LALLEMAND Romain
JESSON Robin

Rapport du projet LO41

Ascenseur

I. Introduction

Ce projet c'est fait de le cadre de l'UV LO41 Systèmes d'exploitation de l'UTBM. Nous nous sommes organisé à deux en nous réunissant fréquemment et en utilisant la plateforme Github.

II. Analyse du problème

Le but de ce projet est de nous amener à simuler un système gérant des ascenseurs. Il faut créer le système en utilisant plusieurs outils vu en cours de LO41.

L'ascenseur doit être capable d'aller chercher une personne à un étage et de le déposer à l'étage souhaité par la personne. De plus il faut s'arrêter en chemin pour pouvoir prendre une personne qui se situerait entre l'ascenseur et une autre personne, également s'arrêter à mi-chemin pour en déposer une autre.

Il faut également respecter la contrainte du nombre maximum qu'une cabine d'ascenseur peut transporter.

III. Méthode proposée et résultats

Premièrement l'ascenseurs est un thread, les clients de l'ascenseur le sont également. Nous avons opté pour ce choix plutôt que des processus lourds par simplicité d'utilisation par la suite.

L'ascenseur est une structure comprenant les champs entre autres son numéro, son étage actuel, le nombre de client à l'intérieur et sa capacité maximum, la liste des étages ayant reçus un appel depuis un bouton, la liste des étages d'arrivés.

Une méthode *genererAscenseur*() permet de créer plusieurs threads sur des fonctions *ascenseur*(). Celles-ci sont utilisées lors de la création des threads et prennent en paramètre une adresse d'instance de la structure *Ascenseur* en les ayant préalablement initialisés sous la forme d'un ascenseur vide au rez-de-chaussée.

Les clients sous sous la même forme, c'est-à-dire des threads agissant sur des structures, avec les champs numéro, étage de départ et d'arrivée, et une heure d'arrivée.

Les structures de clients sont initialisés puis envoyés lors de la création de threads à la fonction *client*().

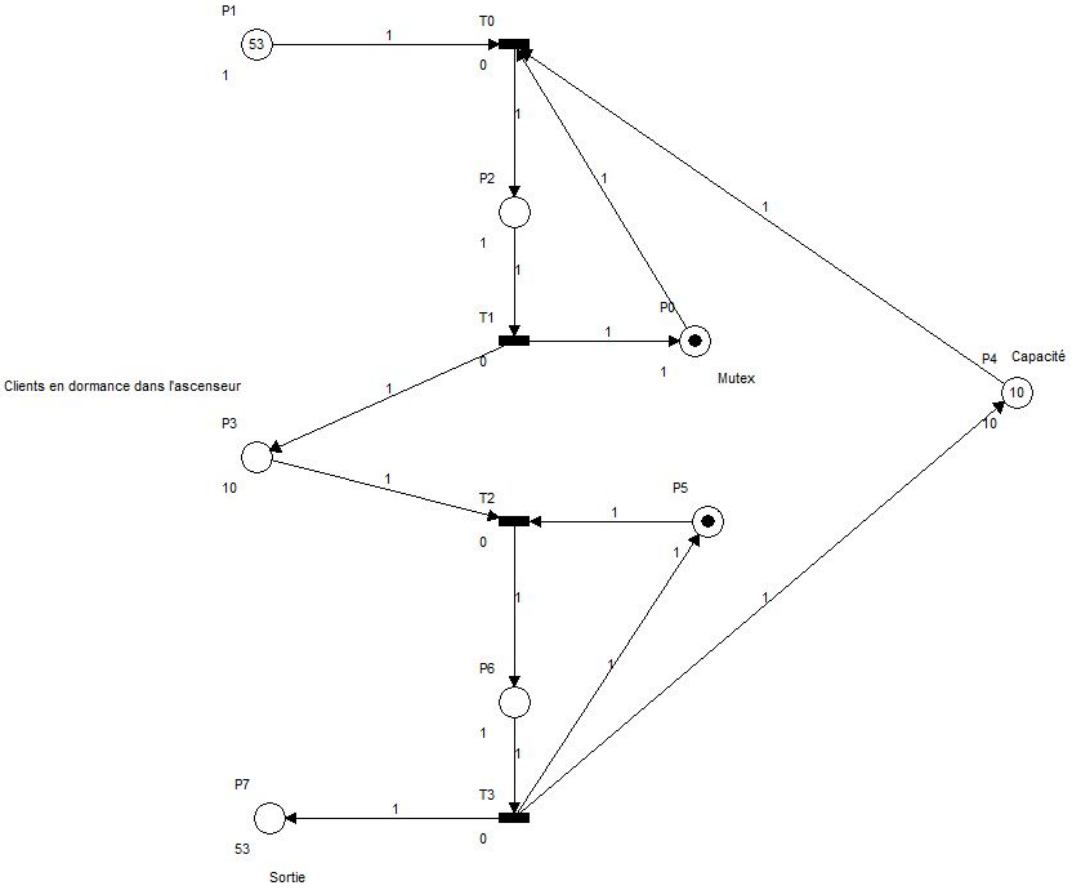
La fonction client doit permettre à un client de dormir un temps donnée, l'heure d'arrivée, avant d'arriver à l'ascenseur. Puis il appelle l'ascenseur via la fonction *appelAscenseur*(). Celle-ci s'occupe d'envoyer un message à l'ascenseur contenant l'étage ou l'appel est effectué et l'étage où le client souhaite se rendre.

Une fois le message envoyé, le client attend l'ascenseur par le biais d'un mutex et d'une condition. L'ascenseur, lorsqu'il arrive devant le client, le notifie avec un *pthread_cond_signal* qu'il peut se réveiller. Ensuite le client peut rentrer dans l'ascenseur. Une liste contenant le nombre de personne qui attendent à cet étage est décrémentée, de même qu'une autre liste pour le nombre de personne qui doivent sortir à l'étage demandé par le client est incrémentée. Puis le client avertis l'ascenseur qu'il peut repartir en déverrouillant un mutex.

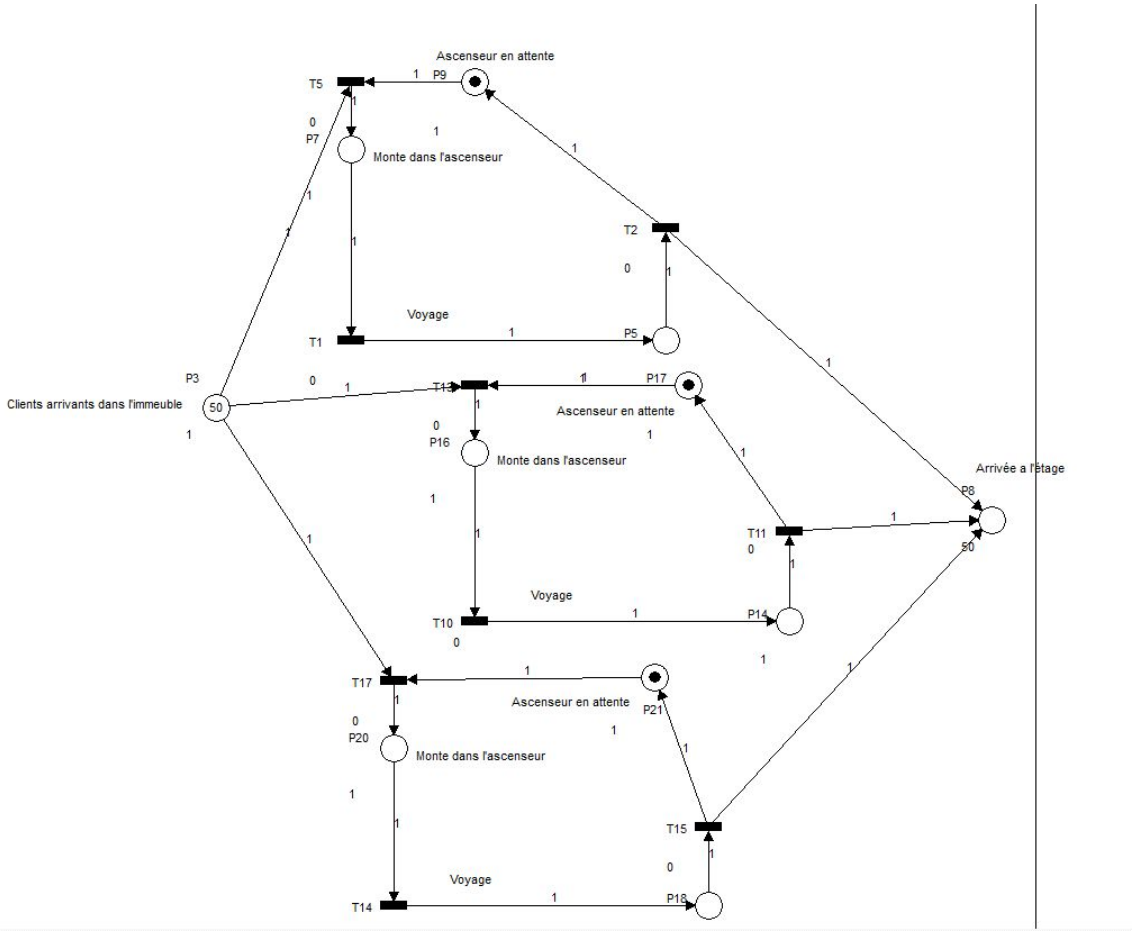
Le client s'endort le temps que l'ascenseur ne lui a pas redonné la main une fois arrivé au bon étage.

Enfin, le client sort de la cabine en prenant soin à bien incrémenter la valeur de personnes arrivées et décrémenté la liste des personnes attendues à cet étage.

Nous avons implémenté trois scénarios. Le premier correspond à notre test de départ avec des valeurs instanciés nous mêmes et un seul client. Le second génère ces personnes à des paliers différents. Le dernier scénario devait en plus les faire apparaître à divers moments.



Entrée et sorties des clients



Répartition des clients sur plusieurs ascenseurs

IV. Difficultés rencontrées

Parmi les difficultés il y a eu la communication de l'ascenseur vers les clients. Nous souhaitions au début, lors d'un appel d'un ascenseur, envoyer un message du type 2 afin de remplir une queue contenant dans l'ordre les appels à l'ascenseur. Egalement, nous voulions qu'un ascenseur, lorsqu'il arrive à un étage, envoie avec un message de type 1 l'étage où il se trouve tous les clients à l'intérieur et à l'extérieur, afin que ces derniers compare avec leurs étages et qu'ils entrent ou sortent selon les cas. Le problème est que si nous avions plusieurs ascenseurs, tous les clients seraient tentés de sortir malgré qu'ils ne soient pas au bon endroit.

Par ailleurs nous nous sommes beaucoup questionnés au début sur le fonctionnement de l'ascenseur, c'est-à-dire quand s'arrêter, monter ou descendre. Nous avons réussi à implémenter une solution pour qu'il aille voir la première qui a fait un appel. Sur sa monté, il peut prendre des personnes ou les déposer tant que c'est dans le sens de voyage actuel.

V. Améliorations possibles

La première amélioration que l'on peut citer est d'ajouter plusieurs ascenseurs. Nous avons privilégié d'abord un ascenseur, et pour le moment, le code n'est pas tout à fait adapté pour plusieurs ascenseurs.

Également le dernier scénario est à améliorer afin de pouvoir simuler presque réellement un immeuble.

VI. Conclusion

Ce projet nous a montré comment sont effectués différentes tâches par un système d'exploitation, notamment comment s'appuyer sur le multi-processus. De plus cela nous a permis de comprendre comment ces mécanismes fonctionnent ou sont implémentés dans les autres langages tel que le Java par exemple, ce qui rend l'ensemble moins flexible du fait que c'est déjà implémenté.

VII. Annexes

ascenseur.c

Fichier comprenant les commandes de gestion du thread d'un ascenseur.

client.c

Fichier pour la gestion des threads Clients.

main.c

Programme principal. A exécuter en appelant le nom du programme + le nombre de client souhaité + le scénario (1 = test ou 2 = tous les clients arrivent en même temps ou 3 = les clients arrivent avec un temps de latence les uns par rapport aux autres).

structure.c

Fichier comprenant les variables globales et structures.