Table des matières

PCP – Travail pratique 00

Ce travail pratique vous amènera à mettre en pratique des techniques pour gérer la concurrence et la synchronisation dans un programme. Vous mandat pour ce travail pratique sera d'implémenter un schéma de conception typique du parallélisme, soit le **pipeline**.

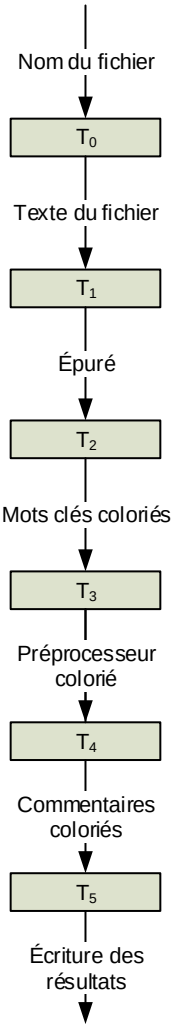
Concrètement, vous devrez lire plusieurs fichiers sources C++ et leur appliquer une séquence de transformations simples menant, en fin de parcours, à un fichier transformé.

Bien que le programme doive traiter du code source C++, vous pouvez rédiger le programme dans un autre langage de (voir Consignes générales, plus bas, pour des détails).

Structure du pipeline

Votre pipeline sera constitué des étapes suivantes :

- une transformation T_0 prendra chaque nom de fichier (extension `.h` ou `.cpp`) et produira son contenu sous la forme d'une chaîne de caractères;
- une transformation T_1 qui prendra chaque chaîne de caractères produite par T_0 et produira une chaîne équivalente mais épurée de certains métacaractères;
- une transformation T_2 qui prendra la chaîne de caractères produite par T_1 et produira une chaîne équivalente, à ceci près qu'elle contiendra une version coloriant les mots clés du langage[1];
- une transformation T_3 qui prendra la chaîne de caractères produite par T_2 et produira une chaîne équivalente, à ceci près qu'elle contiendra une version coloriant les commentaires;
- une transformation T_4 qui prendra la chaîne de caractères produite par T_3 et produira une chaîne équivalente, à ceci près qu'elle contiendra une version coloriant les directives du préprocesseur; et
- une transformation T_5 qui prendra la chaîne de caractères produite par T_4 et l'écrira dans un fichier portant le même nom que le fichier original mais avec extension différente, soit `.sequentiel.html` ou `.parallele.html` selon le cas.



Nous nommerons traitement séquentiel un ensemble de transformations $T_i, i \in (0..5)$ à partir d'une donnée source et menant vers une donnée de destination. Supposant d_0 la donnée originale, chaque T_i transformera d_i en d_{i+1}.

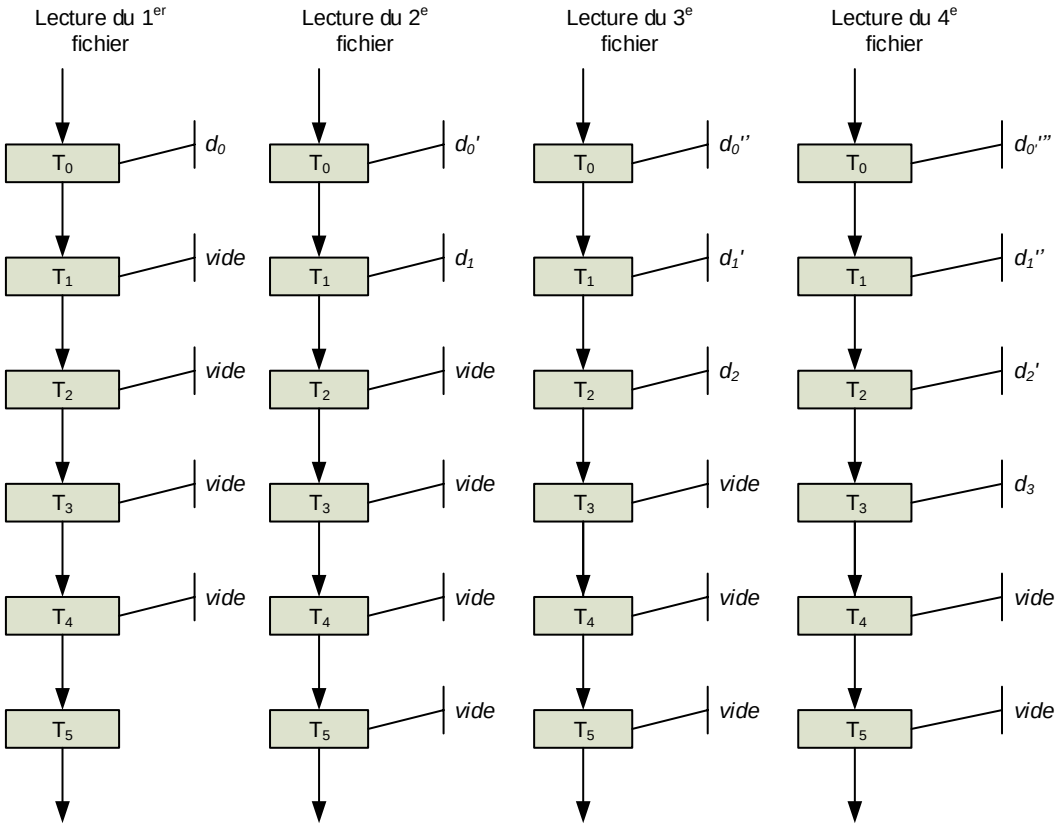
Un exemple d'un traitement séquentiel est donné par le schéma à droite. Si les transformations $T_i, i \in (0..5)$ sont de simples fonctions/ foncteurs/ λ, alors ce traitement séquentiel fonctionne, mais un seul de chaque T_i opère à la fois, ce qui est inefficace (sur une machine à plusieurs cœurs, du moins).

[1] Pour une liste à jour des mots clés du langage, voir http://en.cppreference.com/w/cpp/keyword

Votre travail sera de réaliser cette série de transformations sous la forme d'un pipeline, où, pour une certaine séquence d'opérations :

- le traitement réalisé par T_j dépend du traitement réalisé par $T_i, i < j$ à partir des mêmes données sources, en séquence, mais
- $\forall T_i, i > 0$ le traitement de T_i sur un donnée peut être réalisé en parallèle avec le traitement de T_{i-1} sur une même séquence;
- initialement, le pipeline sera vide;
- on alimentera T_0, qui consommera un premier d_0 pour générer d_1;
- pendant que T_0 transformera un deuxième d_0 (disons $d_0{}'$) en $d_1{}'$, T_1 pourra traiter d_1 pour générer d_2;
- ensuite, alors que T_0 transformera $d_0{}''$ en $d_1{}''$, T_1 transformera $d_1{}'$ en $d_2{}'$ alors que T_2 transformera d_2 en d_3; etc.

Tous les T_i s'exécuteront donc en parallèle (des *threads*), et le traitement se poursuivra tant que le pipeline n'aura pas été épuisé. L'illustration ci-dessous montre l'occupation du pipeline pendant le traitement des quatre premières données sources, de gauche à droite.



Consignes générales

Je vous recommande d'écrire initialement le traitement séquentiel, pour déboguer les trucs simples de la mécanique (consommation d'un fichier, remplacement des sous-chaînes dans une chaîne, ce genre de truc). Profitez-en pour mesurer la vitesse à l'exécution de votre programme en mode séquentiel, à titre indicatif.

Je suis tout à fait ouvert aux travaux dans divers langages de programmation, dans la mesure où vous me consultez au préalable si vous sortez du groupe commercial classique (C, C++, Java, C#). J'ai évidemment utilisé C++ pour ma version, que je vous présenterai une fois votre version livrée; nous pourrons échanger sur les techniques employées dans votre code comme dans le mien, critiquer, réfléchir collectivement, etc.

Ensuite, parallélisez le tout dans un pipeline en ajoutant les structures de données qui s'imposent. Si vous utilisez C++, il est possible que le `Mutex` portable et les zones de transit couvertes en classe vous soient utiles ici, au moins à titre d'inspiration, pour faciliter le passage de données d'un *thread* à l'autre, mais je vous invite à privilégier `std::mutex`, `std::lock_guard`, `std::thread` etc. si votre compilateur est suffisamment à jour, et à investiguer les approches équivalentes (blocs `synchronized` en Java, blocs `lock` en C#, etc.) dans le langage que vous aurez choisi si vous préférez autre chose que C++ (ce qui est tout à fait convenable).

Portez une attention particulière aux conditions d'arrêt : quand un *thread* donné se terminera-t-il exactement? Est-ce que tous les fichiers en entrée ont bel et bien été traités une fois l'exécution du programme terminée? Est-ce que les fichiers traités sont bel et bien complets?

Si vous êtes en équipe de trois personnes

Vous pouvez travailler seul(e) ou en équipe de deux ou trois. Les équipes de trois personnes auront toutefois une contrainte de qualité particulière à respecter, soit la coloration :

- des chaînes de caractères, monoligne (entre `"` et `"`) et multilignes (entre `R"(` et `)"`; pas besoin de traiter les autres cas); et
- des littéraux numériques. Dans ce cas, je ne demande que la couverture des littéraux usuels, soit les entiers et les nombres à virgule flottante primitifs.

Consignes techniques

Je vous recommande de vous familiariser avec les outils standards de votre langage de prédilection. Je vous donnerai en classe un aperçu des opérations sur des flux avec C++ dans le but de vous donner un coup de pouce pour démarrer.

Vous voudrez peut-être explorer les expressions régulières de votre langage favori pour réaliser ces tâches. En C++, l'en-tête standard à ce propos est `<regex>`. Cela dit, il n'est pas nécessaire d'y avoir recours pour en arriver à un bon résultat (ma version personnelle ne s'en sert pas).

Je vous recommande d'utiliser des balises `<span>` et des styles `CSS` pour réaliser le formatage du cours source. Aussi, pour préserver l'indentation d'origine, vous souhaiterez sans doute avoir recours à des balises `<pre>` (*Preformatted*).

Il est possible de spécifier ce que sont des styles `CSS` imbriqués dans d'autres styles `CSS`. Ceci vous facilitera peut-être l'existence quand vous voudrez traiter des mots clés placés entre commentaires, par exemple.

Quoi remettre

Je m'attends à un rapport mettant en relief les éléments suivants :

- l'identité et la contribution de chaque membre de l'équipe;
- une description de la stratégie choisie pour implémenter la partie parallèle (incluant la synchronisation). Cette partie doit inclure le détail des structures de données, idiomes de programmation et schémas de conception que vous aurez choisi d'appliquer pour y arriver (code à l'appui);
- le détail de votre stratégie de mesure pour la version séquentielle, et les métriques obtenues; de même que
- le détail de votre stratégie de mesure pour la version parallèle, et les métriques obtenues.

Bien que toutes les parties soient importantes (n'en négligez aucune!), je m'attarderai plus spécifiquement sur la section indiquant l'implémentation de la version parallèle, de même que sur les stratégies de mesure. La grille de correction sera pondérée en conséquence.

Format de la remise

La remise du travail peut être sous forme papier ou sous forme électronique. Dans le cas d'un document électronique, si vous souhaitez une rétroaction signifiante, je dois être en mesure de l'annoter à l'aide de Microsoft Word 2013 (je n'ai *vraiment* rien contre Open Office, mais ce n'est tout simplement pas ce que j'ai sur mes postes de travail). Il est possible mais un peu déplaisant d'annoter un PDF, alors si vous me livrez un document dans ce format, je limiterai probablement ma rétroaction à quelques remarques par courriel, sans plus.

Si vous souhaitez que je teste votre travail chez moi, tenez compte des spécifications techniques de mon ordinateur[2]. Vous devez aussi vous assurer, par souci d'intégrité, de pouvoir me produire les sources de tous vos modules sur demande et de pouvoir me donner une démonstration sur place, au campus où vous suivez ce cours et un jour où je suis présent, si j'en exprime le désir (si je le fais, je vous donnerai quelques jours d'avis).

Ne me rendez pas la vie trop compliquée je vous prie : il est nécessaire que je puisse installer et désinstaller le tout aussi simplement que possible, que ce soit par un installateur automatisé ou à l'aide de consignes simples et claires. Je corrige énormément de travaux chaque session; il est donc impératif que vous me rendiez la tâche facile à l'installation et que mon ordinateur soit *propre* après la désinstallation.

Tout document remis devra présenter un français de qualité. Le code devra utiliser une police de caractères non proportionnelle (quelque chose comme Courier New ou Lucida Console par exemple) pour l'impression de programmes ou d'extraits de programmes, et indentez à l'aide d'espaces (les tabulations ont une influence qui varie d'un éditeur à l'autre, ce qui complique la présentation du code et sa lisibilité). Ces polices conviennent bien à un tel usage, clarifiant l'indentation appliquée[3]. D'ailleurs, même si vous utilisez des outils de génération automatique de code[4], ce code devra être indenté de manière lisible, sinon il vous sera retourné sans que je ne l'aie lu.

Si vous choisissez de faire une remise en format imprimé, cette remise devra être faite le **lundi 12 juin en début de séance**. Si vous préférez livrer une version électronique par courriel à Patrice.Roy@USherbrooke.ca, alors cette remise devra être faite le **lundi 12 juin à 17 h**.

---

[2] PC au processeur Intel à quatre cœurs cadencés à 2,6 GHz, 7,6 Go RAM, *Microsoft Windows* 7, ce qui devrait suffire.

[3] Sur le site de l'enseignant, voir les articles http://h-deb.clg.qc.ca/Sujets/TrucsScouts/Police-pour-code.html et http://h-deb.clg.qc.ca/Sujets/TrucsScouts/Eviter-Tabulations.html pour plus de détails quant aux exigences en lien avec la présentation du code.

[4] Je pense en particulier aux *aficionados* de Delphi ici. http://h-deb.clg.qc.ca/Sujets/Developpement/Pratique-programmation.html#indentation pour des idées. L'important est d'être conséquent à l'interne dans un programme donné, et de viser la lisibilité.