UNIVERSITÉ DE SHERBROOKE

Faculté des sciences
Département d'informatique

IFT 630

Processus concurrents et parallélisme

Plan de cours

Été 2017

Enseignant : Patrice Roy
Courriel : Patrice.Roy@USherbrooke.ca
Local : D4-1010-14
Téléphone : poste 65561
Site Web pour le cours : http://h-deb.clg.qc.ca/UdeS/PCP/

Note : Privilégiez (de loin!) le courriel (réponses rapides, enseignant matinal)

Horaire : Lundi, 9 h 30 à 12 h 20 salle D3-2040

Description officielle de l'activité pédagogique[1].

Objectifs	Se familiariser avec les concepts de la programmation concurrente. Apprendre à résoudre des problèmes en se servant de la programmation concurrente.
Contenu	Approfondissement des concepts de processus et de fil d'exécution (*thread*). Synchronisation centralisée ou répartie : problématique, techniques et erreurs typiques. Communication pour systèmes centralisés et pour systèmes répartis : problématique et techniques de mise en œuvre. Architecture des systèmes de processus communicants (client/ serveur, P2P, grappes, grilles, …). Coordination de processus.
Crédits	3
Organisation	3 heures d'exposé magistral par semaine 6 heures de travail personnel par semaine
Préalable	IFT 320 ou l'équivalent

[1] Tiré de http://www.usherbrooke.ca/fiches-cours/ift630

Avant d'entreprendre ce cours…

Le cours présume chez l'étudiant(e) une connaissance de la programmation procédurale et orientée objet (OO), de même qu'une solide base avec les langages C, C++ et Java. Avoir été en contact avec des principes de programmation fonctionnelle est un atout.

Aussi, bien que ce cours ne soit pas strictement un cours de programmation de systèmes à haute performance, cet élément ne nous sera pas étranger, étant lié de près avec le propos du cours (à tout le moins sur une base historique).

Il est hautement probable que ces différents *a priori* soient acquis par chacune et chacun d'entre vous, mais personne n'est parfait. Pour les travaux, envisagez fortement vous adjoindre des coéquipières et des coéquipiers ayant des atouts qui seront *complémentaires* aux vôtres. Cela activera la discussion.

Oui, mais j'ai déjà vu cela…

Il y aura inévitablement des éléments de votre formation, dans ce cours comme ailleurs, qui recouperont votre vécu académique (ou professionnel, dû à des stages ou à une expérience de travail antérieure).

De même, dû à des variations selon les parcours académiques et professionnels de chacun(e), certain(e)s parmi vous trouveront évident ce que d'autres estimeront lourd (ou cauchemardesque).

Sachant cela, mettons de l'eau dans notre vin (pas trop, quand même, pour qu'il garde son cachet) et soyons tolérant(e)s les un(e)s envers les autres.

Ce cours demande avant tout une certaine aisance en programmation et une capacité d'analyse rigoureuse et méthodique. Des connaissances en systèmes d'exploitation, réseaux et architecture des ordinateurs seront utiles.

Mise en contexte

Longtemps relégués au monde de la recherche, le parallélisme et la concurrence ont intégré les préoccupations des programmeuses et des programmeurs au sens large durant les années '90, alors que la multiprogrammation à l'aide de plusieurs fils d'exécution concurrents (le *multithreading*) est devenue possible sur les systèmes d'exploitation grand public, en particulier depuis que les processeurs à plusieurs cœurs ont fait leur entrée dans les foyers de tous et chacun.

Il est maintenant bien connu que la soi-disant *Loi de Moore*, qui prédisait un accroissement régulier de la puissance des processeurs, a « frappé un mur » au début des années 2000[2]. Depuis, les processeurs des ordinateurs que met l'industrie à notre disposition, plutôt que d'avoir un processeur dont la vitesse d'horloge est de plus en plus haute, sont munis d'un nombre de plus en plus grand de processeurs (ou de cœurs). Plusieurs algorithmes traditionnels ne tirent pas naturellement profit de cette nouvelle réalité; un peu comme la programmation fonctionnelle ou la programmation orientée objet, *le parallélisme et la concurrence demandent une manière différente d'appréhender les problèmes*.

[2] L'un des articles les plus connus discutant de ce phénomène est probablement *The Free Lunch is Over*, par Herb Sutter, reproduit sur http://www.gotw.ca/publications/concurrency-ddj.htm

Objectif général

Le cours IFT630 vise à développer chez l'étudiant(e) une ***connaissance appliquée*** de la conception (et du développement) de systèmes parallèles et concurrents, à l'aide de plusieurs processus ou de plusieurs *threads*.

Conséquemment, ce cours mettra à la fois l'accent sur les connaissances permettant de comprendre ce que sont ces systèmes, sur les techniques et concepts propres au développement de tels systèmes, ainsi que sur le développement du *savoir-faire* requis pour mettre la main à la pâte ou pour encadrer une équipe de développement faisant face aux problèmes caractéristiques d'un système parallèle et concurrent.

L'approche appliquée, même pour un cours ouvert aux étudiant(e)s de 2e cycle universitaire, vous permettra d'assurer votre crédibilité face à vos pairs et à celles et ceux qui travailleront pour vous. Nous voulons que l'étudiante et l'étudiant ayant réussi IFT630 soit capable de voir ce qui permettra (ou non) de mettre au point un système impliquant de la concurrence et du parallélisme; de résoudre les problèmes propres à ces systèmes; de comprendre et de savoir tirer profit des outils de synchronisation, de débattre avec ses pairs des approches qui rapprocheront son projet et son équipe du succès; de définir les contraintes à respecter et de mesurer quantitativement l'atteinte des résultats souhaités.

Dans ce cours, l'étudiant(e) se familiarisera avec les concepts de programmation concurrente, ceux utilisés dans les systèmes répartis, ainsi que les notions de programmation fiable avec reprise. À la fin du cours, l'étudiant sera capable de résoudre des problèmes en se servant de la programmation concurrente. En particulier, elle/ il saura transformer un programme séquentiel en un programme parallèle à l'aide des différents outils de synchronisation et de communication disponibles lorsque cela s'avérera approprié. La résilience, la gestion des erreurs et la reprise après interruption d'un élément du programme seront des considérations récurrentes.

Objectifs spécifiques

Au terme du cours, l'étudiant(e) sera capable :

{1} de maîtriser la terminologie et les concepts sous-jacents à la programmation concurrente;
{2} de comprendre le fonctionnement des algorithmes de synchronisation;
{3} de développer des programmes concurrents et de les synchroniser avec les divers outils de synchronisation;
{4} de comprendre le fonctionnement de diverses structures de calcul sur un réseau d'ordinateurs;
{5} de comprendre et d'appliquer différentes techniques de *parallélisation* de programmes séquentiels;
{6} de développer des programmes fonctionnant sur un réseau d'ordinateurs ou sur une grappe de calcul;
{7} de comprendre le fonctionnement des systèmes répartis;
{8} de comprendre les principes de fiabilité et de performance associés à ces systèmes.

Nous mettrons donc l'accent sur du code *parallélisable*, sécuritaire, portable, et efficace. Lorsque cela sera possible, les techniques vues en classe pourront être adaptées à d'autres plateformes.

Optiques retenues

Certains langages de programmation sont intimement liés au parallélisme et à la gestion de problèmes de concurrence. Par exemple :

- Java et C# incluent des mots clés[3], des interfaces (p. ex. : `Runnable`) et des classes (p. ex. : `Thread`) voués directement à adresser des questions de parallélisme et de concurrence, et offre à même la classe racine de sa hiérarchie des outils de synchronisation (`wait()`, `notify()`, `notifyAll()`);
- Erlang propose un monde de (typiquement petits) processus disjoints qui ne partagent aucun état et qui communiquent strictement par messages;
- des langages fonctionnels comme Haskell favorisent le recours à des entités immuables, qui peuvent être manipulés de manière concurrente de façon bien moins risquée que ne le sont les objets mutables;
- depuis C++ `11`, le langage C++ offre une gamme d'outils portables axés sur la résolution de problèmes à l'aide de parallélisme et de concurrence[4]. Notez qu'en pratique, les solutions de niveau « bibliothèque » ne suffisent pas[5];
- et ainsi de suite. Rares sont aujourd'hui les langages de programmation et les plateformes qui n'offrent ni soutien au parallélisme et à la concurrence, ni métaphore permettant d'aborder ces thématiques.

Pour des raisons pratiques, l'essentiel des exemples proposés en classe sera en C/ C++ (entre autres raisons, il s'agit du langage le plus utile au professeur dans l'exercice de ses fonctions), mais des exemples et des illustrations seront proposés aussi dans d'autres langages, pour le plaisir de la chose.

---

[3] Par exemple `synchronized` en Java et `lock` en C# pour synchroniser l'accès à une section de code, les variables atomiques, les mémoires transactionnelles, etc.

[4] Par exemple `std::thread` et `std::future`, de même que de la mémoire locale par *thread*.

[5] À ce sujet, voir le texte *Threads Cannot be Implemented as a Library*, par Hans Boehm en `2004` : http://www.hpl.hp.com/techreports/2004/HPL-2004-209.pdf

Organisation du cours

Idéalement étalé sur environ 15 semaines, mais condensé sur 13 semaines dû aux affres du calendrier, le cours sera exigeant du point de vue de la charge de travail (certaines idées s'assimilent mieux par la pratique que par osmose).

Une discussion sera tenue en classe pour déterminer comment seront récupérées les heures qu'aura dévorées ce vilain calendrier.

Le cours comprenant un volet technique important, l'étudiant(e) devra mettre la main à la pâte et y aller d'un effort pratique.

Nous aborderons les systèmes parallèles et concurrents sous plusieurs angles (concepts, façons de faire, comparatifs, technologiques), ce qui pourra donner une impression de désorganisation par moment (ne vous en faites pas, cependant), mais permettra un tour d'horizon plus complet.

Planification des séances

La planification ci-dessous se veut un aperçu plus précis du cours proposé. Cette planification se veut souple : selon le rythme que nous parviendrons à maintenir, des débordements seront possibles d'un cours à l'autre. Il est possible aussi que les questions en classe nous amènent à changer nos plans à l'occasion.

- La colonne **Thématique** offre un titre global pour les divers éléments de contenu.
- La colonne **Contenu** liste brièvement les éléments de contenu prévus pour chaque thématique.
- La colonne **Heures** indique le temps prévu (environ) pour chaque thématique, en heures.
- La colonne **Objectifs** indique auxquels des objectifs spécifiques du cours (plus haut) chaque thématique sera *principalement* arrimée.

La planification utilisée ici est inspirée du plan de cours de **Gabriel Girard**, qui a offert ce cours plusieurs fois. Sa mise en application sera toutefois **fortement** teintée par la personnalité et les caractéristiques de votre enseignant : nous explorerons entre autres des outils récents pour faciliter le développement de systèmes parallèles, et pour tirer profit d'elles, nous couvrirons parfois des techniques de programmation auxiliaires. C'est pourquoi j'ai adjoint à la planification une deuxième « carte » listant les thèmes que je couvre habituellement en classe dans ce cours,

Considérez donc la planification comme étant indicative de ce qui s'en vient, car nous couvrirons essentiellement tout ce qui y apparaît, mais sachez que l'ordre proposé peut être bouleversé à quelques (!) reprises et que plusieurs (!) ajouts y seront faits.

Pour ce qui est des dates clés :

- les séances en classe s'étendront du 1er mai au 14 août inclusivement;
- la date de l'examen final reste à déterminer (la Faculté déterminant celle-ci), mais se situera entre le 8 août et le 18 août inclusivement;
- pour ce qui est des dates de publication des consignes des travaux pratiques, tout comme pour ce qui a trait à leurs dates de remise, le calendrier d'été est particulièrement pervers pour qui enseigne le lundi, et une discussion en classe sera nécessaire avant de fixer ces paramètres.

Un semainier à jour sera proposé sur le site du cours, dont vous trouverez les coordonnées dans la section ***Médiagraphie***, plus bas, de même que sur la page frontispice du présent document.

La planification « classique », en détail, va comme suit.

Thématique	Contenu	Heures	Objectifs
Notions de base	Processus : concept, opérations et relations Fils d'exécution (*threads*) : structure, terminologie, exemples Noyau : fonction, primitives, structure, implantation	2	1
Concurrence et parallélisme	Définitions, types de concurrences, opérations pour la concurrence (fork, join, cobegin/ coend, etc), modélisation de la concurrence	2	1
Synchronisation	Synchronisation, communication Solutions avec attente active : Dekker, Dijkstra, Peterson, etc. Sémaphores : principes, implantation et utilisation Architectures multi-cœurs, atomicité, cohérence séquentielle	8	2, 3
Programmation parallèle	Principes, régions critiques conditionnelles, moniteurs, moniteurs étendus, expressions de chemins, etc.	8	2, 3
Communication inter-processus	Types de communication : mémoire commune; passages de messages Identification des interlocuteurs, synchronisation, protection et implantation	4	2, 3
Systèmes d'exploitation répartis	Systèmes de fichiers répartis, gestion de l'UCT, gestion de la mémoire, synchronisation et communication	10	4, 7
Calcul parallèle	Problématiques et approches; modèles de parallélisation: algorithmique, architecture, communication et synchronisation; outils de programmation	3	4, 5, 6
Algorithmes parallèles	Modèles pour algorithmique parallèle : graphes orientés acycliques (DAG), PRAM, modèle réseau, Hypercube, etc. Performance et complexité des communications	3	4, 5, 6
Fiabilité	Éviter les pannes Détection des erreurs Traitement des pannes Reprise avant et arrière	2	8
« Performance »	Introduction, mesures de « performance », techniques d'évaluation, charge de travail	2	8

En pratique toutefois, l'expérience montre que les thématiques vues concrètement sont les suivantes. Il y a un fort recoupement avec la planification « classique », mais aussi beaucoup d'éléments très contemporains. Notez que ce qui suit groupe les éléments de contenu par thématiques, mais que je ne couvre généralement pas chaque thématique comme un bloc indivise (je tends à y aller de manière plus organique) :

Thématique	Contenu
Programmation parallèle et concurrente	Avec C++, mais aussi avec Java, C#, Erlang, Go, ...
Unités d'exécution	Threads, processus, fibres, etc.
Concepts et enjeux d'ordre général	Algorithmique, portabilité, copie-mouvement-partage, garanties de progression, linéarisabilité, modèles mémoire, résilience et tolérance aux pannes, loi d'Amdahl, loi de Gustafson
Synchronisation	Avec verrous (mutex, sémaphores, variables conditionnelles, etc.), sans verrous, volatilité, atomicité (compare/exchange), pointeurs intelligents (pointeurs intelligents atomiques), loquets (*Latches*), clôtures (*Barriers*), mémoire transactionnelle
Dangers	Conditions de course (*Data Race*, TOCTTOU), interblocage (*deadlock*, *livelock*)
Pratiques et schémas de conception	Modèle fork/join, moniteurs, RAII, valeurs synchronisées, futures et promesses, continuations, coroutines, objets autonomes, when_all(), when_any(), regroupements, exécuteurs, pipeline, map/reduce, etc.
Optimisation	Antémémoire (*Cache*), faux-partage, réduction du blocage, partie sérielle vs partie parallèle
Infrastructures (survols)	MPI, OpenMP, PPL, TBB, AMP, approche RPC
Classiques	Philosophes, algorithmes avec attente active (Peterson, Dekker, Dijkstra, Lamport)

Si le temps le permet, j'ajouterai un peu d'informations et d'exemples sur la programmation parallèle à l'aide du `GPU` (pas de promesses, cela dit; de l'espoir ☺).

Approche pédagogique préconisée

Pour favoriser l'intégration des nombreux concepts au menu, nous suivrons essentiellement le modèle suivant :

- exposés magistraux en classe, où les étudiant(e)s sont *fortement* encouragés à contribuer par leurs questions et commentaires;
- travaux pratiques et exercices à teneur formative, qui permettront aux étudiant(e)s de mesurer concrètement leur compréhension de la matière explorée, et qui pourront être corrigés par le professeur ou autocorrigés à l'aide d'une grille de vérification, selon le cas;
- travaux pratiques à teneur sommative, évalués par le professeur en fonction des mêmes critères que ceux appliqués dans le cadre des activités formatives;
- des questions de réflexion (et parfois à saveur technique) sur une base quasi hebdomadaire; et
- un contrôle théorique récapitulatif, en toute fin de parcours, vérifiant formellement l'atteinte des objectifs.

Les questions et contrôles présumeront que chaque membre d'une équipe a contribué activement à la réalisation de chacun des travaux pratiques, et a bien compris les implications philosophiques et techniques de ces travaux.

Évaluation des apprentissages

Les évaluations sommatives seront réparties et pondérées comme suit.

Minitests (40%)

De petits tests, souvent d'une seule question, auront lieu sur une base relativement régulière, soit une par semaine, et ce la plupart des semaines. Seuls les huit mieux réussis pour chaque étudiant(e) seront comptabilisés.

Chaque question portera sur le thème de la semaine précédente, parfois des deux semaines précédentes. Les questions seront surtout orientées sur la réflexion, mais la technique à proprement dit s'y glissera à l'occasion.

Le poids de chacun de ces petits tests sera 5% de la note finale. En général, le temps alloué pour y répondre sera d'environ 15 minutes, au début du cours.

Travaux pratiques (30%)

Trois travaux pratiques, chacun ayant un poids de 10% sur la note finale, devront être réalisés en cours de session. Le détail de ces travaux et les dates de livraison seront indiqués ultérieurement, mais seront en conformité avec les règles facultaires et départementales.

Examen (30%)

Un examen final récapitulatif valant 30% de la note finale aura lieu lors de la dernière séance de la session.

Plagiat

Un document dont le texte et la structure se rapporte à des textes intégraux tirés d'un livre, d'une publication scientifique ou même d'un site Internet, doit être référencé adéquatement. Lors de la correction de tout travail individuel ou de groupe une attention spéciale sera portée au plagiat, défini dans le Règlement des études comme « le fait, dans une activité pédagogique évaluée, de faire passer indûment pour siens des passages ou des idées tirés de l'œuvre d'autrui ». Le cas échéant, le plagiat est un délit qui contrevient à l'article 8.1.2 du Règlement des études : « tout acte ou manœuvre visant à tromper quant au rendement scolaire ou quant à la réussite d'une exigence relative à une activité pédagogique ». À titre de sanction disciplinaire, les mesures suivantes peuvent être imposées : a) l'obligation de reprendre un travail, un examen ou une activité pédagogique et b) l'attribution de la note E ou de la note 0 pour un travail, un examen ou une activité évaluée. Tout travail suspecté de plagiat sera référé au Secrétaire de la Faculté des sciences.

Médiagraphie

Des notes de cours seront mises à votre disposition, et devraient être votre référence principale pour la session qui s'annonce.

Le site Web du cours devrait être, avec les notes de cours en tant que telles, votre référence principale :

http://h-deb.clg.qc.ca/UdeS/PCP/

Quelques textes sur le sujet au cœur de nos préoccupations dans ce cours sont regroupés sur :

http://h-deb.clg.qc.ca/Sujets/Parallelisme/

Un ensemble toujours croissant de liens portant sur la multiprogrammation peut être consulté ici :

http://h-deb.clg.qc.ca/Liens/Multiprogrammation--Liens.html