GRO 300 - Examen formatif

Processus, fils, et parallélisation du travail (GRO 300-1)

Question 1

Quelle est la relation entre processus et fils ?

Réponse : Il s'agit d'une relation 1 à N. Les fils appartiennent aux processus. Un processus a au moins un fil (le principal). Un fil doit appartenir à un et un seul processus.

Question 2

Est-ce qu'il existe une limite au nombre de fils pouvant être exécutés simultanément ? Si oui, laquelle ? Sinon, pourquoi ? Détaillez votre raisonnement.

Réponse : D'un point de vue matériel, le nombre de fils est limité par le nombre de cœurs le système possède, et le nombre de fils simultanés que ceux-ci peuvent gérer. Or, le système peut, par séquencement temporel, augmenter la limite apparente. Par contre, il existe tout de même une limite au nombre de fils. Premièrement, nous ne disposons pas d'une mémoire illimitée, nous ne pouvons donc pas gérer un nombre illimité de fils d'exécution. Deuxièmement, il y a un maximum raisonnable à ne pas franchir : plus on partage le temps du processeur, plus le temps de réponse pour chaque tâche augmente : chaque tâche sera étalée sur une plus longue durée de temps. On accepterait tout de même une partie des points à une réponse comme "Non, car le partage de temps permet d'exécuter plus de fils qu'il y a de processeurs physiques."

Question 3

Nommez un inconvénient de la communication inter-processus par rapport à la communication inter-fils.

Réponse : Plus lourd, dépend de services du système d'exploitation qui peuvent varier d'une plateforme à l'autre.

Question 4

Est-ce qu'un fil d'exécution peut avoir un espace mémoire bien à lui ? Donnez un exemple en C++.

Réponse : Oui, par exemple pour stocker la pile (mémoire locale aux fonctions en C et autres langages similaires). Un exemple en C++ pourrait être :

```
int val_globale = 0;      // Cette variable est globale et
                          // partagée par les fils.
void thread_func()
{
  int val_priv = 0;       // Cette variable est privée à
                          // chaque fil.
}

int main (int argc, char** argv)
{
  std::thread t1(thread_func);
  std::thread t2(thread_func);

  t1.join();
  t2.join();
}
```

Pas besoin de la syntaxe exacte, seulement de pointer le fait qu'une variable locale à une fonction, même si plusieurs fils l'exécutent simultanément, reste privée à chaque fil. Vous pouviez aussi donner l'exemple du compteur dans les boucles `for` (la variable i) dans la plupart des boucles vues pendant les exercices du laboratoire et la problématique.

Question 5

Qu'est-ce qui peut compliquer la parallélisation du traitement de plusieurs sous-tâches ?

Réponse : Les interdépendances entre les sous-tâches. Par exemple, si la sous-tâche 2 a besoin de la réponse de la sous-tâche 1, alors elle devra nécessairement attendre que la sous-tâche 1 termine son travail, ce qui limite le débit de l'application.

Question 6

Vous êtes ingénieur dans une compagnie qui souhaite utiliser un robot manipulateur pour détecter des pièces défaillantes sur un convoyeur. Un de vos collègues a déjà développé un prototype du système de traitement qui se divise en quatre parties séquentielles :

1. Le rehaussement du contraste de l'image et la suppression de la couleur du convoyeur, pour qu'il ne reste que les objets à évaluer dans l'image. Le traitement est isolé par pixel, c.-à-d. que, pour chaque pixel, le traitement ne dépend pas des pixels voisins ;
2. La détection de points d'intérêts distinctifs dans toutes l'image, comme des coins ou des traits. Cette étape consiste à analyser chaque pixel avec ses voisins de l'image d'origine dans un rayon de 15 pixels. L'analyse par pixel est séquentielle ;
3. Le regroupement des points d'intérêts en objets, en supposant que les points d'intérêts près les uns des autres appartiennent au même objet. La tâche est itérative : Pour *N* points d'intérêts, on connait d'abord le nombre d'objets à détecter (*P*, par exemple 3). On sélectionne *P* points au hasard dans l'image qu'on considère être les centres de ces objets. Ensuite, pour chaque point *n* dans l'ensemble *N*, on trouve le point *p* de l'ensemble *P* qui lui est le plus proche, et on associe ce point à l'objet *p*. Finalement, pour chaque objet *p,* on calcule le barycentre des points (c.-à-d. la moyenne des coordonnées, ce qui devrait correspondre au centre de l'objet) qu'ils lui sont associés, ce qui devient *p',* le nouveau centre de l'objet *p*. On répète le processus d'association des points *N* au objets *P* jusqu'à ce que les ensembles ne changent plus.
4. L'analyse statistique de l'appartenance des points d'intérêts aux objets. Les objets ne possédant pas les bons signes distinctifs sont considérés comme étant défaillants.

Vous avez la responsabilité d'optimiser le prototype en place et d'augmenter le débit du système de traitement de données, l'objectif étant de détecter le plus d'objets défaillants à la minute. On vous suggère d'utiliser la programmation concurrente pour mieux exploiter les capacités de l'ordinateur embarqué de la chaîne de montage.

Répondez aux questions suivantes comme si le système n'avait qu'à traiter une seule image :

a) Peut-on faire une séparation des données en parallèle dans les parties 1 et 2 ? Si oui, comment ? Sinon, pourquoi ?

Réponse : Oui aux deux, puisque chaque pixel est individuel. L'analyse des voisins se fait sur l'image d'entrée dans la partie 2. On dépend des voisins, mais pas de leur traitement.

b) Suggérez au moins deux façons de paralléliser le traitement de l'étape 3.

Réponse : Plusieurs options : Créations des points au hasard, recherche du point P le plus près pour chaque point N, le calcul des barycentres suite au regroupement. Or, on

ne peut pas paralléliser le traitement entre les itérations : ce serait une mauvaise réponse.

c) Croyez-vous qu'il soit possible d'améliorer la performance de l'algorithme **sur une seule image** en parallélisant les quatre parties entre elles ? Expliquez pourquoi et comment.

Réponse : Non, car chaque partie dépend du résultat de la précédente. Faire cela revient à faire du faux parallélisme, ou chaque bloc attend le résultat de l'autre. Si l'étudiant répond « en traitant deux images en même temps, il n'a pas compris les deux indications « sur une seule image ».

Mécanismes de synchronisation (GRO-300-1)

Question 7

En C++, pourquoi préconise-t-on l'usage de verrous comme `std::lock_guard` et `std::unique_lock` plutôt que d'appeler les fonctions « `lock()` » et « `unlock()` » directement sur un mutex ?

Réponse : Car ces verrous se déverrouille automatiquement lors de leur destruction. On peut donc organiser les sections critiques par blocs. Sinon, il faut s'assurer de toujours appeler `unlock()` pour chaque appel de `lock()`.

Question 8

Soit cet exemple (chaque colonne s'exécute dans un fil différent) :

```
int compteur_ = 0;  // Variable globale accessible aux deux fils.
```

```
void fil_A()
{
  while (true) {
    compteur_++;          // A1

    // Pause 1000 ms :
    usleep(1000 * 1000);
  }
}
```

```
void fil_B()
{
  while (true) {
    compteur_++;                  // B1
    printf("Nb. d'accès: %d\n", // B2
           compteur_);
    usleep(1000 * 1000);
  }
}
```

Est-ce que la ligne de code pointée par "A1" représente une condition critique ? Si oui, comment peut-on éliminer cette condition ?

Réponse : Oui, et en protégeant l'accès à cette section à l'aide de verrous. On peut encadrer la ligne par des accolades ("{}"), et créer un `std::lock_guard` sur un mutex (préalablement créé) au début de ce nouveau bloc. Notez que, pendant un véritable examen, dire un verrou / verrouiller un mutex serait suffisant. On ne cherche pas à avoir l'ortographe exacte de "lock_guard".

Question 9

Une fois toutes les sections critiques protégées dans ce code, est-ce la sortie du programme (l'affichage en ligne B2) sera toujours régulière (c.-à-d. des nombres entiers augmentant par 2 à chaque fois) ? Pourquoi ?

Réponse : Non, elle n'augmentera peut-être pas toujours par 2, par exemple si B1-B2 s'exécute 2 fois avant une nouvelle exécution de A1. Par contre, le nombre d'accès sera valide.

Question 10

À quel moment il est plus intéressant d'utiliser un système de variable de condition aux lieux d'utiliser seulement des verrous ?

Réponse : Lorsque on doit attendre après une événement comme une nouvelle événement dans une *queue*.

Mesures de performance (GRO-300-2)

Question 11

Nommez au moins deux raisons pour lesquelles le temps d'exécution d'un programme sur l'unité centrale (CPU) de votre système peut différer du temps réellement écoulé pour réaliser une tâche.

Réponse : Multiple, par exemple le temps d'attente mémoire, les entrées/sorties, partage du temps du CPU, etc.

Question 12

Pour une unité de traitement donnée, comment peut-on mesurer de façon absolue la durée d'exécution d'un programme ?

Réponse : En cycles d'horloge (*CPU clock cycles*).

Question 13

Un programme a été décomposé en trois classes d'instructions (A, B, et C) en donnant la proportion d'instructions appartenant à chaque classe. Le nombre de cycles d'horloge par instruction est donné pour deux unités centrales différentes (1 et 2).

Classe d'instruction	Proportion du programme	Cycles sur unité 1	Cycles sur unité 2
A	10 %	2	3
B	30 %	3	4
C	60 %	2	3

Si les unités 1 et 2 sont cadencées à 3,2 GHz et 3,1 GHz respectivement, laquelle exécutera le programme plus rapidement ? Détaillez vos calculs.

Réponse :
Cycles sur unité 1 : 10 * 2 + 30 * 3 + 60 * 2 = 230 cycles.
Cycles sur unité 2 : 10 * 3 + 30 * 4 + 60 * 3 = 330 cycles.
330 / 3,2 > 230 / 3,1 : l'unité 1 prendra plus de temps que l'unité 2. 4 points pour la bonne réponse (Unité 2),
12 points pour les détails (s'ils sont justes, - 4 points par erreur, -8 points s'ils ont oublié un élément important comme les proportions des instructions).

Pipelines (GRO-300-2)

Question 14

Est-ce qu'un pipeline augmente le débit ou la vitesse d'exécution des instructions d'un système ?

Réponse : Le débit des instructions.

Question 15

Donnez un exemple d'aléa de données dans un pipeline (*data hazard*).

Réponse : Lorsqu'une instruction a besoin des données du résultat d'une instruction précédente.

Question 16

Qu'est-ce qui distingue un aléa de données d'un aléa de contrôle ?

Réponse : Dans un aléa de contrôle, on n'a pas besoin d'attendre que le résultat soit enregistré avant de continuer l'exécution. De plus, on ne peut pas faire de prédiction pour diminuer les aléas de données.

Mémoire (GRO-300-2)

Question 17

Soit ce programme en C++:

```
00 const int LEN_S = 65536;
01 double signal_org[LEN_S] = {…}; // signal_org contient des
mesures.
02 double signal_fin[LEN_S];
03 double noyau[3] = {0.25, 0.50, 0.25};
04
05 signal_fin[0] = signal_org[0];
06 for (int i = 1; i < (LEN_S - 1); ++i) {
07   signal_fin[i] = 0.0;
08   signal_fin[i] += signal_org[i - 1] * noyau[0];
09   signal_fin[i] += signal_org[i] * noyau[1];
10   signal_fin[i] += signal_org[i + 1] * noyau[2];
11 }
12 signal_fin[LEN_S-1] = signal_org[LEN_S - 1];
```

Supposons que les accès en mémoire de ce programme compilé sont optimisés selon le principe de localité, mais seulement à partir de la ligne 5. Indiquez lesquelles des localités (aucune, spatiale, temporelle, ou les deux) ont été considérées et pourquoi, pour :

a) `signal_org[i - 1]` à la première exécution de la ligne 8 (`i = 1`) ?

Réponse : Temporelle, car `signal_org[0]` a été accédé à la ligne 5.

b) `signal_org[i]` à la première exécution de la ligne 9 (`i = 1`) ?

Réponse : Spatiale, car `signal_org[0]`, qui est adjacent à `signal_org[1]`, vient d'être référencé à la ligne 5.

c) `noyau[1]` à la première exécution de la ligne 9 (`i = 1`) ?

Réponse : Spatiale, car `noyau[0]`, qui est adjacent à `noyau[1]`, a été accédé à ligne 8.

d) noyau[1] à la seconde exécution de la ligne 9 (`i = 2`) ?

Réponse : Spatiale pour les mêmes raisons que c), mais aussi temporelle car elle a été accédée au cycle précédent.

Question 18

En IEEE 754, comment est représenté 20,1 ?

Réponse :

Signe : 1
Exp : 1000 000 0011
Mantisse : 0100 0001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1010

Instructions, registres, et cycles d'horloge (GRO-300-2)

Question 19

Soit ce programme en C++:

```
float prix_total;
float code_de_article = 10; // Fruits et Légumes
float prix = 42.69;
float tps = 0.05;
float tvq = 0.09975;

if (code_de_article != 10) {
    float prix_avec_tps = prix * tps;
    float prix_avec_tvq = prix * tvq;
    prix_total = prix_avec_tps + prix_avec_tvq;
} else {
    prix_total = prix;
}
```

Vous disposez de ces instructions, de 6 registres (R1 à R6), et d'une mémoire accessible par nom de variable :

- LDC Rx, VAL # Charge la constante VAL dans le registre Rx
- STO $A, Rx # Stocke Rx en mémoire à la variable $A
- ADD Ra, Rb # Ra = Ra + Rb

- SUB Ra, Rb # Ra = Ra – Rb
- MUL Ra, Rb # Ra = Ra * Rb
- BNE N, Ra, VAL # Branchement à l'adresse N si Ra != VAL
- BE N, Ra, VAL # Branchement à l'adresse N si Ra == VAL

Traduisez le programme en C en pseudo-assembleur à l'aide des instructions données. Associez d'abord chaque variable à un registre séparé en enregistre.

```
01  LDC R1, 10                    # float code_de_article = 10
02  LDC R2, 42.69                 # float prix = 42.69;
03  LDC R3, 0.05                  # float tps = 0.05;
04  LDC R4, 0.09975               # float tvq = 0.09975;

05  BE 11, R1, 10                 # if (code_de_article != 10) {
06  MUL R3, R2                    # float prix_avec_tps = prix * tps;
07  MUL R4, R2                    # float prix_avec_tvq = prix * tvq;
08  ADD R4, R3                    # prix_avec_tps + prix_avec_tvq;
09  STO $prix_total, R4           # prix_total =
10  BNE 12, R1, 10
11  STO $prix_total, R1           # prix_total = prix;
12
```