Denis Morency

De: no-reply@www.usherbrooke.ca
Envoyé: 9 avril 2015 14:22
À: Sciences-CentreImpression@USherbrooke.ca
Objet: COMMANDE EXAMENS
Pièces jointes: IFT209_Final_H2015.pdf

TYPE-EXAMEN	FINAL
SIGLE-COURS	IFT209
TITRE-COURS	Programmation système
PROFESSEUR	Richard St-Denis
DATE-HEURE	Jeudi 23 avril 2015 à 9 h
AUTORISE-PAR	Gabriel Girard et Richard St-Denis
NOMBRE-PAGES	12
NOMBRE-COPIE-PROF	39
IMPRESSION-QUESTIONNAIRE	Recto-verso plié broché en cahier
NOMBRE-FEUILLES-BLANCHES	
NOMBRE-PAPIER-GRAPHIQUE	
NOMBRE-CAHIERS	
CONSENTEMENT-AGES	1
REMARQUES	Les examens doivent être faits en cahier. Une copie doit être remise à Véronique Morin pour un étudiant en situation particulière. Richard.St-Denis@usherbrooke.ca
E-MAIL	
FIRST-NAME	
LAST-NAME	
NICK-NAME	
SPAMSHIELD	true

UNIVERSITÉ DE SHERBROOKE

DÉPARTEMENT D'INFORMATIQUE

IFT 209

Programmation système

EXAMEN FINAL[1]

Le jeudi 23 avril 2015 de 9 h 00 à 12 h 00 Professeur : Richard St-Denis

- Toute documentation est permise.
- **Tout appareil électronique, incluant le téléphone cellulaire et l'ordinateur, est interdit.**
- Ne dégrafez pas ce questionnaire.
- Répondez dans les espaces prévus à cet effet.
- **Personne ne peut quitter la salle d'examen avant 9 h 15.**
- **Personne ne peut quitter son siège entre 11 h 50 et 12 h 00.**
- La correction est, entre autres, basée sur le fait que chacune de vos réponses soit :
  - claire, c'est-à-dire lisible et compréhensible pour le correcteur ;
  - précise, c'est-à-dire exacte ou sans erreur ;
  - complète, c'est-à-dire que toutes les étapes de résolution du problème sont présentes ;
  - concise, c'est-à-dire que la méthode de résolution est la plus courte possible.
- **Les registres de l'interface RS-232 sont fournis sur une feuille séparée.**

Nom : _____ Prénom : _____

Signature : _____ Matricule : _____

Question	Barème
1	/6
2	/6
3	/6
4	./6
5	/6
total :	/30

1. Ce questionnaire comporte douze (12) pages.

1. (6 points) Voici le sous-programme de recherche dichotomique utilisé dans le laboratoire #2. Il s'agit d'une version itérative qui compare la valeur recherchée (N) avec celle de la donnée (T[mid]) au centre d'une table ordonnée (T). S'il y a égalité, la valeur N est trouvée, sinon la recherche se poursuit dans la partie inférieure ou supérieure de la table selon que la valeur N est plus petite ou plus grande que celle de T[mid].

```
/* find:   sous-programme iteratif de recherche dichotomique
   entree: %i0, la valeur recherchee (N)
           %i1, la taille de table en mots (T)
           %i2, l'adresse de la table(A)
   sortie: %o0, le rang du nombre dans la table (entre 1 et N) ou 0 si le nombre est absent */
        .section ".text"
find:
        save    %sp,-208,%sp
        mov     1,%l1           ! low
        mov     %i1,%l2         ! high
find00:
        cmp     %l1,%l2
        bgu     %xcc,find15     ! low > high
        nop

        add     %l1,%l2,%l3     ! mid := (low+high)/2
        udivx   %l3,2,%l3
        mulx    %l3,4,%l4       ! deplacement
        sub     %l4,4,%l4
        lduw    [%i2+%l4],%l5   ! A[mid]
        cmp     %i0,%l5
        be      %xcc,find10     ! N = A[mid]
        nop

        bgu     %xcc,find05     ! N > A[mid]
        nop
        sub     %l3,1,%l2       ! high := mid - 1
        ba      find00
        nop
find05:
        add     %l3,1,%l1       ! low := mid + 1
        ba      find00
        nop

find10: ret
        restore %g0,%l3,%o0
find15: ret
        restore %g0,0,%o0
```

Dans cette question, vous devez écrire une version récursive du sous-programme de recherche dichotomique. Complétez les parties suivantes **en commentant votre code**. Il est possible d'avoir plus d'une séquence de `ret` et `restore`.

```
/* find:   sous-programme recursif de recherche dichotomique
   entree: %i0, la valeur recherchee
           %i1, la taille de table en mots
           %i2, l'adresse de la table
   sortie: %o0, le rang du nombre dans la table (entre 1 et N) ou 0 si le nombre est absent */
        .section ".text"
find:
        save    %sp,-208,%sp
```

Cas de base (1 point) : _____

Prétraitement (ou coeur du sous-programme) (2 points) : _____

Branchement conditionnel vers l'un ou l'autre des deux cas ($\frac{1}{2}$ point) : _____

(suite à la page suivante)

Appel récursif du premier cas (1 point) :

Post-traitement de l'appel récursif du premier cas ($\frac{1}{2}$ point) :

Appel récursif du deuxième cas (1 point) :

Post-traitement de l'appel récursif du deuxième cas ($\frac{1}{2}$ point) :

Retour à l'appelant ($\frac{1}{2}$ point) :

2. (6 points) Voici quatre suites d'instructions en langage d'assemblage *SPARC*.

suite A

```
setx 0XFFFF,%l0,%l1
sllx %l1,48,%l0
andn %l7,%l0,%l0
andn %l7,%l1,%l1
and  %l0,%l1,%l7
```

suite B

```
    setx   0XF000000000000000,%l1,%l0
    srax   %l0,12,%l0
    srlx   %l0,48,%l1
    or     %l0,%l1,%l0
    and    %l7,%l0,%l1
    popc   %l1,%l1
    brnz,a %l1,a
    andn   %l7,%l0,%l7
a:  nop
```

suite C

```
    setx x,%l1,%l0
    srlx %l7,16,%l7
    sth  %l7,[%l0+2]
    srlx %l7,16,%l7
    sth  %l7,[%l0]
    ldx  [%l0-2],%l7
    (...)
    .align 8
    .half  0
x:  .skip  4
    .half  0
```

suite D

```
srlx %l7,16,%l7
sllx %l7,16,%l7
srlx %l7,16,%l7
```

Trois d'entre elles donnent le même résultat (dans le registre %l7).

Quelles sont-elles ? (1 point) _____

_____

_____

Quel est le résultat obtenu (par rapport au contenu du registre %l7) ? (1 point)

_____

_____

Indiquez ou illustrez brièvement pour chacune d'elles le traitement réalisé, incluant la suite d'instructions qui fait un traitement différent des autres (4 points).

_____

_____

_____

_____

_____

_____

_____

_____

_____

(espace disponible sur la page suivante)

3. (6 points) Répondez aux trois questions suivantes et justifiez vos réponses.

a) (2 points) Quelle est la différence entre un bit et un byte ?

b) (2 points) Donnez deux raisons de la présence d'une pile associée à tout programme en exécution (un processus).

c) (2 points) Voici le programme utilisé dans la deuxième partie du laboratoire #6.

```
        .section ".text"
main:
        setx    ptfmt1,%l7,%o0
        call    printf          ! impression d'une message
        nop
        setx    cmpt,%l7,%l0
        jmpl    %l0,%l1
        inc     32,%l1
        inc     8,%l0           ! mise a jour
        inc     8,%l0
        inc     8,%l0
        inc     8,%l0
        inc     8,%l0
        inc     8,%l0
        add     %l0,8,%o0
        call    printf          ! impression d'une message
        nop
        clr     %o0             ! 1er parametre
        call    exit            ! sortie du programme
        nop

        .section ".rodata"
ptfmt1: .asciz  "Debut de la mise a jour des guichets automatiques.\n"
        .align  4
cmpt:   .word   0X81C44000
        .word   0X01000000
        .word   0X496C6C65
        .word   0X67616C20
        .word   0X696E7374
        .word   0X72756374
        .word   0X696F6E0A
        .word   0X00FF1074
```

Expliquez avec le plus de précision possible, soit en commentant le code d'une manière lisible et compréhensible pour le correcteur, soit en écrivant clairement votre réponse dans l'espace suivant, pourquoi le message *Illegal instruction* s'affiche lors de l'exécution du programme.

(espace disponible sur la page suivante)

4. (6 points) Voici le sous-programme `uart0Putch` pour une sortie d'un caractère ASCII **par interrogation d'un bit d'état** sur l'interface série RS-232 tiré de l'exemple fourni par le distributeur de la carte DEV-00268.

```
int uart0Putch(int ch)
{
  while (!(UART0_LSR & ULSR_THRE))      // wait for TX buffer to empty
    continue;

  UART0_THR = (uint8_t)ch;              // put char to Transmit Holding Register
  return (uint8_t)ch;                   // return char
}
```

Le symbole `UART0_LSR` est remplacé par l'adresse numérique `0xE000C014` (l'adresse du registre d'état de la ligne ou *Line Status Register* en anglais) et le symbole `ULSR_THRE`, un masque, est remplacé par l'expression `(1 << 5)`, un décalage de cinq positions vers la gauche de la constante `1`, lors de la compilation du programme. Les opérateurs « `&` » et « `!` » sont respectivement le ET (`and`) bit à bit et le NON logique (`not`). Enfin, le symbole `uint8_t` est équivalent au type `signed char`. Toute donnée de ce type tient sur un octet. Vous devez modifier ce sous-programme de façon à transmettre un caractère de l'ensemble de codes EUC-JP via l'interface série RS-232 branchée à un terminal qui supporte cet ensemble de codes.

Un caractère de l'ensemble de codes EUC-JP tient sur un, deux ou trois octets. Le tableau suivant donne toutes les possibilités. Par exemple, si un caractère tient sur deux octets et que le premier octet est 10001110_2, alors le bit le plus significatif du deuxième octet est 1 et les sept bits les moins significatifs du deuxième octet représentent le code d'un caractère de l'ensemble JISX0201.

n^{bre} d'octets	Code binaire	Ensemble
1	0xxxxxxx	ASCII
2	1xxxxxxx 1xxxxxxx	JISX0208
2	10001110 1xxxxxxx	JISX0201
3	10001111 1xxxxxxx 1xxxxxxx	JISX0212

Bien indiquer le ou les paramètres du sous-programme modifié. Vous pouvez écrire votre solution en pseudocode **très** proche du langage de programmation *C*, en langage de programmation *C* ou en langage d'assemblage *SPARC* (en supposant que ce processeur permet la même programmation de l'interface série RS-232 que sur un processeur *ARM*). Votre code doit être commenté.

(répondez sur la page suivante)

5. (6 points) Voici une description partielle du cheminement d'un caractère ASCII lu à partir du clavier d'un terminal RS-232 dont **les entrées/sorties sont traitées par interruption**.

   — Une personne appuie sur une touche du clavier.

   — Le caractère correspondant est transmis, un bit à la fois, via le câble qui relie le terminal à l'ordinateur.

   — L'interface RS-232 de l'ordinateur reçoit les bits un à un pour reconstituer le caractère qu'elle place à la queue de sa file (tampon) d'entrée interne.

   — Puis après un moment, le caractère est à la tête de la file d'entrée interne de l'interface RS-232, c'est-à-dire dans le registre *Receiver Buffer Register* (RBR).

   — Il y a alors une interruption de type `RDA` (la valeur des bits *IIR1* à *IIR3* du registre *Interrupt Identification Register* (IIR) est `010`) et un branchement vers le sous-programme de gestion d'interruptions en provenance de l'interface RS-232.

   — Le sous-programme de gestion d'interruptions transfère le caractère contenu dans le registre RBR à la queue d'une file d'entrée propre au système d'exploitation.

   a) (4 points) Complétez le cheminement du caractère de façon à ce qu'il soit éventuellement lu par un programme d'application.

b) (2 points) Nous savons qu'un programme d'application qui effectue des entrées et des sorties à partir d'une interface RS-232, via le mécanisme de gestion des interruptions, fonctionne d'une manière asynchrone par rapport au périphérique (ici un terminal), la vitesse de transmission des caractères étant de beaucoup inférieure à celle du processeur qui exécute le programme. Ceci contraste avec le cas dans lequel les entrées/sorties sont réalisées par interrogation de bits d'état, le périphérique et le programme fonctionnant d'une manière synchrone dans ce cas précis.

Supposons maintenant qu'il n'y a qu'un **seul** programme d'application qui s'exécute sur l'ordinateur en plus du système d'exploitation. Ce dernier inclut : i) un ensemble de services (accessibles en provoquant des interruptions logicielles), en particulier la lecture (comme SYS_read) et l'écriture (comme SYS_write) d'une chaîne de caractères via une interface série RS-232 ainsi que ii) les divers sous-programmes de gestion des interruptions. Quelle est la partie de code qui s'exécute le plus longtemps si le programme d'application fait principalement des entrées/sorties, donc très peu de calculs ? Est-ce le programme d'application, le gestionnaire des interruptions d'entrée/sortie, le service pour la lecture et l'écriture d'une chaîne de caractères ou une autre partie du système d'exploitation ? Justifiez très bien votre réponse.

FIN DE L'EXAMEN