Parallélisme à base de thread

(PBT 2022-2023)

TD1

Modèle de programmation Pthread

marc.perache@cea.fr
hugo.taboada@cea.fr

Click here or scan to download TD'files from pcloud link.

Le but de ce TD est de se familiariser avec l'API Pthread pour ensuite comprendre et évaluer différentes bibliothèques de threads. L'utilisation des bibliothèques de threads est l'approche classique pour concevoir une application mémoire partagée. Le niveau auquel est implémenté l'ordonnanceur constitue la caractéristique intrinsèque la plus importante des bibliothèques de threads. Les threads peuvent être gérés dans une bibliothèque entièrement en espace utilisateur, ou bien au sein même du noyau avec l'aide éventuelle d'une bibliothèque en espace utilisateur. Il existe donc trois types de bibliothèques de threads :

— Les bibliothèques de niveau utilisateur.
— Les bibliothèques de niveau système.
— Les bibliothèques mixtes.

Dans ce TD, nous nous intéresserons aux bibliothèques de threads de niveau utilisateur et de niveau système. Les prochains TDs consisterons à implémenter partiellement une bibliothèque mixte. Ce TD n'est pas noté mais les résultats vous servirons pour les prochains TDs qui seront notés.

I Découverte de l'API Pthread

Nous vous proposons une série d'exercice pour découvrir l'API pthread.

1 Hello World !

Les threads POSIX sont créés à partir de la fonction *pthread_create*. L'attente de la terminaison d'un thread ainsi créé se fait à partir de la fonction *pthread_join*

Q.1: Écrire un programme créant un nombre de threads spécifiés en argument, chaque thread devant afficher une chaîne de caractères (par exemple « Hello world ! ») sur la sortie standard.

Modifier le programme précédent pour que chaque thread affiche un entier qui lui est passé en paramètre égal à l'ordre dans lequel il a été créé : le premier thread créé affiche 1, le second 2, etc. La chaîne devra être passée en argument, ainsi que l'identifiant du thread.

Q.2: En créant la chaîne (« Hello... » + identifiant) **avant de la passer** au thread à créer (utilisez la fonction sprintf)

Q.3: **En passant la chaîne et l'identifiant séparément** au thread à créer ; chaque thread se chargera d'appeler printf luimême.

2 multithread-unsafe

Tout programme parallèle doit manier avec précaution des ressources partagées. Dans le cas de la programmation « multithreadée », les variables globales sont des variables communes et visibles de tous les threads. Accéder en écriture de façon concurrente à ce type de variables sans protection peut rendre le programme faux.

Ouvrir le fichier mt_unsafe/exemple.c et étudiez le code.

Q.4: Compilez le code une fois et exécutez le plusieurs fois. Que remarquez-vous ? Expliquer le problème.

Q.5: Rajoutez le mot-clé **volatile** devant la déclaration de count. Que cela signifie t'il ? Cela est-il suffisant pour rendre le programme correct ?

Q.6: Comment résoudre le problème ?

3 Recherche parallèle du max d'un tableau

Ouvrir et étudier les sources du fichier mt_max_tab/max_tab.c. Ce programme prend en argument 2 entiers : le nombre d'éléments d'un tableau d'entiers et un nombre de threads à créer (max_tab.exe nelts nthreads).

Pour l'instant, le programme ne créée aucun thread et recherche en séquentiel le maximum du tableau d'entiers (entiers compris entre 1 et 1000 et tirés aléatoirement). Travail à effectuer : rechercher le maximum du tableau en créant nthreads et en affectant ce maximum dans la variable maxv_mt (déjà déclarée dans les sources). Implémentez 2 méthodes :

Q.7: Chaque thread recherche la valeur maximale dans une souspartie du tableau, et renvoie cette valeur. Le thread principal recherche le maximum parmi les valeurs retournées.

Q.8: Chaque thread met à jour une variable partagée entre les threads contenant le maximum.

4 Les conditions

Nous considérons 2 threads A et B. A doit afficher une valeur calculée par luimême et ainsi qu'une valeur calculée par B : les 2 valeurs affichées doivent être identiques. Le fichier condition.c essaie d'implémenter ce comportement mais n'est pas correct.

Q.9: Après compilation, décrire le comportement obtenu (effectuer plusieurs essais) et expliquer le problème.

Pour corriger le programme, 2 implémentations sont demandées :

Q.10: En utilisant les variables mutex et conditionnelle déjà présentes. En réécrivant le programme avec les sémaphores.

Q.11: En réécrivant le programme avec les sémaphores.

On souhaite désormais effectuer ce travail en boucle tel que dans boucle.c.

Q.12: Rajouter les synchronisations nécessaires pour obtenir le résultat correct.

5 Recherche parallèle du max d'un tableau avec les sémaphores

Q.13: Reprenez l'exercice de la section 3 en remplaçant l'utilisation du mutex par l'utilisation d'un semaphore.

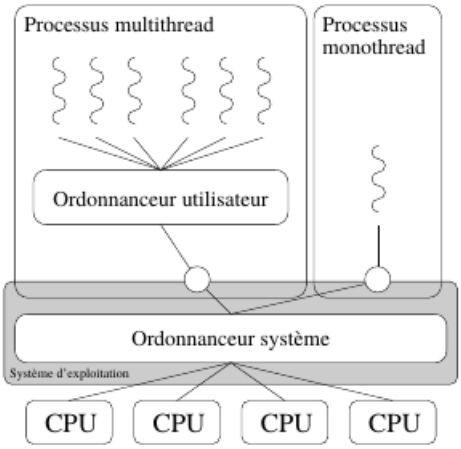
6 Barrière multithread

La bibliothèque pthread fournit un objet *pthread_barrier_t* permettant de définir un point de synchronisation entre threads. On se propose ici d'implémenter nousmême une fonctionnalité similaire en utlisant des variables de condition.

Q.14: Analyser le fichier source barriere.c et le compiler. Implémenter la structure de barrière ainsi que les fonctions d'initialisation et d'attente.

II Fonctionnement d'une bibliothèque de niveau utilisateur : GnuPth

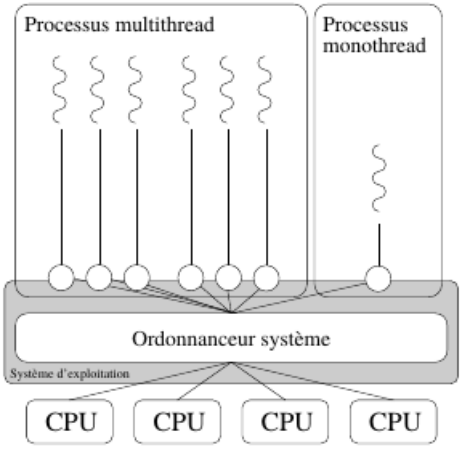
Les bibliothèques de threads de niveau utilisateur telle que GnuPth sont les premières apparues sur les systèmes d'exploitation, sans doute parce qu'il est plus facile de travailler en espace utilisateur que de modifier les systèmes d'exploitation. Il est facile de les adapter à un besoin particulier et, n'ayant pas à dialoguer avec le système d'exploitation, elles sont très efficaces. Le schéma suivant illustre ce type de bibliothèques :



Q.15: D'après vous, quels sont les caractéristiques de ce type de bibliothèque ? (performances, flexibilité, SMP, Appels systèmes bloquants)

III Fonctionnement d'une bibliothèque de niveau système : Pthread

À l'opposé des bibliothèques de niveau utilisateur, les bibliothèques de niveau système sont généralement dédiées à un système particulier. Ces bibliothèques utilisent l'ordonnanceur du système pour gérer leurs threads ; une étroite intégration entre le système et la bibliothèque s'avère donc nécessaire. Les bibliothèques fournies avec les systèmes d'exploitation récents sont généralement des bibliothèques de niveau système (plus rarement à deux niveaux). Le schéma suivant illustre ce type de bibliothèques :

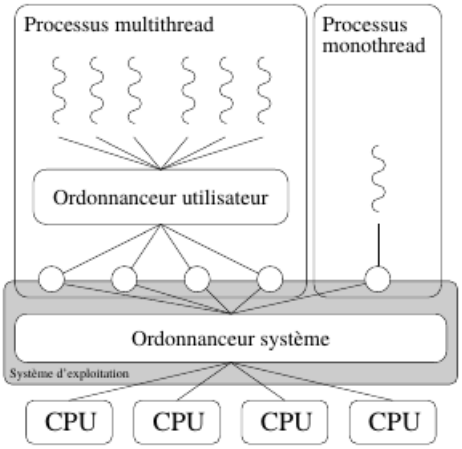


Q.16: D'après vous, quels sont les caractéristiques de ce type de bibliothèque ? (performances, flexibilité, SMP, Appels systèmes bloquants)

IV Fonctionnement d'une bibliothèque mixte : mthread

Les bibliothèque mixte ont l'avantage de garder un ordonnanceur en espace utilisateur et donc d'être efficaces et flexibles. Toutefois, la nécessité de faire co-

opérer deux ordonnanceurs rend l'écriture de telles bibliothèques plus difficile. Le schéma suivant illustre ce type de bibliothèques :



Q.17: D'après vous, quels sont les caractéristiques de ce type de bibliothèque ? (performances, flexibilité, SMP, Appels systèmes bloquants)

V Évaluation des bibliothèques Pthread et GnuPth sur SMP (processeur multi-coeur)

Nous vous proposons d'évaluer deux bibliothèques afin de pouvoir valider les réponses précédentes.

1 Évaluation des capacitées sur architectures SMP

Q.18: Décrire un moyen qui permet de caractériser si une bibliothèque utilise correctement tous les processeurs/coeurs à sa disposition.

Q.19: Évaluer les caractéristiques SMP de la bibliothèque GnuPth.

Q.20: Évaluer les caractéristiques SMP de la bibliothèque Pthread.

2 Évaluation des capacitées envers les appels bloquants

Q.21: Trouver un appel système bloquant permettant d'évaluer la capacité de gestion des appels bloquants des bibliothèques de threads.

Q.22: Évaluer les caractéristiques envers les appels bloquants de la bibliothèque GnuPth.

Q.23: Évaluer les caractéristiques envers les appels bloquants de la bibliothèque Pthread.

$$\frac{Perf_{Predicted} - Perf_{Run}}{Perf_{Run}} * 100$$