Masset Eliot

Munoz Matteo

Cassage d'empreintes MD5 par force brute

Documentation

Fin 2020

Sommaire

Table des matières

Tutoriels :

Pré-requis :

1. **Ouvrir l'invite de commande de votre système**
2. **Si cela n'est pas fait, installer la commande g++ :**
    a. **Pour Windows :**
        i. **Suivre le tuto sur la page : http://www.codebind.com/cprogramming/install-mingw-windows-10-gcc/**
    b. **Pour linux :**
        i. **Exécuter la commande : (sudo) apt install build-essential**
3. **Disposer aussi du package CMake**
    a. **Pour l'installer, utiliser : (sudo) apt install cmake**
4. **Si vous êtes sur Windows et que vous ne l'avez pas encore fait, installer make :**
    a. **Installer make : http://gnuwin32.sourceforge.net/packages/make.htm**
    b. **Exécuter la commande : copy c:\MinGW\bin\mingw32-make.exe c:\MinGW\bin\make.exe**

Compilation :

5. **Exécuter simplement la commande : cmake .**
6. **Puis exécuter la commande : make**
7. **PS : vous pouvez télécharger le programme avec la commande : git clone https://github.com/LeProdrome/Cassage-d-empreinte-MD5-par-force-brute.git**

Exécution :

8. **Exécuter simplement la commande : ./main**
9. **Ou pour faire plusieurs test d'un seul coup : ./bench <nombre_de_tests> <"chaine_à_tester"> <nombre_de_threads>**

Matériel utilisé pour les tests :

Cache : L1 = 256Ko, L2 = 1000Ko, L3 = 6144Ko

RAM: 16 Go

Nombre de cœurs : 8

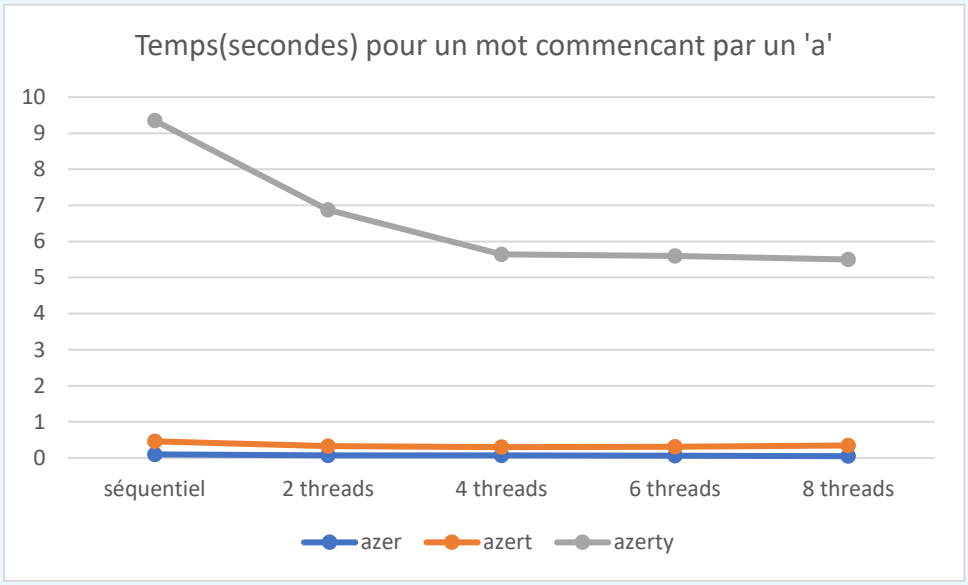
Fréquence : 4 GHz

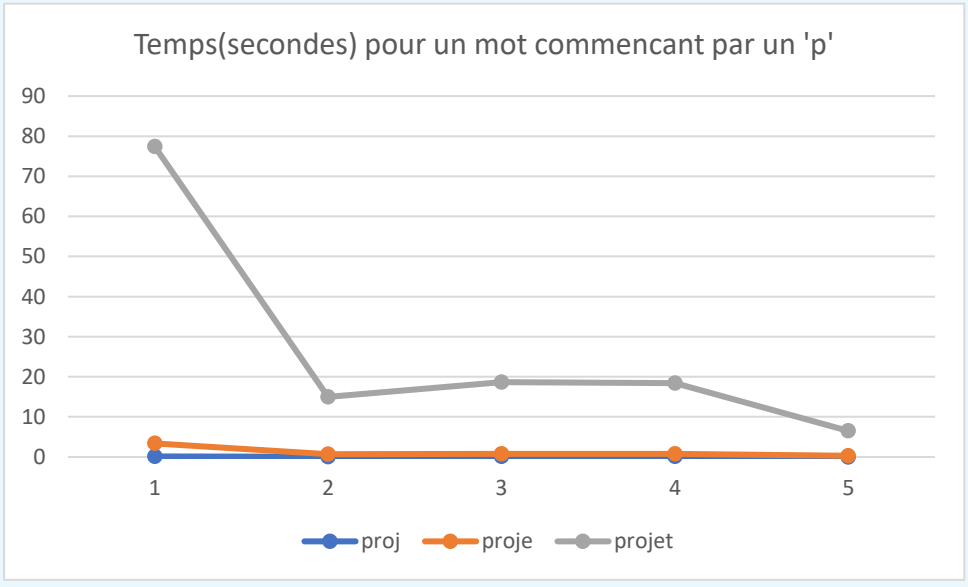
RAM avant le lancement : 1,43 G

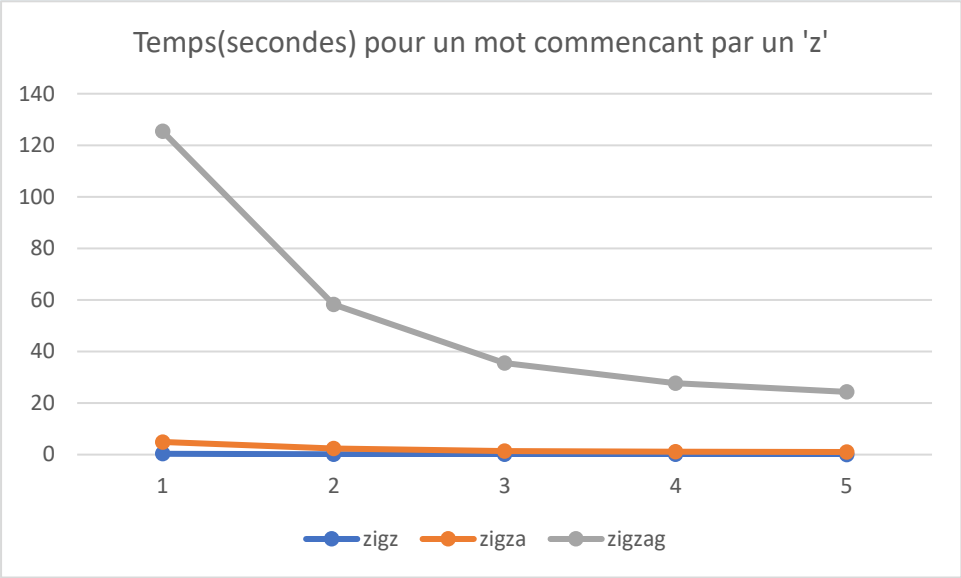
RAM après le lancement : 1,44 G

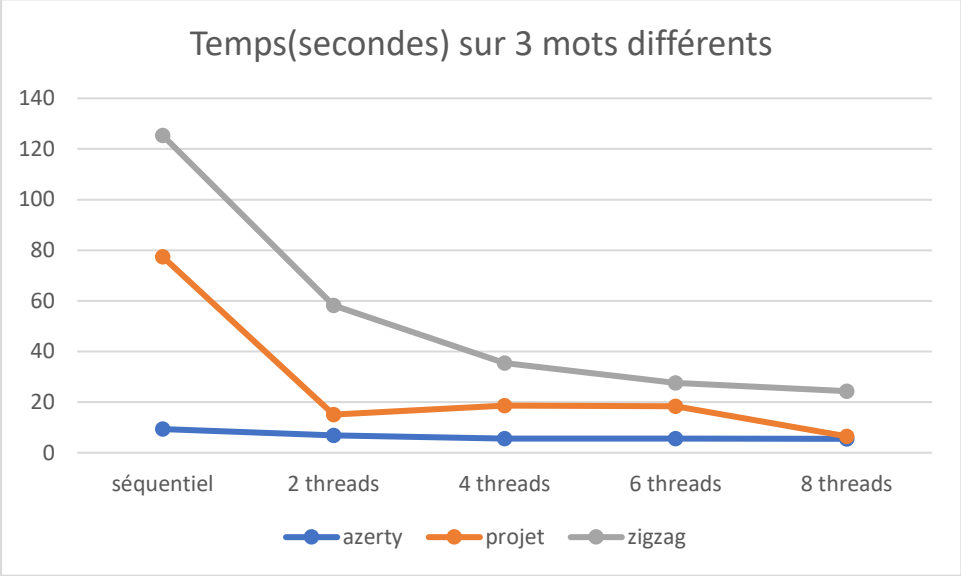
Tableau des valeurs et courbes associées :

	séquentiel	2 threads	4 threads	6 threads	8 threads
azer	0,1	0,071	0,076	0,06	0,05
azert	0,46	0,33	0,3	0,31	0,35
azerty	9,35	6,88	5,64	5,6	5,5
proj	0,13	0,04	0,11	0,101	0,01
proje	3,38	0,68	0,74	0,74	0,3
projet	77,43	15,02	18,66	18,38	6,5
zigz	0,32	0,17	0,12	0,07	0,04
zigza	4,9	2,33	1,43	1,11	1,01
zigzag	125,377	58,215	35,468	27,652	24,342









On voit bien qu'en général, plus il y a de threads, plus le programme est rapide. On peut aussi voir que les mots commençants par un 'a' sont déjà des mots trouvés très rapidement, tout simplement parce que l'alphabet commence par 'a'. Par ailleurs, les mots qui commencent par un 'z' sont eux bien plus longs à trouver, c'est logique quand on sait que 'z' est la dernière lettre de l'alphabet. Il y a bien sûr une exception qui s'explique facilement, au niveau de la lettre 'p'. On voit que pour le mot 'projet', la courbe descend brusquement sur 2 threads, alors qu'elle remonte la pente sur les 4 threads. Même si de première vue, il est difficile de comprendre le comportement, cela s'explique par le fait que lors du découpage de l'alphabet, la lettre 'p' est plus vite atteinte sur 2 threads que sur 4 threads (cf. voir page '6').

Parcours de tous les mots :

a
b
…
y
z

1 caractère

aa
ab
…
ax
az
ba
…
bz
……
za
…
zz

2 caractères

aaa
aab
…
aay
aaz
aba
abb
…
aby
abz
……
Aza

…

Azz

Baa

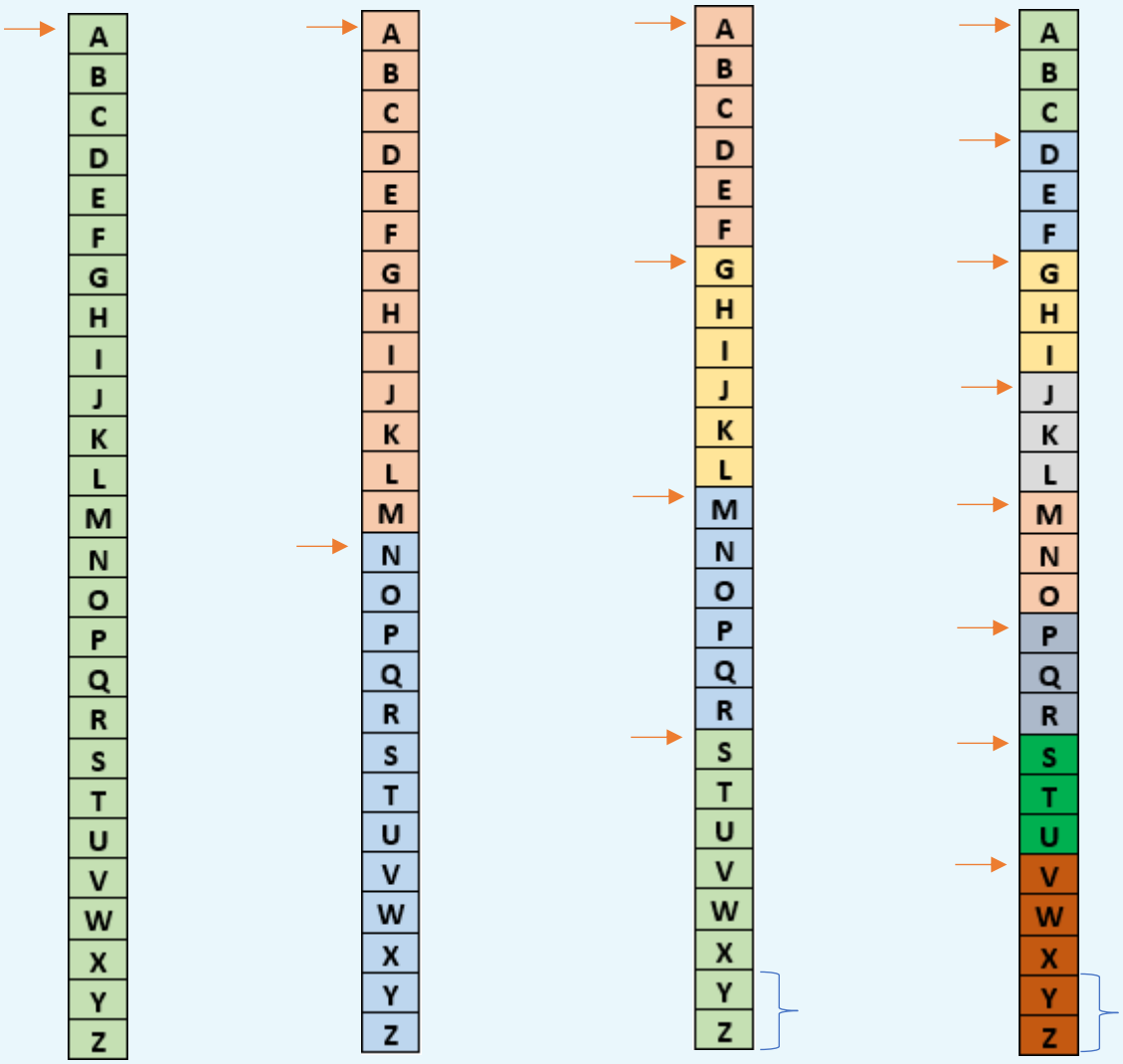
bab

……..

3 caractères

On voit ainsi très bien que le parcours sur des mots allongés peut être exponentiellement long(nombre_itération_max =26^(nombre_de_caractère_du_mot)).

Séquentiel : **2 threads:** **4 threads:** **8 threads:**



→ : **début de chaque processus** } : **reste de la division**

Nous avons décidé de diviser le programme multithread en découpant l'alphabet pour la première lettre du mot. Ainsi, dans l'utilisation du programme avec 4 threads, par exemple, un mot qui commencerait par 'g' sera bien plus rapide qu'avec 2 threads. En effet, avec 2 threads, le programme parcourra les lettres 'ABCDEF' avant d'attendre le 'G'.

<u>L'algorithme de découpe est simple :</u>

Lettre minimale = (26/nombre_de_thread)*(numero_de_thread)

Lettre maximale = (26/nombre_de_thread)*(numero_de_thread+1)

Pour 4 threads numero_de_thread va de 0 à 3.

Mais bien sûr, 26/4 ne donne pas un chiffre rond, on rajoute donc au dernier thread le reste de la division avec cette formule :

Lettre maximale du dernier thread += (26%nombre_de_thread)

Avantage de notre algorithme de multithread :

- **Simple à mettre en place dans le code**
- **Pas lourd en termes de code (temps du programme allégé)**
- **Simple de compréhension**
- **Certains mots seront très rapides à trouver selon la première lettre du mot*[1]**

Inconvénients de notre algorithme de multithread :

- **Répartition sur 26 threads maximum**
- **Certains mots donneront le même temps avec 1 ou plusieurs threads*[1]**
- **Le temps de recherche est déterminé selon la première lettre du mot*[1]**

***[1] : Se référer à la partie « Le problème de la première lettre »**

Le problème de la première lettre :

L'algorithme du multithread est très bien dans le sens où il est simple, mais, il peut aussi être inutile voir moins efficace sur certains mots que sur d'autres. La raison est simple :

Prenons un mot qui commence par 'n': Avec 2 threads, le deuxième thread va être très rapide pour trouver le mot car ce mot-ci commence par la première lettre qu'il va parcourir. Pour l'instant tout va bien.

Prenons maintenant un mot qui commence par une lettre entre 'a' et 'm' : et bien le temps avec 1 seul thread et avec 2 threads sera quasiment équivalent, pourquoi ? et bien le premier thread va parcourir le même nombre d'itérations qu'un programme séquentiel.

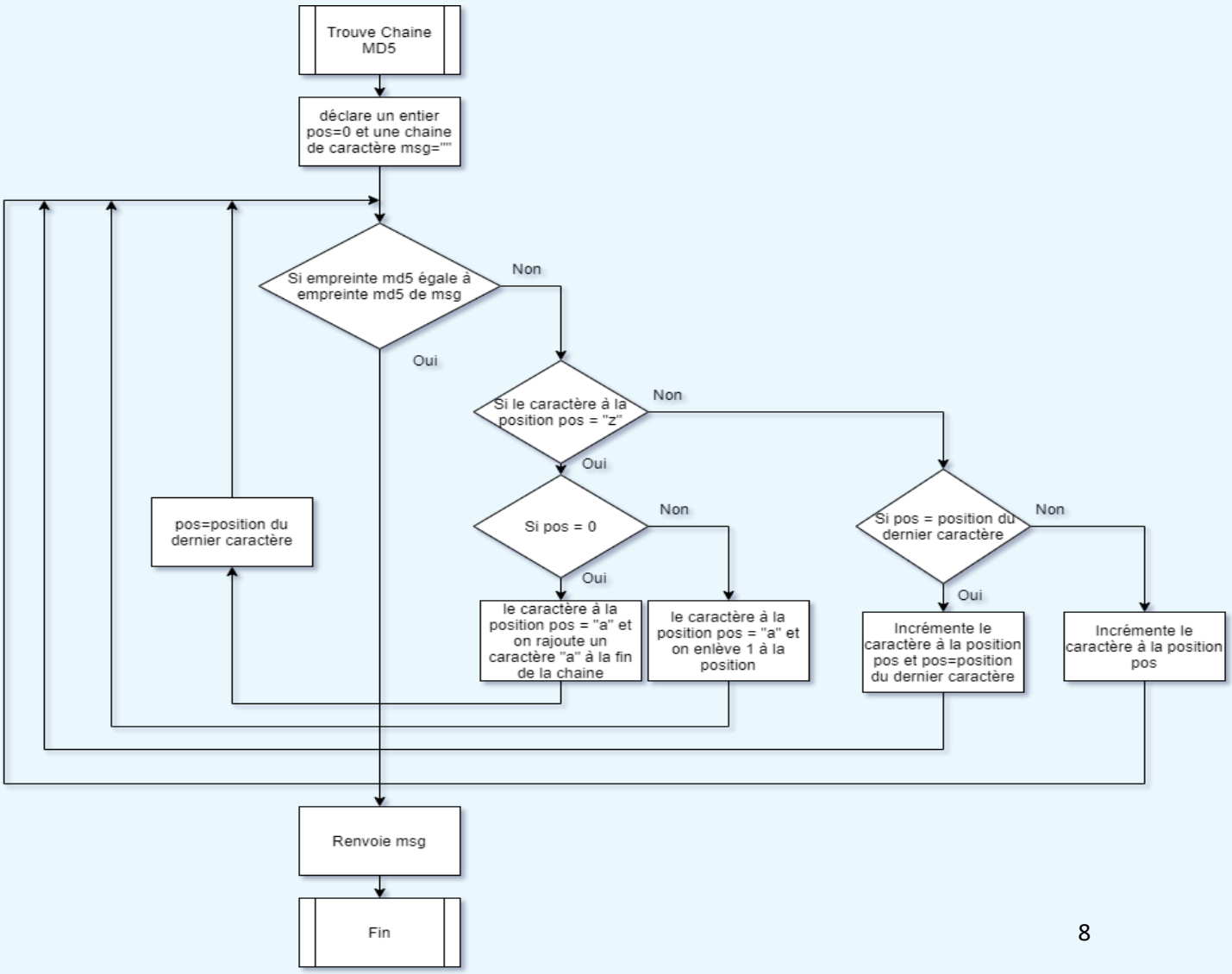
Un problème se pose, la lettre 'a' : Si un mot débute par 'a', que l'on prenne 1, 2, 16 ou même 26 threads, le temps sera quasiment équivalent car la lettre 'a' se trouve au tout début de l'alphabet et il faudra donc toujours parcourir les mots commençant par 'a' en premier.

Le choix de la variable globale sans mutex :

Nous avons décidé, dans le multithread.cpp, d'utiliser deux variables globales, une qui stock la chaine trouvée (string) et une qui indique si un thread a trouvé la chaine (bool).

En partant du principe qu'un seul thread peut trouver la chaine et sur le fait que le thread qui a trouvé la chaine sera le premier qui sortira de sa boucle. Sachant aussi que lorsque le thread sort de sa boucle, il passe le booléen à true et remplit la chaine de caractère. Et pour finir, si dans notre programme (ce qui est le cas), on indique aux autres threads de ne pas accéder ni modifier les variables globales (si booléen égale true), alors le mutex est inutile car les variables globales ne seront modifiées que par le thread qui aura trouvé la bonne chaîne.

Algorigramme du parcours de tous les mots possibles



Algorithmes :

main.cpp:

Inclusion des bibliothèques

Typedef std::chrono::_V2::steady_clock::time_point timePoint

Procédure execute(start : Pointeur sur timepoint , end : Pointeur sur timepoint, programme : Chaine de caractères)
Donnée(s) : Point dans le temps de début, de fin et une chaîne de caractère à envoyer

```
Début
|   param1, nbProc : Chaine de caractère
|   system("clear")
|   Ecrire "Veuillez renseigner une chaine de caractère : "
|   Lire param1
|   Si (programme=="MultiThread")
|   |   Ecrire "Veuillez renseigner le nombre de processeurs à utiliser : "
|   |   Lire nbProc
|   FinSi
|   system("clear")
|   *start ← std::chrono::steady_clock::now()
|   system(("./" + programme +" " + param1 + " " + nbProc). C_str())
|   *end ← std::chrono::steady_clock::now()
Fin
```

Fonction main() : Entier
Résultat : Renvoie une valeur de retour indiquant si le programme s'est bien déroulé.

```
Début
|   selection : char
|   start, end : std::chrono::_V2::steady_clock::time_point
|   Ecrire " Qu'elle methode voulez vous utiliser ? ( s : Sequentiel / m : MultiThread)"
|   Lire selection
|   Si (selection == 's')
|   |   execute(&start, &end, "Sequentielle" )
|   Sinon Si (selection == 'm')
|   |   |   execute(&start, &end, "MultiThread" )
|   |   Sinon
|   |   |   return 0;
|   |   FinSi
|   FinSi
|   elapsed_seconds : std::chrono::duration<double>
|   elapsed_seconds ← end-start
|   Ecrire "Le programme a duré : " + elapsed_seconds.count() + " secondes"
|   return 0 ;
Fin
```

sequentielle.cpp :

```
Inclusion des bibliothèques

Fonction toMd5(↔msg : Chaine de caractères) : Chaine de caractères
Donnée(s) : Chaine du mot à hasher
Résultat : Renvoie l'empreinte md5 calculée

Début
|   digest : Chaine de caractère
|   hash : Chaine MD5
|   hash.Update()
|   digest.resize(hash.DigestSize())
|   hash.Final((byte*)&digest[0])
|   retourne digest
Fin

Fonction FindTextMD5(md5 : Chaine de caractères) : Chaine de caractères
Donnée(s) : Chaine MD5
Résultat : Renvoie le texte associé à l'empreinte MD5

Début
|   Ecrire "Recherche de l'empreinte md5 : "
|   Ecrire "Attention cette opération peut prendre du temps"
|   Variables Locales :  pos  :  size_t
|                        msg : Chaine de caractères
|   pos ← 0
|   msg ← "a"
|   TantQue(md5 != toMD5(msg)) Faire
|   |   Si(msg[pos]== 'a')
|   |   |    Si(pos==0)
|   |   |   |    msg[pos] ← 'a'
|   |   |   |    pos←msg.size()-1
|   |   |   |    msg←msg+'a'
|   |   |   Sinon
|   |   |   |    msg[pos] ←'a'
|   |   |   |    pos--
|   |   |    FinSi
|   |   Sinon Si(pos != msg.size() -1)
|   |   |   msg[pos]++
|   |   |   pos←msg.size()-1
|   |   Sinon
|   |   |   msg[pos]++
|   |   FinSi
|   FinTantQue
|   retourne msg
Fin
```

```
Fonction main(argc : entier, argv[] : tableau de pointeur sur caractères ) : Entier
Donnée(s) : Paramètres de la fonction
Résultat : Renvoie une valeur de retour indiquant si le programme c'est bien déroulé.

Début
|   Si(argc==2)
|   |   encoder(new FileSink(cout)) : HexEncoder
|   |   msg, md5msg, findMsg : Chaine de caractère
|   |   msg ← argv[1]
|   |   md5msg ← toMD5(msg)
|   |   Ecrire " Recherche de l'empreinte md5 :  "
|   |   Ecrire " Attention cette opération peut prendre du temps "
|   |   findMsg ← FindTextMD5(md5msg)
|   |   Ecrire "L'empreinte md5 : "
|   |   StringSource(md5msg, true , encoder : Redirector)
|   |   Ecrire " est associée à la chaine : " + findMsg
|   Sinon
|   |   Ecrire " Erreur : nombre d'arguments non valide "
|   retourne 0
Fin
```

multithread.cpp :

```
Inclusion des bibliothèques

Variables Globales :  estTrouve ← true :  booléen
                      resultat ← "" : Chaine de caractère

Structure paramThread_FindTextMD5

Début
|   md5 : Chaine de caractère
|   minC : caractère
|   maxC : caractère
Fin

Fonction toMd5(↔msg : Chaine de caractères)  : Chaine de caractères
Donnée(s) : Chaine du mot à hasher
Résultat : Renvoie l'empreinte md5 calculée

Début
|   digest : Chaine de caractère
|   hash : Chaine MD5
|   hash.Update()
|   digest.resize(hash.DigestSize())
|   hash.Final((byte*)&digest[0])
|   retourne digest
Fin
```

```
Fonction FindTextMD5(p_struct : pointeur sur vide) : pointeur sur vide
Donnée(s) : Structure de donnée
Résultat : Renvoie le message décodé

Début
|   param : pointeur sur Structure paramThread_FindTextMD5
|   param ← (pointeur sur Structure paramThread_FindTextMD5)p_struct
|   Ecrire "Recherche de l'empreinte md5 : "
|   Ecrire "Attention cette opération peut prendre du temps"
|   Variables Locales :  pos :  size_t
|                        msg : Chaine de caractères
|   pos ← 0
|   msg ← param->minC
|   TantQue(md5 != toMD5(msg) et !estTrouve) Faire
|   |   Si(msg[pos]== 'z')
|   |   |    msg[pos] ← 'a'
|   |   |    pos--
|   |   Sinon Si(msg[pos]==param->maxC et pos==0)
|   |   |   msg[pos] ←param->minC
|   |   |   pos←msg.size()-1
|   |   |   msg←msg+'a'
|   |   Sinon Si(pos != msg.size() -1)
|   |   |   msg[pos]++
|   |   |   pos←msg.size()-1
|   |   Sinon
|   |   |   msg[pos]++
|   |   FinSi
|   FinTantQue
|   Si(!estTrouve)
|   |   resultat←msg;
|   FinSi
|   estTrouve←true;
|   retourne NULL
Fin
```

```
Fonction main(argc : entier, argv[] : tableau de pointeur sur caractères ) : Entier
Donnée(s) : Paramètres de la fonction
Résultat : Renvoie une valeur de retour indiquant si le programme s'est bien déroulé.

Début
|   Si(argc==3)
|   |   Essaye
|   |   |   Si (argv[2]<=0)
|   |   |   |   Ecrire " Erreur : nombre de processus invalides "
|   |   |   FinSi
|   |   FinEssaye
|   |   Catch(e : adresse de la constante d'une exception )
|   |   |   Ecrire " Erreur : " + e.what()
|   |   FinCatch
|   |   encoder(new FileSink(cout)) : HexEncoder
|   |   msg, md5msg : Chaine de caractère
|   |   nbProc : entier désigant une taille
|   |   msg ← argv[1]
|   |   md5msg ← toMD5(msg)
|   |   nbProc ← argv[2]
|   |   t[nbProc] : tableau de thread
|   |   param[nbProc] : paramThread_FindTextMD5
|   |   Pour i : entier allant de 0 à nbProc-1 Faire
|   |   |   param[i].md5←md5msg;
|   |   |   param[i].minC←'a'+(26/nbProc)*i;
|   |   |   if(i+1==nbProc)
|   |   |   |   param[i].maxC←'a'+(26/nbProc)*(i+1)-1+(26%nbProc);
|   |   |   Sinon
|   |   |   |   param[i].maxC←'a'+(26/nbProc)*(i+1)-1;
|   |   |   FinSi
|   |   FinPour
|   |   Ecrire " Recherche de l'empreinte md5 :  "
|   |   Ecrire " Attention cette opération peut prendre du temps "
|   |   Pour i : entier allant de 0 à nbProc-1 Faire
|   |   |   pthread_create(&t[i], NULL, FindTextMD5,(void*)&param[i])
|   |   FinPour
|   |   Pour i : entier allant de 0 à nbProc-1 Faire
|   |   |   pthread_join(t[i], NULL)
|   |   FinPour
|   |   Ecrire " L'empreinte md5 : "
|   |   StringSource(md5msg, true , encoder : Redirector)
|   |   Ecrire " est associée à la chaine : " + findMsg
|   Sinon
|   |   Ecrire " Erreur : nombre d'arguments non valide "
|   retourne 0
Fin
```