Projet C++

Plazza

Dan Baudry baudry_d@epitech.eu
Maxime Montinet zaz@epitech.net
David Giron thor@epitech.net

Résumé: Le but de ce projet (outre un hommage à peine déguisé à notre chère cantine du coin de la rue) est de vous faire manipuler le multi-process/multi-thread, à travers la simulation d'une pizzeria et de ses cuisines. Vous apprendrez à gérer des problématiques variées : la répartition de charge, la synchronisation de processus et de threads, la gestion de cuisiniers, ...

Table des matières

Chapitre I

Informations

Tout d'abord, voici un peu de lecture pour vous mettre en appétit :

- Les pizzerias : http://fr.wikipedia.org/wiki/Pizzeria
- Les pipes nommés : http://en.wikipedia.org/wiki/Named_pipe
- Les processus : `man fork`, `man exit`, `man wait`, `man ...`
- Les threads POSIX : `man pthread_*`

Chapitre II

Generalités

Le but de ce projet est de vous faire réaliser une simulation de pizzeria, qui se présentera sous la forme d'un accueil acceptant des commandes, et de multiples cuisines, elles-mêmes contenant de multiples cuisiniers, qui réaliseront les pizza.

Voici un apercu de l'architecture souhaitée :

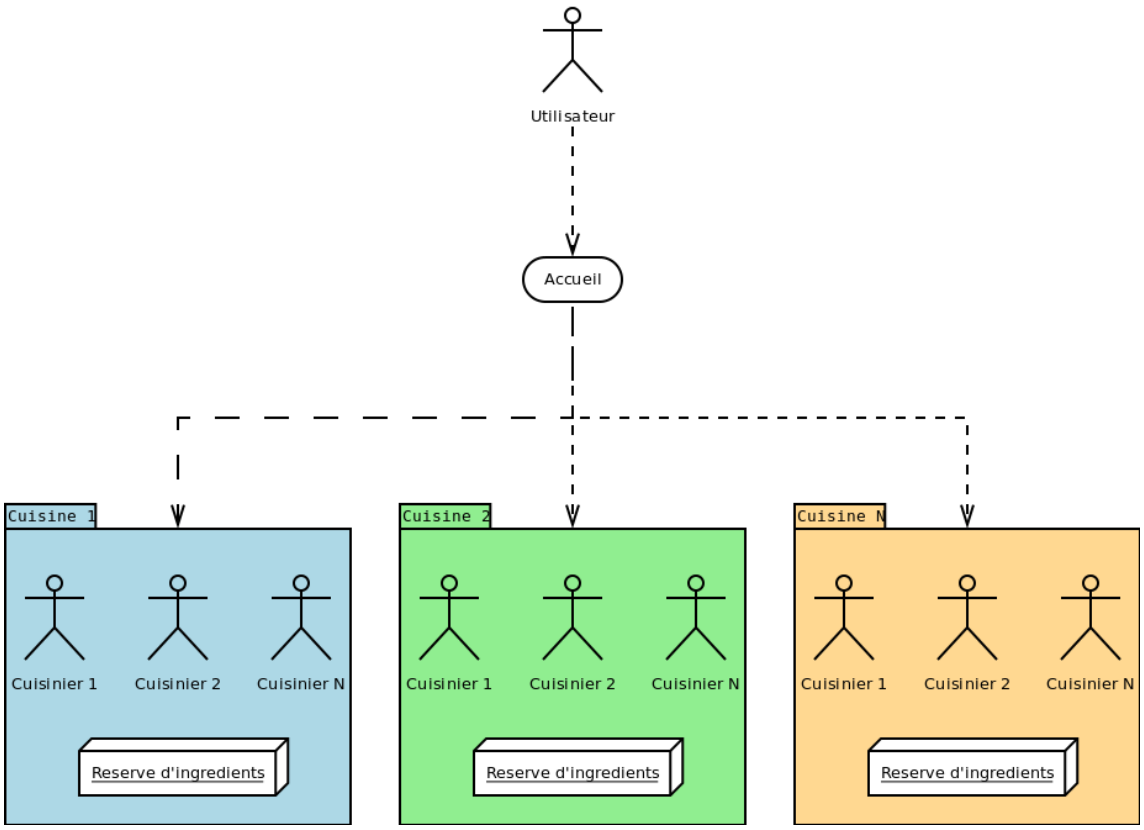

FIGURE II.1 – `Architecture`

Chapitre III

Partie obligatoire

III.1 L'accueil

L'accueil doit pouvoir être lancé en ligne de commande, de la façon suivante :

```
1    >./plazza 2 5 2000
```

- Le premier paramètre est un multiplicateur de temps de cuisson des pizzas. Il servira à observer le fonctionnement de votre programme plus facilement, il DOIT donc **OBLIGATOIREMENT** être implémenté. Il ne sera pas possible de vous évaluer sinon. De plus, ce paramètre DOIT pouvoir interpréter des valeurs comprises entre 0 et 1 pour obtenir un diviseur de temps de cuisson des pizzas...Nous verrons ce que sont les temps de cuisson plus loin.
- Le second paramètre est le nombre de cuisiniers par cuisine. Nous verrons ce que sont les cuisiniers plus loin.
- Le troisième paramètre est le temps en milli-secondes utilisé par les réserves des cuisines pour récupérer un ingrédient. Nous verrons ce que sont les ingrédients plus loin.

L'accueil **DOIT** être interactif, c'est à dire qu'il doit présenter une interface graphique permettant **au moins** les choses suivantes :

- Commande d'une pizza par l'utilisateur mode texte, par exemple "regina XXL x7". Nous verrons le détail plus loin.
- Visualisation de l'état des cuisines, de l'occupation actuelle de leurs cuisiniers et de leur réserve d'ingrédients.

La bibliothèque utilisée pour l'affichage et la prise de commandes n'a pas d'importance. Le barème ne proposera que peu de points à ce sujet.

La partie "graphique" n'est **PAS IMPORTANTE**. Il serait donc avisé d'utiliser quelque chose de simple afin de ne pas perdre de temps (regardez du côté de NCurses, NDK++, Qt, ...).

Les commandes de pizzas DOIVENT respecter la grammaire suivante :

```
1  S := TYPE TAILLE NOMBRE [;TYPE TAILLE NOMBRE]*
2
3  TYPE := [a..zA..Z]+
4
5  TAILLE := S|M|L|XL|XXL
6
7  NOMBRE := x[1..9][0..9]*
```

Exemple de commande grammaticalement valide :

```
1  regina XXL x2; fantasia M x3; margarita S x1
```

La simplicité de la grammaire pour ce projet ne saurait justifier un parseur trop primitif ! Les splits et autres hacks sont donc a éviter absolument ...

- Il DOIT être possible d'ajouter autant de nouvelles commandes que l'utilisateur le souhaite pendant l'éxécution. Le programme DOIT s'adapter.
- Quand il reçoit une commande, l'accueil DOIT la répartir, pizza par pizza, entre les cuisines. Quand toutes les cuisines sont pleines, il DOIT en créer une autre (C'est a dire se forker, voir plus loin).
- L'accueil DOIT toujours charger les cuisines à sa disposition de la manière la plus égale possible. Par exemple, si vous avez 3 cuisines de 3 cuisiniers et 7 pizzas à répartir, vous ne devez pas en donner 6 à une cuisine, 1 à la deuxième, et rien à la troisième.
- Quand une commande est terminée, l'accueil DOIT l'afficher à l'utilisateur et en garder un historique (un fichier de log en plus de toute forme d'affichage pourrait être une bonne idée ...).

III.2 Les cuisines

Les cuisines sont des processus fils de l'accueil, qui les crée au fur et à mesure des besoins. Elles possèdent un certain nombre de cuisiniers, déterminé au lancement du programme.

Les cuisiniers DOIVENT être representés par des threads. Tant qu'ils n'ont rien à faire, ils DOIVENT rester en attente, et se mettre un par un au travail au fur et à mesure que les commandes arrivent.
Ces threads DOIVENT être ordonnancés par un Thread Pool local à chaque cuisine.

Vous DEVEZ proposer des encapsulations objets aux notions suivantes :

- Les processus
- Les threads
- Les mutex
- Les variables conditionnelles

`Ces 4 abstractions représenteront une partie très importante des points disponibles dans le barême. Réalisez donc cette encapsulation intelligemment ...`

De plus :

- Chaque cuisine DOIT avoir un nombre de commandes total (C'est à dire en cours de réalisation et/ou en attente) égal à 2 x N, N étant le nombre de cuisiniers. Une fois que trop de commandes sont accumulées, la cuisine doit les refuser.
- Si toutes les cuisines diponibles ne peuvent plus accepter de commandes, l'accueil DOIT ouvrir une nouvelle cuisine pour réduire la charge et continuer de produire des pizzas.
- Les cuisiniers ont l'amour du travail bien fait. En conséquence, un cuisinier ne DOIT pas faire plus d'une pizza à la fois !
- Les cuisines communiquent avec l'accueil par des `pipes nommés`. Aucun autre moyen de communication n'est autorisé. Vous DEVEZ donc utiliser les `pipes nommés`.
- Vous DEVEZ proposer une encapsulation objet aux `pipes nommés`. Cette encapsulation PEUT proposer des sucharges pour les opérateurs "«" et "»".
- Si une cuisine n'a plus de travail pendant plus de cinq secondes, elle DOIT fermer.
- La cuisine possède une réserve d'ingrédients qui contient à la création 5 unités de chaque ingrédient. La réserve se régenère au rythme de 1 unité de chaque ingrédient

toutes les N milli-secondes. Ce délai est bien sur celui passé a la ligne de commande.

- Les pipes DOIVENT être unidirectionnels. Par cela on entend bien "un pipe par direction", et non "un seul pipe half-duplex". On PEUT donc distinguer des pipes "in" et des pipes "out".

`La création et la destruction d'une cuisine sous-entend des problématiques de communication auxquelles il faut être très attentif ...`

III.3 Les pizzas

Comme expliqué plus haut, l'accueil doit répartir les commandes entre les cuisines, pizza par pizza. Par exemple, si on commande 7 margaritas, il va y avoir de quoi occuper en tout 7 places en cuisine.

Lors du transit par les pipes nommés, les informations sur les commandes et retours de pizzas DOIVENT etre sérialisées. Vous DEVEZ utiliser les valeurs définies ci-dessous :

```
1  enum TypePizza
2  {
3      Regina = 1,
4      Margarita = 2,
5      Americaine = 4,
6      Fantasia = 8
7  };
8
9  enum TaillePizza
10 {
11     S = 1,
12     M = 2,
13     L = 4,
14     XL = 8,
15     XXL = 16
16 };
```

Vos communications feront transiter les pizzas sous la forme d'un type objet opaque de votre choix. Il DOIT être possible d'utiliser des opérations `pack` et `unpack` sur ce type pour y sérialiser ou en déserialiser des données.

Vous DEVEZ gérer les pizzas suivantes :

- La `Margarita` : Contient de la pâte, de la tomate et du gruyère. Cuit en 1 sec * multiplicateur.
- La `Regina` : Contient de la pâte, de la tomate, du gruyère, du jambon et des champignons. Cuit en 2 sec * multiplicateur.
- L'`Américaine` : Contient de la pâte, de la tomate, du gruyère et du steak. Cuit en 2 sec * multiplicateur.
- La `Fantasia` : Contient de la pâte, de la tomate, des aubergines, du fromage de chèvre et l'amour du chef. Cuit en 4 sec * multiplicateur.

`Vous demander très tôt comment représenter le temps peut vous en faire gagner ...`

`Vous arranger pour pouvoir ajouter de nouvelles pizzas très simplement (abstraction?) semble être un bonus simple a faire.`

Chapitre IV

Consignes générales

Vous êtes globalement libres de faire l'implémentation que vous voulez. Cependant, il y a quelques restrictions :

- Les seules et uniques fonctions de la `libc` autorisées sont celles qui encapsulent les appels système et qui n'ont pas d'équivalent en C++.
- Toute reponse à une problématique DOIT être une approche objet.
- Toute valeur passée par copie plutôt que par référence ou par pointeur doit être justifiée, sinon vous perdrez des points.
- Toute valeur non `const` passée en parametre doit être justifiée, sinon vous perdrez des points.
- Toute fonction membre ou méthode ne modifiant pas l'instance courante mais n'étant pas `const` doit être justifiée, sinon vous perdrez des points.
- Il n'existe pas de norme en `C++`. Cependant, tout code que nous jugerons illisible ou trop sale pourra être sanctionné arbitrairement. Soyez sérieux !
- Tout branchement supérieur a `(if ... else if ... else ...)` est INTERDIT ! Factorisez !
- Gardez un œil sur ce sujet régulièrement car il est susceptible d'être modifié.
- Nous attachons une grande importance à la qualité de nos sujets, donc si vous trouvez des fautes de frappe, d'orthographe ou des incohérences, merci de nous contacter à l'adresse koala@epitech.eu pour que nous puissions y remédier dans la journée.
- Vous pouvez joindre les auteurs par mail, voir l'en-tête.
- Le forum de l'intranet contiendra des informations et des réponses à vos questions. Assurez-vous d'abord que la réponse à votre question ne figure pas déjà sur le forum avant de contacter les auteurs.

Chapitre V

Consignes de rendu

Vous DEVEZ rendre votre projet sur le dépôt `SVN` mis à votre disposition par le `Koalab`. Vos dépôts seront ouverts au maximum 48h après la date de fin d'inscription au projet, intranet `Epitech` faisant foi. Il ne sera plus possible de s'inscrire au projet et de passer en soutenance une fois cette date depassée.

Vos dépôts seront fermés en écriture à l'heure exacte de la fin du projet, intranet `Epitech` faisant foi. Venir se plaindre que "sur ma montre il était pas 23h42" ne sert à rien. La seule heure valide à nos yeux est celle d'Epitech car le système est automatique.

Corollaire à loi de Murphy : "Si tu rends ton travail dans la dernière heure, quelque chose va mal se passer".

Seul le code présent sur votre dépôt sera évalué lors de la soutenance. La documentation relative aux dépôts fournis par le `Koalab` est fournie avec ce sujet.

Bon courage !