PROGRAMMATION ORIENTÉE SYSTÈME

Exercice Noté 2

2 avril 12:00 – 14 avril 23:59

INSTRUCTIONS (à lire attentivement)

IMPORTANT ! Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre note annulée dans le cas contraire.

1. Cet exercice doit être réalisée **individuellement !** L'échange de code relatif à cet exercice noté est **strictement interdit** ! Cela inclut la diffusion de code sur le forum.

   Le code rendu doit être le résultat de **votre propre production**. Le plagiat de code, de quelque façon que de soit et quelle qu'en soit la source sera considéré comme de la tricherie (c'est, en plus, illégal et passible de poursuites pénales).

   En cas de tricherie, vous recevrez la note «NA» pour l'entièreté de la branche et serez de plus dénoncés et punis suivant l'ordonnance sur la discipline.
2. Vous devez donc également garder le code produit pour cet exercice dans un endroit à l'accès strictement personnel.
3. Le fichier à fournir pour cet exercice est `vm.c`. Il ne devra plus être modifié après la date et heure limite de rendu.
4. Veuillez à rendre du code anonyme : **pas** de nom ni numéro SCIPER !
5. Utilisez le site Moodle du cours pour rendre votre exercice.

   Vous avez **jusqu'au 14 avril 23:59** (heure du site Moodle du cours faisant foi) pour soumettre votre rendu.
6. Lisez attentivement et *complètement* la question de façon à ne faire que, mais tout, ce qui vous est demandé.

   Si la donnée ne vous paraît pas claire, ou si vous avez un doute, demandez des précisions sur le forum.
7. Pour cet exercice en langage C, vous êtes libres de choisir le standard C11, C99 ou C89. Merci d'indiquer votre choix dans un commentaire en début de fichier.
8. Ne faites **PAS** d'implémentation séparée (i.e. **pas** de fichier « `.h` », ni de `Makefile`) : ne rendre qu'un seul fichier ici.

EXERCICE (C) : Simulateur de machines virtuelles

1 Cadre et types de données

Une machine virtuelle (VM) permet de simuler le fonctionnement d'un microprocesseur et de son environnement sur une machine de type différent. Par exemple, elle permet de faire tourner directement sur un PC des jeux pour console, en émulant l'environnement d'origine : le jeu « croit » qu'il tourne sur une vraie console.

Une VM modélise le processeur, la mémoire et le programme (on ne modélise pas ici le stockage en mémoire du programme : programme et mémoire seront deux entités distinctes).

La mémoire est modélisée comme un tableau (dynamique) d'entiers et le programme comme un tableau (dynamique) de pointeurs sur des instructions, lesquelles sont spécifiées plus bas.

Le processeur est assimilable à une structure de données contenant un pointeur (de pointeur) d'instruction et quelques registres (à spécifier lors de sa construction), lesquels contiennent des nombres entiers. Le processeur travaille dans une boucle qui lit une instruction dans le programme, l'exécute et passe à la suivante en incrémentant son pointeur (de pointeur) d'instruction.

Une instruction peut être modélisée par une structure contenant un nom, un tableau contenant les numéros (type `size_t`) des registres à manipuler et un pointeur sur une fonction effectuant effectivement l'opération. Par exemple, l'addition pourrait contenir un tableau de trois numéros de registres, a, b, et c, et un pointeur sur une fonction effectuant l'addition des registres a et b et mettant le résultat dans le registre c. Cet exemple est détaillé plus bas.

Autre exemple : une instruction de lecture mémoire reçoit un tableau contenant l'adresse mémoire à lire et le numéro du registre devant accueillir la donnée lue et un pointeur sur une fonction effectuant effectivement la copie.

Définissez les types modélisant :
– un processeur, avec ses registres et son pointeur d'instruction ;
– une instruction, avec son nom, le tableau des numéros de ses registres et sa fonction ;
– la mémoire ;
– un programme, avec ses instructions ;
– une machine virtuelle, avec son processeur, sa mémoire et son programme à exécuter.
Vous pouvez utiliser tout type de données supplémentaire que vous jugez utile, en donnant simplement un nom aux autres types plus avancés que vous définiriez.

Faites une conception la plus souple possible en termes de nombre de registres, taille de la mémoire, etc. Par contre, on supposera que toutes les instructions n'ont besoin que de *trois* registres au maximum :

```
#define INSTR_MAX_REG 3
```

Pour les noms des instructions, on les limitera également :

```
#define INSTR_NAME_SIZE 64
```

mais vous devrez bien sûr garantir que cette limite n'est pas dépassée !

2 Représentation et création des instructions

Une « instruction » (au sens défini ci-dessus) représente une instruction élémentaire, atomique, de la VM et est définie par son nom, ses registres et la fonction réalisant effectivement cette instruction dans la VM simulée.

Par exemple, l'instruction réalisant l'addition des registres 1 et 2 dans le registre 3 aurait typiquement :
– comme nom `"ADD(1,2,3)"` ;

– comme tableau de positions de registres, un tableau de `size_t` contant les valeurs `1`, `2` et `3` ;
– et comme (pointeur sur) fonction, (un pointeur sur) la fonction suivante :

```
void do_add(VM* vm, size_t reg_pos[INSTR_MAX_REG]);
```

dont l'effet serait de vérifier que les numéros des registres reçus sont compatibles avec la VM puis faire l'addition, typiquement comme ceci :

```
p_vm->proc.regs[reg_pos[2]] =   p_vm->proc.regs[reg_pos[0]]
                              + p_vm->proc.regs[reg_pos[1]] ;
```

(`proc` est le processeur de la machine virtuelle).

Il sera important, pour chaque famille d'instructions voulue, de bien distinguer deux sortes de fonctions-outils :
– les fonctions permettant la réalisation effective, sur la VM, de l'instruction voulue ; comme par exemple la fonction `do_add` ci-dessus ;
– des fonctions créant des instructions concrètes dans la famille désirée ; par exemple, on pourra vouloir avoir une fonction `create_add` qui prendrait trois numéros de registres et créerait (un pointeur sur) l'instruction représentant l'addition des deux premiers registres reçus dans le troisième ; (un pointeur sur) l'instruction précédente utilisée pour l'exemple ci-dessus aurait alors été crée via l'appel à `create_add(1,2,3)`.

3 Code à fournir

Écrivez les fonctions suivantes :

1. `do_add(VM* vm, size_t regs[INSTR_MAX_REG]);`
   cette fonction permet de faire l'addition des valeurs contenues dans deux registres et de stocker le résultat dans un troisième, comme expliqué dans la section précédente ;
2. `do_memwrite(VM* vm, size_t regs[INSTR_MAX_REG]);`
   cette fonction permet de simuler l'écriture de la valeur d'un registre, dont on a donné le numéro en première position du tableau `regs`, dans la case mémoire dont l'adresse est stockée en seconde position du tableau `regs` ;
3. `do_memread(VM* vm, size_t regs[INSTR_MAX_REG]);`
   cette fonction permet de lire la valeur de la case mémoire dont l'adresse est stockée en première position du tableau `regs` et de l'écrire dans le registre dont le numéro est stocké en seconde position du tableau `regs` ;
4. ```
   Instr* create_add(size_t nb1, size_t nb2, size_t nb3);
   Instr* create_write(size_t reg_nb, size_t where);
   Instr* create_read(size_t where, size_t reg_nb);
   ```
   ces fonctions créent des instructions permettant respectivement :
   – d'ajouter les registres `nb1` et `nb2` dans `nb3` :
   – d'écrire le registre `reg_nb` à l'adresse mémoire `where` ;
   – de lire l'adresse mémoire `where` dans le registre `reg_nb` ;
5. `void exec_instr(VM* p_vm, Instr* p_i);`
   cette fonction réalise l'exécution de l'instruction `p_i` sur la VM `p_vm` ; hormis d'éventuels messages d'affichage pour suivre ce qui se passe (cf exemple de déroulement fourni plus bas), cette fonction s'écrit vraiment en une seule ligne simple ;
6. `void display_vm(VM* p_vm);`
   cette fonction affiche le contenu de la machine virtuelle : le contenu des registres du processeur, le pointeur d'instruction du processeur et le contenu de la mémoire (cf exemple de déroulement plus bas) ;
   ```

7. `void run(VM* p_vm);`
   cette fonction lance l'exécution de la VM, c.-à-d. tant qu'il y a des instructions à exécuter dans le programme de la VM, exécute ces instructions, affiche l'état de la VM (cf exemple de déroulement plus bas) et passe à l'instruction suivante ;
8. `VM* create_vm(size_t nb_regs, size_t mem_size);`
   cette fonction crée les structures de données nécessaires à une machine virtuelle et les initialise.

Vous pouvez bien sûr ajouter toutes les fonctions outils supplémentaires que vous jugez nécessaires pour vous aider dans la modularisation de votre programme (il y en a sûrement qui n'ont pas été mentionnées ci-dessus...).

Enfin, dans la fonction `main()`, créez une machine virtuelle et exécutez un programme qui fait l'addition de deux valeurs contenues dans deux cases mémoire et écrit le résultat dans une troisième (case mémoire).

4 Exemple d'exécution

Si la mémoire a le contenu suivant (format `ADRESSE -> CONTENU`) :

```
44 -> 427
...
56 -> 128
...
123 -> 333
```

et que le programme donné à la VM est :

```
READ(44,1)   // Lit la zone mémoire 44 dans le registre 1
READ(56,2)   // Lit la zone mémoire 56 dans le registre 2
ADD(1,2,3)   // Additionne les registres : R1 + R2 -> R3
WRITE(3,123) // Ecrit le registre 3 dans la zone mémoire 123
```

alors le `main()` pourrait typiquement afficher :

```
================================================
Au départ :
------------------------------------------------
Etat des registres :
  0 -> 0
  1 -> 0
  2 -> 0
  3 -> 0
------------------------------------------------
Pointeur d'instruction :
  pas de programme
------------------------------------------------
Etat de la mémoire :
  0 -> 0
  ...
  44 -> 0
  ...
  56 -> 0
  ...
  123 -> 0
  ...
================================================
Programme chargé et mémoire initialisée :
------------------------------------------------
Etat des registres :
  0 -> 0
```

```
  1 -> 0
  2 -> 0
  3 -> 0
-------------------------------------------------
Pointeur d'instruction :
  vers READ(44,1)
-------------------------------------------------
Etat de la mémoire :
  0 -> 0
  ...
  44 -> 427
  ...
  56 -> 128
  ...
  123 -> 333
  ...
=================================================
J'exécute : READ(44,1)
-------------------------------------------------
Etat des registres :
  0 -> 0
  1 -> 427
  2 -> 0
  3 -> 0
-------------------------------------------------
Pointeur d'instruction :
  vers READ(56,2)
-------------------------------------------------
Etat de la mémoire :
  0 -> 0
  ...
  44 -> 427
  ...
  56 -> 128
  ...
  123 -> 333
  ...
=================================================
J'exécute : READ(56,2)
-------------------------------------------------
Etat des registres :
  0 -> 0
  1 -> 427
  2 -> 128
  3 -> 0
-------------------------------------------------
Pointeur d'instruction :
  vers ADD(1,2,3)
-------------------------------------------------
Etat de la mémoire :
  0 -> 0
  ...
  44 -> 427
  ...
  56 -> 128
  ...
  123 -> 333
  ...
=================================================
J'exécute : ADD(1,2,3)
-------------------------------------------------
```

```
Etat des registres :
  0 -> 0
  1 -> 427
  2 -> 128
  3 -> 555
-------------------------------------------------
Pointeur d'instruction :
  vers WRITE(3,123)
-------------------------------------------------
Etat de la mémoire :
  0 -> 0
  ...
  44 -> 427
  ...
  56 -> 128
  ...
  123 -> 333
  ...
=================================================
J'exécute : WRITE(3,123)
-------------------------------------------------
Etat des registres :
  0 -> 0
  1 -> 427
  2 -> 128
  3 -> 555
-------------------------------------------------
Pointeur d'instruction :
  vers FIN DE PROGRAMME
-------------------------------------------------
Etat de la mémoire :
  0 -> 0
  ...
  44 -> 427
  ...
  56 -> 128
  ...
  123 -> 555
  ...
=================================================
Libération de la mémoire.
```