Programmation Concurrente

Steganography

Travail Pratique

But du projet

Le but de ce travail pratique est de cacher un message textuel au sein d'une image sans que celle-ci n'en soit visiblement affectée.

Steganography : *the art or practice of concealing a message, image, or file within another message, image, or file.*

Plus précisément, il s'agit de développer deux programmes :

- Un « encodeur » permettant de cacher un message au sein d'une image.
- Un « décodeur » permettant d'afficher le message caché dans une image.

Évidemment, comme nous avons un faible pour la programmation concurrente, les deux programmes ci-dessus doivent être multi-threadés.

Encodeur et décodeur

Le premier programme à développer doit être nommé `encode` et sa syntaxe d'utilisation (ligne de commande) est donnée ci-dessous :

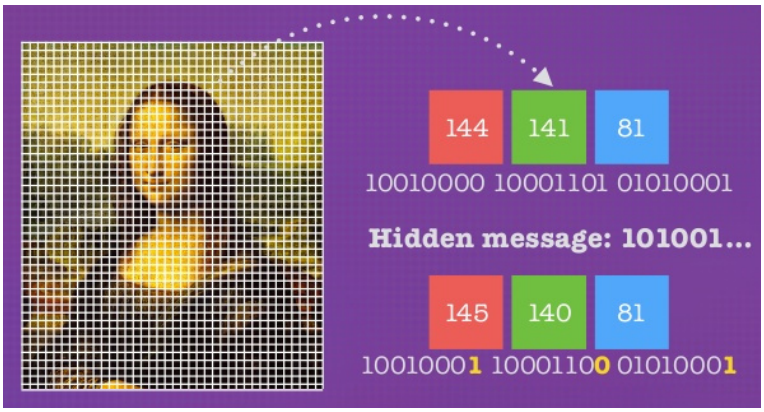
```
$ ./encode
usage: encode text_file input_image output_image thread_count
where input_image and output_image are PPM files
and thread_count the number of threads to use.
```

Le deuxième programme à développer doit être nommé `decode` et sa syntaxe d'utilisation (ligne de commande) est donnée ci-dessous :

```
$ ./decode
usage: decode image thread_count
where image is a PPM file containing an encoded secret message
and thread_count the number of threads to use.
```

Principe

Le principe est de dissimuler le contenu du message dans les pixels de l'image. L'image est donc visuellement altérée par la présence du message caché. Un message texte est composé d'une suite de caractères. Chaque caractère est représenté par un certain nombre de bits (par exemple 7 ou 8). Un message est donc représenté par une suite de bits. Dans le but de minimiser la dégradation visuelle de l'image par le message, les bits constituant le message sont stockés dans les bits de poids faible des composantes RGB (rouge – vert – bleu) des pixels de l'image, comme illustré sur la figure ci-dessous.



Pour une image où chaque pixel est encodé sur 24 bits (8 bits par composante primaire : rouge, vert, bleu), 21 bits sont utilisés pour encoder le contenu de l'image et 3 bits sont utilisés pour encoder une partie du message. L'utilisation de 7 bits par composante primaire au lieu de 8 est imperceptible à l'œil.

Cahier des charges

Encodeur

- L'encodeur, nommé `encode`, doit respecter la syntaxe présentée précédemment.
- Seuls les bits de l'image nécessaires au stockage du message doivent être modifiés. Par exemple, si le message nécessite seulement 700 bits et que l'image peut potentiellement en contenir 1 million, il n'est pas souhaitable de modifier le million de bits de l'image.
- L'encodage doit être multi-threadé.
- Chaque caractère doit être encodé sur 7 bit (les 7 bits de poids faible du code ASCII du caractère en C).
- Chaque bit du code ASCII du message est à stocker dans le bit de poids faible de chaque composante de chaque pixel. Pour rappel, un pixel de l'image est représenté par trois composantes (valeurs) de 8 bits chacune : rouge, verte, et bleue, pour un total de 24 bits par pixel.
- Si l'image ne contient pas suffisamment de bits pour encoder le message complet, le programme doit alors afficher un message d'erreur et se terminer.

Décodeur

- Le décodeur, nommé `decode`, doit respecter la syntaxe présentée précédemment.
- Le décodage doit être multi-threadé.

Encodeur + décodeur

- La division du travail entre les threads doit être aussi équitable que possible.
- La division du travail entre les threads ne doit pas être limitée par la largeur ou la hauteur de l'image.
- La lecture et l'écriture de l'image peuvent être séquentielles.

Généralités

- Un thread ne doit pas réaliser de travail inutile.
- Aucune primitive de synchronisation n'est autorisée.
- Évitez l'utilisation de variables globales.
- Tout programme exécuté sans argument doit afficher un message d'aide affichant la syntaxe de celui-ci et les arguments passés en ligne de commande doivent être validés.
- Le programme ne doit pas crasher si un fichier ne peut être lu ou écrit.
- Gérez le retour de toute fonction pouvant potentiellement échouer (`malloc`, fonctions de pthread, etc.).
- Libérez toute zone mémoire allouée dynamiquement avant de terminer le programme.

Informations utiles

Lecture/écriture d'images

Un exemple de code fonctionnel illustrant la lecture et l'écriture de fichiers images au format PPM couleur 24-bits vous est fourni avec cet énoncé. Celui-ci se trouve dans l'archive `ppm.tar.gz`.

L'outil `convert` de la suite ImageMagick permet de convertir une image d'un format à un autre. Par exemple, pour convertir l'image `azores.jpg` fournie avec le projet au format PPM binaire :

```
convert azores.jpg azores.ppm
```

pour convertir la même image au format PPM ASCII :

```
convert -compress none azores.jpg azores.ppm
```

Travail à rendre

Pour ce travail pratique, chaque groupe me rendra une archive contenant les fichiers suivants :

- Les fichiers sources, incluant un fichier `makefile` pour compiler le projet (et faire le ménage).
- Le rapport au format PDF.

Assurez-vous de respecter les consignes pour l'écriture du code et la rédaction du rapport. Les documents décrivant les consignes pour les travaux pratiques se trouvent sur la page CyberLearn du cours :

- « Consignes pour les travaux pratiques »
- « Consignes générales pour l'écriture du code »
- « Consignes pour l'écriture du code C »

Afin de vous organiser correctement, il est fortement conseillé de lire le document « Gestion de projet en groupe ».