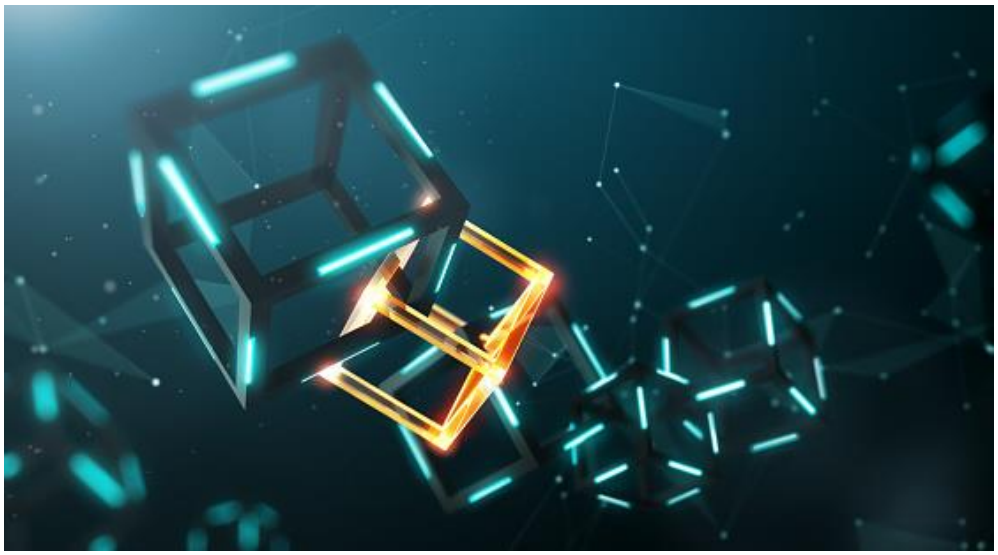
ALGORITHMIQUE ET PROGRAMMATION 2

PROJET : IMPLÉMENTATION DE LA BLOCKCHAIN ET PREUVE DE CONCEPT



DRIANT Arthur – GERARD Hugues

SOMMAIRE

I- Introduction

A l'occasion de notre deuxième projet C dont le but est d'illustrer l'implémentation d'une blockchain en C, nous avons décidé de créer un système de stockage sécurisé de tickets de caisse. En effet, il est obligatoire d'avoir le ticket de caisse de l'objet si l'on veut le ramener en magasin. Il représente une preuve de l'achat et permet de vérifier si l'objet est toujours retournable/échangeable.

De ce fait, si l'on vient à perdre ce dernier, il suffirait d'un peu de créativité pour le reproduire et ainsi pouvoir ramener l'objet acheté. Des personnes malhonnêtes pourraient aussi rééditer leur ticket de caisse pour changer la date et ainsi être dans la période où l'échange est possible.

→ C'est pourquoi nous avons décidé de stocker ces tickets de caisse dans une blockchain afin d'être sûr que la version numérique des tickets ne puisse pas être modifiée.

Pour simplifier notre programme, nous avons décidé de faire un ticket de caisse par bloc en supposant que le temps de validation d'un bloc (le proof of work) est assez rapide pour qu'il n'y ait pas plusieurs tickets en attente d'être ajoutés. Dès lors, chaque ajout d'un bloc à la blockchain représente un passage en caisse.

Pour l'aspect stockage de la blockchain, nous utilisons un fichier .txt dans lequel tous les blocs seront écrits dès qu'ils sont ajoutés à la blockchain. Cela nous permet de la sauvegarder lorsque l'on quitte le programme.

Tout cela est présenté sous la forme d'un menu où l'on peut choisir de créer un ticket de caisse (c'est-à-dire un nouveau bloc), visualiser la blockchain en entier, visualiser tous les tickets édités par un(-e) hôte(-sse) de caisse ou encore chercher un ticket par son index. À chaque fois il est possible de modifier le ticket que vous êtes en train de consulter si vous en êtes l'éditeur. Une modification entraînera le recalcul de tous les hash de la chaîne à partir du bloc modifié.

II- Structure de données, librairies et fonctions utilisées

Notre projet est décomposé en 5 fichiers .c :

- Main.c qui, comme son nom l'indique, contient la fonction main.
- Blockchain.c qui contient toutes les fonctions relatives à la création de blocs, leur remplissage, leur ajout à la blockchain et leur modification.
- Proofofwork.c qui contient toutes les fonctions relatives à la validation des blocs et le proof of work.
- Interface.c qui contient l'interface affichée sur la console lors du lancement du programme.
- Admin.c qui contient la fonction de création de la blockchain à partir d'un fichier texte « chaine.txt »

Le fichier « chaine.txt » contient en première ligne la difficulté de la fonction proof of work, cela représente le nombre de 0 par lequel le hash de chaque bloc devra commencer. Les blocs d'après représentent les blocs que l'on va ajouter à la chaine lors de l'initialisation du programme. Par exemple :

```
1                          ← Difficulté
Hugo                       ← Auteur du bloc (caissier)
Le_25/03/2020_à_15h_45min. ← Date d'édition du ticket
TV____299€                 ← Article et prix
```

L'initialisation de la blockchain à partir de ce fichier correspond à plusieurs ajouts « réglementaires » faits par des hôtes(-sses) de caisse, ils s'ajoutent donc tout seul à la fin de la chaîne et le hash est calculé à ce moment-là.

Pour coder la blockchain, on a créé deux structures :

```
typedef struct
{
        block* tete;
} block_chain;
```

- La structure « block_chain » qui est un pointeur vers le premier bloc de la chaine.

```c
typedef struct _block
{
        int index;
    int nonce;
        char auteur[20];
        char timestamp[30]; //Aura la forme suivante : "Le_DD/MM/AAAA_à_hhH_minminMIN."
        char ticket[100];   //Aura la forme suivante :"objet_acheté___prix€"
        unsigned char hash[5*SHA256_DIGEST_LENGTH];
        unsigned char prev_hash[5*SHA256_DIGEST_LENGTH];
        struct _block* suiv;
} block;
```

- La structure « block » qui contient l'auteur du bloc, la date de création, les données du ticket, le hash du bloc précédent et le hash de ce bloc.

Nous n'avons pas créé de structure pour le premier bloc, c'est un bloc normal avec un prev_hash égal au mot vide. Cependant nous avons quand même dû faire une fonction différente (ValidGenesisBlock) pour vérifier sa validité car c'est le premier bloc, il n'est donc pas possible de comparer le prev_hash avec le hash du bloc précédent.

La fonction de hachage utilisée est la SHA_256 importée à l'aide de la librairie « openssl/sha.h ». Elle prend en argument une chaine de caractères, sa longueur et le type de l'argument 1 et renvoie le hash correspondant. Elle intervient dans la fonction « ajout_hachage » :

```c
void ajout_hachage(block* b)
{
    char index_c[10];   //10 est le nombre de chiffres max
    sprintf(index_c,"%d", b->index);

    char nonce_c[20];  //20 pris au hasard
    sprintf(nonce_c, "%d", b->nonce /*b->nonce*/);

    char s[290];
    strcpy(s,"");   //On la reset (pour éviter de pointer sur des choses déjà là avant)


    /*Concaténation de toutes les données dans s à prendre en compte pour le calcul du hash*/
    strcat(s,index_c);
    strcat(s,nonce_c);
    strcat(s,b->auteur);
    strcat(s,b->timestamp);
    strcat(s,b->ticket);

    strcat(s,b->prev_hash); //A la fin, s contient toutes les infos du bloc concaténées en une chaîne de caractères.
    //printf("tête de s : %s\n",s);


    unsigned char *d = SHA256(s, strlen(s), 0);

    /*On convertit ensuite d qui est en hexa en une chaine de caractères*/
    strcpy(s,"");   //On va se re-servir de s
    char hex[3];
    strcpy(hex,""); //On l'initialise aussi

    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
        strcpy(hex,"");
        sprintf(hex, "%02x", d[i]);
        strcat(s,hex);
    }

    //printf("Hash calculé : %s\n",s);  A NE PAS DECOMMENTER CAR FONCTION UTILISEE DANS BOUCLE POW

    strcpy(b->hash, s);
```

Cette fonction prend un bloc b en argument et complète son champ b->hash. Dans un premier temps on concatène toutes les informations pour calculer le hash dans une variable appelée « s ».

Pour ce faire, on convertit d'abord l'index du bloc et le nonce en char* grâce à la fonction « sprintf » qui prend en argument la variable de destination (ici respectivement index_c et nonce_c), puis le type de donnée que l'on souhaite convertir et enfin la donnée à convertir. On concatène ensuite grâce à strcat de la librairie stdio.

Une fois toutes les informations concaténées (index, nonce, auteur, date, ticket et prev_hash), on appelle la fonction sha_256 et elle nous renvoie une chaine de char que l'on convertit en hexadécimal et que l'on complète dans b->hash.

Une fois le bloc complété, on teste sa validité avec la fonction « POW ».

```c
void POW(block *b)  //A appeler lorsqu'on ajoute un bloc à la blockchain ou lorsque l'on modifie un bloc
{

    FILE* fichier = fopen("chaine.txt","r");
    int difficulty;
    fscanf(fichier,"%d",&difficulty);   //On récupère la difficulty dans le fichier chaine.txt
    fclose(fichier);

    int i;
    char proof[difficulty];     //Proof contient les "difficulty" premiers char du hash
    char list0[difficulty];
    char char_to_chain[2];

    strcpy(list0,"");   //On reset la case mémoire de la liste

    for(i=0;i<difficulty;i++)   //Initialisation de la liste de 0 et de proof
    {

        sprintf(char_to_chain,"%c",b->hash[i]);
        strcat(proof,char_to_chain);
        strcat(list0,"0");      //Liste de "difficulty 0"
    }


    while(strcmp(proof,list0)!=0)   //Tant que le hash ne commence pas par "difficulty" 0, on incrémente nonce et on re-calcule le hash du bloc
    {
        b->nonce+=1;        //Correspond au nombre de fois que le hash a été calculé (le nonce est inclus dans le calcul du hash)

        ajout_hachage(b);       //On utilise la fonction du fichier blockchain.c pour recalculer le hash avec la nouvelle valeur de nonce

        strcpy(proof,"");   //Reset de proof

        for(i=0;i<difficulty;i++)   //On réécrit proof
        {
            sprintf(char_to_chain,"%c",b->hash[i]);
            strcat(proof,char_to_chain);
        }

    }
    printf("\nBloc validé ! ;-)\n\n");
```

La fonction « POW » est la fonction « Proof Of Work » qui permet, comme son nom l'indique, de prouver que l'ordinateur travaille à l'ajout de chaque bloc. En effet, un utilisateur pourrait rajouter des millions de blocs à une chaîne rapidement car le calcul du hash est très rapide. Pour éviter cela, on force la machine à travailler à l'ajout de chaque bloc.

On impose alors une difficulté nommée « difficulty » dans le code qui représente le nombre de zéros par lequel le hash du bloc doit commencer. Le hash ne commence évidemment pas forcément par ce nombre de zéros demandé, on cherche donc à modifier le hash jusqu'à ce qu'il commence par ces zéros. C'est alors qu'intervient le « nonce » cité auparavant. Cet entier est choisi aléatoirement (nous avons choisi de l'initialiser à 0 dans notre projet) et permet, en l'augmentant de 1 à chaque passage dans une boucle, d'obtenir un nouveau hash du bloc sans modifier les données contenues dans ce bloc.

Voici le principe de la fonction « POW » : on vérifie la valeur des « difficulty » premiers éléments du hash de notre bloc (que ce soient des entiers ou des caractères) et tant qu'ils ne sont pas tous égaux à 0, on indente le nonce de 1 et en modifiant cette valeur, on peut recalculer un nouveau hash. Plus on augmente la difficulté, plus le hash doit commencer par un grand nombre de zéros et plus la boucle va tourner longtemps et donc faire travailler la machine. On se rend compte que l'ordinateur travaille pour insérer chaque bloc à partir d'une difficulty de 4.

Nous avons choisi de vous présenter cela sous la forme d'une interface où vous pouvez consulter la blockchain, éditer un bloc (seulement si vous en êtes le créateur) et en ajouter un nouveau.

III- Limites de notre modélisation

Notre modélisation contient tout de même quelques limites.

Tout d'abord, nous avons décidé que seuls les hôtes et hôtesses de caisse pouvaient ajouter un bloc à la chaîne, pour imiter le passage en caisse. Cependant, il pourrait être possible qu'un consommateur puisse ajouter un ticket de caisse lui-même lors d'un problème informatique en magasin par exemple.

De plus, comme expliqué précédemment, à l'échelle de notre modélisation, nous avons décidé de stocker qu'un ticket de caisse par bloc. Cela nous a permis de mieux visualiser les différents éléments d'une blockchain et de l'implémenter plus facilement. Pour un vrai magasin ou une chaîne de supermarché, il paraît plus logique de stocker plusieurs tickets dans un même bloc, pour limiter le nombre de blocs et le proof of work.

Pour finir, afin de nous concentrer sur nos connaissances en C et pour éviter de rendre cette modélisation trop difficile, nous avons pris le choix de négliger le réseau peer-to-peer, permettant à un groupe d'appareil de stocker et partager collectivement des fichiers. Implémenter cette technologie aurait grandement augmenté la difficulté de cette modélisation et nous ne cherchions pas à nous pencher sur cette partie-là de notre formation, c'est-à-dire les cours de réseau mais plutôt à se concentrer sur la partie blockchain afin d'en comprendre parfaitement le fonctionnement et son intérêt.

IV- Conclusion

Dans le cadre de ce deuxième projet en langage C, nous avons choisi de modéliser la blockchain en un service de sauvegarde de tickets de caisse. L'interface proposée dans cette modélisation permet donc, en tant que caissier ou caissière, de rajouter un ticket dans la chaîne ou de modifier les tickets déjà existants.

Les nouveaux tickets sont d'abord vérifiés avant d'être ajouté à la chaîne et oblige l'ordinateur à travailler grâce au proof of work, comme dans une blockchain.

La modification d'un ticket entraîne une modification du hash d'un block et donc celui de tous les blocs suivants, afin de conserver une chaîne valide.

Tout cela a été implémenté à l'aide de différentes fonctions, permettant un bon fonctionnement de l'ensemble de la blockchain.

Ce projet nous a permis de découvrir ce qu'était une blockchain, son importance dans le monde aujourd'hui et tout cet aspect de sécurité que nous n'avions pas abordé auparavant.