



RAPPORT HYPERCUBE

PAR CHARRIER KILLIAN ET
VERPOORTE WILLIAM

L2 Informatique INFO0403

TABLE DES MATIÈRES

01 Fonctionnement et Structure du projet

03 Fonctionnement des Sommets

05 Fonctionnement du Token

07 Gestion des Signaux

09 Compilation du Projet

06 Fichier Principal



FONCTIONNEMENT ET STRUCTURE

Le but du projet Hypercube est de créer une structure en forme d'hypercube où entre chaque sommets de la structure se déplace un jeton

Premièrement, un Hypercube se définit comme une structure à plusieurs dimensions, allant de 0 pour un simple point à quatre ou plus pour un tesseract.

On considère ici que chaque sommet est un processus où pourra passer un jeton sous forme de message ou autre.

Notre code va donc devoir répondre à plusieurs problématique pour résoudre ce problème.

Il faudra donc qu'il crée le nombre de sommet adéquats, les reliées entre eux pour suivre la logique d'un hypercube.

Ensuite, il faut que le token soit créé et envoyer correctement dans l'hypercube.

Puis en plus créer un système de gestion de signaux pour arrêter le déroulement du programme et le reprendre à volonté.




FONCTIONNEMENT ET STRUCTURE

Nous divisons donc notre projet en plusieurs fichiers qui s'occupent chacun d'une des tâches citées.

Le fichiers `sommet.c` qui commence par créer les sommets d'une manière spécifique que nous verrons plus tard.

Le fichiers `Token.c` qui comme son nom l'indique permet de crée le jeton et de gérer son déplacement au sein des sommets.

Et enfin, le fichiers `hypercube.c` qui correspond au main, il a pour but d'utiliser les fichiers précédent pour mettre en place la structure de l'hypercube, de lancer le token et de gérer les signaux.

 `hypercube.c` `sommet.c` `token.c`

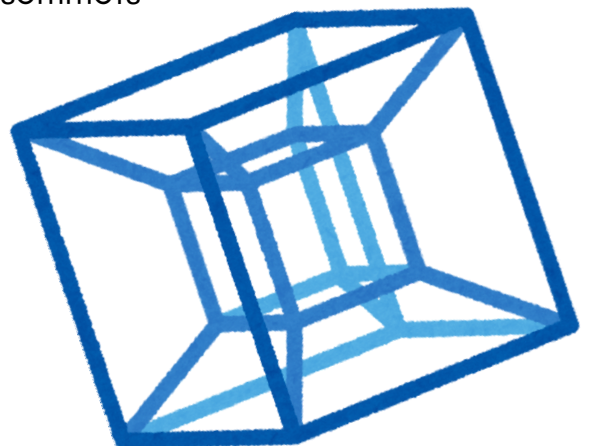
FONCTIONNEMENT DES SOMMETS

Regardons de plus près le fichiers sommet.c.

Nous choisissons de créer nos sommets sous forme de struct pour répondre au différentes contraintes qui se pose tout au long du projet.

Nous retrouvons dans un sommet :

- le pid pour lancer le processus de chaque sommet
- la variable etiq qui est l'identifiant du sommet, qui permettra de retrouver plus facilement ses voisins
- Un tube pipefd pour envoyer ou recevoir le token selon si il le possède ou non
- nb_adj qui contient le nombre de sommets adjacents du sommet, pour faciliter l'envoi du token à un autre sommet
- Un tableau adj qui contient les étiquettes de tous ses sommets adjacents



FONCTIONNEMENT DES SOMMETS

Enfin nous retrouvons dans ce fichiers 2 fonctions :

- `init_sommet` qui initialise les différentes variables du sommet. Il met la variable `etiq` à la valeur d'un compteur qui se trouve dans le main, met le nombre d'adjacents à 0 pour le début puis alloue un tableau dynamique pour stockés les adjacents, en lui donnant la taille du paramètre entré par l'utilisateur lors de l'appel de la commande hypercube.
- `add_adj` qui prend en paramètre un sommet adjacent pour l'ajouter dans le tableau d'adjacents en ajoutant au nombre d'adjacents du sommet courant.

La formation de la structure de l'hypercube se fera elle dans le fichier hypercube que l'on verra plus tard.

FONCTIONNEMENT DU TOKEN

Le fichier token.c se concentre lui sur la gestion du token et la gestion de son déplacement.

Notre token n'est pas réellement un message ou un signal en lui même, il s'agit de notre fonction qui prend en paramètre un sommet et réagit selon ce sommet.

Cette fonction crée en premier temps un fichier du nom de l'étiquette du sommet en paramètre, ce qui montre que le token est "passé" par ce sommet.



En plus du fichier du sommet, la fonction ouvre un fichier LOG, une seule fois si il n'est pas déjà ouvert.

Dans ce fichier la fonction écrit un message contenant le nom du sommet où passe le token ainsi que la date où il y passe.

Ensuite, elle choisi aléatoirement un sommet parmi ceux présent dans le tableau d'adjacents du sommet courant pour lancer à nouveau la fonction dans ce sommet choisi. La fonction est récursive.

Une pause de 0.5 seconde est effectué dans chaque exécution pour éviter des crashes.

FICHER PRINCIPAL

Le fichier principal hypercube.c s'occupe de relier les 2 fichiers précédents pour former le tout du programme.

Il utilise la création des sommets pour former la structure de l'hypercube selon la dimension entrée en paramètre lors de l'appel de hypercube.

En même temps que l'hypercube se forme, la boucle vérifie si un sommet est adjacent à un autre et ajoute dans le tableau d'adjacent d'un sommet si nécessaire.

Tout les sommets créés sont ajoutés dans un tableau de sommets générales.

Ensuite, le lancement du Token se fait dans un processus fils que l'on pourra gérer à côté du processus père.

Enfin, il y a la gestion des signaux que nous verrons après et à la fin du programme, la libération de mémoire des tableaux dynamique alloués dans les sommets et le tableau générales de tout les sommets.

GESTION DES SIGNAUX

Dans la boucle while du processus père, on demande à l'utilisateur de choisir entre suspendre, quitter ou reprendre si le token est en pause.

Un switch permet d'effectuer une action selon l'entrée de l'utilisateur.

Chaque signaux possède un gestionnaire qui est appelé à chaque fois que l'on appel un signal :

- Suspendre : on appel le gestionnaire de SIGSTOP qui applique SIGSTOP avec la commande kill, sur tout les processus des sommets. SIGSTOP est aussi appelé sur le processus du token, un variable pause_execution est mise à 1 pour arrêter le token.
- Reprendre : on appel le gestionnaire de SIGCONT qui applique SIGCONT avec la commande kill, sur tout les processus des sommets. CONT est aussi appelé sur le processus du token, un variable pause_execution est mise à 0 pour empêcher l'arrêt du token et la fonction Token est relancé.

GESTION DES SIGNAUX

- Quitter : aucun gestionnaire n'est appelé, nous exécutons la commande "pkill hypercube" avec la fonction execl pour arrêter tout les processus du programme et y mettre fin.

La variable `pause_execution` est définie dans le fichier `Token` comme un struct `shared_data`, pour pouvoir plus facilement faire communiquer le changement de cette variable dans le main avec la fonction `token`.

En plus de la variable, on y met un sémaphore pour protéger l'accès de cette variable.

Ainsi le déplacement du `Token` se met correctement en pause à l'appel de `SIGSTOP`.



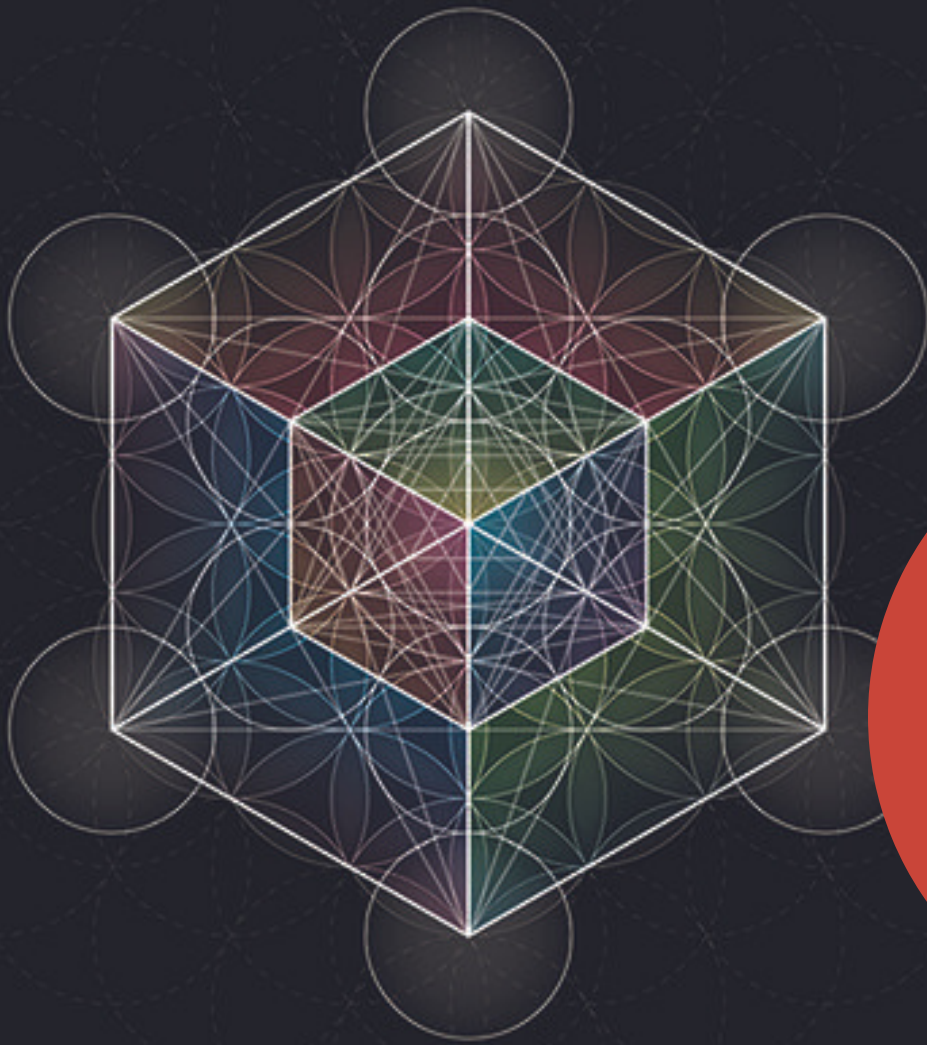
COMPILATION DU PROJET

Notre projet étant formé de 3 fichiers sources, au lieu de compiler séparément les 3 fichiers, nous facilitons le processus avec un CMAKE, qui définit comme sources nos 3 fichiers principaux, les lient entre eux et crée notre exécutable "hypercube".

Comme indiqué dans le README, nous avons effectué le build du projet avec ninja, toutes les commandes nécessaires y sont indiqués.

Aussi, étant donné que le lancement de hypercube crée possiblement de nombreux fichiers selon la dimensions de l'hypercube, nous avons fait en sorte que tout les fichiers commencent par "process" pour faciliter la suppression de ceci, encore une fois la commande pour est écrit dans le README.





NOUS VOUS REMERCIONS DE VOTRE
ATTENTION ET DE VOTRE LECTURE

FIN

PAR CHARRIER KILLIAN ET
VERPOORTE WILLIAM