INF2610

TP #6 : Interblocage et Gestion de la mémoire

Polytechnique Montréal

Automne 2024
Date de remise: Voir le site Moodle du cours
Pondération: 5%

Description	Points
Interblocage (résultats)	8
Gestion de la mémoire (résultats)	12

Section 1 - Interblocage

Dans la première section du travail pratique, des blocs de code sont présentés avec des interblocages potentiels. Votre tâche est d'identifier si un interblocage existe dans le code.

Travail demandé

1. Dans chaque fichier .c, on vous demande d'énoncer les conditions d'interblocage qui sont satisfaites. Vous devez indiquer quelles lignes / blocs de code contribuent au comportement. Soyez le plus précis possible en identifiant le code responsable.
2. Si le code représente une situation commune d'interblocage présentée dans le cours, vous devez l'identifier par son nom.
3. Si un bloc de code contient un interblocage, vous devez faire la correction nécessaire pour l'éliminer. Ensuite, expliquez la solution par rapport aux conditions d'interblocage présentées dans le cours.

Pour les sections justificatives, on vous demande de mettre les résponses en commentaires dans le fichier .c.

Lancer le code

Vous pouvez faire un `make` dans le répertoire de la Section 1 pour créer des exécutables des fichiers `.c`. Ensuite, il suffit de prendre le nom du fichier et l'appeler dans le terminal de la manière suivante :

Exemple: Pour exécuter `deadlock1.c` après le `make` : `./deadlock1`

Section 2 - Gestion de la mémoire

Ce travail pratique vise à mieux comprendre le fonctionnement général de la gestion de la mémoire virtuelle. Il s'agit de complétéer un code fourni qui implémente partiellement un simulateur d'un système de gestion de mémoire **par pagination pure**.

Dans ce système, la taille d'une page / d'un cadre est fixée à 2^{10} (1024) octets.

Travail demandé

Le code fourni prend en charge la génération aléatoire des requêtes d'accès à la mémoire. Par contre, le traitement des requêtes est partiellement implémenté. Il manque le code des fonctions décrites ci-dessous. L'objectif de ces fonctions est d'implémenter la recherche d'une page dans le TLB et dans la table des pages. Votre participation à la réalisation du simulateur consiste donc à coder ces fonctions. Le code à compléter se trouve dans le fichier `GestionMemoire.c`.

Suivez les étapes suivantes pour compléter l'implémentation du simulateur:

- Complétez la fonction `calculerNumeroDePage` du fichier `GestionMemoire.c`. Cette fonction prend en paramètre une adresse virtuelle et doit renvoyer le numéro de page correspondant à cette adresse virtuelle. (Vous pouvez évidemment utiliser cette fonction pour les adresses physiques, car *la taille d'une page correspond à celle d'un cadre*.) (/1)
- Complétez la fonction `calculerDeplacementDansLaPage` du fichier `GestionMemoire.c`. Cette fonction prend en paramètre une adresse virtuelle et doit renvoyer le déplacement dans la page correspondant à cette adresse virtuelle. (Vous pouvez évidemment utiliser cette fonction pour les adresses physiques, car *la taille d'une page correspond à celle d'un cadre*.) (/1)
- Complétez la fonction `calculerAdresseComplète` du fichier `GestionMemoire.c`. Cette fonction prend en paramètres un numéro de cadre ainsi qu'un déplacement (*offset*) dans le cadre. Elle doit renvoyer l'adresse physique correspondant aux paramètres *numéo de cadre* et *déplacement dans le cadre*. (/1)
- Complétez la fonction `rechercherTLB` du fichier `GestionMemoire.c`. Cette fonction prend en paramètres une structure `RequeteMemoire (req)` et une structure `SystemeMemoire (mem)`. Elle doit parcourir le TLB à la recherche de la page indiquée dans le paramètre `req`. Si la page est présente dans le TLB, elle doit calculer l'adresse physique correspondant à l'adresse virtuelle indiquée dans `req` pour la stocker dans l'attribut `adressePhysique` de `req`. Elle doit mettre également à jour la date de dernier accès à l'entrée du TLB à la date de la requête (attribut `date` de `req`), et éventuellement les autres attributs. Si la page n'est pas dans le TLB, elle doit stocker la valeur 0 dans l'attribut `adressePhysique` de `req`. (/2)
- Complétez la fonction `rechercherTableDesPages` du fichier `GestionMemoire.c`. Cette fonction prend en paramètres une structure `RequeteMemoire (req)` et une structure `SystemeMemoire (mem)`. Elle doit vérifier si la page indiquée dans le paramètre `req` est présente en mémoire. Si c'est le cas, elle doit calculer et stocker, dans l'attribut `adressePhysique`, l'adresse physique correspondant à l'adresse virtuelle et mettre à jour éventuellement les autres attributs de `req`. Sinon, c'est la valeur 0 qui doit être stockée dans l'attribut `adressePhysique`. Il n'y a aucune date à modifier dans cette fonction. (/2)
- Complétez la fonction `ajouterDansMemoire` du fichier `GestionMemoire.c`. Cette fonction prend en paramètres une structure `RequeteMemoire (req)` et une structure `SystemeMemoire (mem)`. Elle doit ajouter la page, indiquée dans le paramètre `req`, dans le premier cadre disponible en mémoire principale. Les dates de création et de dernier accès de la page sont identiques à la date de la requête (c'est-à-dire l'attribut `date` du paramètre `req`. *Attention*: cette fonction suppose que la mémoire n'est pas pleine lorsqu'elle est appelée. Elle ne considère pas le cas où la mémoire est pleine. Ce dernier cas nécessite un remplacement de page. Il est donc demandé de coder seulement l'ajout en mémoire. (/2)

On vous informe maintenant que la politique de remplacement du TLB est FIFO (First In First Out).

- Complétez la fonction `mettreAJourTLB` du fichier `GestionMemoire.c`. Cette fonction prend en paramètres une structure `RequeteMemoire (req)` et une structure `SystemeMemoire (mem)`. Elle doit d'abord ap-

pliquer la politique FIFO pour déterminer l'entrée dans le TLB qui va être remplacée. Elle doit ensuite remplacer les informations dans cette entrée par celles déduites de `req`. (/3)

ATTENTION

Toutes les structures sont initialisées. Vous n'avez pas à allouer de la mémoire avec `malloc` ou `calloc`.

ATTENTION

Vous avez accès à la taille du TLB avec la macro `TAILLE_TLB`, à la taille de la mémoire avec la macro `TAILLE_MEMOIRE`, et à la taille de la table des pages avec la macro `TAILLE_PT`. Si vous avez un pointeur `mem` sur la structure `SystemeMemoire`, le bit de validité de la j^{ieme} entrée du TLB peut être désigné par:

`mem->tlb->entreeValide[j].`

Voici la définition de chaque structure:

```
struct RequeteMemoire {
  u_int8_t estDansTLB;            // 0 ou 1
  u_int8_t estDansTablePages;     // 0 ou 1
  unsigned long adressePhysique;
  unsigned long adresseVirtuelle ;
  unsigned long date;
};

struct SystemeMemoire {
  struct TLB* tlb;
  struct TablePages* tp;
  struct Memoire* memoire;
};

struct TLB {
  unsigned int* numeroPage;
  unsigned int* numeroCadre;
  u_int8_t* entreeValide; // pour le bit de validite
  unsigned long* dernierAcces;
  unsigned long* dateCreation;
};

struct TablePages {
  unsigned int* numeroCadre;
  u_int8_t* entreeValide; // pour le bit de presence
};

struct Memoire {
  unsigned int* numeroPage;
  unsigned long* dateCreation;
  unsigned long* dernierAcces;
  u_int8_t* utilisee; // pour le bit de reference
};
```

Test et remise du TP

Utilisez la commande `make` pour compiler votre programme. Il vous est toujours possible de tester et d'évaluer successivement vos fonctions en exécutant le script `./evalue.sh` ou l'exécutable de votre code `./answer`. Si

l'évaluateur ne fonctionne pas comme prévu pour vous, ne vous inquiétez pas, vos chargés de laboratoire vont vérifier votre code.

À la fin du TP, lancez la commande `make handin` dans le répertoire principal du TP afin de créer l'archive `handin.tar.gz` que vous devez remettre sur Moodle (après avoir inséré vos matricules).