensez à bien recompiler tous vos fichiers lorsque vous ajoutez ou retirez ces options) :

```
-g -fsanitize=address,undefined
```

ou (ces options étant malheureusement incompatibles, vous empêchant d'analyser à la fois la mémoire et les accès concurrents) :

```
-g -fsanitize=thread,undefined
```

Assurez-vous également que **tous** vos appels systèmes ont leur valeur de retour correctement traitée (à vous de **gérer correctement les cas où une erreur est survenue**). Cela fait partie de l'évaluation. Un projet dont le code fonctionne mais ne prend pas compte des erreurs pouvant survenir perdra des points.

6.1 Pondération

- Tests automatiques **/3**
- Ce projet de programmation système va principalement évaluer votre compétence à manier correctement les outils liés aux systèmes d'exploitation (processus, *threads*, *sockets*, signaux, ...) dans le langage C ou C++. La pertinence des outils utilisés ainsi que la manière dont ils sont utilisés (trop, pas assez, au mauvais endroit, trop longtemps, ...) sont évalués. Assurez-vous également que les codes d'erreurs sont bien traités. **/9**
- Ce projet doit contenir un rapport dont la longueur attendue est de deux à trois pages (max 5). Ce rapport décrira succinctement le projet, les choix d'implémentation si nécessaire, les difficultés rencontrées et les solutions originales que vous avez fournies et vos choix d'implémentation. **/6**
  - Orthographe
  - Structure
  - Légende des figures
- Votre code sera aussi évidemment examiné en termes de clarté, de documentation, de commentaires et de structure. **/2**

7 Remise du projet

Vous devez remettre un projet par groupe contenant un fichier zip contenant

- les codes source C/C++ :
  - de *img-search* dans le dossier *serveur/* ;
  - de *pokedex-client* dans le dossier *client/* ;
  - des codes partagés par les deux projets (s'ils existent, ce n'est pas une obligation) dans le dossier *commun/*.
- les éventuels scripts Bash que vous utilisez (s'il y en a) ;
- un seul Makefile pour compiler le serveur et le client ;
- votre rapport au format PDF avec le nom des membres du groupe et leur ULBID ;
- les éventuels tests que vous auriez écrits.

N'incluez ni la banque d'images qui vous a été fournie ni les dossiers img-dist/ & test/ et leur contenu !

Vous devez soumettre votre projet sur l'**Université Virtuelle** pour le **17 décembre 2023 23h59** au plus tard.

Retards

Tout retard sera sanctionné d'un point par tranche de 4h de retard, avec un maximum de 24h de retard, et le projet devra être soumis sur l'université virtuelle.

Questions

Vous pouvez poser vos questions par courriel à arnaud.leponce@ulb.be.

A Gestion des signaux dans un contexte multithread

A.1 Assurer l'interruption des appels systèmes

Les signaux n'interrompent pas toujours les appels systèmes bloquants et ce comportement doit donc parfois être exprimé explicitement. Pour s'assurer qu'un signal interrompra bien une fonction bloquante (typiquement avec un code de retour négatif et `EINTR` comme valeur d'*errno*), il est possible d'utiliser la fonction `sigaction()` qui n'a pas été vue en TP. Pour vous faciliter la vie, nous vous donnons directement un morceau de code qui peut remplacer un appel comme `signal(SIGINT, SignalHandler)` mais en s'assurant de la bonne interruption des appels système cette fois :

```
#include <signal.h>

struct sigaction action;

action.sa_handler = SignalHandler; // Précise le gestionnaire de signaux
sigemptyset(&action.sa_mask); // Ne bloque l'arrivée d'aucun signaux
                              // durant l'exécution de SignalHandler.

if (sigaction(SIGINT, &action, NULL) < 0) {
   /* Erreur (errno mis à jour) */
}
```

Attention également, dans un contexte multithreading, si vous aviez plusieurs fonctions bloquantes en cours au moment de l'arrivée d'un signal, seule la fonction bloquante du thread réceptionnant le signal sera interrompue. Si vous souhaitez retransmettre le signal aux autres threads, la Section A.3 pourrait vous être utile...

Vous êtes libres de l'utiliser ou d'utiliser une autre alternative équivalente.

A.2 Réception du signal dans les bons threads

Pour rappel, les signaux sont délivrés aléatoirement à un thread qui ne bloque pas ni n'ignore ce signal. Si vous souhaitez que seuls certains threads aient la possibilité de recevoir un signal, vous pouvez bloquer (`SIG_BLOCK`) ces signaux dans les autres threads. Par exemple, pour bloquer les signaux SIGINT et SIGUSR1 dans un thread, vous pouvez utiliser :

```
sigset_t set;

sigemptyset(&set);          // Ensemble vide de signaux
sigaddset(&set, SIGINT);    // Ajouter le signal SIGINT
sigaddset(&set, SIGUSR1); // Ajouter le signal SIGUSR1

if (pthread_sigmask(SIG_BLOCK, &set, NULL) != 0) {
   /* Erreur (errno mis à jour) */
}
```

Vous pouvez les débloquer après en utilisant `SIG_UNBLOCK` à la place de `SIG_BLOCK` dans le code ci-dessus. Les threads créés héritent du blocage des signaux de leur thread "parent" (mais vous pouvez changer ensuite ceux qui sont bloqués de manière indépendante).

A.3 Envoyer un signal à un thread spécifique

En fonction de vos choix d'implémentation, il pourrait s'avérer judicieux de pouvoir envoyer un signal à des threads spécifiques. Il existe justement une fonction pour cela : `pthread_kill()`. Elle fonctionne de manière similaire à `kill()` sauf que vous devez préciser l'identifiant (un `pthread_t`) du thread qui recevra le signal au lieu d'un PID. Sa signature est la suivante :

```
#include <signal.h>
int pthread_kill(pthread_t thread, int sig);
```

Supposons que vous ayez un thread référencé par `th` et que vous souhaitiez lui envoyer le signal `SIGUSR1`, vous feriez :

```
pthread_kill(th, SIGUSR1);
```